Local GDP Estimates Around the World*

Esteban Rossi-Hansberg
University of Chicago

Jialing Zhang
University of Chicago

February 2, 2025

Replication Steps

This file provides a detailed step-by-step guide to replicating our estimates at each resolution.

Preparation Checklist:

1. **Hardware Requirements**: A computer with at least 10 cores to support parallel processing.
2. **QGIS Installation**: Install QGIS version 3.34.11 from `https://download.qgis.org/downloads/`.
   - For Mac users, download the file macos/ltr/qgis_ltr_final-3_34_11_20240913_170535.dmg.
   - Ensure QGIS is installed on the same machine where you will run the scripts. If using a server to execute the scripts, QGIS must also be installed on the server.
3. **Locale Settings**: Set the system locale to `"en_US.UTF-8"` to ensure consistent handling of character encoding, numeric formatting, and date/time formats. For example, for R users:
   - `Sys.getlocale()`
   - `Sys.setlocale("LC_ALL", "en_US.UTF-8")`

Contents

*Esteban Rossi-Hansberg: earossih@uchicago.edu. Jialing Zhang: jialingzhang@uchicago.edu

1 Global Countries Geometry Shapefile

1. **Obtain and Load Data:** Download the GADM version 4.1 geometry data file as Appendix Section 1.1 describes and read the layer `ADM_0` in R using the command `read_sf`.

2. **Merge Separated Geometry Features:**

   In the geometry data, some features belonging to China, India, or Pakistan are fragmented and separated from their main geometry, so we need to combine them into one line of geometry feature.

   Filter the geometry data to isolate features belonging to China, India, and Pakistan using `filter(COUNTRY %in% c("China", ‘‘India", ‘‘Pakistan"))`.

   Group the features by the `COUNTRY` column and merge them using `summarize( geom = st_union(geom))`.

   Rename the resulting `GID_0` column to `CHN` for China, `IND` for India, and `PAK` for Pakistan using `mutate(GID_0 = case_when(...))`.

3. **Retain Other Countries' Features:** Filter the data to exclude China, India, and Pakistan, and then combine the remaining features with the updated geometries for these three countries using `bind_rows`.

4. **Exclude Antarctica:** Remove Antarctica from the geometry data using `filter(GID_0 != "ATA")`.

5. **Save the Output:** Save the resulting shapefile to `gadm_410_countrylevel0_dissolve_union.gpkg` using the `st_write` command.

2 Global Province-level Geometry Shapefile

1. **Obtain and Load Data:** Now read the GADM version 4.1 geometry data file for the layer `ADM_1` using `read_sf`.

2. **Process China, India, and Pakistan:** Similarly as the Step 1, filter the geometry data for features belonging to China, India, and Pakistan. Group the features by `COUNTRY`, `NAME_1`, and `ENGTYPE_1`, and merge fragmented geometries using `summarize(geom = st_union(geom))`. Rename the `GID_0` column to `CHN`, `IND`, and `PAK` based on the country using `mutate(case_when(...))`.

3. **Process Other Countries:** Filter the remaining features for countries other than China, India, and Pakistan. Retain the `GID_0`, `COUNTRY`, `NAME_1`, and `ENGTYPE_1` columns, and combine these with the processed data for China, India, and Pakistan using `bind_rows`.

4. **Exclude Antarctica:** Remove Antarctica from the data using `filter(GID_0 != "ATA")`.

5. **Save the Output:** Save the processed shapefile as gadm_410_prov_level1_dissolve_union.gpkg using st_write.

3 Remove Large Inland Waters From the Geometry Shapefiles

1. **Download QGIS to Use the qgisprocess R Library:** Install QGIS version 3.34.8 from https://download.qgis.org/downloads/. For Mac users, download at "macos/ltr/qgis_ltr_final–3_34_11_20240913_170535.dmg". Ensure QGIS is installed on the same machine where you run the qgisprocess library. If you use a server to execute the scripts, QGIS must be installed on the server.

2. **Download Large Lakes Shapefile:** Obtain the global large lakes shapefile as described in Appendix Section 1.1. This file will be referred to as glwd_1.shp in subsequent steps.

3. **Process Country-Level Geometry:** Use the qgis_run_algorithm function with the native:difference algorithm to remove large lakes. The input file is the result of Step 1: "gadm_410_countrylevel0_dissolve_union.gpkg", the overlay file is glwd_1.shp, and the output is gadm_country_level0_without_largerwater.gpkg.

4. **Process GADM Province-Level Geometry:** Apply the qgis_run_algorithm function with the native:difference algorithm to remove large lakes. The input file is "gadm_410_prov_level1_dissolve_union.gpkg" (the result of Step 2), the overlay file is glwd_1.shp, and the output is gdam_prov_level1_without_largewater.gpkg.

5. **Process CGAZ Province-Level Geometry:** Use the qgis_run_algorithm function with the native:difference algorithm to remove large lakes. The input file is the CGAZ-ADM1 geometry geojson file (download it as described in the Appendix Section 1.1), the overlay file is glwd_1.shp, and the output is CGAZ_ADM1_without_large_waters.gpkg.

6. **Process County-Level Geometry:** Use the qgis_run_algorithm function with the native:difference algorithm to remove large lakes. The input file is the GADM version 4.1 geometry data file layer ADM_2, the overlay file is glwd_1.shp, and the output is gdam_county_level2_without_water_CHN_IND_PAK_not_union.gpkg.

4 Russia and Brazil Regional GDP Data

Here are the steps to organize regional GDP data before converting to cell-level data. Note: Adjustments may be required when updating to newer years or sourcing data from different websites.

1. Russia's regional data after 2019:

Download Data: Obtain Russia's regional GDP data (post-2019) as detailed in Appendix Section 1.2.1. Save it as RUS.xlsx

Select and Rename Columns: Retain the columns id, name, region_gdp, and year. Rename them to admin_2_id, admin_2_name, and admin_2_rgdp_total, respectively.

Add New Columns: Create columns iso and admin_1_name with values "RUS" and "Russia", respectively.

Calculate Total GDP: Add a column admin_1_rgdp_total by summing admin_2_rgdp_total for each group defined by iso and year.

Remove the grouping

Save: Save the results as RUS_2020_2021.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

2. Brazil's regional GDP data for 2021:

Download Data: Obtain Brazil's regional GDP data for 2021 as described in Appendix Section 1.2.1 and save it as PIB_Otica_Renda_UF.xls.

Select Sheets: Process the following sheets: Tabela3, Tabela4, Tabela5, Tabela6, Tabela7, Tabela8, Tabela9, Tabela11, Tabela12, Tabela13, Tabela14, Tabela15, Tabela16, Tabela17, Tabela18, Tabela19, Tabela21, Tabela22, Tabela23, Tabela24, Tabela26, Tabela27, Tabela28, Tabela30, Tabela31, Tabela32, Tabela33.

Process Each Sheet:

(a) Read the sheet using read_excel with col_names = FALSE.
(b) Remove the first 8 rows (titles and headers) and read from the 9th row.
(c) Select the first 13 columns under Valores correntes (1 000 000 R$)
(d) Use the first row as column names (representing years).
(e) Extract the row named `PIB - Ótica da Renda` (row 9).
(f) Create a new dataframe with the following columns:
  - year: Set to 2021.
  - admin_2_name: Region name from row 7.
  - admin_2_rgdp_total: Value corresponding to `"PIB - Ótica da Renda"` in the 2021 column.

Combine Dataframes: Merge all dataframes generated in the (f) step into a single dataframe.

Add Columns: Append the following columns:

- iso: Set to "BRA".
- admin_1_name: Set to "Brazil".
- admin_1_rgdp_total: Calculate as the sum of admin_2_rgdp_total, grouped by iso and year.

- Remove the grouping

Update Names: Rename "Distrito Federal" to "Distrito Federal (BR)" to align with OECD standards.

Assign Region Codes: Add the admin_2_id column and assign OECD-compliant region codes as follows:
Acre: BR01, Alagoas: BR08, Amapá: BR02, Amazonas: BR03,
Bahia: BR09, Ceará: BR10, Distrito Federal (BR): BR24,
Espírito Santo: BR17, Goiás: BR25, Maranhão: BR11, Mato Grosso: BR26,
Mato Grosso do Sul: BR27, Minas Gerais: BR18, Paraná: BR21,
Paraíba: BR12, Pará: BR04, Pernambuco: BR13, Piauí: BR14,
Rio Grande do Norte: BR15, Rio Grande do Sul: BR22,
Rio de Janeiro: BR19, Rondônia: BR05, Roraima: BR06,
Santa Catarina: BR23, Sergipe: BR16, São Paulo: BR20, Tocantins: BR07

Save: Save the results as BRA_2021.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

5 OECD Regional GDP Data

Here are the steps to organize regional GDP data from OECD before converting to cell-level data. Note: Adjustments may be required when updating to newer years or sourcing data from different websites.

1. **OECD Regional GDP Data (2012–2020):**

Download Data: Obtain OECD's regional GDP data for year 2012 to 2020 as described in Appendix Section 1.2.1 and save it as REGION_ECONOM-2023-1-EN-20240216T100059 2.csv.

Filter Data: Keep rows where:

- Indicator is Regional GDP,
- SERIES is SNA_2008,
- Year is between 2012 and 2020,
- Measure is Millions National currency, current prices,
- POS is ALL,
- TL is in 1, 2, 3.

Select and Rename Columns: Retain TL, REG_ID, Region, TIME, and Value, then rename:

- REG_ID to id,
- Value to rgdp_total,
- TL to admin_unit,
- Region to name.

Data Reshaping:

- Use `pivot_longer` to transform columns starting with `rgdp_` into a long format with a new column `sector`.
- Apply `pivot_wider` to spread `TIME` values into individual columns.
- Use `pivot_longer` to gather year columns (matching `\\d{4}`) into a `year` column and convert to numeric.
- Use `pivot_wider` to spread `sector` values back into individual columns.
- Remove ` $` from `name` using `mutate` with `gsub`.

2. **OECD Data Structure:**

Download Data: Obtain the territorial correspondence table as described in Appendix Section 1.2.1 and save it as `OECD Territorial correspondence - November 2023.xlsx`.

Filter Rows: Retain only rows where `Classification (latest: TL-2021)` equals `TL-2021`.

Select Columns: Extract `REG_ID`, `Regional name (eng)`, and `Hierarchical relations`.

Rename Columns: Rename the selected columns as follows:

- `REG_ID` → `id`,
- `Regional name (eng)` → `name`,
- `Hierarchical relations` → `parent_id`.

Add Columns `parent_name` and `grandparent_id`:

- The base dataset is the result obtained from the previous steps.
- Create a copy of the base dataset and rename `parent_id` to `grandparent_id` and `name` to `parent_name`.
- Perform a `left_join` to add the new columns `parent_name` and `grandparent_id` to the base dataset, using the mapping `by = c("parent_id" = "id")`.

Add Column `grandparent_name`:

- Create a copy of the base dataset and remove the `parent_id` column to avoid conflicts.
- Rename the `name` column in the copied dataset to `grandparent_name`.
- Perform a `left_join` to add the new column `grandparent_name` to the base dataset, using the mapping `by = c("grandparent_id" = "id")`.

Add Column `iso`:

- When both `parent_id` and `grandparent_id` are `NA`, set `iso` to `id` (indicating a top-level region).
- When `parent_id` is not `NA` but `grandparent_id` is `NA`, set `iso` to `parent_id` (indicating a mid-level region).
- When both `parent_id` and `grandparent_id` are not `NA`, set `iso` to `grandparent_id` (indicating a low-level region).

- Then when `grandparent_id` is `NA` and the `iso` value obtained from previous step is `NOMC`, set `iso` to `id`.
- When `grandparent_id` is not `NA` and the `iso` value obtained from previous step is `NOMC`, set `iso` to `parent_id`.
- Otherwise, retain the current value of `iso`.
- After the above step, if `iso` is still `"NOMC"`, update it to `id`.

3. **Process Japan's data in 2019**

   Japan's second administrative level data for 2019 are missing. These values can be derived by summing the corresponding third administrative level data.

   Use the processed OECD regional GDP data from Step 1 and apply the following transformations:

   - Filter the dataset to include only Japan in 2019 (`iso == "JPN", year == "2019"`).
   - Exclude rows where `parent_id == "JPN"` to isolate third administrative level data.
   - Group the data by `parent_id` and `year`, create a new column `sum_gdp` by summing the `rgdp_total` values within the group.
   - Remove the grouping
   - Ensure the result contains one row per region by applying a `distinct` operation.
   - Rename the `parent_id` column to `id` to represent the second administrative level.

4. **Finalize the OECD Regional GDP (2012–2020):**

   Combine Datasets: Merge the processed OECD regional GDP data from Step 1 with the OECD data structure from Step 2 using `left_join`. Keep regional GDP data as starting data.

   Exclude Countries with Insufficient Data: Filter out rows where `iso` or `id` belongs to the following: `"ZAF"`, `"IRL"`, `"ISR"`, `"CHN"`, `"IND"`, `"ISL"`, `"USA"`, `"LUX"`, `"MLT"`.

   Exclude Russia's and Japan's 2020 Data: Remove rows where `iso == "RUS" & year == 2020` or `id == "RUS" & year == 2020`, and exclude all data for Japan in 2020 use `filter(!(iso == "JPN" & year == 2020))`.

   Adjust Missing Value for Norway: Replace the `rgdp_total` value for `id == "NO0B1"` from `NA` to `0`.

   Update Japan's 2019 Data: Replace Japan's second administrative level GDP value for 2019 (`NA`) with the value calculated in Step 3.

   Convert Columns to Numeric: Transform the `year` and `rgdp_total` columns to numeric format.

   Exclude Non-Regionalized GDP: Filter out rows where `name` contains `"not regionalised"` or `"unregionalised"`

Exclude EU Total GDP: Exclude rows where `id` equals `"EU27_2020"`.

Remove Redundant Second Administrative Level Rows: Exclude rows where `admin_unit == 2` duplicates the data in rows with `admin_unit == 1`. For affected countries, reclassify third administrative level data (`admin_unit == 3`) as second administrative level data by changing `admin_unit` from 3 to 2.

Remove Redundant Second Administrative Level Rows: Exclude rows where `admin_unit == 2` duplicates the data in rows with `admin_unit == 3`. Again, for affected countries, update their third administrative level data by:

- Updating `parent_id` to be the corresponding values in `grandparent_id`.
- Updating `parent_name` to to be the corresponding values in `grandparent_name`.

Reclassify Third Administrative Level Data: For countries lacking second administrative level data (`admin_unit == 2`), reclassify third administrative level data to be second administrative level by updating `admin_unit` from 3 to 2.

5. **OECD Regional GDP Data (2021):**

Download Data: Obtain OECD's regional GDP data for 2021 as described in Appendix Section 1.2.1 and save it as `OECD.CFE.EDS,DSD_REG_ECO@DF_GDP,2.0+all.csv`.

Filter Relevant Rows: Select rows where:

- `Measure` is `Gross domestic product`,
- `Price.base` is `Current prices`,
- `Unit.of.measure` is `National currency`,
- `TIME_PERIOD` is `2021`.

Standardize `TERRITORIAL_LEVEL`:

- Replace all occurrences of `"CTRY"` with `"TL1"` to align with the territorial level naming convention.
- Remove the `"TL"` prefix from remaining values and convert them to numeric.

Rename Columns: Update the column names as follows:

- `TERRITORIAL_LEVEL` → `admin_unit`,
- `REF_AREA` → `id`,
- `Reference.area` → `Reference.area`,
- `TIME_PERIOD` → `year`,
- `OBS_VALUE` → `rgdp_total`.

Select Relevant Columns: Retain only `admin_unit`, `id`, `year`, and `rgdp_total`.

Filter by IDs from Previous Data: Keep only rows where `id` appears in the finalized OECD regional GDP data for 2012–2019.

Adjust `admin_unit` for New Zealand: For rows where the first three letters of `id` are `"NZ0"`, set `admin_unit` to `2`. Leave all other rows unchanged.

6. **Merge OECD Regional GDP Data (2012–2020) with 2021 Data:**

   Prepare 2021 Data: Convert the admin_unit column in the 2021 data obtained in Step 5 to a character type using as.character.

   Add Additional Columns: Perform a left_join to add the columns name, parent_id, parent_name, and iso to the 2021 data, with the information for these columns sourced from the finalized OECD GDP data for 2012–2020 from Step 4. Use admin_unit, id, and year as the common keys for left_join.

   Adjust Missing Value for Norway: Update the rgdp_total value to 0 for id == "NO0B1" & year == 2021.

   Exclude Missing Countries: Remove rows corresponding to the following countries due to missing data in the 2021 dataset: "BGR", "BRA", "EST", "HRV", "IDN", "JPN", "LVA", "PER", "ROU", and "RUS".

   Combine Datasets: Use bind_rows to append the processed 2021 data to the finalized OECD regional GDP data for 2012–2020 from Step 4.

7. **Finalized OECD Regional GDP Data (2012-2021):**

   Process First Administrative Level Data:

   - Use the merged OECD GDP data for 2012–2021 from Step 6.
   - Filter rows where admin_unit == 1.
   - Exclude the columns id, parent_id, and parent_name.
   - Restructure the data into a wide format by changing column names to the format `"admin_{admin_unit}_{.value}"` for columns name and those starting with rgdp, using pivot_wider with names_from = admin_unit, values_from = c("name", starts_with("rgdp")), and names_glue.
   - Refer to the resulting dataset as oecd_1.

   Process Second Administrative Level Data:

   - Use the merged OECD GDP data for 2012–2021 from Step 6.
   - Filter rows where admin_unit == 2.
   - Exclude the columns parent_id and parent_name.
   - Restructure the data into a wide format by changing column names to the format `"admin_{admin_unit}_{.value}"` for columns name and those starting with rgdp, using pivot_wider with names_from = admin_unit, values_from = c("name", starts_with("rgdp")), and names_glue.
   - Rename the id column to admin_2_id.
   - Refer to the resulting dataset as oecd_2.

   Process Third Administrative Level Data:

   - Use the merged OECD GDP data for 2012–2021 from Step 6.
   - Filter rows where admin_unit == 3.
   - Rename the columns:

  - parent_id to admin_2_id.
  - id to admin_3_id.
- Exclude the column parent_name.
- Restructure the data into a wide format by changing column names to the format `"admin_{admin_unit}_{.value}"` for columns name and those starting with rgdp, using pivot_wider with names_from = admin_unit, values_from = c("name", starts_with("rgdp")), and names_glue.
- Refer to the resulting dataset as oecd_3.

Merge Administrative Level Data:

- Perform a left_join to merge the dataset oecd_3 with oecd_2. Keep oecd_3 as the starting data.
- Identify rows in oecd_2 where admin_2_id is not present in the current dataset using filter(oecd_2, !admin_2_id %in% .$admin_2_id).
- Append the missing rows using bind_rows.
- Perform another left_join to merge the current dataset with oecd_1. Keep the current dataset as the starting data.
- Convert columns contain the pattern rgdp to numeric using mutate(across()).
- Retain only the columns year, iso, and those starting with admin_3, admin_2, and admin_1.
- Arrange the dataset in ascending order of the columns iso, admin_2_id, admin_3_id, and year.

Incorporate Additional Regional Data:

- Merge the dataset with Russia's 2020–2021 regional data from Section 4
- Merge the dataset with Brazil's 2021 regional data from Section 4.

Correct Geometry and Regional Names:

- **Indonesia:** Add GDP data for the missing region North Kalimantan by subtracting the total GDP of all other regions from the national GDP.
- **Chile:** Update the admin_2_name for `"Biobío (Región)"` to `"Biobío (Región) + Ñuble"`, as region `"Ñuble"`'s GDP is included in Biobio's GDP.
- **New Zealand:** For the same reason, update admin_2_name:
  (a) Tasman-Nelson-Marlborough to Tasman-Nelson-Marlborough + West Coast.
  (b) Gisborne to Gisborne + Hawke's Bay.
  (c) Region Canterbury includes Chatham Islands, but I did not change the name here.

Save Final Dataset:

- Save the finalized dataset as oecd_gdp_clean.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

8. **Select the Training Dataset:**

Create New Columns: Add the following columns:

- min_admin_unit: Set to 2 if admin_3_id is NA, otherwise set to 3.
- unit_name: Assign admin_2_name if min_admin_unit == 2, otherwise assign admin_3_name.
- id: Assign admin_2_id if min_admin_unit == 2, otherwise assign admin_3_id.
- unit_rgdp_total: Assign admin_2_rgdp_total if min_admin_unit == 2, otherwise assign admin_3_rgdp_total.

Set Columns to NA: Replace values in all columns starting with admin_2 with NA where min_admin_unit == 2.

Exclude Columns: Remove all columns starting with admin_3.

Rename Columns: Insert the prefix parent_ after the 8th character for all column names contain the patterns admin_2 or admin_1.

Reshape Data: Use pivot_longer on the following columns:

- Affected columns: admin_2_parent_id, admin_2_parent_name, admin_2_parent_rgdp_total, admin_1_parent_name, and admin_1_parent_rgdp_total.
- Creates new columns:
  - parent_admin_unit: Stores the administrative level (2 or 1).
  - parent_id, parent_name, and parent_rgdp_total: Hold the corresponding values from the affected columns.

Filter Rows: Remove rows where parent_name is NA.

Select Columns: Retain only the following:

- id, year, iso, unit_name, min_admin_unit, parent_admin_unit, parent_id, parent_name.
- Columns starting with unit_rgdp and parent_rgdp.

Adjust Parent ID: Set values of the parent_id column to the corresponding values in column iso for rows with parent_admin_unit == 1.

Align with NUTS 2021: Use recode_nuts function from regions package to align the id column in the dataset with the NUTS (Nomenclature of Units for Territorial Statistics) territorial correspondence for the year 2021. This ensures consistency between the dataset and the official territorial structure.

Remove Unnecessary Columns: Exclude typology, typology_change, and code_2021.

Save the Dataset: Export the final dataset to oecd_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

6 OECD Regional Geometry

1. **Prepare NUTS Geometry Data**

   Fetch Geometry Data: Use the gisco_get_nuts function from the giscoR package to retrieve the geometry for each administrative level (0, 1, 2, and 3). Specify the following parameters:

   - year = 2021: Specifies the version of the NUTS dataset.
   - epsg = 4326: Uses the WGS84 geographic coordinate system.
   - resolution = 10: Sets the spatial resolution to 1:10 million.

   Update Level Codes: Change the value of the column LEVL_CODE from 0 to 1 for rows where LEVL_CODE == 0.

   Combine Geometries: Bind the geometries for all administrative levels (0, 1, 2, and 3) into a single dataset.

   Rename Columns: Rename the column NUTS_ID to id.

   Add ISO3 Codes: Use a left_join to add the column iso3c to the current dataset from the codelist dataset provided by the countrycode package. Match the column CNTR_CODE in the current dataset with eurostat from the codelist dataset.

   Standardize Identifiers: Rename the column iso3c to iso. For rows where id == CNTR_CODE, update the id column to the value from iso.

   Rename Columns: Rename the column geometry to geom.

   Select Relevant Columns: Retain only the columns id, iso, and geom.

   Save Geometry Data: Save the resulting dataset to the file nuts_sf_pre.gpkg.

   Remove Inland Waters: Use the qgis_run_algorithm function from the qgisprocess package with the native:difference algorithm to remove large inland water geometries. Specify:

   - **Input Layer:** nuts_sf_pre.gpkg.
   - **Overlay Layer:** Inland water geometry file named glwd_1.shp.
   - **Output Layer:** Save the result as nuts_sf.gpkg.

2. **Adjust Geometries for Chile and New Zealand:**

   New Zealand:

   - Retrieve geometries from the file CGAZ_ADM1_without_large_waters.gpkg (obtained in Section 3, Step 5).
   - Rename columns: shapeName to name, and shapeGroup to iso.
   - Modify the name column to exclude Region$ from the values.
   - Create a new column region with values:
     - "Tasman-Nelson-Marlborough + West Coast" for regions Tasman, Nelson, Marlborough, and West Coast.
     - "Gisborne + Hawke's Bay" for regions Gisborne and Hawke's Bay.

    - "Canterbury" for regions Canterbury and Chatham Islands Territory.
    - The original name for all other regions.
  - Use st_union to combine geometries with the same values in columns region and iso.
  - Rename the column region to name.

Chile:

  - Retrieve geometries from the file CGAZ_ADM1_without_large_waters.gpkg (obtained in Section 3, Step 5).
  - Rename columns: shapeName to name, and shapeGroup to iso.
  - Convert the encoding of the name column from UTF-8 to LATIN1 using the iconv function.
  - Create a new column region with values:
    - `"Biobío (Región) + Ñuble"` for regions `"Región de Ñuble"` and `"Región del Bío-Bío"`.
    - The original name for all other regions.
  - Use st_union to combine geometries with the same values in columns region and iso.
  - Rename the column region to name.

3. **Other Non-NUTS Regions:**

Identify Non-NUTS Regions: Filter the regions whose id values are present in oecd_training_data.csv but not in nuts_sf.gpkg. Also extract the corresponding country names and exclude New Zealand and Chile.

Retrieve Geometries: Load the geometries for the identified countries from the file CGAZ_ADM1_without_large_waters.gpkg (obtained in Section 3, Step 5).

Rename Columns: Rename the column shapeName to name and shapeGroup to iso.

Update Indonesia Names: Append the suffix Province to the name column for rows where iso equals IDN.

Update Japan Names: Remove the suffix Prefecture from the name column for rows for rows only when it appears at the end of the string and only for rows where the iso column is JPN.

Select Columns: Retain only the columns name, iso, and geom.

Combine with New Zealand and Chile Data: Use bind_rows to combine the processed geometries of New Zealand and Chile obtained in previous steps.

Exclude Unnecessary Rows: Remove rows where the iso column equals AUS and the name column equals Other Territories.

Standardize Names: Modify the name column values for consistency across files:

  - "Gyeonggi" to "Gyeonggi-do",

- "South Gyeongsang" to "Gyeongsangnam-do",
- "North Gyeongsang" to "Gyeongsangbuk-do",
- "North Jeolla" to "Jeollabuk-do",
- "South Jeolla" to "Jeollanam-do",
- "North Chungcheong" to "Chungcheongbuk-do",
- "South Chungcheong" to "Chungcheongnam-do",
- "Gangwon" to "Gangwon-do",
- "Jeju" to "Jeju-do",
- "Australian Capital Territory" to "Canberra region (ACT)",
- "Archipiélago de San Andrés, Providencia y Santa Catalina" to "San Andrés",
- "Bogota Capital District" to "Bogotá Capital District",
- "Córdoba" to "Córdoba (CO)",
- "Amapa" to "Amapá",
- "Ceara" to "Ceará",
- "Espirito Santo" to "Espírito Santo",
- "Goias" to "Goiás",
- "Maranhao" to "Maranhão",
- "Para" to "Pará",
- "Paraiba" to "Paraíba",
- "Parana" to "Paraná",
- "Piaui" to "Piauí",
- "Rondonia" to "Rondônia",
- "Sao Paulo" to "São Paulo",
- "Rio Granda do Norte" to "Rio Grande do Norte",
- "Rio de Jeneiro" to "Rio de Janeiro",
- "Distrito Federal" & iso == "BRA" to "Distrito Federal (BR)",
- "Central Kalimantan Province" to "Middle Kalimantan Province",
- "Central Sulawesi Province" to "Middle Sulawesi Province",
- "Jakarta Special Capital Region Province" to "DKI Jakarta Province",
- "Special Region of Yogyakarta Province" to "D.I. Yogyakarta Province",
- "East Nusa Tenggara Province" to "Eastern Lesser Sundas Province",
- "North Sumatra Province" to "North Sumatera Province",
- "Southeast Sulawesi Province" to "South East Sulawesi Province",
- "West Nusa Tenggara Province" to "Western Lesser Sundas Province",
- "West Sumatra Province" to "West Sumatera Province",
- "South Sumatra Province" to "South Sumatera Province",
- "Bangka-Belitung Islands Province" to "Bangka Belitung Province",
- "North Kalimantan Province" to "North Kalimantan",
- "Riau Islands Province" to "Riau Mainland Province",

- "Riau Province" to "Riau Province",
- "MichoacÃ¡n de Ocampo" to "Michoacan",
- "Nuevo LeÃ³n" to "Nuevo Leon",
- "QuerÃ©taro de Arteaga" to "Queretaro",
- "San Luis PotosÃ" to "San Luis Potosi",
- "Veracruz de Ignacio de la Llave" to "Veracruz",
- "YucatÃ¡n" to "Yucatan",
- "Coahuila de Zaragoza" to "Coahuila",
- "MÃ©xico" to "Mexico",
- "Distrito Federal" & iso == "MEX" to "Mexico City",
- "Ingushetia" to "Republic of Ingushetia",
- "Khanty-Mansiysk Autonomous Okrug – Ugra" to "Khanty-Mansi Autonomous Okrug",
- "Adygea" to "Republic of Adygea",
- "Khakassia" to "Republic of Khakassia",
- "Tatarstan" to "Republic of Tatarstan",
- "Buryatia" to "Republic of Buryatia",
- "Chechnya" to "Chechen Republic",
- "Chuvashia" to "Chuvash Republic",
- "North Ossetia–Alania" to "Republic of North Ossetia-Alania",
- "Kabardino-Balkaria" to "Kabardino-Balkar Republic",
- "Leningrad oblast" to "Leningrad Oblast",
- "Mari El" to "Mari El Republic",
- "Tuva" to "Tuva Republic",
- "Udmurtia" to "Udmurt Republic",
- "Kalmykia" to "Republic of Kalmykia",
- "Karachay-Cherkessia" to "Karachay-Cherkess Republic",
- "Bashkortostan" to "Republic of Bashkortostan",
- "Dagestan" to "Republic of Dagestan",
- "Kaliningrad" to "Kaliningrad Oblast",
- "Amazonas" & iso == "PER" to "Amazonas (PE)",
- "Ancash" to "Áncash",
- "Madre de Dios" to "Madre de dios",
- "Callao" to "Prov. const. del Callao",
- "San Martín" to "San Martin",
- "Región de Antofagasta" to "Antofagasta",
- "Región de Arica y Parinacota" to "Arica y Parinacota",
- "Región de Atacama" to "Atacama",
- "Región de Aysén del Gral.Ibañez del Campo" to "Aysén",

- "Región de Coquimbo" to "Coquimbo",
- "Región de La Araucanía" to "Araucanía",
- "Región de Los Lagos" to "Los Lagos",
- "Región de Los Ríos" to "Los Ríos",
- "Región de Magallanes y Antártica Chilena" to "Magallanes and Chilean Antarctica",
- "Región de Tarapacá" to "Tarapacá",
- "Región de Valparaíso" to "Valparaíso",
- "Región del Libertador Bernardo O'Higgins" to "O'Higgins",
- "Región del Maule" to "Maule",
- "Región Metropolitana de Santiago" to "Santiago Metropolitan Region",

Save the File: Export the resulting dataset as `non_nuts_base_regions.gpkg`.

4. **Second Administrative Geometries for Japan and Korea:**

Purpose: The second administrative geometries for Japan and Korea are required.

Retrieve Administrative Data: Use the file `oecd_gdp_clean.csv` obtained in Section 5 to extract the columns `iso`, `admin_2_id`, `admin_2_name`, and `admin_3_name` for each second administrative level in Japan and Korea.

Obtain Third Administrative Geometries: Extract Japan and Korea's third administrative geometries from the file `non_nuts_base_regions.gpkg`.

Merge Geometries: Use `st_union` to combine the third administrative geometries into their corresponding second administrative levels based on the columns `iso`, `admin_2_id`, and `admin_2_name`.

Select Columns: Retain only the columns `id`, `iso`, and `geom`.

Save Result: Refer to the resulting file as `non_nuts_aggregate_regions`.

5. **Non-NUTS Regions' Country Geometry:**

Aggregate Geometries: Use the `qgis_run_algorithm` function from the `qgisprocess` package with the `native:aggregate` algorithm to generate country-level geometries for non-NUTS regions. The input file is `non_nuts_base_regions.gpkg`, and the parameters are set as follows:

- `GROUP_BY = "iso"` to group geometries by the `iso` column.
- `AGGREGATES`: Specify `list("aggregate" = "concatenate", "input" = '"iso"', "delimiter" = ",", "name" = "iso", "type" = 10, "length" = 0, "precision" = 0)`.
- `Output`: Set the `OUTPUT` as `non_nuts_nations.gpkg`.

Process Aggregated Geometries: Read the `non_nuts_nations.gpkg` file and:

- Update the `iso` column to contain only the first three characters of the original values.

- Create a new column id with the same values as the updated iso.
- Retain only the columns id, iso, and geom.

Save Result: Refer to the resulting file as non_nuts_nations.

6. **Finalize the Regional GDP Geometries:**

Identify Non-NUTS Regions: Filter the regions from oecd_training_data.csv whose id values are not present in nuts_sf.gpkg. Extract their id, iso, and unit_name columns and refer to the resulting dataset as non_nuts_regions.

Join with Base Regions: Start with the non_nuts_base_regions.gpkg file and perform a left_join with non_nuts_regions, using by = c("name" = "unit_name", "iso" = "iso").

Select Relevant Columns: Retain only the columns id, iso, and geom.

Combine Geometries:

- Use the rbind function to combine the resulting dataset with non_nuts_aggregate_regions obtained in the previous steps.
- Use the rbind function to further combine the dataset with non_nuts_nations obtained in the previous steps.
- Set the Coordinate Reference System (CRS) of the geom column in the dataset with the CRS of the nuts_sf.gpkg file obtained in the previous steps.
- Finally, use the rbind function to merge the dataset with nuts_sf.gpkg.

Save Result: Save the finalized dataset as oecd_poly.gpkg.

7. **Finalize the Training Dataset Regional GDP Geometries:**

Filter Geometries: Extract the rows from oecd_poly.gpkg whose id values belong to those in oecd_training_data.csv obtained in Section 5.

Save Result: Save the filtered dataset as oecd_training_poly.gpkg.

7 USA Regional GDP Data

Here are the steps to organize USA regional GDP data. Note: Adjustments may be required when updating to newer years or sourcing data from different websites.

1. **State and Country Level GDP:**

Filter and Prepare Data:

- Use the same downloaded file CAGDP2__ALL_AREAS_2001_2021.csv.
- Select rows where the column LineCode is 1, representing all industries.
- Remove the columns Region, TableName, IndustryClassification, Description, and Unit.

Reshape Data to Long Format:

- Transform the dataset from wide to long format, where each year column becomes a row, creating two new columns:
  - `year`: Extracted from column names starting with `X`, removing the prefix `X`.
  - `value`: Contains the data values corresponding to the original columns.
- Use the `pivot_longer` function for this transformation.

Rename and Filter Columns:

- Rename columns:
  - `LineCode` to `variable`.
  - `GeoName` to `name`.
  - `GeoFIPS` to `state_fips`.
- Change the value of `variable` column to `total` for rows where `variable` equals `1`.
- Select rows where the `name` column does not contain a comma `,`. This isolates the state-level data.
- Exclude rows where the `name` column values are `"Mideast"`, `"Great Lakes"`, `"New England"`, `"Plains"`, `"Southeast"`, `"Southwest"`, `"Rocky Mountain"`, or `"Far West"` since these are not states.
- Modify the `state_fips` column by removing its first character.

Reshape Data to Wide Format:

- Use the `pivot_wider` function to create new columns based on unique values in the `variable` column.
- Each new column name is prefixed with `admin_2_rgdp_` in front, and the corresponding values from the original dataset are assigned to these new columns.

Data Cleanup:

- Arrange the dataset in ascending order of the columns `state_fips` and `year`.
- Modify the `state_fips` column to contain only the first two characters of its original values.
- Remove commas `,` from the `admin_2_rgdp_total` column values, convert the cleaned strings to numeric values, and divide by 1000 to express the values in millions.
- Change the value of the `name` column to `United States` for rows where the current value of `name` is `United States *`.

Separate Country and State-Level GDP Data:

- **Country Level GDP Dataset:**
  - Keep only the row where `name` equals `United States`.
  - Rename all columns containing `admin_2`, replacing the number `2` with `1`
  - Remove the column `state_fips`.

  - Rename the column name to admin_1_name.
  - Create a new column iso with the value USA.
  - Refer to this processed dataset as country_gdp.
- **State Level GDP Dataset:**
  - Exclude rows where name equals United States.
  - Rename the column name to admin_2_name.
  - Create a new column iso with the value USA.
  - Use left_join to add country-level columns from country_gdp.
  - Refer to this processed dataset as state_gdp. And save the dataset as usa_state_gdp.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

2. **County Level GDP:**

Download Data: Obtain USA regional GDP data as described in Appendix Section 1.2.1 and save the file as CAGDP2__ALL_AREAS_2001_2021.csv.

Filter Data: Select rows where the column LineCode equals 1, which represents all industries.

Remove Unnecessary Columns: Exclude columns Region, TableName, Industry Classification, Description, and Unit.

Reshape to Long Format: Transform columns from wide format (with each year as a separate column) into a long format using pivot_longer, creating two new columns:

- year: Extracted from column names that start with X and remove the prefix X.
- value: Contains the data values from the corresponding columns.

Rename Columns: Rename LineCode to variable, GeoName to name, and GeoFIPS to fips.

Update Variable Column: Change the values in the column variable to total for rows where variable is equal to 1.

Filter County-Level Data: Select rows where the name column contains a comma (,), as these correspond to county-level data.

Clean FIPS Codes: Remove the first character of the fips column values.

Correct Misencoded Name: Change the value of the column name where it equals `"Do\xf1a Ana, NM"` to the correctly encoded string `"Doña Ana, NM"`.

Remove Asterisks: Remove asterisks that appear at the end of the name column values.

Trim Name Values: Remove the last four characters from the name column.

Reshape to Wide Format: Reshape the dataset from long to wide format using pivot_wider, spreading the variable column values into multiple columns, each prefixed with rgdp_.

Arrange Dataset: Arrange the dataset in ascending order of the columns fips and year.

Correct and Convert rgdp_total: Replace values of rgdp_total that are (D) or (NA) with 0, and convert the column to numeric using as.numeric.

Create State FIPS Column: Create a new column state_fips containing the first two characters of the fips column.

Adjust GDP Units: Divide all columns starting with rgdp_ by 1000 to convert values into millions.

Exclude Alaska's County Data: Use filter(substr(fips, 1, 2) != "02") to exclude Alaska's county-level data, as the county geometries change frequently and will be handled separately later.

Reference the Dataset: Refer to the processed dataset as county_gdp.

Finalize the County GDP Dataset: Perform the following steps:

- Start with the file county_gdp.
- Create a new column min_admin_unit with a value of 3.
- Add admin_3_ to all column names containing the substring rgdp_.
- Rename the name column to admin_3_name.
- Combine the current dataframe with the state-level GDP dataset state_gdp using left_join. Keep the current dataframe as the starting data.
- Save the resulting dataset as usa_gdp_clean.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

3. **USA Training Dataset:**

Start with Base File: Begin with the file usa_gdp_clean.csv.

Create Column id: Add a new column id by converting the fips column to character format using as.character.

Rename Columns with Substrings admin_2 or admin_1: Rename all column names containing the substrings admin_2 or admin_1 by inserting parent_ after the first 8 characters of the column names.

Reshape Data: Use pivot_longer to target columns whose names contain "admin_1" or "admin_2". This operation creates three new columns:

- parent_admin_unit: Stores the administrative level (1 or 2) extracted from the column name.
- parent_name: Contains the parent name values from the affected columns.
- parent_rgdp_total: Holds the corresponding GDP values.

Rename Columns Starting with admin_3: Replace the substring admin_3 with unit in all column names starting with admin_3.

Select Relevant Columns: Keep only the columns id, year, iso, unit_name, min_admin_unit, columns contain unit_rgdp, parent_admin_unit, parent_name, and columns contain parent_rgdp.

Save Result: Save the resulting dataset as usa_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

8 USA Regional Geometry

1. **County Geometry:**

Most County Geometry:

- **Retrieve Geographic Data:** Apply the counties function from the tigris package to retrieve geographic data from the U.S. Census Bureau with the following parameters:
  - cb = T: Use "cartographic boundary" shapefiles, which provide simplified geometries suitable for mapping and visualization.
  - year = 2020: Fetch county boundaries as they existed in 2020.
- **Rename Columns:** Rename the following columns:
  - NAME → name
  - GEOID → fips
  - STATEFP → state_fips
- **Modify name Column:** Update the values in the name column as follows:
  - If fips = "51600", change the value to "Fairfax City".
  - If fips = "51620", change the value to "Franklin City".
  - If fips = "51770", change the value to "Roanoke City".
- **Exclude Unnecessary Rows:** Remove rows where the name column are the following values: "United States", "New England", "Mideast", "Great Lakes", "Plains", "Southeast", "Southwest", "Rocky Mountain", or "Far West".
- **Remove U.S. Territories:** Exclude rows where the fips column starts with "69", "78", "60", or "66", as these represent U.S. territories.
- **Exclude Alaska Data:** Remove rows where the state_fips column equals "02", as Alaska will be processed separately.
- **Select Relevant Columns:** Retain only the columns name, fips, state_fips, and geometry.
- **Transform CRS:** Convert the coordinate reference system (CRS) to epsg:4326 using the st_transform function.
- **Refer as county_sf_2020:** Save the result as county_sf_2020.
- **Combine with GDP Data:** Start with the county_gdp file obtained in Section 7, and apply a left_join to combine it with county_sf_2020.
- **Convert to SF Object:** Use the st_as_sf function to convert the geometry column into a simple feature (sf) object.
- **Refer as county_sf_pre:** Save the resulting dataset as county_sf_pre.

Special County Geometry

- **Adjust Geometries to Match GDP Data:** The Bureau of Economic Analysis (BEA) has made modifications to the FIPS codes, resulting in some GDP data being aggregated for multiple counties. To align the geometries with the GDP data, modifications are required for certain regions in the `county_sf_2020` file.
- **Combine the Geometry of Kalawao and Maui Counties in Hawaii:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Kalawao` and `Maui`, and the `state_fips` column value is `15`.
  - Replace these rows with a single new row containing:
    - `name` column: Set to `Maui + Kalawao`.
    - `fips` column: Set to `15901`.
    - `state_fips` column: Retain the value `15`.
    - `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Update the Name of Fremont County to Include Yellowstone Park:** Use the file `county_sf_2020` and perform the following steps:
  - Select the row where the `name` column value is `Fremont` and the `fips` column value is `16043`.
  - Change the `name` column value to `Fremont (includes Yellowstone Park)`.
- **Update the Name of LaGrange County:** Use the file `county_sf_2020` and perform the following steps:
  - Select the row where the `name` column value is `LaGrange`.
  - Change the `name` column value to `Lagrange`.
- **Update Names for Independent Cities:** Use the file `county_sf_2020` and perform the following steps:
  - Select the rows where the `fips` column values are `24510`, `29510`, or `32510`.
  - Append ` (Independent City)` to the current `name` column values to align with the names in the GDP dataset.
- **Update Names for Independent Cities in Virginia:** Take the file `county_sf_2020` and perform the following steps:
  - Select rows where the `name` column values are in `c("Baltimore", "St. Louis", "Carson City", "Alexandria", "Chesapeake", "Hampton", "Newport News", "Norfolk", "Portsmouth", "Richmond", "Roanoke City", "Suffolk", "Virginia Beach")` and the `state_fips` column value is `51`.
  - Exclude the row where the `fips` column value is `51159`, as this corresponds to a different Richmond County in Virginia that should not be changed.
  - For the selected rows:
    - If the `name` column value is `Roanoke City`, change it to `Roanoke (Independent City)`.

    * For all other rows, append `" (Independent City)"` to the existing `name` column values.

- **Combine the Geometry of `Albemarle` and `Charlottesville` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Albemarle` and `Charlottesville`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Albemarle + Charlottesville`.
    * `fips` column: Set to `51901`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Alleghany` and `Covington` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Alleghany` and `Covington`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Alleghany + Covington`.
    * `fips` column: Set to `51903`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Augusta`, `Staunton` and `Waynesboro` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the three rows where the `name` column values are `Augusta`, `Staunton` and `Waynesboro`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Augusta, Staunton + Waynesboro`.
    * `fips` column: Set to `51907`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Campbell` and `Lynchburg` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Campbell` and `Lynchburg`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Campbell + Lynchburg`.
    * `fips` column: Set to `51911`.

    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.

- **Combine the Geometry of** Carroll **and** Galax **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Carroll and Galax, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Carroll + Galax.
    * fips column: Set to 51913.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Dinwiddie**,** ColonialHeights **and** Petersburg **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the three rows where the name column values are Dinwiddie, Colonial Heights and Petersburg, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Dinwiddie, Colonial Heights + Petersburg.
    * fips column: Set to 51918.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Fairfax**,** FairfaxCity **and** FallsChurch **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the three rows where the name column values are Fairfax, Fairfax City and Falls Church, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Fairfax, Fairfax City + Falls Church.
    * fips column: Set to 51919.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Frederick **and** Winchester **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Frederick and Winchester, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:

    * `name` column: Set to `Frederick + Winchester`.
    * `fips` column: Set to `51921`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Greensville` and `Emporia` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Greensville` and `Emporia`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Greensville + Emporia`.
    * `fips` column: Set to `51923`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Henry` and `Martinsville` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Henry` and `Martinsville`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `Henry + Martinsville`.
    * `fips` column: Set to `51929`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `JamesCity` and `Williamsburg` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `James City` and `Williamsburg`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:
    * `name` column: Set to `James City + Williamsburg`.
    * `fips` column: Set to `51931`.
    * `state_fips` column: Retain the value `51`.
    * `geometry` column: Combine the geometries of the two rows using the `st_union` function.
- **Combine the Geometry of `Montgomery` and `Radford` Counties in Virginia:** Use the file `county_sf_2020` and perform the following steps:
  - Select the two rows where the `name` column values are `Montgomery` and `Radford`, and the `state_fips` column value is `51`.
  - Replace these rows with a single new row containing:

    * name column: Set to Montgomery + Radford.
    * fips column: Set to 51933.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.

- **Combine the Geometry of Pittsylvania and Danville Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Pittsylvania and Danville, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Pittsylvania + Danville.
    * fips column: Set to 51939.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of PrinceGeorge and Hopewell Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Prince George and Hopewell, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Prince George + Hopewell.
    * fips column: Set to 51941.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of PrinceWilliam, Manassas and ManassasPark Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the three rows where the name column values are Prince William, Manassas and Manassas Park, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Prince William, Manassas + Manassas Park.
    * fips column: Set to 51942.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of Roanoke and Salem Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Roanoke and Salem, and the state_fips column value is 51.

- Replace these rows with a single new row containing:
  * name column: Set to Roanoke + Salem.
  * fips column: Set to 51944.
  * state_fips column: Retain the value 51.
  * geometry column: Combine the geometries of the two rows using the st_union function.

- **Combine the Geometry of** Rockbridge**,** BuenaVista **and** Lexington **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the three rows where the name column values are Rockbridge, Buena Vista and Lexington, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Rockbridge, Buena Vista + Lexington.
    * fips column: Set to 51945.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Rockingham **and** Harrisonburg **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Rockingham and Harrisonburg, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Rockingham + Harrisonburg.
    * fips column: Set to 51947.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Southampton **and** FranklinCity **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Southampton and Franklin City, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    * name column: Set to Southampton + Franklin.
    * fips column: Set to 51949.
    * state_fips column: Retain the value 51.
    * geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Spotsylvania **and** Fredericksburg **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Spotsylvania and Fredericksburg, and the state_fips column value is 51.

- Replace these rows with a single new row containing:
  - name column: Set to Spotsylvania + Fredericksburg.
  - fips column: Set to 51951.
  - state_fips column: Retain the value 51.
  - geometry column: Combine the geometries of the two rows using the st_union function.

- **Combine the Geometry of** Washington **and** Bristol **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Washington and Bristol, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    - name column: Set to Washington + Bristol.
    - fips column: Set to 51953.
    - state_fips column: Retain the value 51.
    - geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** Wise **and** Norton **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are Wise and Norton, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    - name column: Set to Wise + Norton.
    - fips column: Set to 51955.
    - state_fips column: Retain the value 51.
    - geometry column: Combine the geometries of the two rows using the st_union function.
- **Combine the Geometry of** York **and** Poquoson **Counties in Virginia:** Use the file county_sf_2020 and perform the following steps:
  - Select the two rows where the name column values are York and Poquoson, and the state_fips column value is 51.
  - Replace these rows with a single new row containing:
    - name column: Set to York + Poquoson.
    - fips column: Set to 51958.
    - state_fips column: Retain the value 51.
    - geometry column: Combine the geometries of the two rows using the st_union function.

Finalize the County Level Geometry:

- **Remove Empty Geometries:** Start with the file county_sf_pre, and remove rows with empty geometry.

- **Ensure Distinct Rows:** Use distinct to retain unique combinations of name, fips, state_fips, and geometry.
- **Add Special Geometries:** Use bind_rows to include all the special geometry regions processed earlier.
- **Rename Columns:** Rename the fips column to id.
- **Add ISO Column:** Create a new column iso with the value USA.
- **Select Relevant Columns:** Retain only the columns id and iso.
- **Rename Geometry Column:** Rename the geometry column to geom.
- **Reference the File:** Refer the resulting dataset as usa_admin_3_with_waters.gpkg.
- **Exclude Inland Waters:** Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:difference to remove large inland waters. The input file is usa_admin_3_with_waters.gpkg, the overlay file is glwd_1.shp, and the output file is saved as usa_admin_3.gpkg.

Finalize the Country Geometry:

- **Filter Relevant States:** Start with the file county_sf_2020, and select only the rows where the state_fips column values are also present in the state_gdp file obtained in Section 7.
- **Summarize United States Geometry:** Create a single summarized geometry for the United States at the first administrative level:
  - Assign name = United States to label the resulting geometry.
  - Set admin_unit = 1 to indicate the first administrative level.
  - Combine all input region geometries into a unified geometry using geom = st_union(geometry).
- **Add ISO Column:** Create a new column iso with the value USA.
- **Reference the File:** Refer the resulting dataset as usa_admin_1_with_waters.gpkg.
- **Exclude Inland Waters:** Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:difference to exclude large inland waters. The input file is usa_admin_1_with_waters.gpkg, the overlay file is glwd_1.shp, and the output file is saved as usa_admin_1.gpkg.

9 China Regional GDP Data and Geometry

1. **Province GDP Data:**

Download Data: Obtain China's GDP data as detailed in Appendix Section 1.2.1. Save it as AnnualbyProvince.xls.

Read Data: Read the file starting from the fourth row and limiting the read to the first 31 rows, using the read_xls function from the readxl package.

Reshape Data to Long Format: Use `pivot_longer` to reshape the data from wide to long format by selecting all columns with names matching a four-digit year, consolidating their values into a single column named `year` while preserving the corresponding values in other columns.

Convert Year to Numeric: Change the values in the `year` column to numeric using `as.numeric`.

Add Administrative Unit Column: Create a new column `admin_unit` with values set to `2`.

Rename Value Column: Rename the `value` column to `rgdp_total`.

Arrange Data: Arrange the dataset in ascending order of the `year` column.

Reshape Data to Wide Format: Use `pivot_wider` to reshape the dataset from long to wide format by creating new columns for each unique value in the `admin_unit` column, extracting values from columns with names containing `rgdp`, and naming the new columns using the template `"admin_{admin_unit}_{.value}"`, where `admin_unit` represents the administrative level and `.value` represents the original column name.

Convert GDP Column to Numeric: Change the values in the `admin_2_rgdp_total` column to numeric using `as.numeric`.

Calculate Aggregate GDP: Create a new column `admin_1_rgdp_total` with values as the sum of the `admin_2_rgdp_total` column grouped by the `year` column.

Remove the grouping

Generate ID Column: Create a new column `id` by appending the string `_CHN` to the end of each value in the `Region` column.

Add ISO Code: Create a new column `iso` with values set to `CHN`.

Define Minimum Administrative Unit: Create a new column `min_admin_unit` with values set to `2`.

Add National Name: Create a new column `admin_1_name` with values set to `China`.

Rename Region Column: Rename the `Region` column to `admin_2_name`.

Select Relevant Columns: Select only the columns `id`, `iso`, `year`, `min_admin_unit`, and columns with names starting with `admin_2` or `admin_1`.

Save Processed File: Save the file as `chn_gdp_clean.csv` and ensure the output excludes row names by setting the parameter `row.names = FALSE`.

2. **China Training Data:**

Create New Column: From the `chn_gdp_clean.csv` file, create a new column `parent_admin_unit` with the value `1`.

Rename Columns:

- Rename all columns whose names start with admin_1 by replacing the substring "admin_1" with "parent".
- Rename all columns whose names start with admin_2 by replacing the substring "admin_2" with "unit".

Select Columns: Retain only the columns id, year, iso, unit_name, min_admin_unit, parent_admin_unit, parent_name, and columns with names containing unit_rgdp or parent_rgdp.

Save Processed File: Save the file as chn_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

3. **Province Geometry:**

Load Initial File: Start from the file gdam_prov_level1_without_largewater.gpkg obtained in Section 3 Step 4.

Filter Rows: Select rows where the GID_0 column value is CHN.

Rename Columns: Rename the NAME_1 column as name and the GID_0 column as iso.

Exclude Rows: Remove rows where the name column has values Hong Kong or Macau, as these regions are not included in China's national GDP.

Modify Names: Update values in the name column as follows:

- "Nei Mongol" becomes "Inner Mongolia".
- "Ningxia Hui" becomes "Ningxia".
- "Xinjiang Uygur" becomes "Xinjiang".
- "Xizang" becomes "Tibet".
- Other values remain unchanged.

Filter Rows by GDP Data: Select rows where the name column values are in the admin_2_name column values from the chn_gdp_clean.csv file.

Create ID Column: Generate a new id column by concatenating the values of the name and iso columns using paste0(name, "_", iso).

Select Final Columns: Retain only the columns id, iso, and geom.

Save Final File: Save the result as chn_admin_2.gpkg.

10 India Regional GDP Data and Geometry

1. **Special Regions:**

Regions of Interest: Process GDP data separately for the regions "Jammu & Kashmir*" and "Jammu & Kashmir-U.T.".

Download Data: Obtain India's GDP data as described in Appendix Section 1.2.1 and save it as 27T_15112023E301A02422494F73BFAFD6CDD84EEEAE.XLSX.

Extract Relevant Sheets: Focus on sheets named T_27(iii) and T_27(iv), which contain GDP data for 2012 to 2021. For each sheet:

- Use read_excel to read 34 rows of GDP data, skipping the first 5 rows.
- Reshape columns containing four-digit years into a long format, creating a new column year with corresponding GDP values.

Combine Data: Use map_dfr to combine the results from both sheets into a single dataset by binding rows together.

Rename Columns: Rename the column ...1 to admin_2_name and value to rgdp_total.

Process Year Column: Update year values to the first four characters of the original strings, and convert them to numeric using as.numeric.

Administrative Unit: Add a new column admin_unit with values set to 2.

Filter Rows: Retain only rows where admin_2_name is in c("Jammu & Kashmir* ", "Jammu & Kashmir-U.T.").

Convert GDP to Numeric: Ensure rgdp_total values are numeric using as.numeric.

Group and Summarize: Group by year and summarize as follows:

- Use the coalesce function to combine rgdp_total values from rows with admin_2_name equal to either "Jammu & Kashmir-U.T." or "Jammu & Kashmir*".
- Assign "Jammu & Kashmir" as the value for admin_2_name column.
- Set the admin_unit column value to 2.

Exclude 2022 Data: Remove rows corresponding to the year 2022.

Save Result: Refer to the processed data as jk.

2. **Province GDP Data:**

Download Data: Obtain India's GDP data as detailed in Appendix Section 1.2.1 and save it as 27T_15112023E301A02422494F73BFAFD6CDD84EEEAE.XLSX.

Extract Relevant Sheets: Process data from the sheets T_27(iii) and T_27(iv) because they contain GDP data for the years 2012 to 2021. For each sheet:

- Use read_excel to read 34 rows of GDP data, skipping the first 5 rows.
- Reshape columns containing four-digit years into a long format with a new column year for the year and corresponding GDP values.

Combine Data: Use map_dfr to combine the processed data from both sheets into a single data frame by binding rows together.

Rename Columns: Rename the column ...1 to admin_2_name and value to rgdp_total.

Process Year Column: Extract the first four characters of the year column values and convert them to numeric using as.numeric.

Administrative Unit: Add a new column `admin_unit` with values set to `2`.

Filter Rows: Exclude rows where `admin_2_name` is in `c("Jammu & Kashmir*", "Jammu & Kashmir-U.T.")`, as they will be processed separately.

Exclude 2022 Data: Remove rows where `year` equals `2022` because this version does not predict GDP data for that year.

Convert GDP to Numeric: Ensure `rgdp_total` values are numeric using `as.numeric`.

Combine with Special Province Data: Use `bind_rows` to merge the processed data with the `jk` dataset obtained in the previous step.

Reshape Data to Wide Format: Use `pivot_wider` to reshape the dataset from long to wide format by creating new columns for each unique value in the `admin_unit` column, extracting values from columns with names containing `rgdp`, and naming the new columns using the template `"admin_{admin_unit}_{.value}"`, where `admin_unit` represents the administrative level and `.value` represents the original column name.

Convert GDP Column to Numeric: Change the values in the `admin_2_rgdp_total` column to numeric using `as.numeric`.

Calculate Aggregate GDP: Create a new column `admin_1_rgdp_total` with values as the sum of the `admin_2_rgdp_total` column grouped by the `year` column.

Remove the grouping

Generate ID Column: Create a new column `id` by appending the string `_IND` to the end of each value in the `admin_2_name` column.

Add ISO Code: Create a new column `iso` with values set to `IND`.

Define Minimum Administrative Unit: Create a new column `min_admin_unit` with values set to `2`.

Add National Name: Create a new column `admin_1_name` with values set to `India`.

Select Relevant Columns: Select only the columns `id`, `iso`, `year`, `min_admin_unit`, and columns with names starting with `admin_2` or `admin_1`.

Save Processed File: Save the file as `ind_gdp_clean.csv` and ensure the output excludes row names by setting the parameter `row.names = FALSE`.

3. **India Training Data:**

Create New Column: From the `ind_gdp_clean.csv` file, create a new column `parent_admin_unit` with the value `1`.

Rename Columns:

- Rename all columns whose names start with `admin_1` by replacing the substring `"admin_1"` with `"parent"`.

- Rename all columns whose names start with admin_2 by replacing the substring "admin_2" with "unit".

Select Columns: Retain only the columns id, year, iso, unit_name, min_admin_unit, parent_admin_unit, parent_name, and columns with names containing unit_rgdp or parent_rgdp.

Save Processed File: Save the file as ind_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

4. **Province Geometry:**

Load Initial File: Start from the file gdam_prov_level1_without_largewater.gpkg obtained in Section 3 Step 4.

Filter Rows: Select rows where the GID_0 column value is IND.

Rename Columns: Rename the NAME_1 column as name and the GID_0 column as iso.

Modify Names: Update values in the name column as follows:

- "Andaman and Nicobar" becomes "Andaman & Nicobar Islands".
- "NCT of Delhi" becomes "Delhi".
- "Jammu and Kashmir" becomes "Jammu & Kashmir".
- Other values remain unchanged.

Filter Rows by GDP Data: Select rows where the name column values are in the admin_2_name column values from the ind_gdp_clean.csv file.

Create ID Column: Generate a new id column by concatenating the values of the name and iso columns using paste0(name, "_", iso).

Select Final Columns: Retain only the columns id, iso, and geom.

Save Final File: Save the result as ind_admin_2.gpkg.

11 Kyrgyzstan Regional GDP Data and Geometry

1. **Regional GDP Data:**

Download Data: Obtain the regional GDP data for Kyrgyzstan as detailed in Appendix Section 1.2.1 and save the file as "1010009 Валовой региональный продукт (ВРП) в текущих ценах..xlsx".

Read Data: Use the read_excel function to read the xlsx file, skipping the first 3 rows and reading the next 13 rows.

Exclude Columns: Remove columns with names "Көрсөткүчтөрдүн аталыштары" and "Наименование показателей", as they contain region names not in English.

Remove Irrelevant Rows: Exclude the first three rows, as they are blank or pertain to national GDP and are not relevant for regional-level analysis.

Reshape Data to Long Format: Use `pivot_longer` to reshape the data from wide to long format by selecting all columns with names matching a four-digit year, consolidating their values into a single column named `year` while preserving the corresponding values in other columns.

Convert Year to Numeric: Change the values in the `year` column to numeric using `as.numeric`.

Add Administrative Unit Column: Create a new column `admin_unit` with values set to `2`.

Rename Value Column: Rename the `value` column to `rgdp_total`.

Reshape Data to Wide Format: Use `pivot_wider` to reshape the dataset from long to wide format by creating new columns for each unique value in the `admin_unit` column, extracting values from columns with names containing `rgdp`, and naming the new columns using the template `"admin_{admin_unit}_{.value}"`, where `admin_unit` represents the administrative level and `.value` represents the original column name.

Calculate Aggregate GDP: Create a new column `admin_1_rgdp_total` with values as the sum of the `admin_2_rgdp_total` column grouped by the `year` column.

Remove the grouping

Generate ID Column: Create a new column `id` by appending the string `_KGZ` to the end of each value in the `Items` column.

Add ISO Code: Create a new column `iso` with values set to `KGZ`.

Define Minimum Administrative Unit: Create a new column `min_admin_unit` with values set to `2`.

Add National Name: Create a new column `admin_1_name` with values set to `Kyrgyzstan`.

Rename Column: Rename the `Items` column to be `admin_2_name`

Select Relevant Columns: Select only the columns `id`, `iso`, `year`, `min_admin_unit`, and columns with names starting with `admin_2` or `admin_1`.

Save Processed File: Save the file as `kgz_gdp_clean.csv` and ensure the output excludes row names by setting the parameter `row.names = FALSE`.

2. **Kyrgyzstan Training Data:**

Create New Column: From the `kgz_gdp_clean.csv` file, create a new column `parent_admin_unit` with the value `1`.

Rename Columns:

- Rename all columns whose names start with `admin_1` by replacing the substring `"admin_1"` with `"parent"`.
- Rename all columns whose names start with `admin_2` by replacing the substring `"admin_2"` with `"unit"`.

Select Columns: Retain only the columns id, year, iso, unit_name, min_admin_unit, parent_admin_unit, parent_name, and columns with names containing unit_rgdp or parent_rgdp.

Save Processed File: Save the file as kgz_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

3. **Geometry:**

Load Initial File: Start from the file gdam_prov_level1_without_largewater.gpkg obtained in Section 3 Step 4.

Filter Rows: Select rows where the GID_0 column value is KGZ.

Rename Columns: Rename the NAME_1 column as name and the GID_0 column as iso.

Modify Names: Update values in the name column as follows:

- "Batken" becomes "Batken oblast".
- "Biškek" becomes "Bishkek city". Note that the "c" in "Bishkek city" is character U+0441 "c", you can copy it here.
- "Chüy" becomes "Chui oblast".
- "Jalal-Abad" becomes "Jalal-Abat oblast"
- "Naryn" becomes "Naryn oblast"
- "Osh" becomes "Osh oblast"
- "Osh (city)" becomes "Osh city". Note that the "c" in "Osh city" is character U+0441 "c", you can copy it here.
- "Talas" becomes "Talas oblast"
- "Ysyk-Köl" becomes "Yssyk-Kul oblast"
- Other values remain unchanged.

Filter Rows by GDP Data: Select rows where the name column values are in the admin_2_name column values from the kgz_gdp_clean.csv file.

Create ID Column: Generate a new id column by concatenating the values of the name and iso columns using paste0(name, "_", iso).

Select Final Columns: Retain only the columns id, iso, and geom.

Save Final File: Save the result as kgz_admin_2.gpkg.

12 Philippines Regional GDP Data and Geometry

1. **Regional GDP Data:**

Download Data: Obtain the regional GDP data for Philippines as detailed in Appendix Section 1.2.1 and save the file as GRDP_Reg_2018PSNA_2000-2023.xlsx.

Read Data: Use the read_excel function to read the xlsx file, skipping the first 9 rows and reading the next 18 rows.

Remove Irrelevant Row: Exclude the first row, as they are missing values.

Remove Irrelevant Column: Exclude the first column because we don't need it.

Rename Column: Rename the column ...2 to admin_2_name

Reshape Data to Long Format: Use pivot_longer to reshapes the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removes the prefix X from those column names, and stores the resulting values in a new column named year.

Convert Year to Numeric: Change the values in the year column to numeric using as.numeric.

Add Administrative Unit Column: Create a new column admin_unit with values set to 2.

Rename Value Column: Rename the value column to rgdp_total.

Reshape Data to Wide Format: Use pivot_wider to reshape the dataset from long to wide format by creating new columns for each unique value in the admin_unit column, extracting values from columns with names containing rgdp, and naming the new columns using the template `"admin_{admin_unit}_{.value}"`, where admin_unit represents the administrative level and .value represents the original column name.

Calculate Aggregate GDP: Create a new column admin_1_rgdp_total with values as the sum of the admin_2_rgdp_total column grouped by the year column.

Remove the grouping

Generate ID Column: Create a new column id by appending the string _PHL to the end of each value in the admin_2_name column.

Add ISO Code: Create a new column iso with values set to PHL.

Define Minimum Administrative Unit: Create a new column min_admin_unit with values set to 2.

Add National Name: Create a new column admin_1_name with values set to Philippines.

Select Relevant Columns: Select only the columns id, iso, year, min_admin_unit, and columns with names starting with admin_2 or admin_1.

Save Processed File: Save the file as phl_gdp_clean.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

2. **Philippines Training Data:**

Create New Column: From the phl_gdp_clean.csv file, create a new column parent_admin_unit with the value 1.

Rename Columns:

- Rename all columns whose names start with admin_1 by replacing the substring "admin_1" with "parent".
- Rename all columns whose names start with admin_2 by replacing the substring "admin_2" with "unit".

Select Columns: Retain only the columns id, year, iso, unit_name, min_admin_unit, parent_admin_unit, parent_name, and columns with names containing unit_rgdp or parent_rgdp.

Save Processed File: Save the file as phl_training_data.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

3. **Geometry:**

Load Initial File: Start from the file CGAZ_ADM1_without_large_waters.gpkg obtained in Section 3 Step 5.

Rename Columns: Rename the shapeName column as name and the shapeGroup column as iso.

Filter Rows: Select rows where the iso column value is PHL.

Modify Names: Update values in the name column as follows:

- "ARMM" becomes "Bangsamoro Autonomous Region\r\nin Muslim Mindanao".
- "NCR" becomes "National Capital Region".
- "Calabarzon" becomes "CALABARZON".
- "Mimaropa" becomes "MIMAROPA Region"
- "Soccsksargen" becomes "SOCCSKSARGEN"
- "CAR" becomes "Cordillera Administrative Region"
- Other values remain unchanged.

Filter Rows by GDP Data: Select rows where the name column values are in the admin_2_name column values from the phl_gdp_clean.csv file.

Create ID Column: Generate a new id column by concatenating the values of the name and iso columns using paste0(name, "_", iso).

Select Final Columns: Retain only the columns id, iso, and geom.

Save Final File: Save the result as phl_admin_2.gpkg.

13 Kazakhstan Regional GDP Data and Geometry

1. **Special Regions:**

The boundaries and the number of Kazakhstan's regions, as reflected in the regional GDP data, changed between 2008 and 2022, but we aim to use constant regions and geometry files across all years. This requires adjustments to align the GDP data with consistent geometries. When updating to newer years, ensure that the GDP data corresponds to the same geometries currently in use.

Download GDP Data: Download the regional GDP data as described in Appendix Section 1.2.1 and save the file as `1. Gross regional product.xlsx`.

Aggregate GDP Data for "Ontustik Kazakhstan": Starting in 2018, `Ontustik Kazakhstan` was divided into `Shymkent city` and `Turkistan`. To maintain consistent geometries across all years, the GDP data for these regions needs to be aggregated back into a single entry, as outlined below:

- Read the sheet named `2008-2023` from the file `1. Gross regional product.xlsx` using the `read_excel` function, skipping the first two rows and reading the next 22 rows.
- Remove the first row, as it represents the GDP for the entire country.
- Rename the column `...1` to `admin_2_name`.
- Select the following columns: `admin_2_name`, `2008`, `2009`, `2010`, `2011`, `2012`, `2013`, `2014`, `2015`, `2016`, `2017`, `2018`, `2019`, `2020`, `2021`, `2022`.
- Filter rows where the `admin_2_name` column contains values in `c("Ontustik Kazakhstan", "Shymkent city", "Turkistan")`.
- Adjust the GDP data as follows:
  - Each column named with a year represents the GDP data for that specific year.
  - For the `Ontustik Kazakhstan` region, values are missing after 2018 due to its division into `Shymkent city` and `Turkistan`.
  - Fill these missing values (i.e., for years after 2018) by summing the GDP values of `Shymkent city` and `Turkistan`, retaining the existing values for `Ontustik Kazakhstan` in earlier years if not missing.
- Select only the row where the `admin_2_name` column has the value `Ontustik Kazakhstan`.
- Refer the processed dataframe as `ost`.

Aggregate GDP Data for "Shygys Kazakhstan": Start 2021, region `Abay` is separated from `Shygys Kazakhstan`

- Read the sheet named `2008-2023` from the file `1. Gross regional product.xlsx` using the `read_excel` function, skipping the first two rows and reading the next 22 rows.
- Remove the first row, as it represents the GDP for the entire country.
- Rename the column `...1` to `admin_2_name`.
- Select the following columns: `admin_2_name`, `2008`, `2009`, `2010`, `2011`, `2012`, `2013`, `2014`, `2015`, `2016`, `2017`, `2018`, `2019`, `2020`, `2021`, `2022`.
- Filter rows where the `admin_2_name` column contains values in `c("Abay", "Shygys Kazakhstan")`.
- Replace all columns `NA` values to `0`. This will make region `Abay`'s GDP to 0 for years before 2021.
- Adjust the GDP data as follows:

  - Each column named with a year represents the GDP data for that specific year.
  - For all such columns, if the admin_2_name column value is Shygys Kazakhstan, update its GDP value to the sum of the GDP values for Abay and Shygys Kazakhstan.
- Select only the row where the admin_2_name column has the value Shygys Kazakhstan.
- Refer the processed dataframe as as.

Aggregate GDP Data for "Almaty": Start 2021, region "Zhetisu" is separated from region "Almaty". Note, the "A" in "Almaty" is character U+0441 "A", you can copy it here.

- Read the sheet named 2008-2023 from the file 1. Gross regional product.xlsx using the read_excel function, skipping the first two rows and reading the next 22 rows.
- Remove the first row, as it represents the GDP for the entire country.
- Rename the column ...1 to admin_2_name.
- Select the following columns: admin_2_name, 2008, 2009, 2010, 2011, 2012 , 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022.
- Filter rows where the admin_2_name column contains values in "c("Zhetisu", "Almaty")".
- Replace all columns NA values to 0. This will make region Zhetisu's GDP to 0 for years before 2021.
- Adjust the GDP data as follows:
  - Each column named with a year represents the GDP data for that specific year.
  - For all such columns, if the admin_2_name column value is "Almaty", update its GDP value to the sum of the GDP values for "Zhetisu" and "Almaty".
- Select only the row where the admin_2_name column has the value "Almaty".
- Refer the processed dataframe as za.

Aggregate GDP Data for "Karagandy": Start 2021, region "Ulytau" is separated from region "Karagandy". Note, the "K" in "Karagandy" is character U+0441 "K", you can copy it here.

- Read the sheet named 2008-2023 from the file 1. Gross regional product.xlsx using the read_excel function, skipping the first two rows and reading the next 22 rows.
- Remove the first row, as it represents the GDP for the entire country.
- Rename the column ...1 to admin_2_name.
- Select the following columns: admin_2_name, 2008, 2009, 2010, 2011, 2012 , 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022.

- Filter rows where the admin_2_name column contains values in c("Ulytau", "Karagandy").
- Replace all columns NA values to 0. This will make region Ulytau's GDP to 0 for years before 2021.
- Adjust the GDP data as follows:
  - Each column named with a year represents the GDP data for that specific year.
  - For all such columns, if the admin_2_name column value is "Karagandy", update its GDP value to the sum of the GDP values for "Ulytau" and "Karagandy".
- Select only the row where the admin_2_name column has the value "Karagandy".
- Refer the processed dataframe as uk.

2. **Finalize Regional GDP Data:**

Read the Data: Read the sheet named 2008-2023 from the file 1. Gross regional product.xlsx using the read_excel function, skipping the first two rows and reading the next 22 rows.

Remove Unnecessary Row: Remove the first row, as it represents the GDP for the entire country.

Rename the column: Rename the column ...1 to admin_2_name.

Select Necessary Columns: Select the following columns: admin_2_name, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022.

Remove Unnecessary Rows: Exclude rows where the admin_2_name column value belong to c("Ontustik Kazakhstan", "Shymkent city", "Turkistan", "Abay", "Shygys Kazakhstan", "Zhetisu", "Almaty", "Ulytau", "Karagandy")

Combine Data: Use bind_rows to combine with the above processed dataframe named ost, as, za, and uk.

Reshape Data to Long Format: Use pivot_longer to reshapes the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removes the prefix X from those column names, and stores the resulting values in a new column named year.

Convert Year to Numeric: Change the values in the year column to numeric using as.numeric.

Add Administrative Unit Column: Create a new column admin_unit with values set to 2.

Rename Value Column: Rename the value column to rgdp_total.

Reshape Data to Wide Format: Use pivot_wider to reshape the dataset from long to wide format by creating new columns for each unique value in the admin_unit column, extracting values from columns with names containing rgdp, and naming the new columns using the template `"admin_{admin_unit}_{.value}"`, where `admin_unit` represents the administrative level and `.value` represents the original column name.

Calculate Aggregate GDP: Create a new column `admin_1_rgdp_total` with values as the sum of the `admin_2_rgdp_total` column grouped by the `year` column.

Remove the grouping

Generate ID Column: Create a new column `id` by appending the string `_KAZ` to the end of each value in the `admin_2_name` column.

Add ISO Code: Create a new column `iso` with values set to `KAZ`.

Define Minimum Administrative Unit: Create a new column `min_admin_unit` with values set to `2`.

Add National Name: Create a new column `admin_1_name` with values set to `Kazakhstan`.

Select Relevant Columns: Select only the columns `id`, `iso`, `year`, `min_admin_unit`, and columns with names starting with `admin_2` or `admin_1`.

Save Processed File: Save the file as `kaz_gdp_clean.csv` and ensure the output excludes row names by setting the parameter `row.names = FALSE`.

3. **Kazakhstan Training Data:**

Create New Column: From the `kaz_gdp_clean.csv` file, create a new column `parent_admin_unit` with the value `1`.

Rename Columns:

- Rename all columns whose names start with `admin_1` by replacing the substring `"admin_1"` with `"parent"`.
- Rename all columns whose names start with `admin_2` by replacing the substring `"admin_2"` with `"unit"`.

Select Columns: Retain only the columns `id`, `year`, `iso`, `unit_name`, `min_admin_unit`, `parent_admin_unit`, `parent_name`, and columns with names containing `unit_rgdp` or `parent_rgdp`.

Save Processed File: Save the file as `kaz_training_data.csv` and ensure the output excludes row names by setting the parameter `row.names = FALSE`.

4. **Geometry:**

Load Initial File: Start from the file `CGAZ_ADM1_without_large_waters.gpkg` obtained in Section 3 Step 5.

Rename Columns: Rename the `shapeName` column as `name` and the `shapeGroup` column as `iso`.

Filter Rows: Select rows where the `iso` column value is `KAZ`.

Modify Names: Update values in the `name` column as follows:

- "Akmola Region" becomes "Akmola".
- "Aktobe Region" becomes "Aktobe". Note "A" and "e" in Aktobe" are in character U+0410, you can copy them here.
- "Almaty" becomes "Almaty city".
- "Almaty Region" becomes "Almaty". Note "A" in "Almaty" are in character U+0410, you can copy it here.
- "Astana" becomes "Astana city"
- "Atyrau Region" becomes "Atyrau". Note "A" in "Atyrau" are in character U+0410, you can copy it here.
- "East Kazakhstan Region" becomes "Shygys Kazakhstan"
- "Jambyl Region" becomes "Zhambyl"
- "Karaganda Region" becomes "Karagandy". Note "K" in "Karagandy" are in character U+0410, you can copy it here.
- "Kostanay Region" becomes "Kostanay". Note "Ko" in "Kostanay" are in character U+0410, you can copy it here.
- "Kyzylorda Region" becomes "Kyzylorda". Note "K" in "Kyzylorda" are in character U+0410, you can copy it here.
- "Mangystau Region" becomes "Mangystau". Note "M" in "Mangystau" are in character U+0410, you can copy it here.
- "North Kazakhstan Region" becomes "Soltustik Kazakhstan". Note "K" in "Kazakhstan" are in character U+0410, you can copy it here.
- "Pavlodar Region" becomes "Pavlodar"
- "South Kazakhstan Region" becomes "Ontustik Kazakhstan"
- "West Kazakhstan Region" becomes "Batys Kazakhstan"
- Other values remain unchanged.

Filter Rows by GDP Data: Select rows where the `name` column values are in the `admin_2_name` column values from the `kaz_gdp_clean.csv` file.

Create ID Column: Generate a new `id` column by concatenating the values of the `name` and `iso` columns using `paste0(name, "_", iso)`.

Select Final Columns: Retain only the columns `id`, `iso`, and `geom`.

Save Final File: Save the result as `kaz_admin_2.gpkg`.

14 National GDP Data

1. **IMF National GDP Data:**

Download Data: Download the IMF national GDP data as described in Appendix Section 1.2.2 and save it as `WEO_Data.xlsx`.

IMF Population Data:

- Read the `WEO_Data.xlsx` file using `read_excel`.

- Select rows where the `Subject Descriptor` column value is `Population` and the `Units` column value is `Persons`.
- Exclude columns `"Subject Descriptor"`, `"Units"`, `"Scale"`, `"Country/Series-specific Notes"`, and `"Estimates Start After"`.
- Use `pivot_longer` to reshape the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removing the prefix `X` from those column names, and storing the resulting values in a new column named `year`.
- Rename the column `ISO` to `iso`.
- Exclude rows where the `iso` column values are either `ERI` or `SYR`, as these contain missing values and data must be sourced elsewhere.
- Convert the `year` column values to numeric using `as.numeric`.
- Create a new column `population`, where the values are calculated by converting the `value` column to numeric using `as.numeric` and multiplying each value by `1e6` (to scale the population to its full value in persons)
- Select only the columns `iso`, `year`, `population`, and `Country`.
- Refer to the resulting dataframe as `imf_pop`.

IMF GDP Data in Constant 2017 International Dollars:

- Read the `WEO_Data.xlsx` file using `read_excel`.
- Select rows where the `Subject Descriptor` column value is `Gross domestic product per capita, constant prices` and the `Units` column value is `Purchasing power parity; 2017 international dollar`.
- Exclude columns `"Country"`, `"Subject Descriptor"`, `"Units"`, `"Scale"`, `"Country/Series-specific Notes"`, and `"Estimates Start After"`.
- Use `pivot_longer` to reshape the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removing the prefix `X` from those column names, and storing the resulting values in a new column named `year`.
- Rename the column `ISO` to `iso`.
- Exclude rows where the `iso` column values are either `ERI` or `SYR`, as these contain missing values and data must be sourced elsewhere.
- Convert the `year` column values to numeric using `as.numeric`.
- Create a new column `rgdp_total`, where the values are calculated by converting the `value` column to numeric using `as.numeric`.
- Select only the columns `iso`, `year`, `rgdp_total`
- Use `left_join` to combine the current dataframe with the above processed `imf_pop` dataframe. Keep the current dataframe as the starting data.
- Update the `rgdp_total` column by recalculating its values as `rgdp_total * population / 1e9`, converting the unit to billions of constant 2017 international dollars.
- Refer to the resulting dataframe as `imf_gdp_const_2017_PPP`.

IMF GDP Data in Current International Dollars:

- Read the WEO_Data.xlsx file using read_excel.
- Select rows where the Subject Descriptor column value is Gross domestic product, current prices and the Units column value is Purchasing power parity; international dollars.
- Exclude columns "Country", "Subject Descriptor", "Units", "Scale", "Country/Series-specific Notes", and "Estimates Start After".
- Use pivot_longer to reshape the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removing the prefix X from those column names, and storing the resulting values in a new column named year.
- Rename the column ISO to iso.
- Exclude rows where the iso column values are either ERI or SYR, as these contain missing values and data must be sourced elsewhere.
- Convert the year column values to numeric using as.numeric.
- Create a new column rgdp_total, where the values are calculated by converting the value column to numeric using as.numeric.
- Select only the columns iso, year, rgdp_total
- Use left_join to combine the current dataframe with the above processed imf_pop dataframe. Keep the current dataframe as the starting data.
- Refer to the resulting dataframe as imf_gdp_crt_PPP.

IMF GDP Data in Current USD:

- Read the WEO_Data.xlsx file using read_excel.
- Select rows where the Subject Descriptor column value is Gross domestic product, current prices and the Units column value is U.S. dollars.
- Exclude columns "Country", "Subject Descriptor", "Units", "Scale", "Country/Series-specific Notes", and "Estimates Start After".
- Use pivot_longer to reshape the dataset from wide to long format by gathering all columns with names matching a four-digit year (e.g., 2020, 2021), removing the prefix X from those column names, and storing the resulting values in a new column named year.
- Rename the column ISO to iso.
- Exclude rows where the iso column values are either ERI or SYR, as these contain missing values and data must be sourced elsewhere.
- Convert the year column values to numeric using as.numeric.
- Create a new column rgdp_total, where the values are calculated by converting the value column to numeric using as.numeric.
- Select only the columns iso, year, rgdp_total
- Use left_join to combine the current dataframe with the above processed imf_pop dataframe. Keep the current dataframe as the starting data.
- Refer to the resulting dataframe as imf_gdp_crt_us.

IMF GDP Data in Constant 2017 USD:

- **Download Data:** Obtain the "Annual Index Value" data from the US Census Bureau as described in Appendix Section 1.2.2 and save it as annual-index-value_annual-percent-change.xls.
- **Process Index Data:**
  - Read the file using read_excel and skip the first two rows.
  - Rename the column Income Year to year and the column "C–CPI–U1\nIndex\n(Dec 1999=100)" to index.
  - Select only the columns year and index.
  - Use na.omit to exclude rows with missing values.
  - Change the year column values to numeric.
  - Refer to the resulting dataframe as index.
- **Adjust GDP Data:**
  - Start with the imf_gdp_crt_us dataframe processed in the previous step.
  - Apply left_join to combine the current dataframe with the index dataframe. Keep the current dataframe as the starting data.
  - Create a new column index_2017 containing the value of the index column value for the row where the year column equals 2017.
  - Update the rgdp_total column by recalculating its values as rgdp_total * (index_2017 / index).
  - Select only the columns iso, year, rgdp_total, population, and Country.
  - Refer to the resulting dataframe as imf_gdp_const_2017.

2. **World Bank National GDP Data:**

Here we supplement the above IMF national GDP data using the World Bank dataset for the following countries: BMU, CYM, CUW, GRL, XKX, LIE, MCO, SXM, SYR, TCA, PSE.

World Bank Population Data:

- Obtain the World Bank population dataset as described in Appendix Section 1.2.2 and save it as API_SP.POP.TOTL_DS2_en_excel_v2_294626.xls.
- Use read_excel to read the sheet named Data, skipping the first three rows.
- Rename the Country Code column to iso.
- Select only the rows where the iso column values are within c("BMU", "CYM", "CUW", "GRL", "XKX", "LIE", "MCO", "SXM", "SYR", "TCA", "PSE").
- Exclude the columns Indicator Name and Indicator Code.
- Use pivot_longer to reshape the dataset from wide to long format by:
  - Gathering all columns with names matching a four-digit year (e.g., 2020, 2021).
  - Removing the prefix X from those column names.

  - Storing the resulting values in a new column named `year`.
- Filter the `year` column to retain only values within the range 2012 to 2021 for this 2024 dataset version.
- Change the `year` column values to numeric.
- Rename the `value` column to `population` and the `Country Name` column to `Country`.
- Refer to the resulting dataframe as `wb_pop`.

World Bank GDP Data in Current USD:

- **Download Data:** Obtain the World Bank GDP data in current USD as described in Appendix Section 1.2.2 and save it as `API_NY.GDP.MKTP.CD_DS2_en_excel_v2_287504.xls`
- Rename the `Country Code` column to `iso`.
- Select only the rows where the `iso` column values are within `c("BMU", "CYM", "CUW", "GRL", "XKX", "LIE", "MCO", "SXM", "SYR", "TCA", "PSE")`.
- Exclude the columns `Country Name`, `Indicator Name` and `Indicator Code`.
- Use `pivot_longer` to reshape the dataset from wide to long format by:
  - Gathering all columns with names matching a four-digit year (e.g., 2020, 2021).
  - Removing the prefix `X` from those column names.
  - Storing the resulting values in a new column named `year`.
- Filter the `year` column to retain only values within the range 2012 to 2021 for this 2024 dataset version.
- Change the `year` column values to numeric.
- Rename the `value` column to `rgdp_total`.
- Change the `rgdp_total` column values to be `rgdp_total/1e9`.
- Apply `left_join` to combine the current dataframe with the `wb_pop` dataframe. Keep the current dataframe as the starting data.
- Refer to the resulting dataframe as `wb_gdp_crt_us`.

World Bank GDP Data in Constant 2017 USD:

- Start with `wb_gdp_crt_us` file obtained in the previous step.
- Apply `left_join` to combine the current dataframe with the `index` dataframe. Keep the current dataframe as the starting data.
- Create a new column `index_2017` containing the value of the `index` column value for the row where the `year` column equals `2017`.
- Update the `rgdp_total` column by recalculating its values as `rgdp_total * (index_2017 / index)`.
- Select only the columns `iso`, `year`, `rgdp_total`, `population`, and `Country`.
- Refer to the resulting dataframe as `wb_gdp_const_2017`.

World Bank GDP Data in Current International Dollars:

- **Download Data:** Obtain the World Bank GDP data in current international dollar as described in Appendix Section 1.2.2 and save it as `API_NY.GDP.MKTP.PP.CD_DS2_en_excel_v2_287316.xls`
- Read the sheet named `Data`, skipping the first three rows.
- Rename the `Country Code` column to `iso`.
- Select only the rows where the `iso` column values are within `c("BMU", "CYM", "CUW", "GRL", "XKX", "LIE", "MCO", "SXM", "SYR", "TCA", "PSE")`. Note however, "SYR", "LIE", "MCO" have missing data for GDP in current international dollars.
- Exclude the columns `Country Name`, `Indicator Name` and `Indicator Code`.
- Use `pivot_longer` to reshape the dataset from wide to long format by:
  - Gathering all columns with names matching a four-digit year (e.g., 2020, 2021).
  - Removing the prefix `X` from those column names.
  - Storing the resulting values in a new column named `year`.
- Filter the `year` column to retain only values within the range 2012 to 2021 for this 2024 dataset version.
- Change the `year` column values to numeric.
- Rename the `value` column to `rgdp_total`.
- Change the `rgdp_total` column values to be `rgdp_total/1e9`.
- Apply `left_join` to combine the current dataframe with the `wb_pop` dataframe. Keep the current dataframe as the starting data.
- Refer to the resulting dataframe as `wb_gdp_crt_PPP`.

World Bank GDP Data in Constant 2017 International Dollars:

- **Download Data:** Obtain the World Bank GDP data in constant 2021 international dollars as described in Appendix Section 1.2.2 and save it as `API_NY.GDP.MKTP.PP.KD_DS2_en_excel_v2_288920.xls`. We will change the constant 2021 international dollar to constant 2017 international dollar.
- **Process Data:**
  - Read the sheet named `Data`, skipping the first three rows.
  - Rename the `Country Code` column to `iso`.
  - Select only the rows where the `iso` column values are within `c("BMU", "CYM", "CUW", "GRL", "XKX", "LIE", "MCO", "SXM", "SYR", "TCA", "PSE")`. Note however, "LIE", "MCO" have missing data for this one.
  - Exclude the columns `Country Name`, `Indicator Name` and `Indicator Code`.
  - Use `pivot_longer` to reshape the dataset from wide to long format by:
    - Gathering all columns with names matching a four-digit year (e.g., 2020, 2021).
    - Removing the prefix `X` from those column names.

        * Storing the resulting values in a new column named `year`.
    - Filter the `year` column to retain only values within the range 2012 to 2021 for this 2024 dataset version.
    - Change the `year` column values to numeric.
    - Rename the `value` column to `rgdp_total`.
    - Change the `rgdp_total` column values to be `rgdp_total/1e9`.
    - Apply `left_join` to combine the current dataframe with the `wb_pop` dataframe. Keep the current dataframe as the starting data.
    - Refer to the resulting dataframe as `wb_gdp_const_2021_PPP`.
  - **Process Index:**
    - Start from the `wb_gdp_const_2021_PPP` dataframe.
    - Rename the `rgdp_total` column to `rgdp_total_const_2021_PPP`.
    - Perform a `left_join` to merge the current dataframe with `wb_gdp_crt_PPP` after renaming its `rgdp_total` column to `rgdp_total_current_PPP`. Keep the current dataframe as the starting data.
    - Filter the data to include only rows where the `year` column equals `2017`
    - Create a new column `conv_fact` with values calculated as `rgdp_total_current_PPP/rgdp_total_const_2021_PPP`.
    - Select only the columns `iso` and `conv_fact`.
    - Refer to the resulting file as `convers_factor`.
  - **Adjust GDP Data:**
    - Start from the `wb_gdp_const_2021_PPP` dataframe.
    - Perform a `left_join` to merge the current dataframe with `convers_factor`. Keep the current dataframe as the starting data.
    - Adjust the `rgdp_total` column values by multiplying them with `conv_fact` column.
    - Remove the `conv_fact` column.
    - Refer to the resulting file as `wb_gdp_const_2017_PPP`.

3. **UNdata National GDP:**

   Here we supplement the above IMF and World Bank national GDP data using the UNdata for the following countries: `CUB`, `ERI`, and `PRK`.

   Prepare Population:

   - Use World Bank population data file `API_SP.POP.TOTL_DS2_en_excel_v2_294626.xls`.
   - Use `read_excel` to read the sheet named `Data`, skipping the first three rows.
   - Rename the `Country Code` column to `iso`, rename the `Country Name` column to `Country`
   - Select only the rows where the `iso` column values are within `c("CUB", "ERI", "PRK")`.

- Use pivot_longer to reshape the dataset from wide to long format by:
  - Gathering all columns with names matching a four-digit year (e.g., 2020, 2021).
  - Removing the prefix X from those column names.
  - Storing the resulting values in a new column named year.
- Rename the value column to population
- Select the columns iso, year, population, and Country
- Change the year column values to numeric.
- Filter the year column to retain only values within the range 2012 to 2022.
- Refer to the resulting dataframe as pop_cub_eri_prk.

UNdata GDP at Current USD:

- **Download Data:** Obtain the UNdata GDP at current USD as described in Appendix Section 1.2.2 and save it as UNdata_Export_20240613_003754887.csv. Read the file using read.csv.
- Create a new iso column with values set to CUB when the Country.or.Area column is "Cuba", PRK when Country.or.Area is "Democratic People's Republic of Korea", and ERI when Country.or.Area is "Eritrea", assigning NA to all other rows.
- Rename the Year column to year and the Value column to rgdp_pc.
- Convert the year column and the rgdp_pc column to numeric.
- Select only the columns iso, year, and rgdp_pc.
- Apply left_join to combine the current dataframe with the previously processed pop_cub_eri_prk dataset to obtain population data. Keep the current dataframe as the starting data.
- Create a new column rgdp_total with values calculated as rgdp_pc * population / 1e9.
- Select only the columns iso, year, rgdp_total, population, and Country.
- Refer to the resulting dataframe as un_gdp_const_2017.

UNdata GDP at Constant 2017 USD:

- Start with un_gdp_const_2017 file obtained in the previous step.
- Apply left_join to combine the current dataframe with the index dataframe. Keep the current dataframe as the starting data.
- Create a new column index_2017 containing the value of the index column value for the row where the year column equals 2017.
- Update the rgdp_total column by recalculating its values as rgdp_total * (index_2017 / index).
- Select only the columns iso, year, rgdp_total, population, and Country.
- Refer to the resulting dataframe as un_gdp_const_2017.

4. **Finalize National GDP Datasets:**

National GDP at Current USD:

- Apply bind_rows to combine dataframes imf_gdp_crt_us, wb_gdp_crt_us, and un_gdp_crt_us
- Save the resulting file as national_gdp_current_USD.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP at Constant 2017 USD:

- Apply bind_rows to combine dataframes imf_gdp_const_2017, wb_gdp_const_2017, and un_gdp_const_2017
- Save the resulting file as national_gdp_const_2017_USD.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP at Current International Dollars:

- Apply bind_rows to combine dataframes imf_gdp_crt_PPP and wb_gdp_crt_PPP
- Save the resulting file as national_gdp_current_PPP.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP at Constant 2017 International Dollars:

- Apply bind_rows to combine dataframes imf_gdp_const_2017_PPP and wb_gdp_const_2017_PPP
- Save the resulting file as national_gdp_const_2017_PPP.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

5. **Finalize National GDP per Capita Datasets:**

National GDP per Capita at Current USD:

- Take the file national_gdp_current_USD.csv
- Create a new column national_gdpc with values as rgdp_total/population
- Save the resulting file as national_gdpc_current_USD.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP per Capita at Constant 2017 USD:

- Take the file national_gdp_const_2017_USD.csv
- Create a new column national_gdpc with values as rgdp_total/population
- Save the resulting file as national_gdpc_const_2017_USD.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP per Capita at Current International Dollars:

- Take the file national_gdp_current_PPP.csv
- Create a new column national_gdpc with values as rgdp_total/population
- Save the resulting file as national_gdpc_current_PPP.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

National GDP per Capita at Constant 2017 International Dollars:

- Take the file national_gdp_const_2017_PPP.csv
- Create a new column national_gdpc with values as rgdp_total/population
- Save the resulting file as national_gdpc_const_2017_PPP.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

15 Regional GDP Data for Certain Developing Countries from the DOSE Dataset

1. Obtain regional GDP data from the DOSE dataset for the following developing countries: THA, MOZ, UZB, KEN, VNM, SRB, ECU, BLR, ALB, LKA, BIH. Note, however, that this dataset is not regularly updated and does not have complete data for the years 2012 to 2021.

2. Download the DOSE dataset as described in Appendix Section 1.2.1 and save it as DOSE_V2.csv.

3. Download the DOSE spatial geometry data as described in Appendix Section 1.1 and save them as all_non_GADM_regions.shp and gadam36_1.shp

4. **Geometry:**

   - Use st_read to read the gadm36_1.shp file and exclude rows where the GID_0 column value belongs to "KAZ", "MKD", "NPL", "PHL", "LKA".
   - Apply rbind to combine the data with the all_non_GADM_regions.shp file excluding the fid column.
   - Rename the geometry column to geom.
   - Select only the columns GID_0 and GID_1.
   - Rename the GID_0 column to iso and the GID_1 column to id.
   - Filter the rows where the iso column value belongs to "THA", "MOZ", "UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH".
   - Save the resulting file as DOSE_certain_developing_isos.gpkg.
   - Exclude larger inland waters by applying the qgis_run_algorithm function from the qgisprocess package with the algorithm native:difference. Use DOSE_certain_developing_isos.gpkg as the input layer, glwd_1.shp as the overlay layer, and save the output as DOSE_certain_developing_isos_without_large_water.gpkg.

5. **GDP Data:**

   - Read the downloaded DOSE_V2.csv file using read.csv.
   - Select only the columns GID_0, GID_1, year, grp_lcu, pop, and grp_pc_lcu.

- Rename the column GID_0 to iso and GID_1 to id.
- Select only the rows where the year column is greater than or equal to 2012 .
- Select only the rows where the iso column value belongs to "THA", "MOZ", " UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH".
- Arrange the dataset in the order of iso, year, and id.
- Identify combinations of iso and year where the grp_lcu data is missing. Exclude all rows corresponding to these iso-year combinations.
- Save the resulting file as DOSE_gdp_full.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

16 Rescale Regional GDP Data

Regional GDP data are provided in varying units, so it is necessary to rescale them to align with national GDP data.

1. **Adjust the Training Regional GDP Data:** For the countries BEL, DNK, ESP, FRA, GBR, ITA, NLD, and NOR, some portions of the national GDP are not regionalized according to the OECD dataset. As no clear explanation is provided for this discrepancy, we rescale the regional GDP data to ensure that their sum aligns with the national GDP data obtained in Section 14 through the following steps:

   Process the files: chn_training_data.csv, ind_training_data.csv, kaz_training_data.csv, kgz_training_data.csv, oecd_training_data.csv, phl_training_data.csv, and usa_training_data.csv.

   Read Data: For each file, use read_csv to read the data and convert the id column to character using as.character.

   Combine Data: Combine all files into a single dataframe using bind_rows.

   Remove Unnecessary Columns: Remove the parent_id column.

   Filter Years: Select rows where the year column is between 2012 and 2021 (inclusive).

   Resulting Dataframe: Refer to the resulting dataframe as training_data_complete_pre.

   Create the Scalar:

   - Filter training_data_complete_pre to include only rows where parent_admin_unit = 1.
   - Create a new column aggre_regio_GDP by summing unit_rgdp_total grouped by iso and year.
   - Remove the grouping
   - Use distinct to retain unique combinations of year, iso, parent_rgdp_total, and aggre_regio_GDP.

- Create a new column scalr with values calculated as parent_rgdp_total / aggre_regio_GDP.
- Refer to the resulting dataframe as scalar.

Merge Data: Start with training_data_complete_pre file, apply left_join to merge with the file scalar to add the scalr column.

Update Regional GDP: Update the unit_rgdp_total column values by multiplying them by scalr, but only for rows where the iso column values belong to: "BEL", "DNK", "ESP", "FRA", "GBR", "ITA", "NLD", or "NOR".

Update Parent GDP: Update the parent_rgdp_total column values by multiplying them by scalr, but only for rows where the parent_admin_unit column equals 2 and the iso column values belong to: "BEL", "DNK", "ESP", "FRA", "GBR", "ITA", "NLD", or "NOR".

Finalize Data: Remove the scalr column and refer to the resulting dataframe as training_data_complete.

2. **Rescale Regional GDP Data**

Create the Scalar:

- Filter training_data_complete to include only rows where parent_admin_unit = 1.
- Retain unique combinations of year, iso, parent_name, and parent_rgdp_total.
- Apply left_join to combine the current dataframe with national_gdp_const_2017_USD.csv with rgdp_total renamed to rgdp_2017_USD. Keep the current dataframe as the starting data.
- Create a new column scale_factor with values as rgdp_2017_USD / parent_rgdp_total.
- Refer to the resulting dataframe as national_gdp_scale_factor.

Perform Rescaling:

- Start with the training_data_complete dataframe.
- Apply left_join to merge it with national_gdp_scale_factor to add the column scale_factor.
- Update all columns with names containing rgdp_total by multiplying their values by the corresponding scale_factor column values.
- Remove the columns scale_factor and rgdp_2017_USD.
- Rename the population column to national_population.
- Refer to the resulting dataframe as training_data_rescaled.

Process Alaska:

- Start with the usa_state_gdp.csv data.
- Select only the rows where the admin_2_name column value is Alaska.
- Filter for year values between 2012 and 2021, inclusive.

- Apply left_join to merge the current dataframe with national_gdp_scale_factor to add the scale_factor column. Keep the current dataframe as the starting data.
- Update all columns with names containing rgdp_total by multiplying their values by the corresponding scale_factor column values.
- Create new columns: id with the value 02, unit_name with the value Alaska, iso with the value Ala, min_admin_unit with the value 2, parent_admin_unit with the value 1, and parent_name with the value United States.
- Rename the admin_1_rgdp_total column to parent_rgdp_total, the population column to national_population, and the admin_2_rgdp_total column to unit_rgdp_total.
- Remove the columns: state_fips, admin_2_name, admin_1_name, rgdp_2017_USD, and scale_factor.
- Refer to the resulting dataframe as alaska_state.

Process Countries without any Regional GDP Data:

- Start with the national_gdp_const_2017_USD.csv file obtained in Section 14.
- Rename the rgdp_total column to rgdp_2017_USD.
- Select only rows where the iso column value is not present in the iso column values of the training_data_rescaled dataframe.
- Create new columns: id with values equal to the iso column, min_admin_unit with the value 1, unit_rgdp_total with NA, parent_admin_unit with the value 1, and parent_name with values equal to the Country column.
- Rename the Country column to unit_name, the rgdp_2017_USD column to parent_rgdp_total, and the population column to national_population.
- Refer to the resulting dataframe as national_gdp_rest.

3. **Finalize the Rescaled Regional GDP:**

   Apply rbind to combine the dataframes: training_data_rescaled, national_gdp_rest, and alaska_state.

   Arrange the columns in the order of iso, year, and parent_name.

   Select only rows where year is between 2012 and 2021, inclusive.

   Save the resulting file as rgdp_total_rescaled.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

4. **Rescale DOSE Regional GDP Data:**

   Apply read.csv to read the file DOSE_gdp_full.csv obtained in Section 15.

   Apply left_join to combine the current dataframe with national_gdp_const_2017_USD.csv obtained in Section 14, renaming its rgdp_total column to rgdp_2017_USD. Keep the current dataframe as the starting data.

Create a new column unit_rgdp_total with values calculated as rgdp_2017_USD * grp_lcu / sum(grp_lcu), grouped by iso and year.

Remove the grouping

Rename rgdp_2017_USD column to parent_rgdp_total and population column to national_population.

Select only columns iso, id, year, unit_rgdp_total, parent_rgdp_total, and national_population.

Save the resulting file as DOSE_certain_developing_isos_total_rescaled.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

17 Organizing Regional and National Geometries

In this section, we organize the geometry data obtained in previous sections to create comprehensive geometry files. These include geometries for regional GDP data in the training sample, administrative levels used for predictions in non-training samples, and world national boundaries.

1. **Geometry for Regional GDP Data in Training Sample**

   For the geometry files obtained in earlier sections: "chn_admin_2.gpkg", "ind_admin_2.gpkg", "kaz_admin_2.gpkg", "kgz_admin_2.gpkg", "phl_admin_2.gpkg", and "usa_admin_3.gpkg":

   - Use read_sf to read each gpkg file and set the CRS to 4326 using st_set_crs(4326).
   - Combine all files using rbind.

   Refer to the resulting dataframe as regional_subnational_poly.

   For the oecd_training_poly.gpkg file:

   - Use read_sf to read the file.
   - Select only the columns id, iso, and geom.

   Use rbind to append the processed oecd_training_poly to regional_subnational_poly.

   Set the CRS to 4326 using st_set_crs.

   Save the resulting file as training_poly_full.gpkg.

   Select the rows in training_poly_full.gpkg where the iso column values belong to the following list: "AUT", "BEL", "BGR", "CHE", "CZE", "DEU", "DNK", "ESP", "FIN", "FRA", "GBR", "GRC", "HUN", "ITA", "JPN", "KOR", "LTU", "NLD", "NOR", "POL", "PRT", "ROU", "SVK", "HRV", "LVA", "SVN", "SWE", "TUR", "NZL", "IDN", "COL", "PER", "CHL", "EST", "USA", "KGZ", "PHL", "THA", "MOZ", "UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH".

Use bind_rows to append the following geometry data from the DOSE file:

- Read DOSE_certain_developing_isos_without_large_water.gpkg.
- Filter rows where iso column values belong to "THA", "MOZ", "UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH".

Save the resulting file as training_poly_sample.gpkg.

2. **Construct Global National Geometries**

Process HongKong and Macau:

- Treat Hong Kong and Macau as two separate "countries" since their GDP values are not included in China's GDP.
- Start with the gdam_prov_level1_without_largewater.gpkg file obtained in Section 3, Step 4.
- Select rows where GID_0 equals CHN and NAME_1 equals either Hong Kong or Macau.
- Remove the columns NAME_1 and ENGTYPE_1.
- Rename the GID_0 column to iso.
- Update the iso column values to HKG for Hong Kong and MAC for Macau.
- Set the CRS to 4326 using st_set_crs.
- Retain only the iso column. Note that since the dataframe is an sf object, the geometry column will still be included automatically.
- Refer to the resulting dataframe as hkg_mac.

Process CHN, IND, KAZ, KGZ, PHL, USA

- Start with the regional_subnational_poly file.
- Remove the id column.
- Set the CRS to 4326 using st_set_crs.
- Refer to the resulting dataframe as regional_subnational_poly_noid.

Process OECD

- Read the oecd_poly.gpkg file obtained in Section 6.
- Exclude rows where the iso column values are "CHN", "IND", "KAZ", "KGZ", "PHL", or "USA".
- Set the CRS to 4326 using st_set_crs.
- Exclude rows where the iso column value equals the id column value.
- Remove the id column.
- Refer to the resulting dataframe as oecd.

Process Pakistan

- Read the file gadm_country_level0_without_largerwater.gpkg obtained in Section 3 Step 3.
- Select only the rows where the GID_0 column equals PAK.

- Rename the GID_0 column to iso.
- Retain only the iso column.
- Set the CRS to 4326 using st_set_crs.
- Refer to the resulting dataframe as pak.

Process Island Countries

- Read the file gadm_country_level0_without_largerwater.gpkg obtained in Section 3 Step 3.
- Select only the rows where the GID_0 column value is one of "ABW", "BMU", "CUW", "CYM", "PRI", "PSE", "SXM", or "TCA".
- Rename the GID_0 column to iso.
- Retain only the iso column.
- Set the CRS to 4326 using st_set_crs.
- Refer to the resulting dataframe as islands.

Process Alaska

- Read the file gdam_prov_level1_without_largewater.gpkg obtained in Section 3 Step 4.
- Select only the rows where the GID_0 column equals USA and the NAME_1 column equals Alaska.
- Rename the GID_0 column to iso.
- Retain only the iso column.
- Set the CRS to 4326 using st_set_crs.
- Refer to the resulting dataframe as alaska.

Finalize the National Geometries

- Read the file CGAZ_ADM1_without_large_waters.gpkg obtained in Section 3 Step 5.
- Rename the shapeGroup column to iso.
- Set the CRS to 4326 using st_set_crs.
- Remove rows where the iso column has NA values.
- Exclude rows where the iso column value is among the values already processed: "HKG", "MAC", "AUS", "COL", "BRA", "CAN", "IDN", "KOR", "JPN", "MEX", "PER", "RUS", "NZL", "CHL", "AUT", "GRC", "EST", "FRA", "HUN", "HRV", "IRL", "ALB", "DEU", "DNK", "ESP", "BEL", "BGR", "CHE", "CYP", "NLD", "ITA", "FIN", "CZE", "GBR", "PRT", "ROU", "LIE", "LTU", "LUX", "TUR", "ISL", "LVA", "MNE", "MKD", "MLT", "SRB", "NOR", "SWE", "POL", "SVN", "SVK", "CHN", "IND", "KAZ", "KGZ", "PHL", "USA", "PAK".
- Select only the columns iso and geom.
- Use rbind to combine the previously processed files: hkg_mac, oecd, regional_subnational_poly_noid, pak, islands, alaska.
- Refer to the resulting file as world_poly_pre.gpkg.

- Aggregate the geometries to the iso level using the qgis_run_algorithm function from the qgisprocess package with the algorithm native:aggregate, setting the input layer to world_poly_pre.gpkg. Use the following additional parameters:
  - GROUP_BY = "iso"
  - AGGREGATES = list(list("aggregate" = "concatenate", "input" = '"iso"', "delimiter" = ",", "name" = "iso", "type" = 10, "length" = 0, "precision" = 0))
- For the output file, change the iso column values to the first three characters of their original values.
- Exclude rows where the iso column value is among the following: "ATA", "111", "112", "113", "114", "115", "116", "117", "118", "119", "120", "121", "122", "123", "124", "125", "126", "127", "128", "129".
- Save the resulting file as world_poly.gpkg.

3. **Complete Geometry File for Training and Non-Training Samples**

For the world_poly.gpkg file, create the id column with values to "02" for rows where the iso column equals "Ala", and for all other rows, set the id value equal to the iso column value.

Use rbind to combine the dataframe with the training_poly_full.gpkg file obtained in previous steps.

Update the row where the id column value is "Bangsamoro Autonomous Region\r\nin Muslim Mindanao_PHL" to "Bangsamoro Autonomous Region\nin Muslim Mindanao_PHL".

Select only the rows where the id column values are also in the id column values of the file rgdp_total_rescaled.csv from Section 16, ensuring the following steps are completed before performing the selection:

- Read the file rgdp_total_rescaled.csv using read.csv with the encoding parameter set to "UTF-8".
- Exclude rows where the iso column values are "UVK" or "WBG", as these are already represented by "XKX" and "PSE" respectively.

Save the resulting combined geometry file as complete_poly.gpkg.

18 Construct Cell True GDP

1. **GDP Data in the Training Sample at the Originally Collected Administrative Level**

Start with Rescaled GDP Data: Start with the file rgdp_total_rescaled.csv obtained in Section 16, Step 3.

Filter by Training Sample ISO Codes: Filter the data to include only rows where the iso column value is one of the following values: "AUT", "BEL", "BGR", "CHE", "CZE", "DEU", "DNK", "ESP", "FIN", "FRA", "GBR", "GRC", "HUN", "ITA", "JPN", "KOR", "LTU", "NLD", "NOR", "POL", "PRT", "ROU", "SVK", "HRV", "LVA", "SVN", "SWE", "TUR", "NZL", "IDN", "COL", "PER", "CHL", "EST", "USA", "KGZ", "PHL", "THA", "MOZ", "UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH".

Filter by Minimum Administrative Unit: Select only rows where the min_admin_unit column value does not equal to 1.

Filter by Parent Administrative Unit: Select only rows where the parent_admin_unit column value equals to 1.

Remove Unnecessary Columns: Exclude the columns unit_name, min_admin_unit, parent_admin_unit, and parent_name.

Combine with Additional Dataset: Use bind_rows to combine the resulting dataframe with the DOSE_certain_developing_isos_total_rescaled.csv dataset obtained in Section 16, Step 4.

Save Dataset: Save the resulting dataset as rgdp_total_training_data.csv, ensuring the output excludes row names by setting the parameter row.names = FALSE.

2. **Population in the Administrative Regions of the Training Sample's GDP Data**

Download Population Data: Download population data as described in Appendix Section 1.3. The files should be in tif format and named landscan-global-20xx, with each file representing a specific year.

Handle Alaska Population: Note that Alaska is treated as a separate "country" in our analysis, but the official national population data still includes Alaska. To accurately calculate each US county's national population share, Alaska must be included in the US during this calculation. For each population file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 5 for concurrent processing):

- Read the population tif file using rast.
- From the world_poly.gpkg file, select only the row where the iso column has the value Ala. Refer to the resulting file as alaska
- Perform a spatial extraction using the exact_extract function from the exactextractr package with the sum operation to compute the population in the alaska polygon.
- Rename the extracted population column to pop for clarity.
- Create a new column year with the value extracted from the tif file name.
- Use cbind to combine the above pop and year columns with the iso and geom columns from alaska.

Combine Dataframes: Use bind_rows to combine the resulting dataframes for all years into a single dataframe.

Convert and Save Data: Convert the combined sf object into a dataframe using as.data.frame and exclude the geom column.

Add Columns: Add a new column id with the value Ala and a new column iso with the value USA.

Save Dataset: Save the resulting dataframe as alaska_population.csv.

Process Other Polygons: Next, process the population data for other polygons. For each population file from 2012 to 2021, perform the following steps (again, use mclapply with mc.cores = 5 for concurrent processing):

- Read the population tif file using rast.
- Perform a spatial extraction using the exact_extract function from the exactextractr package with the sum operation to compute the population for each polygon in the training_poly_sample.gpkg file obtained in Section 17, Step 1.
- Rename the extracted population column to pop for clarity.
- Use cbind to combine the above pop column with the id, iso, and geom columns from training_poly_sample.gpkg.
- Add a new column year with value as the integer form of the corresponding year from the tif file name.

Combine Dataframes: Use bind_rows to combine the resulting dataframes for all years into a single dataframe.

Merge Data: Combine the current dataframe with alaska_population.csv using bind_rows.

Calculate Population Share: For each group defined by iso and year, create a new column pop_share with values pop/sum(pop). Remove the grouping afterward.

Exclude Alaska Data: Exclude rows where the id column equals Ala, as Alaska's data is not used in the training sample and is included only to calculate the correct US county national population share.

Save File: Save the final file as land_pop_extracted_train_county.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

3. **GDP per Capita at the Originally Collected Administrative Level**

Start with Training Data: Start from the file rgdp_total_training_data.csv.

Exclude Missing Data for Thailand: Exclude rows where the iso value is THA and the year value is in (2012, 2013, 2019) due to missing data.

Exclude Missing Data for Ecuador: Exclude rows where the iso value is ECU and the year value is in (2012, 2013, 2014) due to missing data.

Update Specific Row: Update the row where the id column value is "Bangsamoro Autonomous Region\nin Muslim Mindanao_PHL" to "Bangsamoro Autonomous Region\r\nin Muslim Mindanao_PHL".

Join with Population Data: Ensure that the current dataframe is used as the starting file. Apply left_join with parameters by = c("id", "iso", "year") to merge it with a dataset created through the following steps:

- Read the land_pop_extracted_train_county.RData file using load.
- Select columns "id", "iso", "year", "pop_share", "geom".

Rescale Population Data: Create a new column pop_rescaled equal to round(pop_share * national_population). This step rescales the extracted population data so that the sum matches the official national population data.

Calculate GDP per Capita: Create a new column county_GDPC, where the value is 0 if pop_rescaled equals 0; otherwise, calculate it as unit_rgdp_total / pop_rescaled.

Select Relevant Columns: Select only the columns "id", "iso", "year", "county_GDPC", "geom".

Convert to Spatial File: Turn the dataframe into an sf file using st_as_sf.

Save the Final File: Save the resulting file as county_GDPC.gpkg.

4. **Obtain Cell Grids**

1-degree Cell Grids:

- Generate a raster grid with 1° x 1° resolution using rast(resolution = 1, crs = "epsg:4326") from the terra package, dividing the world into 64800 cells.
- Set all grid cells to NA using setValues(NA).
- Convert the raster to vector format using st_as_stars.
- Transform the vector into an sf object while retaining NA values using st_as_sf(na.rm = F).
- Add a new column cell_id which adds a unique ID to each grid cell using rownames(.).
- Exclude the column lyr.1.
- Save the 1-degree grid as just_grid_1degree.gpkg.

0.5-degree Cell Grids:

- Create 0.5-degree grids using qgis_run_algorithm with the algorithm native:creategrid and the following parameters:
  - TYPE = 2,
  - EXTENT = "-180,180,-90,90",
  - HSPACING = 0.5,
  - VSPACING = 0.5,
  - HOVERLAY = 0,
  - VOVERLAY = 0,
  - CRS = "EPSG:4326",
  - OUTPUT = tempfile(fileext = ".gpkg")

- Determine which 0.5-degree grids belong to which 1-degree grid using qgis_run_algorithm with the algorithm native:intersection, where the input layer is the 0.5-degree grids created above and the overlay layer is just_grid_1degree.gpkg. Save the output as subcell_0_5grid.gpkg.
- Read subcell_0_5grid.gpkg using read_sf.
- Select the column cell_id.
- Group by cell_id, create a new column subcell_id with sequential numbers assigned using row_number().
- Remove the grouping.
- Arrange rows in ascending order of as.numeric(cell_id) column.
- Save the resulting grid as just_grid_0_5degree.gpkg.

0.25-degree Cell Grids:

- Create 0.25-degree grids using qgis_run_algorithm with the algorithm native:creategrid and the following parameters:
  - TYPE = 2,
  - EXTENT = "-180,180,-90,90",
  - HSPACING = 0.25,
  - VSPACING = 0.25,
  - HOVERLAY = 0,
  - VOVERLAY = 0,
  - CRS = "EPSG:4326",
  - OUTPUT = tempfile(fileext = ".gpkg")
- Determine which 0.25-degree grids belong to which 0.5-degree grid using qgis_run_algorithm with the algorithm native:intersection, where the input layer is the 0.25-degree grids created above and the overlay layer is just_grid_0_5degree.gpkg. Save the output as subcell_0_25grid.gpkg.
- Read subcell_0_25grid.gpkg using read_sf.
- Select the columns cell_id, subcell_id, and id.
- Group by cell_id and subcell_id, arrange rows in each group in ascending order of id, and create a new column subcell_id_0_25 with sequential numbers assigned using row_number().
- Remove the grouping.
- Exclude the column id.
- Sort the data in ascending order by the numeric values of cell_id and then subcell_id.
- Save the resulting grid as just_grid_0_25degree.gpkg.

5. **Intersect Administrative-Level GDP Geometry with Grids**

Intersect with 1-degree Grid: Use the qgis_run_algorithm function with the algorithm "native:intersection". The input layer is the county_GDPC.gpkg file obtained in previous steps, the overlay layer is `just_grid_1degree.gpkg`, and the output is saved as `county_gridded_1degree.gpkg`.

Intersect with 0.5-degree Grid: Use the `qgis_run_algorithm` function with the algorithm `"native:intersection"`. The input layer is the `county_GDPC.gpkg` file obtained in previous steps, the overlay layer is `just_grid_0_5degree.gpkg`, and the output is saved as `county_gridded_0_5degree.gpkg`.

Intersect with 0.25-degree Grid: Use the `qgis_run_algorithm` function with the algorithm `"native:intersection"`. The input layer is the `county_GDPC.gpkg` file obtained in previous steps, the overlay layer is `just_grid_0_25degree.gpkg`, and the output is saved as `county_gridded_0_25degree.gpkg`.

6. **Extract Population Data for Intersected Polygons**

- **1-degree:**
  - For each population `tif` file from 2012 to 2021, perform the following steps (use `mclapply` with `mc.cores = 5` for efficiency):
    * Read the `county_gridded_1degree.gpkg` file use `read_sf`
    * Exclude the `fid_2` column
    * Select rows where the `year` column matches the year value in the population file name.
    * refer to the resulting file as `county_with_1cellid_year`
    * Read the population `tif` file using `rast`.
    * Use the `exact_extract` function with the `sum` operation to calculate the population for each polygon in the `county_with_1cellid_year` file.
    * Rename the extracted population column to `pop` for clarity.
    * Use `cbind` to combine the above `pop` column with the `id`, `iso`, `year`, `county_GDPC`, `cell_id`, and `geom` columns from the `county_with_1cellid_year` file.
  - Use `bind_rows` to combine the resulting dataframes for all years into one single dataframe.
  - Replace `NA` values in the `pop` column with `0`.
  - Save the resulting file as `county_cell_pop_extracted_1deg.RData`. Ensure that the dataframe name matches the name of the `RData` file when saving it.
- **0.5-degree:**
  - For each population `tif` file from 2012 to 2021, perform the following steps (use `mclapply` with `mc.cores = 5` for efficiency):
    * Read the `county_gridded_0_5degree.gpkg` file use `read_sf`
    * Exclude the `fid_2` column
    * Select rows where the `year` column matches the year value in the population file name.
    * refer to the resulting file as `county_with_0_5cellid_year`

- * Read the population tif file using rast.
  * Use the exact_extract function with the sum operation to calculate the population for each polygon in the county_with_0_5cellid_year file.
  * Rename the extracted population column to pop for clarity.
  * Use cbind to combine the above pop column with the id, iso, year, county_GDPC, cell_id, subcell_id, and geom columns from the county_with_0_5cellid_year file.
- Use bind_rows to combine the resulting dataframes for all years into one single dataframe.
- Replace NA values in the pop column with 0.
- Save the resulting file as county_cell_pop_extracted_0_5deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

- **0.25-degree:**
  - For each population tif file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 5 for efficiency):
    * Read the county_gridded_0_25degree.gpkg file use read_sf
    * Exclude the fid_2 column
    * Select rows where the year column matches the year value in the population file name.
    * refer to the resulting file as county_with_0_25cellid_year
    * Read the population tif file using rast.
    * Use the exact_extract function with the sum operation to calculate the population for each polygon in the county_with_0_25cellid_year file.
    * Rename the extracted population column to pop for clarity.
    * Use cbind to combine the pop column with the id, iso, year, county_GDPC, cell_id, subcell_id, and geom columns from the county_with_0_25cellid_year file.
  - Use bind_rows to combine the resulting dataframes for all years into one single dataframe.
  - Replace NA values in the pop column with 0.
  - Save the resulting file as county_cell_pop_extracted_0_25deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

7. **Obtain Cell GDP**

1-degree:

- Start with the file rgdp_total_training_data.csv.
- Exclude rows where iso is THA and year belongs to (2012, 2013, 2019) due to missing data.

- Exclude rows where iso is ECU and year belongs to (2012, 2013, 2014) due to missing data.
- Create a new column iso_real with values equal to the iso column.
- For rows where iso equals USA, update the iso value to paste0("USA_", substr(id, 1, 2)). Other rows remain unchanged.
- Group by iso and year columns, and create a new column state_total_GDP with values equal to the sum of unit_rgdp_total within each group.
- Remove the grouping.
- Update the row where the id column equals "Bangsamoro Autonomous Region \nin Muslim Mindanao_PHL" to "Bangsamoro Autonomous Region\r\nin Muslim Mindanao_PHL".
- Refer to the resulting dataframe as county_GDP_change_USA.
- Read alaska_population.csv using read.csv.
- Rename the iso column to iso_real.
- Exclude the X column.
- Refer to the resulting dataframe as alaska_pop.
- Read county_cell_pop_extracted_1deg.RData using load.
- Convert the data to a dataframe using as.data.frame().
- Select only the columns cell_id, id, iso, year, county_GDPC, and pop.
- For rows where iso equals USA, update the iso value to paste0("USA_", substr(id, 1, 2)). Other rows remain unchanged.
- Combine the current dataframe with county_GDP_change_USA using left_join. Ensure that the current dataframe is the starting file.
- Combine with alaska_pop using bind_rows.
- Group by year and iso_real columns, create a new column GDP_subcell calculated as county_GDPC * round(national_population * pop / sum(pop)).
- Remove the grouping.
- Exclude rows where id equals Ala.
- Group by iso and year columns, create a new column GDP_subcell_rescl calculated as GDP_subcell * state_total_GDP / sum(GDP_subcell).
- Remove the grouping.
- Group by year, iso, and cell_id columns, and create a new column GCP_1deg with values equal to sum(GDP_subcell_rescl).
- Remove the grouping.
- Select only the columns cell_id, iso, year, GCP_1deg, state_total_GDP, parent_rgdp_total, and national_population.
- Select distinct rows based on year, iso, and cell_id, retaining all other columns using .keep_all = TRUE.
- Save the resulting file as training_iso_1deg_cell_GCP.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

0.5-degree:

- Read county_cell_pop_extracted_0_5deg.RData using load.
- Convert the data to a dataframe using as.data.frame().
- Select only the columns cell_id, subcell_id, id, iso, year, county_GDPC, and pop.
- For rows where iso equals USA, update the iso value to paste0("USA_", substr(id, 1, 2)). Other rows remain unchanged.
- Combine the current dataframe with county_GDP_change_USA using left_join. Ensure that the current dataframe is the starting file.
- Combine with alaska_pop using bind_rows.
- Group by year and iso_real columns, create a new column GDP_subcell calculated as county_GDPC * round(national_population * pop / sum(pop)).
- Remove the grouping.
- Exclude rows where id equals Ala.
- Group by iso and year columns, create a new column GDP_subcell_rescl calculated as GDP_subcell * state_total_GDP / sum(GDP_subcell).
- Remove the grouping.
- Group by year, iso, cell_id, and subcell_id columns, and create a new column GCP_0_5deg with values equal to sum(GDP_subcell_rescl).
- Remove the grouping.
- Select only the columns cell_id, subcell_id, iso, year, GCP_0_5deg, state_total_GDP, parent_rgdp_total, and national_population.
- Select distinct rows based on year, iso, cell_id, and subcell_id, retaining all other columns using .keep_all = TRUE.
- Save the resulting file as training_iso_0_5deg_cell_GCP.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

0.25-degree:

- Read county_cell_pop_extracted_0_25deg.RData using load.
- Convert the data to a dataframe using as.data.frame().
- Select only the columns cell_id, subcell_id, subcell_id_0_25, id, iso, year, county_GDPC, and pop.
- For rows where iso equals USA, update the iso value to paste0("USA_", substr(id, 1, 2)). Other rows remain unchanged.
- Combine the current dataframe with county_GDP_change_USA using left_join. Ensure that the current dataframe is the starting file.
- Combine with alaska_pop using bind_rows.
- Group by year and iso_real columns, create a new column GDP_subcell_0_25 calculated as county_GDPC * round(national_population * pop / sum(pop)).

- Remove the grouping.
- Exclude rows where id equals Ala.
- Group by iso and year columns, create a new column GDP_subcell_0_25_rescl calculated as GDP_subcell_0_25 * state_total_GDP / sum(GDP_subcell_0_25).
- Remove the grouping.
- Group by year, iso, cell_id, subcell_id, and subcell_id_0_25 columns, and create a new column GCP_0_25deg with values equal to sum(GDP_subcell_0_25_rescl).
- Remove the grouping.
- Select only the columns cell_id, subcell_id, subcell_id_0_25, iso, year, GCP_0_25deg, state_total_GDP, parent_rgdp_total, and national_population.
- Select distinct rows based on year, iso, cell_id, subcell_id, and subcell_id_0_25, retaining all other columns using .keep_all = TRUE.
- Save the resulting file as training_iso_0_25deg_cell_GCP.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

19 Retrieve the Geometry and GDP of administrative regions used for GDP share and predictor share calculations

1. **Geometry**

Read Input Files: Read the files complete_poly.gpkg and world_poly.gpkg obtained in Section 17 using read_sf.

US States as Reference Units: For the United States, states serve as the reference units for calculating GDP and predictor shares:

- Select only rows where iso equals USA from the complete_poly.gpkg file.
- Update the id column to contain the character version of the first two characters of its original values.
- Group by the id column and use reframe to create a single combined geometry for each group using st_union(geom).
- Add a new column iso with the value USA.
- Refer to the resulting dataframe as USA.

Other Training Countries as Reference Units: For the remaining training countries, entire countries serve as the reference units for calculating GDP and predictor shares:

- Start with the world_poly.gpkg file.

- Select rows where the iso column value is one of the following training countries: "AUT", "BEL", "BGR", "CHE", "CZE", "DEU", "DNK", "ESP", "FIN", "FRA", "GBR", "GRC", "HUN", "ITA", "JPN", "KOR", "LTU", "NLD", "NOR", "POL", "PRT", "ROU", "SVK", "HRV", "LVA", "SVN", "SWE", "TUR", "NZL", "IDN", "THA", "MOZ", "UZB", "KEN", "VNM", "SRB", "ECU", "BLR", "ALB", "LKA", "BIH", "COL", "PER", "CHL", "KGZ", "PHL", "EST".
- Add a new column id with values equal to iso.
- Select only the columns id, iso, and geom.
- Refer to the resulting dataframe as rest_train_iso_county_bound.

Non-Training Countries as Reference Units: For countries not included in the training sample, use their reference units directly from the complete_poly.gpkg file:

- Select rows where the iso value is not part of the training sample, i.e., iso values that do not appear in either rest_train_iso_county_bound or USA dataframe.
- Select only the columns id and iso.
- Refer to the resulting dataframe as poly_nontraining.

Combine All Reference Units: Use bind_rows to combine the dataframes poly_nontraining, USA, and rest_train_iso_county_bound.

Adjust for Alaska: Change the id column value to Ala for rows where iso equals Ala.

Save the Resulting File: Save the final dataframe as world_model8_poly.gpkg.

2. **Intersect with Grids**

1-degree:

- Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection. The input layer is world_model8_poly.gpkg, and the overlay layer is just_grid_1degree.gpkg obtained in Section 18. Save the output as world_province_1deg_with_cellid.gpkg.

0.5-degree:

- Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection. The input layer is world_model8_poly.gpkg, and the overlay layer is just_grid_0_5degree.gpkg obtained in Section 18. Save the output as world_province_0_5deg_with_cellid.gpkg.

0.25-degree:

- Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection. The input layer is world_model8_poly.gpkg, and the overlay layer is just_grid_0_25degree.gpkg obtained in Section 18. Save the output as world_province_0_25deg_with_cellid.gpkg.

3. **GDP**

Read Input Data: Read the file rgdp_total_rescaled.csv obtained in Section 16, Step 3, using read.csv with encoding = "UTF-8".

USA:

- From rgdp_total_rescaled.csv, select rows where iso is USA and parent_admin_unit is 2.
- Update the id column values to the first two characters of their original values.
- Select distinct combinations of id, iso, year, parent_name, parent_rgdp_total, and national_population.
- Add a new column rescale_level with the value 2.
- Rename the column parent_rgdp_total to unit_gdp_af_sum_rescl.
- Ensure that the current dataframe is used as the starting file. Apply left_join to merge it with a dataset created through the following steps:
  - From rgdp_total_rescaled.csv, select rows where iso is USA and parent_admin_unit is 1.
  - Create a new column country_total_GDP with values equal to parent_rgdp_total.
  - Select only the columns iso, year, and country_total_GDP.
  - Remove duplicate rows by selecting distinct rows based on the year column, keeping all other columns intact, using distinct(year, .keep_all = TRUE).
- Exclude the parent_name column.
- Refer to the resulting dataframe as USA_GDP_final.

Countries Using National GDP Only:

- From world_model8_poly.gpkg, identify a list of iso values where the id column value equals the iso column value, excluding rows where id equals Ala.
- From rgdp_total_rescaled.csv, select rows where parent_admin_unit equals 1 and iso value belongs to the identified list.
- Add new columns:
  - rescale_level with the value 1.
  - unit_gdp_af_sum_rescl with values equal to parent_rgdp_total.
  - country_total_GDP with values equal to parent_rgdp_total.
  - id with values equal to iso.
- Select columns id, iso, year, rescale_level, unit_gdp_af_sum_rescl, country_total_GDP, and national_population.
- Select distinct rows based on year and iso using distinct(iso, year, .keep_all = TRUE).
- Refer to the resulting dataframe as iso_use_countryGDP_GDP.

Alaska:

- From rgdp_total_rescaled.csv, select rows where iso equals Ala.
- Add new columns:
  - rescale_level with the value 2.
  - unit_gdp_af_sum_rescl with values equal to unit_rgdp_total.
  - country_total_GDP with values equal to parent_rgdp_total.
  - id with values equal to iso.
- Update the iso column value to USA.
- Select columns id, iso, year, rescale_level, unit_gdp_af_sum_rescl, country_total_GDP, and national_population.
- Refer to the resulting dataframe as Ala.

Countries Using Second Administrative Level GDP:

- From world_model8_poly.gpkg, identify a list of id values where the id column value does not equal the iso column value, excluding id values belonging to USA.
- From rgdp_total_rescaled.csv, select rows where id belongs to the identified list.
- Add new columns:
  - rescale_level with the value 2.
  - unit_gdp_af_sum_rescl with values equal to unit_rgdp_total.
  - country_total_GDP with values equal to parent_rgdp_total.
- Select columns id, iso, year, rescale_level, unit_gdp_af_sum_rescl, country_total_GDP, and national_population.
- Refer to the resulting dataframe as iso_use_provinceGDP_GDP.

Combine Processed Dataframes:

- Use bind_rows to combine iso_use_countryGDP_GDP, USA_GDP_final, iso_use_provinceGDP_GDP, and Ala.
- Refer to the resulting dataframe as rgdp_total_af_sum_rescl_pre.

Fulfill Missing Data: Some countries in the training sample do not have regional GDP data from the OECD for 2020 and 2021. Consequently, their regional GDP data is missing from the rgdp_total_rescaled.csv file and, therefore, absent from the rgdp_total_af_sum_rescl_pre dataframe. However, for these countries, GDP and predictor calculations rely on country-level reference units, which require only country-level information that is already available. These column values can be completed at this step.

- Identify missing iso and year combinations in rgdp_total_af_sum_rescl_pre.
- From national_gdp_const_2017_USD.csv, rename rgdp_total to rgdp_2017_USD.

- Perform a semi_join to retain rows matching the identified iso and year combinations.
- Add new columns:
  - id with values equal to iso.
  - rescale_level with the value 1.
  - unit_gdp_af_sum_rescl with values equal to rgdp_2017_USD.
  - country_total_GDP with values equal to rgdp_2017_USD.
  - national_population with values equal to population.
- Exclude columns Country, rgdp_2017_USD, and population.
- Refer to the resulting dataframe as fulfill_gdp.

Final Output:

- Use rbind to combine fulfill_gdp and rgdp_total_af_sum_rescl_pre.
- Save the final dataframe as rgdp_total_af_sum_rescl.RData. Ensure that the dataframe name matches the name of the RData file when saving it. Also save it as rgdp_total_af_sum_rescl.csv, ensuring row names are excluded with row.names = FALSE.

20 Extract Cell Population

1. **1-degree**

Download Population Data: Download population data as described in Appendix Section 1.3. The files should be in tif format and named landscan-global-20xx, with each file representing a specific year.

Extract Population: For each population file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 5 for concurrent processing):

- Read the population tif file using rast.
- Perform a spatial extraction using the exact_extract function from the exactextractr package with the sum operation to compute the population for each polygon in the world_province_1deg_with_cellid.gpkg file obtained in Section 19.
- Rename the extracted population column to pop for clarity.
- Use cbind to combine the above pop column with the id, iso, cell_id, and geom columns from world_province_1deg_with_cellid.gpkg. Column fid_2 is not needed.
- Add a new column year with value as the integer form of the corresponding year from the tif file name.

Combine Dataframes: Use bind_rows to combine the resulting dataframes for all years into a single dataframe.

Adjust Values: Replace NA values in the pop column with 0.

Save: Save the resulting file as land_pop_extracted_region_level_1deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

2. **0.5-degree**

Extract Population: For each population file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 5 for concurrent processing):

- Read the population tif file using rast.
- Perform a spatial extraction using the exact_extract function from the exactextractr package with the sum operation to compute the population for each polygon in the world_province_0_5deg_with_cellid.gpkg file obtained in Section 19.
- Rename the extracted population column to pop for clarity.
- Use cbind to combine the above pop column with the id, iso, cell_id, subcell_id, and geom columns from world_province_0_5deg_with_cellid.gpkg. Column fid_2 is not needed.
- Add a new column year with value as the integer form of the corresponding year from the tif file name.

Combine Dataframes: Use bind_rows to combine the resulting dataframes for all years into a single dataframe.

Adjust Values: Replace NA values in the pop column with 0.

Save: Save the resulting file as land_pop_extracted_region_level_0_5deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

3. **0.25-degree**

Extract Population: For each population file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 5 for concurrent processing):

- Read the population tif file using rast.
- Perform a spatial extraction using the exact_extract function from the exactextractr package with the sum operation to compute the population for each polygon in the world_province_0_25deg_with_cellid.gpkg file obtained in Section 19.
- Rename the extracted population column to pop for clarity.
- Use cbind to combine the above pop column with the id, iso, cell_id, subcell_id, subcell_id_0_25, and geom columns from world_province_0_25deg_with_cellid.gpkg. Column fid_2 is not needed.
- Add a new column year with value as the integer form of the corresponding year from the tif file name.

Combine Dataframes: Use bind_rows to combine the resulting dataframes for all years into a single dataframe.

Adjust Values: Replace `NA` values in the `pop` column with `0`.

Save: Save the resulting file as `land_pop_extracted_region_level_0_25deg.RData`. Ensure that the dataframe name matches the name of the `RData` file when saving it.

21 Extract Land Use Area by Type for Each Cell

1. **Prepare Landcover Datasets**

   Download Landcover Data: Download the landcover data as described in Appendix Section 1.3. The files should be in `hdf` format and saved into corresponding year folders. Each year folder should consist of multiple files, each representing a portion of the Earth's surface. The goal is to combine these pieces into a single raster map for each year and reproject the data from the Sino coordinate reference system (CRS) to `EPSG:4326`.

   Process Landcover Data for Each Year: For each year folder, perform the following steps using `mclapply` with `mc.cores = 5` for parallel processing:

   - List all `hdf` files in the folder.
   - For each file in the folder:
     - Read the 10th layer of the file using the `rast` function.
     - Reproject the raster data from its original CRS to `EPSG:4326` using the `project("epsg:4326", method = "near")` function.
     - Save the resulting reprojected raster as a `.tif` file named `paste0("test", i, "_", year, ".tif")` using the `writeRaster` function.
   - Combine all the `.tif` files created for the year into a single virtual raster (`VRT`) file using the `vrt` function from the `terra` package. This combines the spatial extents of the `.tif` files into a single reference without physically merging them. Save the `VRT` file as `paste0("test", year, ".vrt")`.

2. **1-degree Cells**

   Extract Land Cover Areas: For each `paste0("test", year, ".vrt")` file from 2012 to 2021, perform the following steps (use `mclapply` with `mc.cores = 10` for parallel processing):

   - Read the `paste0("test", year, ".vrt")` file using `rast`.
   - Perform a spatial extraction with the `exact_extract` function to calculate the area (in square kilometers) of each land cover type within each polygon in the `world_province_1deg_with_cellid.gpkg` file. Use the following parameters:
     - `coverage_area = T`
     - `include_cols = c("id", "iso", "cell_id")`
     - `summarize_df = T`

- fun = function(df_in){
  out_df <- df_in %>%
  mutate(value = ifelse(is.na(value), 3, value)) %>%
  group_by(id, iso, cell_id, value) %>%
  summarize(lc_area = sum(coverage_area)/1e6, .groups = "drop")
  return(out_df)
  }
- The resulting file will include columns: `id`, `iso`, `cell_id`, `value`, and `lc_area`.

* Reshape the above dataframe into a wider format using `pivot_wider`, creating separate columns for each unique land cover type id (`value` column) with names prefixed by `class_`, and filling these columns with the corresponding `lc_area` values.
* Replace any `NA` values in the resulting land cover columns with `0`.
* Rename the land cover columns as follows:
  - `class_1` → `barren`
  - `class_2` → `snow_ice`
  - `class_3` → `water`
  - `class_9` → `urban`
  - `class_10` → `dense_forest`
  - `class_20` → `open_forest`
  - `class_25` → `forest_cropland`
  - `class_30` → `herbaceous`
  - `class_35` → `herbaceous_cropland`
  - `class_36` → `cropland`
  - `class_40` → `shrub`
* Add a new column `year` with value as the integer form of the corresponding year from the `vrt` file name.

Combine Dataframes: Use `bind_rows` to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as `lc_full_1deg.RData`. Ensure that the dataframe name matches the name of the `RData` file when saving it.

3. **0.5-degree Cells**

Extract Land Cover Areas: For each `paste0("test", year, ".vrt")` file from 2012 to 2021, perform the following steps (use `mclapply` with `mc.cores = 10` for parallel processing):

* Read the `paste0("test", year, ".vrt")` file using `rast`.
* Perform a spatial extraction with the `exact_extract` function to calculate the area (in square kilometers) of each land cover type within each polygon in the `world_province_0_5deg_with_cellid.gpkg` file. Use the following parameters:

- coverage_area = T
- include_cols = c("id", "iso", "cell_id", "subcell_id")
- summarize_df = T
- fun = function(df_in){
  out_df <- df_in %>%
  mutate(value = ifelse(is.na(value), 3, value)) %>%
  group_by(id, iso, cell_id, subcell_id, value) %>%
  summarize(lc_area = sum(coverage_area)/1e6, .groups = "drop")
  return(out_df)
  }
- The resulting file will include columns: id, iso, cell_id, subcell_id, value, and lc_area.

* Reshape the above dataframe into a wider format using pivot_wider, creating separate columns for each unique land cover type id (value column) with names prefixed by class_, and filling these columns with the corresponding lc_area values.
* Replace any NA values in the resulting land cover columns with 0.
* Rename the land cover columns as follows:
  - class_1 → barren
  - class_2 → snow_ice
  - class_3 → water
  - class_9 → urban
  - class_10 → dense_forest
  - class_20 → open_forest
  - class_25 → forest_cropland
  - class_30 → herbaceous
  - class_35 → herbaceous_cropland
  - class_36 → cropland
  - class_40 → shrub
* Add a new column year with value as the integer form of the corresponding year from the vrt file name.

Combine Dataframes: Use bind_rows to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as lc_full_0_5deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

4. **0.25-degree Cells**

Extract Land Cover Areas: For each paste0("test", year, ".vrt") file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 10 for parallel processing):

- Read the `paste0("test", year, ".vrt")` file using `rast`.
- Perform a spatial extraction with the `exact_extract` function to calculate the area (in square kilometers) of each land cover type within each polygon in the `world_province_0_25deg_with_cellid.gpkg` file. Use the following parameters:
  - `coverage_area = T`
  - `include_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25")`
  - `summarize_df = T`
  - fun = function(df_in){
    out_df <- df_in %>%
    mutate(value = ifelse(is.na(value), 3, value)) %>%
    group_by(id, iso, cell_id, subcell_id, subcell_id_0_25, value) %>%
    summarize(lc_area = sum(coverage_area)/1e6, .groups = "drop")
    return(out_df)
    }
  - The resulting file will include columns: `id`, `iso`, `cell_id`, `subcell_id`, `subcell_id_0_25`, `value`, and `lc_area`.
- Reshape the above dataframe into a wider format using `pivot_wider`, creating separate columns for each unique land cover type id (`value` column) with names prefixed by `class_`, and filling these columns with the corresponding `lc_area` values.
- Replace any `NA` values in the resulting land cover columns with `0`.
- Rename the land cover columns as follows:
  - `class_1` → `barren`
  - `class_2` → `snow_ice`
  - `class_3` → `water`
  - `class_9` → `urban`
  - `class_10` → `dense_forest`
  - `class_20` → `open_forest`
  - `class_25` → `forest_cropland`
  - `class_30` → `herbaceous`
  - `class_35` → `herbaceous_cropland`
  - `class_36` → `cropland`
  - `class_40` → `shrub`
- Add a new column `year` with value as the integer form of the corresponding year from the `vrt` file name.

Combine Dataframes: Use `bind_rows` to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as `lc_full_0_25deg.RData`. Ensure that the dataframe name matches the name of the `RData` file when saving it.

22 Extract Net Primary Productivity Values for Each Cell

1. **Prepare NPP Datasets**

   Download NPP Data: Download the NPP data as described in Appendix Section 1.3. The files should be in `hdf` format and saved into corresponding year folders. Each year folder should consist of multiple files, each representing a portion of the Earth's surface. The goal is to combine these pieces into a single raster map for each year and reproject the data from the Sino coordinate reference system (CRS) to `EPSG:4326`.

   Process NPP Data for Each Year: For each year folder, perform the following steps using `mclapply` with `mc.cores = 2` for parallel processing:

   - List all `hdf` files in the folder.
   - For each file in the folder:
     - Read the 2nd layer of the file using the `rast` function.
     - Reproject the raster data from its original CRS to `EPSG:4326` using the `project("epsg:4326")` function.
     - Save the resulting reprojected raster as a `.tif` file named `paste0("test", i, "_", year, ".tif")` using the `writeRaster` function.
   - Combine all the `.tif` files created for the year into a single virtual raster (`VRT`) file using the `vrt` function from the `terra` package. This combines the spatial extents of the `.tif` files into a single reference without physically merging them. Save the `VRT` file as `paste0("test", year, ".vrt")`.

2. **1-degree Cells**

   Extract NPP Values: For each `paste0("test", year, ".vrt")` file from 2012 to 2021, perform the following steps (use `mclapply` with `mc.cores = 11` for parallel processing):

   - Read the `paste0("test", year, ".vrt")` file using `rast`.
   - Perform a spatial extraction with the `exact_extract` function to calculate the NPP values of each polygon in the `world_province_1deg_with_cellid.gpkg` file. Use the following parameters:
     - `coverage_area = T`
     - `include_cols = c("id", "iso", "cell_id")`
     - `summarize_df = T`
     - fun = function(df_in){
       out_df <- df_in %>%
       mutate(value = ifelse(value >= 3.2760 | value < -3 | is.na(value), 0, value)) %>%
       group_by(cell_id, iso, id) %>%
       summarize(NPP = sum(value * coverage_area), .groups = "drop")

return(out_df)
}
  - The resulting file will include columns: id, iso, cell_id, and NPP.
- Add a new column year with value as the integer form of the corresponding year from the vrt file name.

Combine Dataframes: Use bind_rows to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as NPP_full_1deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

3. **0.5-degree Cells**

Extract NPP Values: For each paste0("test", year, ".vrt") file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 11 for parallel processing):

- Read the paste0("test", year, ".vrt") file using rast.
- Perform a spatial extraction with the exact_extract function to calculate the NPP values of each polygon in the world_province_0_5deg_with_cellid.gpkg file. Use the following parameters:
  - coverage_area = T
  - include_cols = c("id", "iso", "cell_id", "subcell_id")
  - summarize_df = T
  - fun = function(df_in){
    out_df <- df_in %>%
    mutate(value = ifelse(value >= 3.2760 | value < -3 | is.na(value), 0, value)) %>%
    group_by(subcell_id, cell_id, iso, id) %>%
    summarize(NPP = sum(value * coverage_area), .groups = "drop")
    return(out_df)
    }
  - The resulting file will include columns: id, iso, cell_id, subcell_id, and NPP.
- Add a new column year with value as the integer form of the corresponding year from the vrt file name.

Combine Dataframes: Use bind_rows to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as NPP_full_0_5deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

4. **0.25-degree Cells**

Extract NPP Values: For each paste0("test", year, ".vrt") file from 2012 to 2021, perform the following steps (use mclapply with mc.cores = 11 for parallel processing):

- Read the paste0("test", year, ".vrt") file using rast.
- Perform a spatial extraction with the exact_extract function to calculate the NPP values of each polygon in the world_province_0_25deg_with_cellid.gpkg file. Use the following parameters:
  - coverage_area = T
  - include_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25")
  - summarize_df = T
  - fun = function(df_in){
    out_df <- df_in %>%
    mutate(value = ifelse(value >= 3.2760 | value < -3 | is.na(value), 0, value)) %>%
    group_by(subcell_id_0_25, subcell_id, cell_id, iso, id) %>%
    summarize(NPP = sum(value * coverage_area), .groups = "drop")
    return(out_df)
    }
  - The resulting file will include columns: id, iso, cell_id, subcell_id, subcell_id_0_25, and NPP.
- Add a new column year with value as the integer form of the corresponding year from the vrt file name.

Combine Dataframes: Use bind_rows to combine the processed dataframes for all years into a single dataframe.

Save the Dataset: Save the combined dataframe as NPP_full_0_25deg.RData. Ensure that the dataframe name matches the name of the RData file when saving it.

23 Gas Flare Spots

In this section, we create a 0.2-degree square grid centered around each gas flare spot. In later sections, all nighttime lights within these squares will be omitted.

1. **Download Data:** Download the gas flare data as described in Appendix Section 1.3 and save it as GGFR-Flaring-Dashboard-Data-March292023.xlsx.

2. **Read Data:** Use read_excel to read the excel file.

3. **Filter Data:** Select rows where the Flaring Vol (million m3) column values are not equal to 0.

4. **Convert to Spatial Object:** Use st_as_sf(coords = c("Longitude", "Latitude"), crs = 4326) to convert the filtered dataframe into an sf spatial object, using the Longitude and Latitude columns as coordinates.

5. **Refer to the Spatial Data:** Save the resulting spatial object as spot_sf.

6. **Create Square Grids:** Use the qgis_run_algorithm function from the qgisprocess package with the algorithm native:rectanglesovalsdiamonds to create squares, setting the parameters WIDTH = 0.2 and HEIGHT = 0.2. The input file is spot_sf, and the output is saved as gas_flare_spot_sf_square_0_2deg.gpkg.

24 Extract Cell Nighttime Light Emissions

1. **Prepare NTL Datasets**

   Download Data: Download the NTL data as described in Appendix Section 1.3. The files should be in h5 format and saved into folders corresponding to each year. Each year folder should contain multiple files, with each file representing a specific portion of the Earth's surface. The objective is to combine these files into a single raster map for each year. Since the original data is in a linear lat-lon grid format, reprojection to EPSG:4326 is not required.

   Process NTL Data for Each Year: For each year folder, follow the steps below using mclapply with mc.cores = 2 for parallel processing:

   - List all h5 files in the folder.
   - Process each file in the folder by performing the following:
     - Extract longitude use lon <- h5read(h5_file, "/HDFEOS/GRIDS/VIIRS_Grid_DNB_2d/Data Fields/lon")
     - Extract latitude use lat <- h5read(h5_file, "/HDFEOS/GRIDS/VIIRS_Grid_DNB_2d/Data Fields/lat")
     - Read the raster data from the file using the rast function.
     - Set the spatial extent of the raster to c(min(lon), max(lon), min(lat), max(lat)).
     - Set the CRS (Coordinate Reference System) of the raster to "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0".
     - Save the raster as a .tif file named paste0("test", i, "_", year, ".tif") using the writeRaster function.
   - Combine all .tif files generated for the year into a single virtual raster (VRT) file using the vrt function from the terra package. This VRT file references the spatial extents of the .tif files without merging them physically. Save the VRT file as paste0("test", year, ".vrt").

2. **1-degree Cells**

For each `paste0("test", year, ".vrt")` file from 2012 to 2021, perform the following steps:

- Read the `paste0("test", year, ".vrt")` file into a raster using the `rast` function.
- Rename the raster layers using the `setNames` function in the following order:
  - "AllAngle_Composite_Snow_Covered",
    "AllAngle_Composite_Snow_Covered_Num",
    "AllAngle_Composite_Snow_Covered_Quality",
    "AllAngle_Composite_Snow_Covered_Std",
    "AllAngle_Composite_Snow_Free",
    "AllAngle_Composite_Snow_Free_Num",
    "AllAngle_Composite_Snow_Free_Quality",
    "AllAngle_Composite_Snow_Free_Std",
    "DNB_Platform",
    "Land_Water_Mask",
    "NearNadir_Composite_Snow_Covered",
    "NearNadir_Composite_Snow_Covered_Num",
    "NearNadir_Composite_Snow_Covered_Quality",
    "NearNadir_Composite_Snow_Covered_Std",
    "NearNadir_Composite_Snow_Free",
    "NearNadir_Composite_Snow_Free_Num",
    "NearNadir_Composite_Snow_Free_Quality",
    "NearNadir_Composite_Snow_Free_Std",
    "OffNadir_Composite_Snow_Covered",
    "OffNadir_Composite_Snow_Covered_Num",
    "OffNadir_Composite_Snow_Covered_Quality",
    "OffNadir_Composite_Snow_Covered_Std",
    "OffNadir_Composite_Snow_Free",
    "OffNadir_Composite_Snow_Free_Num",
    "OffNadir_Composite_Snow_Free_Quality",
    "OffNadir_Composite_Snow_Free_Std"
- Select only two layers: `"AllAngle_Composite_Snow_Covered"` and `"AllAngle_Composite_Snow_Free"`, and refer to the resulting raster as `vv`.
- Read the gas flare spots from the file `gas_flare_spot_sf_square_0_2deg.gpkg` using `st_read`. Select rows corresponding to the same year as the `vv` raster file.
- Apply the `exact_extract` function from the `exactextractr` package to identify pixels in `vv` that overlap with gas flare spots. Use the following parameters:
  - `coverage_area` = FALSE.
  - `include_cell` = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:
  - `AllAngle_Composite_Snow_Covered`: NTL value for the pixel.

- AllAngle_Composite_Snow_Free: Snow-free NTL value for the pixel.
- cell: Pixel ID as defined in the vv file.
- coverage_fraction: Fraction of the pixel overlapping with the polygon.

- Extract the unique pixel IDs from the cell column in the list and refer to them as unique_indices.
- For each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
  - Perform a spatial extraction using the exact_extract function to calculate NTL values for each polygon in world_province_1deg_with_cellid.gpkg. Use the following parameters:
    * SpatRaster file: vv[[i]], where i is the layer index.
    * Polygon file: world_province_1deg_with_cellid.gpkg, excluding the column fid_2.
    * coverage_area = TRUE.
    * include_cols = c("id", "iso", "cell_id").
    * summarize_df = TRUE.
    * include_cell = TRUE.
    * fun = function(df_in){
      out_df <- df_in %>%
      mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
      group_by(id, iso, cell_id) %>%
      summarize(NTL = sum(coverage_area * value), .groups = "drop")
      return(out_df)}.
  - The resulting file will include columns: id, iso, cell_id, and NTL.
  - Save the resulting file for the current year and layer as paste0("NTL_extracted_1deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

3. **0.5-degree Cells**

For each paste0("test", year, ".vrt") file from 2012 to 2021, perform the following steps:

- Read the paste0("test", year, ".vrt") file into a raster using the rast function.
- Rename the raster layers using the setNames function in the following order:
  - `"AllAngle_Composite_Snow_Covered",`
    `"AllAngle_Composite_Snow_Covered_Num",`
    `"AllAngle_Composite_Snow_Covered_Quality",`
    `"AllAngle_Composite_Snow_Covered_Std",`
    `"AllAngle_Composite_Snow_Free",`
    `"AllAngle_Composite_Snow_Free_Num",`
    `"AllAngle_Composite_Snow_Free_Quality",`

```
"AllAngle_Composite_Snow_Free_Std",
"DNB_Platform",
"Land_Water_Mask",
"NearNadir_Composite_Snow_Covered",
"NearNadir_Composite_Snow_Covered_Num",
"NearNadir_Composite_Snow_Covered_Quality",
"NearNadir_Composite_Snow_Covered_Std",
"NearNadir_Composite_Snow_Free",
"NearNadir_Composite_Snow_Free_Num",
"NearNadir_Composite_Snow_Free_Quality",
"NearNadir_Composite_Snow_Free_Std",
"OffNadir_Composite_Snow_Covered",
"OffNadir_Composite_Snow_Covered_Num",
"OffNadir_Composite_Snow_Covered_Quality",
"OffNadir_Composite_Snow_Covered_Std",
"OffNadir_Composite_Snow_Free",
"OffNadir_Composite_Snow_Free_Num",
"OffNadir_Composite_Snow_Free_Quality",
"OffNadir_Composite_Snow_Free_Std"
```

- Select only two layers: `"AllAngle_Composite_Snow_Covered"` and `"AllAngle_Composite_Snow_Free"`, and refer to the resulting raster as `vv`.
- Read the gas flare spots from the file `gas_flare_spot_sf_square_0_2deg.gpkg` using `st_read`. Select rows corresponding to the same year as the `vv` raster file.
- Apply the `exact_extract` function from the `exactextractr` package to identify pixels in `vv` that overlap with gas flare spots. Use the following parameters:
  - `coverage_area` = FALSE.
  - `include_cell` = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:
  - `AllAngle_Composite_Snow_Covered`: NTL value for the pixel.
  - `AllAngle_Composite_Snow_Free`: Snow-free NTL value for the pixel.
  - `cell`: Pixel ID as defined in the `vv` file.
  - `coverage_fraction`: Fraction of the pixel overlapping with the polygon.
- Extract the unique pixel IDs from the `cell` column in the list and refer to them as `unique_indices`.
- For each of the two layers in `vv`, perform the following steps using `mclapply` with `mc.cores = 2` for parallel processing:
  - Perform a spatial extraction using the `exact_extract` function to calculate NTL values for each polygon in `world_province_0_5deg_with_cellid.gpkg`. Use the following parameters:
    * SpatRaster file: `vv[[i]]`, where `i` is the layer index.

      * Polygon file: world_province_0_5deg_with_cellid.gpkg, excluding the column fid_2.
      * coverage_area = TRUE.
      * include_cols = c("id", "iso", "cell_id", "subcell_id").
      * summarize_df = TRUE.
      * include_cell = TRUE.
      * fun = function(df_in){
        out_df <- df_in %>%
        mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
        group_by(id, iso, cell_id, subcell_id) %>%
        summarize(NTL = sum(coverage_area * value), .groups = "drop")
        return(out_df)}.
    - The resulting file will include columns: id, iso, cell_id, subcell_id, and NTL.
    - Save the resulting file for the current year and layer as paste0("NTL_extracted_0_5deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

4. **0.25-degree Cells**

   For each paste0("test", year, ".vrt") file from 2012 to 2021, perform the following steps:

   - Read the paste0("test", year, ".vrt") file into a raster using the rast function.
   - Rename the raster layers using the setNames function in the following order:
     - `"AllAngle_Composite_Snow_Covered",`
       `"AllAngle_Composite_Snow_Covered_Num",`
       `"AllAngle_Composite_Snow_Covered_Quality",`
       `"AllAngle_Composite_Snow_Covered_Std",`
       `"AllAngle_Composite_Snow_Free",`
       `"AllAngle_Composite_Snow_Free_Num",`
       `"AllAngle_Composite_Snow_Free_Quality",`
       `"AllAngle_Composite_Snow_Free_Std",`
       `"DNB_Platform",`
       `"Land_Water_Mask",`
       `"NearNadir_Composite_Snow_Covered",`
       `"NearNadir_Composite_Snow_Covered_Num",`
       `"NearNadir_Composite_Snow_Covered_Quality",`
       `"NearNadir_Composite_Snow_Covered_Std",`
       `"NearNadir_Composite_Snow_Free",`
       `"NearNadir_Composite_Snow_Free_Num",`
       `"NearNadir_Composite_Snow_Free_Quality",`
       `"NearNadir_Composite_Snow_Free_Std",`

```
"OffNadir_Composite_Snow_Covered",
"OffNadir_Composite_Snow_Covered_Num",
"OffNadir_Composite_Snow_Covered_Quality",
"OffNadir_Composite_Snow_Covered_Std",
"OffNadir_Composite_Snow_Free",
"OffNadir_Composite_Snow_Free_Num",
"OffNadir_Composite_Snow_Free_Quality",
"OffNadir_Composite_Snow_Free_Std"
```

- Select only two layers: `"AllAngle_Composite_Snow_Covered"` and `"AllAngle_Composite_Snow_Free"`, and refer to the resulting raster as `vv`.
- Read the gas flare spots from the file `gas_flare_spot_sf_square_0_2deg.gpkg` using `st_read`. Select rows corresponding to the same year as the `vv` raster file.
- Apply the `exact_extract` function from the `exactextractr` package to identify pixels in `vv` that overlap with gas flare spots. Use the following parameters:
  - `coverage_area` = FALSE.
  - `include_cell` = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:
  - `AllAngle_Composite_Snow_Covered`: NTL value for the pixel.
  - `AllAngle_Composite_Snow_Free`: Snow-free NTL value for the pixel.
  - `cell`: Pixel ID as defined in the `vv` file.
  - `coverage_fraction`: Fraction of the pixel overlapping with the polygon.
- Extract the unique pixel IDs from the `cell` column in the list and refer to them as `unique_indices`.
- For each of the two layers in `vv`, perform the following steps using `mclapply` with `mc.cores = 2` for parallel processing:
  - Perform a spatial extraction using the `exact_extract` function to calculate NTL values for each polygon in `world_province_0_25deg_with_cellid.gpkg`. Use the following parameters:
    * SpatRaster file: `vv[[i]]`, where `i` is the layer index.
    * Polygon file: `world_province_0_25deg_with_cellid.gpkg`, excluding the column `fid_2`.
    * `coverage_area` = TRUE.
    * `include_cols` = `c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25")`.
    * `summarize_df` = TRUE.
    * `include_cell` = TRUE.
    * fun = function(df_in){
      out_df <- df_in %>%

mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
group_by(id, iso, cell_id, subcell_id, subcell_id_0_25) %>%
summarize(NTL = sum(coverage_area * value), .groups = "drop")
return(out_df)}.

  - The resulting file will include columns: `id`, `iso`, `cell_id`, `subcell_id`, `subcell_id_0_25`, and `NTL`.
  - Save the resulting file for the current year and layer as `paste0("NTL_extracted_0_25deg_", i, year, ".RData")`. Ensure that the dataframe name matches the name of the `RData` file when saving it. Data across years and layers will be combined later.

5. **Combine Data Across Years and Layers**

1-degree:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file `paste0("NTL_extracted_1deg_1", year, ".RData")` using the `load` function.
  - Add a new column `year` with values corresponding to the respective year.
  - Rename the column `NTL` to `NTL_snow_covered_period`.
  - Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_extracted_1deg_2", year, ".RData")` file.
- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_full_1deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

0.5-degree:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file `paste0("NTL_extracted_0_5deg_1", year, ".RData")` using the `load` function.
  - Add a new column `year` with values corresponding to the respective year.
  - Rename the column `NTL` to `NTL_snow_covered_period`.
  - Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_extracted_0_5deg_2", year, ".RData")` file.
- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_full_0_5deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

0.25-degree:

- For each year from 2012 to 2021, perform the following steps:

  - Load the file `paste0("NTL_extracted_0_25deg_1", year, ".RData")` using the `load` function.
  - Add a new column `year` with values corresponding to the respective year.
  - Rename the column `NTL` to `NTL_snow_covered_period`.
  - Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_extracted_0_25deg_2", year, ".RData")` file.
- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_full_0_25deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

25 Extract Cell Ruggedness

1. **Download Data:** Download the terrain ruggedness index data as described in Appendix Section 1.3 and save it as the file `tri.txt`.

2. **1-degree**

   Extract Mean Ruggedness: Use the `exact_extract` function to calculate the mean ruggedness for each polygon in the `world_province_1deg_with_cellid.gpkg` file with the following parameters:

   - **SpatRaster file**: Read the `tri.txt` file using the `rast` function and assign the CRS using `crs("epsg:4326")`.
   - **Polygon file**: Read the `world_province_1deg_with_cellid.gpkg` file and exclude the column `fid_2`.
   - **Function**: `fun = "mean"`
   - **Other parameters**:
     - `coverage_area = TRUE`
     - `progress = TRUE`
     - `append_cols = c("cell_id", "id", "iso")`
   - The resulting dataframe will include the columns: `cell_id`, `id`, `iso`, and `mean`.

   Adjust Mean Ruggedness: Create a new column `mean_rug` by dividing the values in the `mean` column by 1000.

   Finalize Dataframe: Exclude the `mean` column.

   Save File: Save the resulting dataframe as `mean_ruggedness_1deg.csv`, ensuring that row names are excluded by setting the parameter `row.names = FALSE`.

3. **0.5-degree**

   Extract Mean Ruggedness: Use the `exact_extract` function to calculate the mean ruggedness for each polygon in the `world_province_0_5deg_with_cellid.gpkg` file with the following parameters:

- **SpatRaster file**: Read the `tri.txt` file using the `rast` function and assign the CRS using `crs("epsg:4326")`.
- **Polygon file**: Read the `world_province_0_5deg_with_cellid.gpkg` file and exclude the column `fid_2`.
- **Function**: `fun = "mean"`
- **Other parameters**:
  - `coverage_area = TRUE`
  - `progress = TRUE`
  - `append_cols = c("cell_id", "subcell_id", "id", "iso")`
- The resulting dataframe will include the columns: `cell_id`, `subcell_id`, `id`, `iso`, and `mean`.

Adjust Mean Ruggedness: Create a new column `mean_rug` by dividing the values in the `mean` column by 1000.

Finalize Dataframe: Exclude the `mean` column.

Save File: Save the resulting dataframe as `mean_ruggedness_0_5deg.csv`, ensuring that row names are excluded by setting the parameter `row.names = FALSE`.

4. **0.25-degree**

Extract Mean Ruggedness: Use the `exact_extract` function to calculate the mean ruggedness for each polygon in the `world_province_0_25deg_with_cellid.gpkg` file with the following parameters:

- **SpatRaster file**: Read the `tri.txt` file using the `rast` function and assign the CRS using `crs("epsg:4326")`.
- **Polygon file**: Read the `world_province_0_25deg_with_cellid.gpkg` file and exclude the column `fid_2`.
- **Function**: `fun = "mean"`
- **Other parameters**:
  - `coverage_area = TRUE`
  - `progress = TRUE`
  - `append_cols = c("cell_id", "subcell_id", "subcell_id_0_25", "id", "iso")`
- The resulting dataframe will include the columns: `cell_id`, `subcell_id`, `subcell_id_0_25`, `id`, `iso`, and `mean`.

Adjust Mean Ruggedness: Create a new column `mean_rug` by dividing the values in the `mean` column by 1000.

Finalize Dataframe: Exclude the `mean` column.

Save File: Save the resulting dataframe as `mean_ruggedness_0_25deg.csv`, ensuring that row names are excluded by setting the parameter `row.names = FALSE`.

26 Extract Cell CO2 Emissions from Biofuels

1. **Download Data:** Download the CO2 emissions from biofuels data as described in Appendix Section 1.3. Save the data files in their respective sector folders, ensuring the files are in nc format.

2. **1-degree**

   Process Each Sector Folder: For each sector folder, perform the following steps:

   - List all nc files in the sector folder. Each nc file corresponds to a specific year. For each nc file, do the following:
     - Use the exact_extract function to calculate the CO2 emissions for each polygon in the world_province_1deg_with_cellid.gpkg file, with the following parameters:
       * **SpatRaster file**: Read the nc file using the rast function.
       * **Polygon file**: Read the world_province_1deg_with_cellid.gpkg file and exclude the column fid_2.
       * **Function**: fun = "sum"
       * **Additional Parameters**: append_cols = c("id", "iso", "cell_id")
       * The resulting dataframe will include the columns: cell_id, id, iso, and sum.
     - Rename the sum column to CO2_bio.
     - Create a new column year with values extracted from the year in the nc file name.
   - Combine the dataframes from all nc files within the sector using rbind.
   - Rename the CO2_bio column to paste0("CO2_bio_", sector).

   Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the full_join function with the parameter by = c("id", "iso", "cell_id", "year").

   Resulting Dataframe: The final dataframe will include the columns: id, iso, cell_id, year, CO2_bio_combustion_for_manufacturing, CO2_bio_fuel_exploitation, CO2_bio_oil_refine_transf, CO2_bio_power_industry, CO2_bio_road_transp, CO2_bio_shipping.

   Save File: Save the final dataframe as CO2_bio_full_1deg.RData. Ensure the dataframe name matches the file name when saving it.

3. **0.5-degree**

   Process Each Sector Folder: For each sector folder, perform the following steps:

   - List all nc files in the sector folder. Each nc file corresponds to a specific year. For each nc file, do the following:
     - Use the exact_extract function to calculate the CO2 emissions for each polygon in the world_province_0_5deg_with_cellid.gpkg file, with the following parameters:

    * **SpatRaster file**: Read the nc file using the rast function.
    * **Polygon file**: Read the world_province_0_5deg_with_cellid.gpkg file and exclude the column fid_2.
    * **Function**: fun = "sum"
    * **Additional Parameters**: append_cols = c("id", "iso", "cell_id", "subcell_id")
    * The resulting dataframe will include the columns: subcell_id, cell_id, id, iso, and sum.
  - Rename the sum column to CO2_bio.
  - Create a new column year with values extracted from the year in the nc file name.
- Combine the dataframes from all nc files within the sector using rbind.
- Rename the CO2_bio column to paste0("CO2_bio_", sector).

Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the full_join function with the parameter by = c("id", "iso", "cell_id", "subcell_id", "year").

Resulting Dataframe: The final dataframe will include the columns: id, iso, cell_id, subcell_id, year, CO2_bio_combustion_for_manufacturing, CO2_bio_fuel_exploitation, CO2_bio_oil_refine_transf, CO2_bio_power_industry, CO2_bio_road_transp, CO2_bio_shipping.

Save File: Save the final dataframe as CO2_bio_full_0_5deg.RData. Ensure the dataframe name matches the file name when saving it.

4. **0.25-degree**

Process Each Sector Folder: For each sector folder, perform the following steps:

- List all nc files in the sector folder. Each nc file corresponds to a specific year. For each nc file, do the following:
  - Use the exact_extract function to calculate the CO2 emissions for each polygon in the world_province_0_25deg_with_cellid.gpkg file, with the following parameters:
    * **SpatRaster file**: Read the nc file using the rast function.
    * **Polygon file**: Read the world_province_0_25deg_with_cellid.gpkg file and exclude the column fid_2.
    * **Function**: fun = "sum"
    * **Additional Parameters**: append_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25")
    * The resulting dataframe will include the columns: subcell_id_0_25, subcell_id, cell_id, id, iso, and sum.
  - Rename the sum column to CO2_bio.
  - Create a new column year with values extracted from the year in the nc file name.

- Combine the dataframes from all nc files within the sector using rbind.
- Rename the CO2_bio column to paste0("CO2_bio_", sector).

Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the full_join function with the parameter by = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25", "year").

Resulting Dataframe: The final dataframe will include the columns: id, iso, cell_id, subcell_id, subcell_id_0_25, year, CO2_bio_combustion_for_manufacturing, CO2_bio_fuel_exploitation, CO2_bio_oil_refine_transf, CO2_bio_power_industry, CO2_bio_road_transp, CO2_bio_shipping.

Save File: Save the final dataframe as CO2_bio_full_0_25deg.RData. Ensure the dataframe name matches the file name when saving it.

27 Extract Cell CO2 Emissions from Non-organic Fuels

1. **Download Data:** Download the CO2 emissions from non-organic fuels data as described in Appendix Section 1.3. Save the data files in their respective sector folders, ensuring the files are in nc format.

2. **1-degree**

Process Each Sector Folder: For each sector folder, perform the following steps:

- List all nc files in the sector folder. Each nc file corresponds to a specific year. For each nc file, do the following:
  - Use the exact_extract function to calculate the CO2 emissions for each polygon in the world_province_1deg_with_cellid.gpkg file, with the following parameters:
    - **SpatRaster file**: Read the nc file using the rast function.
    - **Polygon file**: Read the world_province_1deg_with_cellid.gpkg file and exclude the column fid_2.
    - **Function**: fun = "sum"
    - **Additional Parameters**: append_cols = c("id", "iso", "cell_id")
    - The resulting dataframe will include the columns: cell_id, id, iso, and sum.
  - Rename the sum column to CO2_non_org.
  - Create a new column year with values extracted from the year in the nc file name.
- Combine the dataframes from all nc files within the sector using rbind.
- Rename the CO2_non_org column to paste0("CO2_non_org_", sector).

Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the full_join function with the parameter by = c("id", "iso", "cell_id", "year").

Resulting Dataframe: The final dataframe will include the columns: id, iso, cell_id, year, CO2_non_org_combustion_for_manufacturing, CO2_non_org_fuel_exploitation, CO2_non_org_iron_steel, CO2_non_org_non_ferrous_metal, CO2_non_org_non_metallic_mineral, CO2_non_org_oil_refine_transf, CO2_non_org_power_industry, CO2_non_org_road_transp, CO2_non_org_shipping.

Save File: Save the final dataframe as CO2_non_org_full_1deg.RData. Ensure the dataframe name matches the file name when saving it.

3. **0.5-degree**

Process Each Sector Folder: For each sector folder, perform the following steps:

- List all nc files in the sector folder. Each nc file corresponds to a specific year. For each nc file, do the following:
  - Use the exact_extract function to calculate the CO2 emissions for each polygon in the world_province_0_5deg_with_cellid.gpkg file, with the following parameters:
    - **SpatRaster file**: Read the nc file using the rast function.
    - **Polygon file**: Read the world_province_0_5deg_with_cellid.gpkg file and exclude the column fid_2.
    - **Function**: fun = "sum"
    - **Additional Parameters**: append_cols = c("id", "iso", "cell_id", "subcell_id")
    - The resulting dataframe will include the columns: subcell_id, cell_id, id, iso, and sum.
  - Rename the sum column to CO2_non_org.
  - Create a new column year with values extracted from the year in the nc file name.
- Combine the dataframes from all nc files within the sector using rbind.
- Rename the CO2_non_org column to paste0("CO2_non_org_", sector).

Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the full_join function with the parameter by = c("id", "iso", "cell_id", "subcell_id", "year").

Resulting Dataframe: The final dataframe will include the columns: id, iso, cell_id, subcell_id, year, CO2_non_org_combustion_for_manufacturing, CO2_non_org_fuel_exploitation, CO2_non_org_iron_steel, CO2_non_org_non_ferrous_metal, CO2_non_org_non_metallic_mineral, CO2_non_org_oil_refine_transf, CO2_non_org_power_industry, CO2_non_org_road_transp, CO2_non_org_shipping..

Save File: Save the final dataframe as CO2_non_org_full_0_5deg.RData. Ensure the dataframe name matches the file name when saving it.

4. **0.25-degree**

Process Each Sector Folder: For each sector folder, perform the following steps:

- List all `nc` files in the sector folder. Each `nc` file corresponds to a specific year. For each `nc` file, do the following:
  - Use the `exact_extract` function to calculate the CO2 emissions for each polygon in the `world_province_0_25deg_with_cellid.gpkg` file, with the following parameters:
    * **SpatRaster file**: Read the `nc` file using the `rast` function.
    * **Polygon file**: Read the `world_province_0_25deg_with_cellid.gpkg` file and exclude the column `fid_2`.
    * **Function**: `fun = "sum"`
    * **Additional Parameters**: `append_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25")`
    * The resulting dataframe will include the columns: `subcell_id_0_25`, `subcell_id`, `cell_id`, `id`, `iso`, and `sum`.
  - Rename the `sum` column to `CO2_non_org`.
  - Create a new column `year` with values extracted from the year in the `nc` file name.
- Combine the dataframes from all `nc` files within the sector using `rbind`.
- Rename the `CO2_non_org` column to `paste0("CO2_non_org_", sector)`.

Combine Sector Dataframes: Merge all sector dataframes into a single dataframe using the `full_join()` function with the parameter `by = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25", "year")`.

Resulting Dataframe: The final dataframe will include the columns: `id`, `iso`, `cell_id`, `subcell_id`, `subcell_id_0_25`, `year`, `CO2_non_org_combustion_for_manufacturing`, `CO2_non_org_fuel_exploitation`, `CO2_non_org_iron_steel`, `CO2_non_org_non_ferrous_metal`, `CO2_non_org_non_metallic_mineral`, `CO2_non_org_oil_refine_transf`, `CO2_non_org_power_industry`, `CO2_non_org_road_transp`, `CO2_non_org_shipping`.

Save File: Save the final dataframe as `CO2_non_org_full_0_25deg.RData`. Ensure the dataframe name matches the file name when saving it.

28 Extract Cell Nighttime Light Emissions from Urban and Cropland Areas

In this section, we isolate the urban and cropland geometries within each cell. Then we extract nighttime light (NTL) emissions exclusively from the urban areas or from the cropland areas.

1. **Prepare Land Cover Polygons:**

   Recall that the `paste0("test", year, ".vrt")` files for each year (2012–2021) were obtained from Section 24 Step 1.

For each `paste0("test", year, ".vrt")` file:

- Read the file using `rast`.
- Convert the raster into a polygon geometry file using the `as.polygons()` function from the `terra` package (`values = TRUE`, `na.rm = TRUE`).
- Apply `st_as_sf` to ensure the polygons are in `sf` format.
- Save the resulting file as `paste0("lc_polygons_", year, ".gpkg")`.

2. **Fix Land Cover Geometries:** Now fix the geometries of the above `paste0("lc_polygons_", year, ".gpkg")` polygons because some of those polygons are not regular enough to be processed for intersection. Apply `mclapply()` function with `mc.cores = 5` for parallel processing:

For each `paste0("lc_polygons_", year, ".gpkg")` file:

- Use `qgis_run_algorithm` from the `qgisprocess` package with the algorithm `native:fixgeometries`.
  - **Input Layer:** `paste0("lc_polygons_", year, ".gpkg")` file and select only rows where the column named `paste0("test", year)` equals `9` (representing urban geometries).
  - **Output:** Save as `paste0("temp_urban_", year, ".gpkg")`.
- Similarly, use `qgis_run_algorithm` from the `qgisprocess` package with the algorithm `native:fixgeometries`.
  - **Input Layer:** `paste0("lc_polygons_", year, ".gpkg")` file and select only rows where the column named `paste0("test", year)` contains values in `(25, 35, 36)` (representing forest cropland, herbaceous cropland, and cropland, respectively).
  - **Output:** Save as `paste0("temp_cropland_", year, ".gpkg")`.

3. **Intersect Land Cover Polygons with Grids:** For each year from 2012 to 2021, process through the following steps (apply `mclapply()` function with `mc.cores = 5` for parallel processing):

Use `qgis_run_algorithm` from the `qgisprocess` package with the algorithm `native:intersection`.

- **Input Layer:** `paste0("temp_urban_", year, ".gpkg")`.
- **Overlay Layer:** `world_province_1deg_with_cellid.gpkg`.
- **Output:** Save as `paste0("lc_urban_inters_id_1deg_", year, ".gpkg")`.

Use `qgis_run_algorithm` from the `qgisprocess` package with the algorithm `native:intersection`.

- **Input Layer:** `paste0("temp_cropland_", year, ".gpkg")`.
- **Overlay Layer:** `world_province_1deg_with_cellid.gpkg`.
- **Output:** Save as `paste0("lc_cropland_inters_id_1deg_", year, ".gpkg")`.

Use `qgis_run_algorithm` from the `qgisprocess` package with the algorithm `native:intersection`.

- **Input Layer:** paste0("temp_urban_", year, ".gpkg").
- **Overlay Layer:** world_province_0_5deg_with_cellid.gpkg.
- **Output:** Save as paste0("lc_urban_inters_id_0_5deg_", year, ".gpkg").

Use qgis_run_algorithm from the qgisprocess package with the algorithm native:intersection.

- **Input Layer:** paste0("temp_cropland_", year, ".gpkg").
- **Overlay Layer:** world_province_0_5deg_with_cellid.gpkg.
- **Output:** Save as paste0("lc_cropland_inters_id_0_5deg_", year, ".gpkg").

Use qgis_run_algorithm from the qgisprocess package with the algorithm native:intersection.

- **Input Layer:** paste0("temp_urban_", year, ".gpkg").
- **Overlay Layer:** world_province_0_25deg_with_cellid.gpkg.
- **Output:** Save as paste0("lc_urban_inters_id_0_25deg_", year, ".gpkg").

Use qgis_run_algorithm from the qgisprocess package with the algorithm native:intersection.

- **Input Layer:** paste0("temp_cropland_", year, ".gpkg").
- **Overlay Layer:** world_province_0_25deg_with_cellid.gpkg.
- **Output:** Save as paste0("lc_cropland_inters_id_0_25deg_", year, ".gpkg").

4. **1-degree Cells**

For each paste0("test", year, ".vrt") file from 2012 to 2021 obtained from Section 24 Step 1, perform the following steps:

- Read the paste0("test", year, ".vrt") file into a raster using the rast() function.
- Rename the raster layers using the setNames() function in the following order:
  - "AllAngle_Composite_Snow_Covered", "AllAngle_Composite_Snow_Covered_Num", "AllAngle_Composite_Snow_Covered_Quality", "AllAngle_Composite_Snow_Covered_Std", "AllAngle_Composite_Snow_Free", "AllAngle_Composite_Snow_Free_Num", "AllAngle_Composite_Snow_Free_Quality", "AllAngle_Composite_Snow_Free_Std", "DNB_Platform", "Land_Water_Mask", "NearNadir_Composite_Snow_Covered", "NearNadir_Composite_Snow_Covered_Num", "NearNadir_Composite_Snow_Covered_Quality",

```
"NearNadir_Composite_Snow_Covered_Std",
"NearNadir_Composite_Snow_Free",
"NearNadir_Composite_Snow_Free_Num",
"NearNadir_Composite_Snow_Free_Quality",
"NearNadir_Composite_Snow_Free_Std",
"OffNadir_Composite_Snow_Covered",
"OffNadir_Composite_Snow_Covered_Num",
"OffNadir_Composite_Snow_Covered_Quality",
"OffNadir_Composite_Snow_Covered_Std",
"OffNadir_Composite_Snow_Free",
"OffNadir_Composite_Snow_Free_Num",
"OffNadir_Composite_Snow_Free_Quality",
"OffNadir_Composite_Snow_Free_Std"
```

- Select only two layers: "AllAngle_Composite_Snow_Covered" and "AllAngle_Composite_Snow_Free", and refer to the resulting raster as vv.
- Read the gas flare spots from the file gas_flare_spot_sf_square_0_2deg.gpkg using st_read. Select rows corresponding to the same year as the vv raster file.
- Apply the exact_extract() function from the exactextractr package to identify pixels in vv that overlap with gas flare spots. Use the following parameters:
  - coverage_area = FALSE.
  - include_cell = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:
  - AllAngle_Composite_Snow_Covered: NTL value for the pixel.
  - AllAngle_Composite_Snow_Free: Snow-free NTL value for the pixel.
  - cell: Pixel ID as defined in the vv file.
  - coverage_fraction: Fraction of the pixel overlapping with the polygon.
- Extract the unique pixel IDs from the cell column in the list and refer to them as unique_indices.
- For each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
  - Perform a spatial extraction using the exact_extract() function to calculate NTL values for each polygon in paste0("lc_urban_inters_id_1deg_", year, ".gpkg"). Use the following parameters:
    * SpatRaster file: vv[[i]], where i is the layer index.
    * Polygon file: paste0("lc_urban_inters_id_1deg_", year, ".gpkg"), excluding the column fid_3, and rename column paste0("test",year) to land_type.
    * coverage_area = TRUE.
    * include_cols = c("id", "iso", "cell_id", "land_type").

        * summarize_df = TRUE.
        * include_cell = TRUE.
        * fun = function(df_in){
          out_df <- df_in %>%
          mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
          group_by(id, iso, cell_id, land_type) %>%
          summarize(NTL = sum(coverage_area * value), .groups = "drop")
          return(out_df)}.
    - The resulting file will include columns: id, iso, cell_id, land_type, and NTL.
    - Save the resulting file for the current year and layer as paste0("NTL_urban_extracted_1deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.
- Again, for each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
    - Perform a spatial extraction using the exact_extract() function to calculate NTL values for each polygon in paste0("lc_cropland_inters_id_1deg_", year, ".gpkg"). Use the following parameters:
        * SpatRaster file: vv[[i]], where i is the layer index.
        * Polygon file: paste0("lc_cropland_inters_id_1deg_", year, ".gpkg"), excluding the column fid_3, and rename column paste0("test",year) to land_type.
        * coverage_area = TRUE.
        * include_cols = c("id", "iso", "cell_id", "land_type").
        * summarize_df = TRUE.
        * include_cell = TRUE.
        * fun = function(df_in){
          out_df <- df_in %>%
          mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
          group_by(id, iso, cell_id, land_type) %>%
          summarize(NTL = sum(coverage_area * value), .groups = "drop")
          return(out_df)}.
    - The resulting file will include columns: id, iso, cell_id, land_type, and NTL.
    - Save the resulting file for the current year and layer as paste0("NTL_cropland_extracted_1deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

5. **0.5-degree Cells**

For each paste0("test", year, ".vrt") file from 2012 to 2021 obtained from Section 24 Step 1, perform the following steps:

- Read the paste0("test", year, ".vrt") file into a raster using the rast() function.
- Rename the raster layers using the setNames() function in the following order:
  - "AllAngle_Composite_Snow_Covered",
    "AllAngle_Composite_Snow_Covered_Num",
    "AllAngle_Composite_Snow_Covered_Quality",
    "AllAngle_Composite_Snow_Covered_Std",
    "AllAngle_Composite_Snow_Free",
    "AllAngle_Composite_Snow_Free_Num",
    "AllAngle_Composite_Snow_Free_Quality",
    "AllAngle_Composite_Snow_Free_Std",
    "DNB_Platform",
    "Land_Water_Mask",
    "NearNadir_Composite_Snow_Covered",
    "NearNadir_Composite_Snow_Covered_Num",
    "NearNadir_Composite_Snow_Covered_Quality",
    "NearNadir_Composite_Snow_Covered_Std",
    "NearNadir_Composite_Snow_Free",
    "NearNadir_Composite_Snow_Free_Num",
    "NearNadir_Composite_Snow_Free_Quality",
    "NearNadir_Composite_Snow_Free_Std",
    "OffNadir_Composite_Snow_Covered",
    "OffNadir_Composite_Snow_Covered_Num",
    "OffNadir_Composite_Snow_Covered_Quality",
    "OffNadir_Composite_Snow_Covered_Std",
    "OffNadir_Composite_Snow_Free",
    "OffNadir_Composite_Snow_Free_Num",
    "OffNadir_Composite_Snow_Free_Quality",
    "OffNadir_Composite_Snow_Free_Std"
- Select only two layers: "AllAngle_Composite_Snow_Covered" and "AllAngle_Composite_Snow_Free", and refer to the resulting raster as vv.
- Read the gas flare spots from the file gas_flare_spot_sf_square_0_2deg.gpkg using st_read. Select rows corresponding to the same year as the vv raster file.
- Apply the exact_extract() function from the exactextractr package to identify pixels in vv that overlap with gas flare spots. Use the following parameters:
  - coverage_area = FALSE.
  - include_cell = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:

  - AllAngle_Composite_Snow_Covered: NTL value for the pixel.
  - AllAngle_Composite_Snow_Free: Snow-free NTL value for the pixel.
  - cell: Pixel ID as defined in the vv file.
  - coverage_fraction: Fraction of the pixel overlapping with the polygon.
- Extract the unique pixel IDs from the cell column in the list and refer to them as unique_indices.
- For each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
  - Perform a spatial extraction using the exact_extract() function to calculate NTL values for each polygon in paste0("lc_urban_inters_id_0_5deg_", year, ".gpkg"). Use the following parameters:
    * SpatRaster file: vv[[i]], where i is the layer index.
    * Polygon file: paste0("lc_urban_inters_id_0_5deg_", year, ".gpkg"), excluding the column fid_3, and rename column paste0("test",year) to land_type.
    * coverage_area = TRUE.
    * include_cols = c("id", "iso", "cell_id", "subcell_id", "land_type").
    * summarize_df = TRUE.
    * include_cell = TRUE.
    * fun = function(df_in){
      out_df <- df_in %>%
      mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_indices, 0, value)) %>%
      group_by(id, iso, cell_id, subcell_id, land_type) %>%
      summarize(NTL = sum(coverage_area * value), .groups = "drop")
      return(out_df)}.
  - The resulting file will include columns: id, iso, cell_id, subcell_id, land_type, and NTL.
  - Save the resulting file for the current year and layer as paste0("NTL_urban_extracted_0_5deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.
- Again, for each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
  - Perform a spatial extraction using the exact_extract() function to calculate NTL values for each polygon in paste0("lc_cropland_inters_id_0_5deg_", year, ".gpkg"). Use the following parameters:
    * SpatRaster file: vv[[i]], where i is the layer index.
    * Polygon file: paste0("lc_cropland_inters_id_0_5deg_", year, ".gpkg"), excluding the column fid_3, and rename column paste0("test", year) to land_type.
    * coverage_area = TRUE.

- * include_cols = c("id", "iso", "cell_id", "subcell_id", "land_type").
  * summarize_df = TRUE.
  * include_cell = TRUE.
  * fun = function(df_in){
    out_df <- df_in %>%
    mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%
    group_by(id, iso, cell_id, subcell_id, land_type) %>%
    summarize(NTL = sum(coverage_area * value), .groups = "drop")
    return(out_df)}.
- The resulting file will include columns: id, iso, cell_id, subcell_id, land_type, and NTL.
- Save the resulting file for the current year and layer as paste0("NTL_crop land_extracted_0_5deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

6. **0.25-degree Cells**

For each paste0("test", year, ".vrt") file from 2012 to 2021 obtained from Section 24 Step 1, perform the following steps:

- Read the paste0("test", year, ".vrt") file into a raster using the rast() function.
- Rename the raster layers using the setNames() function in the following order:
  - `"AllAngle_Composite_Snow_Covered",`
    `"AllAngle_Composite_Snow_Covered_Num",`
    `"AllAngle_Composite_Snow_Covered_Quality",`
    `"AllAngle_Composite_Snow_Covered_Std",`
    `"AllAngle_Composite_Snow_Free",`
    `"AllAngle_Composite_Snow_Free_Num",`
    `"AllAngle_Composite_Snow_Free_Quality",`
    `"AllAngle_Composite_Snow_Free_Std",`
    `"DNB_Platform",`
    `"Land_Water_Mask",`
    `"NearNadir_Composite_Snow_Covered",`
    `"NearNadir_Composite_Snow_Covered_Num",`
    `"NearNadir_Composite_Snow_Covered_Quality",`
    `"NearNadir_Composite_Snow_Covered_Std",`
    `"NearNadir_Composite_Snow_Free",`
    `"NearNadir_Composite_Snow_Free_Num",`
    `"NearNadir_Composite_Snow_Free_Quality",`
    `"NearNadir_Composite_Snow_Free_Std",`
    `"OffNadir_Composite_Snow_Covered",`
    `"OffNadir_Composite_Snow_Covered_Num",`

```
"OffNadir_Composite_Snow_Covered_Quality",
"OffNadir_Composite_Snow_Covered_Std",
"OffNadir_Composite_Snow_Free",
"OffNadir_Composite_Snow_Free_Num",
"OffNadir_Composite_Snow_Free_Quality",
"OffNadir_Composite_Snow_Free_Std"
```

- Select only two layers: `"AllAngle_Composite_Snow_Covered"` and `"AllAngle_Composite_Snow_Free"`, and refer to the resulting raster as `vv`.
- Read the gas flare spots from the file `gas_flare_spot_sf_square_0_2deg.gpkg` using `st_read`. Select rows corresponding to the same year as the `vv` raster file.
- Apply the `exact_extract()` function from the `exactextractr` package to identify pixels in `vv` that overlap with gas flare spots. Use the following parameters:
  - `coverage_area` = FALSE.
  - `include_cell` = TRUE.
- The resulting file is a list where each item represents a polygon from the gas flare spots file. Each list item contains a dataframe with the following columns:
  - `AllAngle_Composite_Snow_Covered`: NTL value for the pixel.
  - `AllAngle_Composite_Snow_Free`: Snow-free NTL value for the pixel.
  - `cell`: Pixel ID as defined in the `vv` file.
  - `coverage_fraction`: Fraction of the pixel overlapping with the polygon.
- Extract the unique pixel IDs from the `cell` column in the list and refer to them as `unique_indices`.
- For each of the two layers in `vv`, perform the following steps using `mclapply` with `mc.cores = 2` for parallel processing:
  - Perform a spatial extraction using the `exact_extract()` function to calculate NTL values for each polygon in `paste0("lc_urban_inters_id_0_25deg_", year, ".gpkg")`. Use the following parameters:
    - SpatRaster file: `vv[[i]]`, where `i` is the layer index.
    - Polygon file: `paste0("lc_urban_inters_id_0_25deg_", year, ".gpkg")`, excluding the column `fid_3`, and rename column `paste0("test",year)` to `land_type`.
    - `coverage_area` = TRUE.
    - `include_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25", "land_type")`.
    - `summarize_df` = TRUE.
    - `include_cell` = TRUE.
    - fun = function(df_in){
      out_df <- df_in %>%
      mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_-indices, 0, value)) %>%

group_by(id, iso, cell_id, subcell_id, subcell_id_0_25, land_type) %>%
summarize(NTL = sum(coverage_area * value), .groups = "drop")
return(out_df)}.

  - The resulting file will include columns: id, iso, cell_id, subcell_id, subcell_id_0_25, land_type, and NTL.
  - Save the resulting file for the current year and layer as paste0("NTL_urban_extracted_0_25deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

- Again, for each of the two layers in vv, perform the following steps using mclapply with mc.cores = 2 for parallel processing:
  - Perform a spatial extraction using the exact_extract() function to calculate NTL values for each polygon in paste0("lc_cropland_inters_id_0_25deg_", year, ".gpkg"). Use the following parameters:
    * SpatRaster file: vv[[i]], where i is the layer index.
    * Polygon file: paste0("lc_cropland_inters_id_0_25deg_", year, ".gpkg"), excluding the column fid_3, and rename column paste0("test", year) to land_type.
    * coverage_area = TRUE.
    * include_cols = c("id", "iso", "cell_id", "subcell_id", "subcell_id_0_25", "land_type").
    * summarize_df = TRUE.
    * include_cell = TRUE.
    * fun = function(df_in){
      out_df <- df_in %>%
      mutate(value = ifelse(is.na(value) | value == 65535 | cell %in% unique_indices, 0, value)) %>%
      group_by(id, iso, cell_id, subcell_id, subcell_id_0_25, land_type) %>%
      summarize(NTL = sum(coverage_area * value), .groups = "drop")
      return(out_df)}.
  - The resulting file will include columns: id, iso, cell_id, subcell_id, subcell_id_0_25, land_type, and NTL.
  - Save the resulting file for the current year and layer as paste0("NTL_cropland_extracted_0_25deg_", i, year, ".RData"). Ensure that the dataframe name matches the name of the RData file when saving it. Data across years and layers will be combined later.

7. **Combine Data Across Years and Layers**

   1-degree Urban Areas:

   - For each year from 2012 to 2021, perform the following steps:

- Load the file `paste0("NTL_urban_extracted_1deg_1", year, ".RData")` using the `load()` function.
- Add a new column `year` with values corresponding to the respective year.
- Rename the column `NTL` to `NTL_snow_covered_period`.
- Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_urban_extracted_1deg_2", year, ".RData")` file.

- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_urban_full_1deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

1-degree Cropland Areas:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file `paste0("NTL_cropland_extracted_1deg_1", year, ".RData")` using the `load()` function.
  - Add a new column `year` with values corresponding to the respective year.
  - Rename the column `NTL` to `NTL_snow_covered_period`.
  - Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_cropland_extracted_1deg_2", year, ".RData")` file.
- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_cropland_full_1deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

0.5-degree Urban Areas:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file `paste0("NTL_urban_extracted_0_5deg_1", year, ".RData")` using the `load()` function.
  - Add a new column `year` with values corresponding to the respective year.
  - Rename the column `NTL` to `NTL_snow_covered_period`.
  - Create a new column `NTL_snow_free_period` with values taken from the `NTL` column of the corresponding `paste0("NTL_urban_extracted_0_5deg_2", year, ".RData")` file.
- Combine the dataframes for all years into a single dataframe using `bind_rows`.
- Save the final dataframe as `NTL_urban_full_0_5deg.RData`, ensuring that the dataframe name matches the name of the saved `RData` file.

0.5-degree Cropland Areas:

- For each year from 2012 to 2021, perform the following steps:

  - Load the file paste0("NTL_cropland_extracted_0_5deg_1", year, ".RData") using the load() function.
  - Add a new column year with values corresponding to the respective year.
  - Rename the column NTL to NTL_snow_covered_period.
  - Create a new column NTL_snow_free_period with values taken from the NTL column of the corresponding paste0("NTL_cropland_extracted_0_5deg_2", year, ".RData") file.
- Combine the dataframes for all years into a single dataframe using bind_rows.
- Save the final dataframe as NTL_cropland_full_0_5deg.RData, ensuring that the dataframe name matches the name of the saved RData file.

0.25-degree Urban Areas:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file paste0("NTL_urban_extracted_0_25deg_1", year, ".RData") using the load() function.
  - Add a new column year with values corresponding to the respective year.
  - Rename the column NTL to NTL_snow_covered_period.
  - Create a new column NTL_snow_free_period with values taken from the NTL column of the corresponding paste0("NTL_urban_extracted_0_25deg_2", year, ".RData") file.
- Combine the dataframes for all years into a single dataframe using bind_rows.
- Save the final dataframe as NTL_urban_full_0_25deg.RData, ensuring that the dataframe name matches the name of the saved RData file.

0.25-degree Cropland Areas:

- For each year from 2012 to 2021, perform the following steps:
  - Load the file paste0("NTL_cropland_extracted_0_25deg_1", year, ".RData") using the load() function.
  - Add a new column year with values corresponding to the respective year.
  - Rename the column NTL to NTL_snow_covered_period.
  - Create a new column NTL_snow_free_period with values taken from the NTL column of the corresponding paste0("NTL_cropland_extracted_0_25deg_2", year, ".RData") file.
- Combine the dataframes for all years into a single dataframe using bind_rows.
- Save the final dataframe as NTL_cropland_full_0_25deg.RData, ensuring that the dataframe name matches the name of the saved RData file.

29 Finalize the Training and Predicting Dataset

In this section, we combine all the predictors and GDP values (when available) obtained from the previous sections into unified datasets.

1. **1-degree Predictors**

Population Share:

- Read the land_pop_extracted_region_level_1deg.RData file obtained in Section 20 Step 1 using load.
- Group the dataset by the id and year columns.
- Create a new column pop_share, where each value is the proportion of the pop value to the total pop within the group, calculated as pop/sum(pop) .
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as pop_put_in_model

CO2 Emissions From Biofuels Share:

- Read the CO2_bio_full_1deg.RData file obtained in Section 26 Step 2 using load.
- Create a new column CO2_bio_heavy_indus with values equal to the sum of corresponding values of columns CO2_bio_fuel_exploitation, CO2_bio_oil_refine_transf, and CO2_bio_power_industry.
- Create a new column CO2_bio_tspt with values equal to the sum of corresponding values of columns CO2_bio_road_transp and CO2_bio_shipping.
- Group the dataset by the id and year columns.
- Create a new column CO2_bio_manuf_conbust_share, where each value is calculated as:
  - If CO2_bio_combustion_for_manufacturing equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_bio_combustion_for_manufacturing/sum(CO2_bio_combustion_for_manufacturing) within the group.
- Create a new column CO2_bio_heavy_indus_share, where each value is calculated as:
  - If CO2_bio_heavy_indus equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_bio_heavy_indus/sum(CO2_bio_heavy_indus) within the group.
- Create a new column CO2_bio_tspt_share, where each value is calculated as:
  - If CO2_bio_tspt equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_bio_tspt/sum(CO2_bio_tspt) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as CO2_bio_put_in_model

CO2 Emissions From Non-organic Fuels Share:

- Read the `CO2_non_org_full_1deg.RData` file obtained in Section 27 Step 2 using `load`.
- Create a new column `CO2_non_org_heavy_indus` with values equal to the sum of corresponding values of columns `CO2_non_org_fuel_exploitation`, `CO2_non_org_iron_steel`, `CO2_non_org_non_ferrous_metal` `CO2_non_org_non_metallic_mineral`, `CO2_non_org_oil_refine_transf`, and `CO2_non_org_power_industry`.
- Create a new column `CO2_non_org_tspt` with values equal to the sum of corresponding values of columns `CO2_non_org_road_transp` and `CO2_non_org_shipping`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `CO2_non_org_manuf_conbust_share`, where each value is calculated as:
  - If `CO2_non_org_combustion_for_manufacturing` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_non_org_combustion_for_manufacturing/sum(CO2_non_org_combustion_for_manufacturing)` within the group.
- Create a new column `CO2_non_org_heavy_indus_share`, where each value is calculated as:
  - If `CO2_non_org_heavy_indus` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_non_org_heavy_indus/sum(CO2_non_org_heavy_indus)` within the group.
- Create a new column `CO2_non_org_tspt_share`, where each value is calculated as:
  - If `CO2_non_org_tspt` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_non_org_tspt/sum(CO2_non_org_tspt)` within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as `CO2_non_org_put_in_model`

NPP Share:

- Read the `NPP_full_1deg.RData` file obtained in Section 22 Step 2 using `load`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `NPP_share`, where each value is the proportion of the `NPP` value to the total `NPP` within the group, calculated as `NPP/sum(NPP)`
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as `NPP_put_in_model`

Urban NTL Share:

- Read the `NTL_urban_full_1deg.RData` file obtained in Section 28 Step 7 using `load`.

- Group the dataset by the id and year columns.
- Create a new column NTL_urban_snow_covered_period_share, where each value is calculated as:
  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_urban_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_urban_snow_covered_period
- Rename NTL_snow_free_period column to NTL_urban_snow_free_period
- Exclude the column land_type
- Refer to the resulting dataframe as NTL_urban_put_in_model

Cropland NTL Share:

- Read the NTL_cropland_full_1deg.RData file obtained in Section 28 Step 7 using load.
- Group the dataset by the cell_id, iso, id, and year columns
- Summarize the dataset by updating the NTL_snow_covered_period column values to be the sum of NTL_snow_covered_period values within each group, and similarly update the NTL_snow_free_period column. Remove grouping use .groups = "drop".
- Group the dataset by the id and year columns.
- Create a new column NTL_cropland_snow_covered_period_share, where each value is calculated as:
  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_cropland_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_cropland_snow_covered_period

- Rename NTL_snow_free_period column to NTL_cropland_snow_free_period
- Refer to the resulting dataframe as NTL_cropland_put_in_model

Other Lands NTL Share:

- Read the NTL_full_1deg.RData file obtained in Section 24 Step 5 using load.
- Rename the NTL_snow_covered_period column to NTL_full_snow_covered_period.
- Rename the NTL_snow_free_period column to NTL_full_snow_free_period.
- Select only the rows where the year column values are between 2012 and 2021 inclusive.
- Apply left_join to combine the current dataframe with NTL_urban_put_in_model, adding the following columns: NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share. Ensure the current dataframe is the base file.
- Replace any NA values with 0 for the columns NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share.
- Apply left_join again to combine the current dataframe with NTL_cropland_put_in_model, adding the columns: NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Replace any NA values with 0 for the columns NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Create a new column NTL_other_snow_covered_period with values equal to NTL_full_snow_covered_period - NTL_urban_snow_covered_period - NTL_cropland_snow_covered_period.
- Similarly, create a new column NTL_other_snow_free_period with values equal to NTL_full_snow_free_period - NTL_urban_snow_free_period - NTL_cropland_snow_free_period.
- Group the dataset by id and year.
- Create a new column NTL_other_snow_covered_period_share, where each value is calculated as:
  - If NTL_other_snow_covered_period equals 0, set the value to 0.
  - Otherwise, calculate it as NTL_other_snow_covered_period/sum(NTL_other_snow_covered_period) within the group.
- Create a new column NTL_other_snow_free_period_share, where each value is calculated as:
  - If NTL_other_snow_free_period equals 0, set the value to 0.

- Otherwise, calculate it as NTL_other_snow_free_period/sum(NTL_other_snow_free_period) within the group.

- Remove the grouping to return to an ungrouped dataframe.
- Select only the following columns: cell_id, iso, id, year, NTL_urban_snow_covered_period_share, NTL_urban_snow_free_period_share, NTL_cropland_snow_covered_period_share, NTL_cropland_snow_free_period_share, NTL_other_snow_covered_period_share, NTL_other_snow_free_period_share.
- Convert the resulting dataframe to a regular dataframe using as.data.frame.
- Refer to the resulting dataframe as NTL_put_in_model.

Share of Different Land Use Types:

- Read the file lc_full_1deg.RData obtained in Section 21 Step 2 using load.
- Create a new column forest_full with values equal to the sum of the corresponding row values from the columns open_forest and dense_forest.
- Create a new column cropland_full with values equal to the sum of the corresponding row values from the columns cropland, forest_cropland, and herbaceous_cropland.
- Exclude the columns open_forest, dense_forest, cropland, forest_cropland, and herbaceous_cropland.
- Group the dataset by id and year.
- Create a new column barren_share, where each value is the proportion of the barren value to the total barren within the group, calculated as barren/sum(barren)
- Create a new column snow_ice_share, where each value is the proportion of the snow_ice value to the total snow_ice within the group, calculated as snow_ice/sum(snow_ice)
- Create a new column water_share, where each value is the proportion of the water value to the total water within the group, calculated as water/sum(water)
- Create a new column urban_share, where each value is the proportion of the urban value to the total urban within the group, calculated as urban/sum(urban)
- Create a new column forest_share, where each value is the proportion of the forest_full value to the total forest_full within the group, calculated as forest_full/sum(forest_full)
- Create a new column herbaceous_share, where each value is the proportion of the herbaceous value to the total herbaceous within the group, calculated as herbaceous/sum(herbaceous)
- Create a new column cropland_share, where each value is the proportion of the cropland_full value to the total cropland_full within the group, calculated as cropland_full/sum(cropland_full)

- Create a new column shrub_share, where each value is the proportion of the shrub value to the total shrub within the group, calculated as shrub/sum(shrub)
- Remove the grouping to return to an ungrouped dataframe.
- Replace any NA values to 0
- Refer to the resulting dataframe as lc_put_in_model.

Ruggedness:

- Read the mean_ruggedness_1deg.csv file obtained in Section 25 Step 2 using read.csv.
- Convert the cell_id column values to characters using as.character.

GDP per Capita:

- Read the national_gdpc_const_2017_USD.csv file obtained in Section 14 Step 5 using read.csv.
- Add a new row with iso value as Ala, Country value as Alaska, and all other column values the same as the USA.

2. **Finalize 1-degree Model Prediction Dataset:**

   Start with the file pop_put_in_model, considering it as the base file.

   Combine it with the following dataframes one by one using left_join with the argument by = join_by(cell_id, id, iso, year): CO2_bio_put_in_model, CO2_non_org_put_in_model, NPP_put_in_model, NTL_put_in_model, lc_put_in_model.

   Combine it with the rug_put_in_model file using left_join, but with the argument by = join_by(cell_id, id, iso).

   Replace any NA values with 0.

   Select only rows where year values are between 2012 and 2021, inclusive.

   Combine the current dataframe with national_GDPC using left_join. Again, ensure that the current dataframe is the base file.

   Select only the columns: id, iso, cell_id, year, mean_rug, national_gdpc, pop, and any columns whose names contain "share".

   Create a new column original_order with values representing the row number using row_number().

   Arrange the dataset in ascending order based on the columns cell_id, id, iso, and year.

   Group the dataset by the columns cell_id, id, and iso.

   Create new columns representing the previous year's values by using the lag() function. If no previous year's data exists, use the value from the first available year (2012) within the group. The following new columns are created:

- lag_NTL_urban_share = lag(NTL_urban_snow_free_period_share, default = first(NTL_urban_snow_free_period_share))
- lag_urban_share = lag(urban_share, default = first(urban_share))
- lag_cropland_share = lag(cropland_share, default = first(cropland_share))

- lag_NTL_other_share = lag(NTL_other_snow_free_period_share, default = first(NTL_other_snow_free_period_share))
- lag_NTL_cropland_share = lag(NTL_cropland_snow_free_period_share, default = first(NTL_cropland_snow_free_period_share))
- lag_CO2_bio_mc_share = lag(CO2_bio_manuf_conbust_share, default = first(CO2_bio_manuf_conbust_share))
- lag_CO2_nonorg_mc_share = lag(CO2_non_org_manuf_conbust_share, default = first(CO2_non_org_manuf_conbust_share))
- lag_CO2_bio_heavy_indus_share = lag(CO2_bio_heavy_indus_share, default = first(CO2_bio_heavy_indus_share))
- lag_CO2_non_org_heavy_indus_share = lag(CO2_non_org_heavy_indus_share, default = first(CO2_non_org_heavy_indus_share))
- lag_CO2_bio_tspt_share = lag(CO2_bio_tspt_share, default = first(CO2_bio_tspt_share))
- lag_CO2_non_org_tspt_share = lag(CO2_non_org_tspt_share, default = first(CO2_non_org_tspt_share))
- lag_pop_share = lag(pop_share, default = first(pop_share))
- lag_NPP_share = lag(NPP_share, default = first(NPP_share))

Remove the grouping to return to an ungrouped dataframe.

Arrange the dataset by the original_order column.

Exclude the column original_order.

Save the resulting dataframe as new_predictors_put_in_model_1deg.RData.

3. **Finalize 1-degree Model Training Dataset:**

Read the training_iso_1deg_cell_GCP.RData file obtained in Section 18 Step 7 using the load() function.

Group the dataset by the iso and year columns.

Create a new column GCP_share_1deg, where each value is calculated as:

- If GCP_1deg equals 0, set the value to 0.
- Otherwise, calculate it as GCP_1deg/sum(GCP_1deg) within each group.

Remove the grouping to return to an ungrouped dataframe.

Apply left_join to combine the current dataframe with the following dataframe, ensuring that the current dataframe is the base file:

- Read the new_predictors_put_in_model_1deg.RData file using load.

- Convert it to a dataframe using `as.data.frame`.
- Remove the `geom` column.
- Create a new column `iso_change`, where for rows with `USA` in the `iso` column, it concatenates `USA_` with the first two characters of the `id` column. For all other rows, the `iso_change` column will retain the value from the `id` column.
- Select only the columns `cell_id`, `iso_change`, `year`, `mean_rug`, `national_gdpc`, `pop`, and any columns containing the substring `share`.
- Rename the column `iso_change` to `iso`.

Omit any rows containing `NA` values.

Save the resulting file as `new_predict_data_complete_1deg.RData`.

4. **0.5-degree Predictors**

Population Share:

- Read the `land_pop_extracted_region_level_0_5deg.RData` file obtained in Section 20 Step 2 using `load`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `pop_share`, where each value is the proportion of the `pop` value to the total `pop` within the group, calculated as `pop/sum(pop)`.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as `pop_put_in_model`

CO2 Emissions From Biofuels Share:

- Read the `CO2_bio_full_0_5deg.RData` file obtained in Section 26 Step 3 using `load`.
- Create a new column `CO2_bio_heavy_indus` with values equal to the sum of corresponding values of columns `CO2_bio_fuel_exploitation`, `CO2_bio_oil_refine_transf`, and `CO2_bio_power_industry`.
- Create a new column `CO2_bio_tspt` with values equal to the sum of corresponding values of columns `CO2_bio_road_transp` and `CO2_bio_shipping`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `CO2_bio_manuf_conbust_share`, where each value is calculated as:
  - If `CO2_bio_combustion_for_manufacturing` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_bio_combustion_for_manufacturing/sum(CO2_bio_combustion_for_manufacturing)` within the group.
- Create a new column `CO2_bio_heavy_indus_share`, where each value is calculated as:
  - If `CO2_bio_heavy_indus` equals `0`, the value is set to `0`.

- Otherwise, calculate it as CO2_bio_heavy_indus/sum(CO2_bio_heavy_indus) within the group.
- Create a new column CO2_bio_tspt_share, where each value is calculated as:
  - If CO2_bio_tspt equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_bio_tspt/sum(CO2_bio_tspt) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as CO2_bio_put_in_model

CO2 Emissions From Non-organic Fuels Share:

- Read the CO2_non_org_full_0_5deg.RData file obtained in Section 27 Step 3 using load.
- Create a new column CO2_non_org_heavy_indus with values equal to the sum of corresponding values of columns CO2_non_org_fuel_exploitation, CO2_non_org_iron_steel, CO2_non_org_non_ferrous_metal CO2_non_org_non_metallic_mineral, CO2_non_org_oil_refine_transf, and CO2_non_org_power_industry.
- Create a new column CO2_non_org_tspt with values equal to the sum of corresponding values of columns CO2_non_org_road_transp and CO2_non_org_shipping.
- Group the dataset by the id and year columns.
- Create a new column CO2_non_org_manuf_conbust_share, where each value is calculated as:
  - If CO2_non_org_combustion_for_manufacturing equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_non_org_combustion_for_manufacturing/sum(CO2_non_org_combustion_for_manufacturing) within the group.
- Create a new column CO2_non_org_heavy_indus_share, where each value is calculated as:
  - If CO2_non_org_heavy_indus equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_non_org_heavy_indus/sum(CO2_non_org_heavy_indus) within the group.
- Create a new column CO2_non_org_tspt_share, where each value is calculated as:
  - If CO2_non_org_tspt equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_non_org_tspt/sum(CO2_non_org_tspt) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as CO2_non_org_put_in_model

NPP Share:

- Read the NPP_full_0_5deg.RData file obtained in Section 22 Step 3 using load.
- Group the dataset by the id and year columns.
- Create a new column NPP_share, where each value is the proportion of the NPP value to the total NPP within the group, calculated as NPP/sum(NPP)
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as NPP_put_in_model

Urban NTL Share:

- Read the NTL_urban_full_0_5deg.RData file obtained in Section 28 Step 7 using load.
- Group the dataset by the id and year columns.
- Create a new column NTL_urban_snow_covered_period_share, where each value is calculated as:
  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_urban_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_urban_snow_covered_period
- Rename NTL_snow_free_period column to NTL_urban_snow_free_period
- Exclude the column land_type
- Refer to the resulting dataframe as NTL_urban_put_in_model

Cropland NTL Share:

- Read the NTL_cropland_full_0_5deg.RData file obtained in Section 28 Step 7 using load.
- Group the dataset by the subcell_id, cell_id, iso, id, and year columns
- Summarize the dataset by updating the NTL_snow_covered_period column values to be the sum of NTL_snow_covered_period values within each group, and similarly update the NTL_snow_free_period column. Remove grouping use .groups = "drop".
- Group the dataset by the id and year columns.
- Create a new column NTL_cropland_snow_covered_period_share, where each value is calculated as:

  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_cropland_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_cropland_snow_covered_period
- Rename NTL_snow_free_period column to NTL_cropland_snow_free_period
- Refer to the resulting dataframe as NTL_cropland_put_in_model

Other Lands NTL Share:

- Read the NTL_full_0_5deg.RData file obtained in Section 24 Step 5 using load.
- Rename the NTL_snow_covered_period column to NTL_full_snow_covered_period.
- Rename the NTL_snow_free_period column to NTL_full_snow_free_period.
- Select only the rows where the year column values are between 2012 and 2021 inclusive.
- Apply left_join to combine the current dataframe with NTL_urban_put_in_model, adding the following columns: NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share. Ensure the current dataframe is the base file.
- Replace any NA values with 0 for the columns NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share.
- Apply left_join again to combine the current dataframe with NTL_cropland_put_in_model, adding the columns: NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Replace any NA values with 0 for the columns NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Create a new column NTL_other_snow_covered_period with values equal to NTL_full_snow_covered_period - NTL_urban_snow_covered_period - NTL_cropland_snow_covered_period.

- Similarly, create a new column NTL_other_snow_free_period with values equal to NTL_full_snow_free_period - NTL_urban_snow_free_period - NTL_cropland_snow_free_period.
- Group the dataset by id and year.
- Create a new column NTL_other_snow_covered_period_share, where each value is calculated as:
  - If NTL_other_snow_covered_period equals 0, set the value to 0.
  - Otherwise, calculate it as NTL_other_snow_covered_period/sum(NTL_other_snow_covered_period) within the group.
- Create a new column NTL_other_snow_free_period_share, where each value is calculated as:
  - If NTL_other_snow_free_period equals 0, set the value to 0.
  - Otherwise, calculate it as NTL_other_snow_free_period/sum(NTL_other_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Select only the following columns: cell_id, subcell_id, iso, id, year, NTL_urban_snow_covered_period_share, NTL_urban_snow_free_period_share, NTL_cropland_snow_covered_period_share, NTL_cropland_snow_free_period_share, NTL_other_snow_covered_period_share, NTL_other_snow_free_period_share.
- Convert the resulting dataframe to a regular dataframe using as.data.frame.
- Refer to the resulting dataframe as NTL_put_in_model.

Share of Different Land Use Types:

- Read the file lc_full_0_5deg.RData obtained in Section 21 Step 3 using load.
- Create a new column forest_full with values equal to the sum of the corresponding row values from the columns open_forest and dense_forest.
- Create a new column cropland_full with values equal to the sum of the corresponding row values from the columns cropland, forest_cropland, and herbaceous_cropland.
- Exclude the columns open_forest, dense_forest, cropland, forest_cropland, and herbaceous_cropland.
- Group the dataset by id and year.
- Create a new column barren_share, where each value is the proportion of the barren value to the total barren within the group, calculated as barren/sum(barren)
- Create a new column snow_ice_share, where each value is the proportion of the snow_ice value to the total snow_ice within the group, calculated as snow_ice/sum(snow_ice)
- Create a new column water_share, where each value is the proportion of the water value to the total water within the group, calculated as water/sum(water)

- Create a new column urban_share, where each value is the proportion of the urban value to the total urban within the group, calculated as urban/sum(urban)
- Create a new column forest_share, where each value is the proportion of the forest_full value to the total forest_full within the group, calculated as forest_full/sum(forest_full)
- Create a new column herbaceous_share, where each value is the proportion of the herbaceous value to the total herbaceous within the group, calculated as herbaceous/sum(herbaceous)
- Create a new column cropland_share, where each value is the proportion of the cropland_full value to the total cropland_full within the group, calculated as cropland_full/sum(cropland_full)
- Create a new column shrub_share, where each value is the proportion of the shrub value to the total shrub within the group, calculated as shrub/sum(shrub)
- Remove the grouping to return to an ungrouped dataframe.
- Replace any NA values to 0
- Refer to the resulting dataframe as lc_put_in_model.

Ruggedness:

- Read the mean_ruggedness_0_5deg.csv file obtained in Section 25 Step 3 using read.csv.
- Convert the cell_id column values to characters using as.character.

GDP per Capita:

- Read the national_gdpc_const_2017_USD.csv file obtained in Section 14 Step 5 using read.csv.
- Add a new row with iso value as Ala, Country value as Alaska, and all other column values the same as the USA.

5. **Finalize 0.5-degree Model Prediction Dataset:**

Start with the file pop_put_in_model, considering it as the base file.

Combine it with the following dataframes one by one using left_join with the argument by = join_by(cell_id, subcell_id, id, iso, year): CO2_bio_put_in_model, CO2_non_org_put_in_model, NPP_put_in_model, NTL_put_in_model, lc_put_in_model.

Combine it with the rug_put_in_model file using left_join, but with the argument by = join_by(cell_id, subcell_id, id, iso).

Replace any NA values with 0.

Select only rows where year values are between 2012 and 2021, inclusive.

Combine the current dataframe with national_GDPC using left_join. Again, ensure that the current dataframe is the base file.

Select only the columns: id, iso, cell_id, subcell_id, year, mean_rug, national_gdpc, pop, and any columns whose names contain "share".

Create a new column original_order with values representing the row number using row_number().

Arrange the dataset in ascending order based on the columns subcell_id, cell_id, id, iso, and year.

Group the dataset by the columns subcell_id, cell_id, id, and iso.

Create new columns representing the previous year's values by using the lag() function. If no previous year's data exists, use the value from the first available year (2012) within the group. The following new columns are created:

- lag_NTL_urban_share = lag(NTL_urban_snow_free_period_share, default = first(NTL_urban_snow_free_period_share))
- lag_urban_share = lag(urban_share, default = first(urban_share))
- lag_cropland_share = lag(cropland_share, default = first(cropland_share))
- lag_NTL_other_share = lag(NTL_other_snow_free_period_share, default = first(NTL_other_snow_free_period_share))
- lag_NTL_cropland_share = lag(NTL_cropland_snow_free_period_share, default = first(NTL_cropland_snow_free_period_share))
- lag_CO2_bio_mc_share = lag(CO2_bio_manuf_conbust_share, default = first(CO2_bio_manuf_conbust_share))
- lag_CO2_nonorg_mc_share = lag(CO2_non_org_manuf_conbust_share, default = first(CO2_non_org_manuf_conbust_share))
- lag_CO2_bio_heavy_indus_share = lag(CO2_bio_heavy_indus_share, default = first(CO2_bio_heavy_indus_share))
- lag_CO2_non_org_heavy_indus_share = lag(CO2_non_org_heavy_indus_share, default = first(CO2_non_org_heavy_indus_share))
- lag_CO2_bio_tspt_share = lag(CO2_bio_tspt_share, default = first(CO2_bio_tspt_share))
- lag_CO2_non_org_tspt_share = lag(CO2_non_org_tspt_share, default = first(CO2_non_org_tspt_share))
- lag_pop_share = lag(pop_share, default = first(pop_share))
- lag_NPP_share = lag(NPP_share, default = first(NPP_share))

Remove the grouping to return to an ungrouped dataframe.

Arrange the dataset by the original_order column.

Exclude the column original_order.

Save the resulting dataframe as new_predictors_put_in_model_0_5deg.RData.

6. **Finalize 0.5-degree Model Training Dataset:**

Read the training_iso_0_5deg_cell_GCP.RData file obtained in Section 18 Step 7 using the load() function.

Group the dataset by the iso and year columns.

Create a new column GCP_share_0_5deg, where each value is calculated as:

- If GCP_0_5deg equals 0, set the value to 0.
- Otherwise, calculate it as GCP_0_5deg/sum(GCP_0_5deg) within each group.

Remove the grouping to return to an ungrouped dataframe.

Apply left_join to combine the current dataframe with the following dataframe, ensuring that the current dataframe is the base file:

- Read the new_predictors_put_in_model_0_5deg.RData file using load.
- Convert it to a dataframe using as.data.frame.
- Remove the geom column.
- Create a new column iso_change, where for rows with USA in the iso column, it concatenates USA_ with the first two characters of the id column. For all other rows, the iso_change column will retain the value from the id column.
- Select only the columns cell_id, subcell_id, iso_change, year, mean_rug, national_gdpc, pop, and any columns containing the substring share.
- Rename the column iso_change to iso.

Omit any rows containing NA values.

Save the resulting file as new_predict_data_complete_0_5deg.RData.

7. **0.25-degree Predictors**

Population Share:

- Read the land_pop_extracted_region_level_0_25deg.RData file obtained in Section 20 Step 3 using load.
- Group the dataset by the id and year columns.
- Create a new column pop_share, where each value is the proportion of the pop value to the total pop within the group, calculated as pop/sum(pop).
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as pop_put_in_model

CO2 Emissions From Biofuels Share:

- Read the CO2_bio_full_0_25deg.RData file obtained in Section 26 Step 4 using load.
- Create a new column CO2_bio_heavy_indus with values equal to the sum of corresponding values of columns CO2_bio_fuel_exploitation, CO2_bio_oil_refine_transf, and CO2_bio_power_industry.

- Create a new column `CO2_bio_tspt` with values equal to the sum of corresponding values of columns `CO2_bio_road_transp` and `CO2_bio_shipping`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `CO2_bio_manuf_conbust_share`, where each value is calculated as:
  - If `CO2_bio_combustion_for_manufacturing` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_bio_combustion_for_manufacturing/sum(CO2_bio_combustion_for_manufacturing)` within the group.
- Create a new column `CO2_bio_heavy_indus_share`, where each value is calculated as:
  - If `CO2_bio_heavy_indus` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_bio_heavy_indus/sum(CO2_bio_heavy_indus)` within the group.
- Create a new column `CO2_bio_tspt_share`, where each value is calculated as:
  - If `CO2_bio_tspt` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_bio_tspt/sum(CO2_bio_tspt)` within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as `CO2_bio_put_in_model`

CO2 Emissions From Non-organic Fuels Share:

- Read the `CO2_non_org_full_0_25deg.RData` file obtained in Section 27 Step 4 using `load`.
- Create a new column `CO2_non_org_heavy_indus` with values equal to the sum of corresponding values of columns `CO2_non_org_fuel_exploitation`, `CO2_non_org_iron_steel`, `CO2_non_org_non_ferrous_metal` `CO2_non_org_non_metallic_mineral`, `CO2_non_org_oil_refine_transf`, and `CO2_non_org_power_industry`.
- Create a new column `CO2_non_org_tspt` with values equal to the sum of corresponding values of columns `CO2_non_org_road_transp` and `CO2_non_org_shipping`.
- Group the dataset by the `id` and `year` columns.
- Create a new column `CO2_non_org_manuf_conbust_share`, where each value is calculated as:
  - If `CO2_non_org_combustion_for_manufacturing` equals `0`, the value is set to `0`.
  - Otherwise, calculate it as `CO2_non_org_combustion_for_manufacturing/sum(CO2_non_org_combustion_for_manufacturing)` within the group.
- Create a new column `CO2_non_org_heavy_indus_share`, where each value is calculated as:

  - If CO2_non_org_heavy_indus equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_non_org_heavy_indus/sum(CO2_non_org_heavy_indus) within the group.
- Create a new column CO2_non_org_tspt_share, where each value is calculated as:
  - If CO2_non_org_tspt equals 0, the value is set to 0.
  - Otherwise, calculate it as CO2_non_org_tspt/sum(CO2_non_org_tspt) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as CO2_non_org_put_in_model

NPP Share:

- Read the NPP_full_0_25deg.RData file obtained in Section 22 Step 4 using load.
- Group the dataset by the id and year columns.
- Create a new column NPP_share, where each value is the proportion of the NPP value to the total NPP within the group, calculated as NPP/sum(NPP)
- Remove the grouping to return to an ungrouped dataframe.
- Refer to the resulting dataframe as NPP_put_in_model

Urban NTL Share:

- Read the NTL_urban_full_0_25deg.RData file obtained in Section 28 Step 7 using load.
- Group the dataset by the id and year columns.
- Create a new column NTL_urban_snow_covered_period_share, where each value is calculated as:
  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_urban_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_urban_snow_covered_period
- Rename NTL_snow_free_period column to NTL_urban_snow_free_period
- Exclude the column land_type
- Refer to the resulting dataframe as NTL_urban_put_in_model

Cropland NTL Share:

- Read the NTL_cropland_full_0_25deg.RData file obtained in Section 28 Step 7 using load.
- Group the dataset by the subcell_id_0_25, subcell_id, cell_id, iso, id, and year columns
- Summarize the dataset by updating the NTL_snow_covered_period column values to be the sum of NTL_snow_covered_period values within each group, and similarly update the NTL_snow_free_period column. Remove grouping use .groups = "drop".
- Group the dataset by the id and year columns.
- Create a new column NTL_cropland_snow_covered_period_share, where each value is calculated as:
  - If NTL_snow_covered_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_covered_period/sum(NTL_snow_covered_period) within the group.
- Create a new column NTL_cropland_snow_free_period_share, where each value is calculated as:
  - If NTL_snow_free_period equals 0, the value is set to 0.
  - Otherwise, calculate it as NTL_snow_free_period/sum(NTL_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Rename NTL_snow_covered_period column to NTL_cropland_snow_covered_period
- Rename NTL_snow_free_period column to NTL_cropland_snow_free_period
- Refer to the resulting dataframe as NTL_cropland_put_in_model

Other Lands NTL Share:

- Read the NTL_full_0_25deg.RData file obtained in Section 24 Step 5 using load.
- Rename the NTL_snow_covered_period column to NTL_full_snow_covered_period.
- Rename the NTL_snow_free_period column to NTL_full_snow_free_period.
- Select only the rows where the year column values are between 2012 and 2021 inclusive.
- Apply left_join to combine the current dataframe with NTL_urban_put_in_model, adding the following columns: NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share. Ensure the current dataframe is the base file.

- Replace any NA values with 0 for the columns NTL_urban_snow_covered_period, NTL_urban_snow_free_period, NTL_urban_snow_covered_period_share, and NTL_urban_snow_free_period_share.
- Apply left_join again to combine the current dataframe with NTL_cropland_put_in_model, adding the columns: NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Replace any NA values with 0 for the columns NTL_cropland_snow_covered_period, NTL_cropland_snow_free_period, NTL_cropland_snow_covered_period_share, and NTL_cropland_snow_free_period_share.
- Create a new column NTL_other_snow_covered_period with values equal to NTL_full_snow_covered_period - NTL_urban_snow_covered_period - NTL_cropland_snow_covered_period.
- Similarly, create a new column NTL_other_snow_free_period with values equal to NTL_full_snow_free_period - NTL_urban_snow_free_period - NTL_cropland_snow_free_period.
- Group the dataset by id and year.
- Create a new column NTL_other_snow_covered_period_share, where each value is calculated as:
  - If NTL_other_snow_covered_period equals 0, set the value to 0.
  - Otherwise, calculate it as NTL_other_snow_covered_period/sum(NTL_other_snow_covered_period) within the group.
- Create a new column NTL_other_snow_free_period_share, where each value is calculated as:
  - If NTL_other_snow_free_period equals 0, set the value to 0.
  - Otherwise, calculate it as NTL_other_snow_free_period/sum(NTL_other_snow_free_period) within the group.
- Remove the grouping to return to an ungrouped dataframe.
- Select only the following columns: subcell_id_0_25, cell_id, subcell_id, iso, id, year, NTL_urban_snow_covered_period_share, NTL_urban_snow_free_period_share, NTL_cropland_snow_covered_period_share, NTL_cropland_snow_free_period_share, NTL_other_snow_covered_period_share, NTL_other_snow_free_period_share.
- Convert the resulting dataframe to a regular dataframe using as.data.frame.
- Refer to the resulting dataframe as NTL_put_in_model.

Share of Different Land Use Types:

- Read the file lc_full_0_25deg.RData obtained in Section 21 Step 4 using load.
- Create a new column forest_full with values equal to the sum of the corresponding row values from the columns open_forest and dense_forest.

- Create a new column cropland_full with values equal to the sum of the corresponding row values from the columns cropland, forest_cropland, and herbaceous_cropland.
- Exclude the columns open_forest, dense_forest, cropland, forest_cropland, and herbaceous_cropland.
- Group the dataset by id and year.
- Create a new column barren_share, where each value is the proportion of the barren value to the total barren within the group, calculated as barren/sum(barren)
- Create a new column snow_ice_share, where each value is the proportion of the snow_ice value to the total snow_ice within the group, calculated as snow_ice/sum(snow_ice)
- Create a new column water_share, where each value is the proportion of the water value to the total water within the group, calculated as water/sum(water)
- Create a new column urban_share, where each value is the proportion of the urban value to the total urban within the group, calculated as urban/sum(urban)
- Create a new column forest_share, where each value is the proportion of the forest_full value to the total forest_full within the group, calculated as forest_full/sum(forest_full)
- Create a new column herbaceous_share, where each value is the proportion of the herbaceous value to the total herbaceous within the group, calculated as herbaceous/sum(herbaceous)
- Create a new column cropland_share, where each value is the proportion of the cropland_full value to the total cropland_full within the group, calculated as cropland_full/sum(cropland_full)
- Create a new column shrub_share, where each value is the proportion of the shrub value to the total shrub within the group, calculated as shrub/sum(shrub)
- Remove the grouping to return to an ungrouped dataframe.
- Replace any NA values to 0
- Refer to the resulting dataframe as lc_put_in_model.

Ruggedness:

- Read the mean_ruggedness_0_25deg.csv file obtained in Section 25 Step 4 using read.csv.
- Convert the cell_id column values to characters using as.character.

GDP per Capita:

- Read the national_gdpc_const_2017_USD.csv file obtained in Section 14 Step 5 using read.csv.
- Add a new row with iso value as Ala, Country value as Alaska, and all other column values the same as the USA.

8. **Finalize 0.25-degree Model Prediction Dataset:**

Start with the file pop_put_in_model, considering it as the base file.

Combine it with the following dataframes one by one using left_join with the argument by = join_by(cell_id, subcell_id, subcell_id_0_25, id, iso, year): CO2_bio_put_in_model, CO2_non_org_put_in_model, NPP_put_in_model, NTL_put_in_model, lc_put_in_model.

Combine it with the rug_put_in_model file using left_join, but with the argument by = join_by(cell_id, subcell_id, subcell_id_0_25, id, iso).

Replace any NA values with 0.

Select only rows where year values are between 2012 and 2021, inclusive.

Combine the current dataframe with national_GDPC using left_join. Again, ensure that the current dataframe is the base file.

Select only the columns: id, iso, cell_id, subcell_id, subcell_id_0_25, year, mean_rug, national_gdpc, pop, and any columns whose names contain "share".

Create a new column original_order with values representing the row number using row_number().

Arrange the dataset in ascending order based on the columns subcell_id, subcell_id_0_25, cell_id, id, iso, and year.

Group the dataset by the columns subcell_id, subcell_id_0_25, cell_id, id, and iso.

Create new columns representing the previous year's values by using the lag() function. If no previous year's data exists, use the value from the first available year (2012) within the group. The following new columns are created:

- lag_NTL_urban_share = lag(NTL_urban_snow_free_period_share, default = first(NTL_urban_snow_free_period_share))
- lag_urban_share = lag(urban_share, default = first(urban_share))
- lag_cropland_share = lag(cropland_share, default = first(cropland_share))
- lag_NTL_other_share = lag(NTL_other_snow_free_period_share, default = first(NTL_other_snow_free_period_share))
- lag_NTL_cropland_share = lag(NTL_cropland_snow_free_period_share, default = first(NTL_cropland_snow_free_period_share))
- lag_CO2_bio_mc_share = lag(CO2_bio_manuf_conbust_share, default = first(CO2_bio_manuf_conbust_share))
- lag_CO2_nonorg_mc_share = lag(CO2_non_org_manuf_conbust_share, default = first(CO2_non_org_manuf_conbust_share))
- lag_CO2_bio_heavy_indus_share = lag(CO2_bio_heavy_indus_share, default = first(CO2_bio_heavy_indus_share))
- lag_CO2_non_org_heavy_indus_share = lag(CO2_non_org_heavy_indus_share, default = first(CO2_non_org_heavy_indus_share))

- lag_CO2_bio_tspt_share = lag(CO2_bio_tspt_share, default = first(CO2_bio_tspt_share))
- lag_CO2_non_org_tspt_share = lag(CO2_non_org_tspt_share, default = first(CO2_non_org_tspt_share))
- lag_pop_share = lag(pop_share, default = first(pop_share))
- lag_NPP_share = lag(NPP_share, default = first(NPP_share))

Remove the grouping to return to an ungrouped dataframe.

Arrange the dataset by the original_order column.

Exclude the column original_order.

Save the resulting dataframe as new_predictors_put_in_model_0_25deg.RData.

9. **Finalize 0.5-degree Model Training Dataset:**

Read the training_iso_0_25deg_cell_GCP.RData file obtained in Section 18 Step 7 using the load() function.

Group the dataset by the iso and year columns.

Create a new column GCP_share_0_25deg, where each value is calculated as:

- If GCP_0_25deg equals 0, set the value to 0.
- Otherwise, calculate it as GCP_0_25deg/sum(GCP_0_25deg) within each group.

Remove the grouping to return to an ungrouped dataframe.

Apply left_join to combine the current dataframe with the following dataframe, ensuring that the current dataframe is the base file:

- Read the new_predictors_put_in_model_0_25deg.RData file using load.
- Convert it to a dataframe using as.data.frame.
- Remove the geom column.
- Create a new column iso_change, where for rows with USA in the iso column, it concatenates USA_ with the first two characters of the id column. For all other rows, the iso_change column will retain the value from the id column.
- Select only the columns cell_id, subcell_id, subcell_id_0_25, iso_change, year, mean_rug, national_gdpc, pop, and any columns containing the substring share.
- Rename the column iso_change to iso.

Omit any rows containing NA values.

Save the resulting file as new_predict_data_complete_0_25deg.RData.

30 Training, Validation, and Test Datasets

In this section, we divide the data into training, validation, and test datasets, but we do not use them for the final model training. Instead, we train the model on all available data for the final models due to two reasons: 1) Group K-fold Cross-Validation is applied to tune the hyperparameters, and 2) China is treated as entirely non-training data, with additional tests conducted to assess out-of-sample performance. The separation into training, validation, and test datasets is only to replicate the steps in the R scripts and is not necessary for the final model training.

1. **1-degree Train, Validation, and test datasets:**

   Define Random Countries: Define the list `random_countries` by randomly selecting two countries from the following list of country iso codes using a fixed seed for reproducibility `set.seed(12345678)`: "AUT", "BEL", "BGR", "CHE", "CZE", "DEU", "DNK", "ESP", "EST", "FIN", "FRA", "GBR", "GRC", "HRV", "HUN", "ITA", "JPN", "KOR", "LTU", "LVA", "NLD", "NOR", "NZL", "POL", "PRT", "ROU", "SVK", "SVN", "SWE", "TUR".

   Define Developed Countries Group: Define the list `developed_group` as the set of countries represented by the iso codes: "AUT", "BEL", "BGR", "CHE", "CZE", "DEU", "DNK", "ESP", "EST", "FIN", "FRA", "GBR", "GRC", "HRV", "HUN", "ITA", "JPN", "KOR", "LTU", "LVA", "NLD", "NOR", "NZL", "POL", "PRT", "ROU", "SVK", "SVN", "SWE", "TUR", and any unique `iso` values that begin with "US".

   Load Data for Prediction: Read the `new_predict_data_complete_1deg.RData` file obtained in Section 29 Step 3 using `load`.

   Define Developing Countries Group: Define the list `developing_group` as the set of unique `iso` values from the `predict_data_complete_1deg` dataset that are not present in the `developed_group`.

   Define Random Developing Countries: Define the list `random_country_developing` by randomly selecting two countries from the `developing_group` list using a fixed seed for reproducibility `set.seed(12345678)`.

   Developed Countries in Training Dataset:

   - From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developed_group`.
   - Select rows where `year` values are either less than or equal to 2018 or equal to 2021.
   - Remove rows where `iso` is present in the `random_countries` list.
   - Refer to the resulting file as `training_df_developed`.

   Developed Countries Validation Dataset 1:

   - From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developed_group`.
   - Remove rows where `iso` is in the `random_countries` list.
   - Select rows where `year` is 2019.
   - Refer to the resulting file as `validation_df_year_developed`.

Developed Countries Validation Dataset 2:

- From the `new_predict_data_complete_1deg.RData` file, select rows where `iso` matches the first value in the `random_countries` list.
- Refer to the resulting file as `validation_df_iso_developed`.

Developed Countries Testing Dataset 1:

- From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developed_group`.
- Remove rows where `iso` is in the `random_countries` list.
- Select rows where `year` is 2020.
- Refer to the resulting file as `testing_df_year_developed`.

Developed Countries Testing Dataset 2:

- From the `new_predict_data_complete_1deg.RData` file, select rows where `iso` matches the second value in the `random_countries` list.
- Refer to the resulting file as `testing_df_iso_developed`.

Developing Countries in Training Dataset:

- From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developing_group`.
- Select rows where `year` is not equal to 2018 or 2019.
- Remove rows where `iso` is in the `random_country_developing` list.
- Refer to the resulting file as `training_df_developing`.

Developing Countries Validation Dataset 1:

- From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developing_group`.
- Remove rows where `iso` is in the `random_country_developing` list.
- Select rows where `year` is 2018.
- Refer to the resulting file as `validation_df_year_developing`.

Developing Countries Validation Dataset 2:

- From the `new_predict_data_complete_1deg.RData` file, select rows where `iso` matches the first value in the `random_country_developing` list.
- Refer to the resulting file as `validation_df_iso_developing`.

Developing Countries Testing Dataset 1:

- From the `new_predict_data_complete_1deg.RData` file, select rows with `iso` values from the `developing_group`.
- Remove rows where `iso` is in the `random_country_developing` list.
- Select rows where `year` is 2019.
- Refer to the resulting file as `testing_df_year_developing`.

Developing Countries Testing Dataset 2:

- From the new_predict_data_complete_1deg.RData file, select rows where iso matches the second value in the random_country_developing list.
- Refer to the resulting file as testing_df_iso_developing.

Finalize the 1-degree Training Dataset:

- Use bind_rows to combine training_df_developed and training_df_developing files.
- Save the resulting file as new_data_train_1deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 1-degree Validation Dataset 1:

- Use bind_rows to combine validation_df_year_developed and validation_df_year_developing files.
- Save the resulting file as new_data_valid_year_1deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 1-degree Validation Dataset 2:

- Use bind_rows to combine validation_df_iso_developed and validation_df_iso_developing files.
- Save the resulting file as new_data_valid_iso_1deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 1-degree Testing Dataset 1:

- Use bind_rows to combine testing_df_year_developed and testing_df_year_developing files.
- Save the resulting file as new_data_test_year_1deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 1-degree Testing Dataset 2:

- Use bind_rows to combine testing_df_iso_developed and testing_df_iso_developing files.
- Save the resulting file as new_data_test_iso_1deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

2. **0.5-degree Train, Validation, and test datasets:**

Load Data for Prediction: Read the new_predict_data_complete_0_5deg.RData file obtained in Section 29 Step 6 using load.

Developed Countries in Training Dataset:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developed_group.
- Select rows where year values are either less than or equal to 2018 or equal to 2021.
- Remove rows where iso is present in the random_countries list.
- Refer to the resulting file as training_df_developed.

Developed Countries Validation Dataset 1:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developed_group.
- Remove rows where iso is in the random_countries list.
- Select rows where year is 2019.
- Refer to the resulting file as validation_df_year_developed.

Developed Countries Validation Dataset 2:

- From the new_predict_data_complete_0_5deg.RData file, select rows where iso matches the first value in the random_countries list.
- Refer to the resulting file as validation_df_iso_developed.

Developed Countries Testing Dataset 1:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developed_group.
- Remove rows where iso is in the random_countries list.
- Select rows where year is 2020.
- Refer to the resulting file as testing_df_year_developed.

Developed Countries Testing Dataset 2:

- From the new_predict_data_complete_0_5deg.RData file, select rows where iso matches the second value in the random_countries list.
- Refer to the resulting file as testing_df_iso_developed.

Developing Countries in Training Dataset:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developing_group.
- Select rows where year is not equal to 2018 or 2019.
- Remove rows where iso is in the random_country_developing list.
- Refer to the resulting file as training_df_developing.

Developing Countries Validation Dataset 1:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developing_group.
- Remove rows where iso is in the random_country_developing list.
- Select rows where year is 2018.
- Refer to the resulting file as validation_df_year_developing.

Developing Countries Validation Dataset 2:

- From the new_predict_data_complete_0_5deg.RData file, select rows where iso matches the first value in the random_country_developing list.
- Refer to the resulting file as validation_df_iso_developing.

Developing Countries Testing Dataset 1:

- From the new_predict_data_complete_0_5deg.RData file, select rows with iso values from the developing_group.
- Remove rows where iso is in the random_country_developing list.
- Select rows where year is 2019.
- Refer to the resulting file as testing_df_year_developing.

Developing Countries Testing Dataset 2:

- From the new_predict_data_complete_0_5deg.RData file, select rows where iso matches the second value in the random_country_developing list.
- Refer to the resulting file as testing_df_iso_developing.

Finalize the 0.5-degree Training Dataset:

- Use bind_rows to combine training_df_developed and training_df_developing files.
- Save the resulting file as new_data_train_0_5deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.5-degree Validation Dataset 1:

- Use bind_rows to combine validation_df_year_developed and validation_df_year_developing files.
- Save the resulting file as new_data_valid_year_0_5deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.5-degree Validation Dataset 2:

- Use bind_rows to combine validation_df_iso_developed and validation_df_iso_developing files.
- Save the resulting file as new_data_valid_iso_0_5deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.5-degree Testing Dataset 1:

- Use bind_rows to combine testing_df_year_developed and testing_df_year_developing files.
- Save the resulting file as new_data_test_year_0_5deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.5-degree Testing Dataset 2:

- Use bind_rows to combine testing_df_iso_developed and testing_df_iso_developing files.
- Save the resulting file as new_data_test_iso_0_5deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

3. **0.25-degree Train, Validation, and test datasets:**

Load Data for Prediction: Read the new_predict_data_complete_0_25deg.RData file obtained in Section 29 Step 9 using load.

Developed Countries in Training Dataset:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developed_group.
- Select rows where year values are either less than or equal to 2018 or equal to 2021.
- Remove rows where iso is present in the random_countries list.
- Refer to the resulting file as training_df_developed.

Developed Countries Validation Dataset 1:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developed_group.
- Remove rows where iso is in the random_countries list.
- Select rows where year is 2019.
- Refer to the resulting file as validation_df_year_developed.

Developed Countries Validation Dataset 2:

- From the new_predict_data_complete_0_25deg.RData file, select rows where iso matches the first value in the random_countries list.
- Refer to the resulting file as validation_df_iso_developed.

Developed Countries Testing Dataset 1:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developed_group.
- Remove rows where iso is in the random_countries list.
- Select rows where year is 2020.
- Refer to the resulting file as testing_df_year_developed.

Developed Countries Testing Dataset 2:

- From the new_predict_data_complete_0_25deg.RData file, select rows where iso matches the second value in the random_countries list.
- Refer to the resulting file as testing_df_iso_developed.

Developing Countries in Training Dataset:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developing_group.
- Select rows where year is not equal to 2018 or 2019.
- Remove rows where iso is in the random_country_developing list.
- Refer to the resulting file as training_df_developing.

Developing Countries Validation Dataset 1:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developing_group.
- Remove rows where iso is in the random_country_developing list.
- Select rows where year is 2018.
- Refer to the resulting file as validation_df_year_developing.

Developing Countries Validation Dataset 2:

- From the new_predict_data_complete_0_25deg.RData file, select rows where iso matches the first value in the random_country_developing list.
- Refer to the resulting file as validation_df_iso_developing.

Developing Countries Testing Dataset 1:

- From the new_predict_data_complete_0_25deg.RData file, select rows with iso values from the developing_group.
- Remove rows where iso is in the random_country_developing list.
- Select rows where year is 2019.
- Refer to the resulting file as testing_df_year_developing.

Developing Countries Testing Dataset 2:

- From the new_predict_data_complete_0_25deg.RData file, select rows where iso matches the second value in the random_country_developing list.
- Refer to the resulting file as testing_df_iso_developing.

Finalize the 0.25-degree Training Dataset:

- Use bind_rows to combine training_df_developed and training_df_developing files.
- Save the resulting file as new_data_train_0_25deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.25-degree Validation Dataset 1:

- Use bind_rows to combine validation_df_year_developed and validation_df_year_developing files.
- Save the resulting file as new_data_valid_year_0_25deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.25-degree Validation Dataset 2:

- Use bind_rows to combine validation_df_iso_developed and validation_df_iso_developing files.
- Save the resulting file as new_data_valid_iso_0_25deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.25-degree Testing Dataset 1:

- Use bind_rows to combine testing_df_year_developed and testing_df_year_developing files.
- Save the resulting file as new_data_test_year_0_25deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

Finalize the 0.25-degree Testing Dataset 2:

- Use bind_rows to combine testing_df_iso_developed and testing_df_iso_developing files.
- Save the resulting file as new_data_test_iso_0_25deg.csv and ensure the output excludes row names by setting the parameter row.names = FALSE.

31 Train the 1-degree Random Forest Model: Include Years 2012 to 2021

In this section, we build the algorithm to train the 1-degree models.

1. **Calculate the Share of Cells Representing Developed and Developing Countries in the Actual World**

   Read the `new_predictors_put_in_model_1deg.RData` file using the `load()` function.

   Adjust the `iso` column values: if the original `iso` value is `Ala`, change it to `USA`, and leave all other values unchanged.

   Convert the dataset into a data frame using `as.data.frame`.

   Exclude the `geom` column.

   Select unique combinations of `cell_id` and `iso` columns using `distinct(cell_id, iso)` .

   Create a new column `is_developing` with a value of `1` if the corresponding `iso` value is found in the `developing` column from the `list_developed_developing.xlsx` file located in the `step4_train_and_tune_log_change/inputs` folder. For all other rows, assign a value of `0`.

   Group the dataset by the `is_developing` column, then use the `summarise()` function to create a new column `count`, which calculates the number of rows for each group.

   Create a new column `share` by dividing the `count` column by the total sum of `count`.

   Refer to the resulting dataset as `real_share`.

2. **Full Training Dataset Put in the Model**

   Define the `developing_group` list as: "CHL", "COL", "IDN", "KGZ", "PER", "PHL", "ALB", "BIH", "BLR", "MOZ", "SRB", "UZB", "VNM", "KEN", "LKA", "THA", "ECU".

   Use `read.csv` to read the following files: `new_data_train_1deg.csv`, `new_data_valid_year_1deg.csv`, `new_data_valid_iso_1deg.csv`, `new_data_test_year_1deg.csv`, and `new_data_test_iso_1deg.csv`, then combine them using `bind_rows`.

   Create a new column, `unit_gdp_af_sum_rescl`, with values equal to the corresponding values from the column `state_total_GDP`.

   Create a new column, `original_order`, with values equal to the row number, using the `row_number()` function.

   Create a new column, `is_developing`, with a value of `1` if the corresponding `iso` value belongs to the `developing_group` list; otherwise, set the value to `0`.

   Group the dataset by the `is_developing` column, and create a new column, `sample_count`, with values equal to the number of rows in each group, using the `n()` function.

Ungroup the dataset.

Create a new column, sample_share, with values equal to sample_count / n(), which represents the proportion of samples in each group.

Use left_join to combine the current dataframe with the real_share dataframe obtained in the previous step, ensuring that the current dataframe is the base file.

Create a new column, normalized_weight, with values calculated as share / sample_share.

Exclude the columns sample_count, sample_share, count, and share.

Create a new column, import_weight, with values generated by applying the importance_weights(normalized_weight) function. This will signal to the random forest model that the normalized_weight column is to be used as a weight column.

Arrange the dataset in ascending order based on the original_order column created earlier.

Exclude the original_order column.

Refer the resulting dataframe as data_full.

3. **Training Algorithm:**

Define Cross-Validation:

- Set the random seed to 1234567 for reproducibility.
- Define the cross-validation splits using the function group_vfold_cv().
- Group by the iso column to ensure that data from the same country appears in only one fold. This is done by set group = "iso" in group_vfold_cv
- Set the number of folds to 5 by passing v = 5 to group_vfold_cv.
- Store the cross-validation splits in the folds variable.

Define the Random Forest Training Function:

- Define the target variable as GCP_share_1deg.
- Define the predictor variables as a list of columns: "pop_share", "CO2_bio_manuf_conbust_share", "CO2_bio_heavy_indus_share", "CO2_bio_tspt_share", "CO2_non_org_manuf_conbust_share", "CO2_non_org_heavy_indus_share", "CO2_non_org_tspt_share", "NPP_share", "NTL_urban_snow_free_period_share", "NTL_cropland_snow_free_period_share", "NTL_other_snow_free_period_share", "snow_ice_share", "water_share", "urban_share", "forest_share", "cropland_share", "mean_rug", "national_gdpc", "lag_NTL_urban_share", "lag_urban_share", "lag_cropland_share", "lag_NTL_other_share", "lag_NTL_cropland_share", "lag_CO2_bio_mc_share", "lag_CO2_nonorg_mc_share", "lag_CO2_bio_heavy_indus_share", "lag_CO2_non_org_heavy_indus_share", "lag_CO2_bio_tspt_share", "lag_CO2_non_org_tspt_share", "lag_pop_share", "lag_NPP_share", "import_weight"

- Create a formula for the model use: formula = as.formula(paste(target_var, "~", paste(predictor_vars, collapse = " + "))).

Defining Reasonable Hyperparameter Combinations:

- Print a message indicating that hyperparameter tuning is starting.
- **Define a Grid of Hyperparameters to Tune:** Due to memory limitations, it is not feasible to try every single possible combination of `mtry`, `trees`, and `min_n`, so we increase `mtry` and `min_n` with an increment of 3. As long as the values lie within a reasonable range, they should be sufficient. Avoid using values that are either too low or too high. Define the following hyperparameter ranges:
  - `mtry` (the number of variables to sample for each split) with possible values: {7, 10, 13, 16}
  - `trees` (the number of trees in the forest) with possible values: {500, 750, 1000}
  - `min_n` (the minimum node size) with possible values: {10, 13, 16, 19}
- Use `expand.grid` to define combinations of `mtry`, `trees`, and `min_n`.
- Define a helper function `tune_hyperparameters()` to tune the hyperparameters.
- In the `tune_hyperparameters()` function, loop through each combination of hyperparameters, perform cross-validation, and calculate the performance metrics (MSE, R2) which will be defined below.
- Then save the best results based on the weighted R2 of annual changes (`wgt_chan_r2`) which will be defined below.

Calculate Performance Metrics for Each Combination of Hyperparameters:

- Initialize empty vectors for storing model evaluation metrics, including:
  - `mse_developed`, `mse_developing`, and `mse_all` for storing mean squared error (MSE).
  - `r2_developed`, `r2_developing`, and `r2_all` for storing R-squared values.
  - `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all` for storing annual changes R-squared values.
- Define the function `calculate_r2()` to compute and return the R-squared value between the true and predicted values.
  - The function takes two inputs: `true_values` and `predicted_values`.
  - First, filter out any zero or negative values from both the `true_values` and `predicted_values` to ensure valid data.
  - It then takes the natural logarithm of the valid values of `true_values` and `predicted_values` (stored as `true_log` and `predicted_log` respectively).
  - The R-squared value is then calculated using the formula:

$$1 - \frac{\sum(\text{true\_log} - \text{predicted\_log})^2}{\sum(\text{true\_log} - \text{mean}(\text{true\_log}))^2}$$

- Define the function calculate_mse() to compute and return the Mean Squared Error (MSE) between the true and predicted values.
  - The function takes two inputs: true_values and predicted_values.
  - The MSE value is then computed as the average of these squared differences, using the formula:

$$\frac{1}{n}\sum_{i=1}^{n}(\text{true\_values}_i - \text{predicted\_values}_i)^2$$

    where n is the number of data points.
- Define the function calculate_chan_r2() to compute and return the R-squared value of the annual changes for evaluating the model's performance on time series data.
  - The function takes four inputs: true_values, true_last, predicted_values, and predicted_last.
  - First, filter out any zero or negative values from all four inputs to ensure valid data.
  - Then, take the natural logarithm of the valid values of true_values, true_last, predicted_values, and predicted_last, which are stored as true_log, true_last_log, pred_log, and pred_last_log, respectively.
  - Calculate the differences in the logs: true_log_diff = true_log - true_last_log and pred_log_diff = pred_log - pred_last_log.
  - The R-squared value of the annual changes is calculated using the formula:

$$1 - \frac{\sum(\text{true\_log\_diff} - \text{pred\_log\_diff})^2}{\sum(\text{true\_log\_diff} - \text{mean}(\text{true\_log\_diff}))^2}$$

- Recall that the training countries are divided into five groups, creating five folds using the group_vfold_cv() function, stored in the folds file. Each fold consists of four groups used as the training set and one group used as the testing set. For each of the five folds, perform the following steps:
  - **Extract the Training and Testing Sets:** Extract the training and testing sets for the current fold using the analysis() and assessment() functions, respectively. To do this, use the following code: analysis <− as.data.frame(analysis(current_fold$splits [[i]])) and assessment <− as .data.frame(assessment(current_fold$splits[[i]]))
  - **Fit the Random Forest Model:**
    - **Define the Random Forest Model:** Use the rand_forest() function from the parsnip package with the parameters mtry, trees, and min_n, setting them to the current combination of hyperparameters in the loop.
    - **Set the Engine:** Use the set_engine() function from the parsnip package to specify the engine. Set the parameters as verbose = FALSE and seed = 1234567 to ensure reproducibility and suppress verbose output.

    * **Specify the Mode of the Model:** Use the set_mode("regression") function to indicate that this model is a regression model.
    * **Fit the Model:** Fit the model using the formula defined earlier, with the data argument set to the training set and weights argument set to the import_weight column from the training data.
    * **Complete Example Code:**
      ```
      fit <- rand_forest(mtry = params$mtry, trees = params$trees, min_n
       = params$min_n)%>%
      set_engine("ranger", verbose = FALSE, seed = 1234567)%>%
      set_mode("regression")%>%
      fit (formula, data = analysis, weights = import_weight)
      ```
- **Predict on the Testing Set:** Use the predict command from the stats package to predict on the testing set. The code is: preds <- as. data.frame(predict(fit, assessment)), where fit refers to the fitted random forest model described above, and assessment is the testing set obtained for the current fold.
- **Add the Prediction to the Testing Set:**
    * Begin with the assessment file.
    * Create a new column named pred_GCP_share_1deg and populate it with the predictions from the preds file.
    * Adjust the values in the pred_GCP_share_1deg column: set the value to 0 where the pop_share column equals 0; otherwise, retain the original values.
    * Refer to the updated file as assessment_with_preds.
    * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
- **Calculate MSE:** Use the calculate_mse() function defined earlier to compute the mean squared error (MSE) for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_share_1deg column. For the predicted_values argument, use the pred_GCP_share_1deg column. Store the computed MSE values in the initialized empty vectors defined earlier: mse_developed, mse_developing, and mse_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
- **Adjust the assessment_with_preds File:**
    * Start with the assessment_with_preds file obtained in the previous steps.
    * Group the dataset by the iso and year columns.
    * Create a new column, pred_GCP_share_1deg_rescaled, with values calculated as pred_GCP_share_1deg / sum(pred_GCP_share_1deg) within each group.

        * Ungroup the dataframe.
        * Create a new column, pred_GCP_1deg, with values calculated as pred_GCP_share_1deg_rescaled * state_total_GDP.
        * Refer to the updated file as the new assessment_with_preds file.
        * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
    - **Calculate R2 for log(GDP):** Use the calculate_r2() function defined earlier to compute the R2 for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_1deg column. For the predicted_values argument, use the pred_GCP_1deg column. Store the computed R2 values in the initialized empty vectors defined earlier: r2_developed, r2_developing, and r2_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
    - **Adjust the assessment_with_preds File Again:**
        * Start with the updated assessment_with_preds file obtained in the previous steps.
        * Arrange the dataset in ascending order by the columns iso, cell_id, and year.
        * Group the dataset by the iso and cell_id columns.
        * Create a new column, prev_year_pred, with values calculated as: ifelse (year − 1 %in% year, pred_GCP_1deg[match(year − 1, year)], NA).
        * Create a new column, prev_year_true, with values calculated as: ifelse (year − 1 %in% year, GCP_1deg[match(year − 1, year)], NA).
        * Ungroup the dataframe.
        * Keep only rows where the prev_year_pred and prev_year_true columns values are not NA.
        * Refer to the updated file as the new assessment_with_preds file.
        * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
    - **Calculate R2 for log(GDP in t) - log(GDP in t-1):** Use the calculate_chan_r2() function defined earlier to compute the R2 for the three new files: developed, developing, and assessment_with_preds. For the true_values argument, use the GCP_1deg column. For the true_last argument, use the prev_year_true column. For the predicted_values argument, use the pred_GCP_1deg column. For the predicted_last argument, use the prev_year_pred column. Store the computed R2 values in the initialized empty vectors defined earlier: chan_r2_developed, chan_r2_developing, and chan_r2_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.

- Under the previous loop step, we now have the following vectors for the current combination of hyperparameters: `mse_developed`, `mse_developing`, `mse_all`, `r2_developed`, `r2_developing`, `r2_all`, `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all`. Each vector contains five values corresponding to the five folds.
- Calculate the single value `wgt_chan_r2` as the weighted average of `mean(chan_r2_developed)` and `mean(chan_r2_developing)`, using the respective shares from the `real_share` dataframe for developed (`is_developing == 0`) and developing (`is_developing == 1`) groups.
- Return the `wgt_chan_r2` value.

Select the Best Hyperparameters:

- After the above steps, we obtain `wgt_chan_r2` value for each combination of hyperparameters.
- Identify the combination of hyperparameters that yields the highest `wgt_chan_r2` value.

Fit the Final Model:

- **Define the Final Random Forest Model:**
  - Now we use the best combination of hyperparameters to fit the final Random Forest model.
  - Use the `rand_forest()` function with the best values for `mtry`, `trees`, and `min_n`.
  - Set the engine to `"ranger"` use `set_engine()` with the following additional configurations:
    * `importance = "impurity"` to compute feature importance.
    * `verbose = TRUE` to enable verbose output during training.
    * `num.threads = 20` to utilize parallel processing with 20 threads.
    * `seed = 1234567` to ensure reproducibility.
  - Set the mode of the model to `"regression"` using `set_mode()`
  - Refer to this as `rf_model_final`
- **Create the Workflow:**
  - Initialize a workflow object using `workflow()`.
  - Use the `recipe()` function to create the recipe for the Random Forest model by specifying the `formula` as generated in the previous steps and setting the training dataset to `data_full`. The code is `rf_recipe <- recipe(formula = formula, data = data_full)`.
  - Add the preprocessing recipe (`rf_recipe`) to the workflow using `add_recipe(rf_recipe)`.
  - Add the defined random forest model (`rf_model_final`) to the workflow using `add_model(rf_model_final)`.
  - Include case weights (`import_weight` column) in the workflow using `add_case_weights(import_weight)`.

  - Refer to this as rf_workflow_final.
- **Fit the Model:**
  - Use the fit() function to train the workflow rf_workflow_final on the complete dataset data_full.
  - Save the resulting fitted workflow as rf_model9_good_grid_search_1deg.RData. This is the final trained random forest model that will be used to predict global 1-degree cell GDP.

32 Train the 0.5-degree Random Forest Model: Include Years 2012 to 2021

In this section, we build the algorithm to train the 0.5-degree models.

1. **Calculate the Share of Cells Representing Developed and Developing Countries in the Actual World**

   Read the new_predictors_put_in_model_0_5deg.RData file using the load() function.

   Adjust the iso column values: if the original iso value is Ala, change it to USA, and leave all other values unchanged.

   Convert the dataset into a data frame using as.data.frame.

   Exclude the geom column.

   Select unique combinations of cell_id, subcell_id and iso columns using distinct(cell_id, subcell_id, iso).

   Create a new column is_developing with a value of 1 if the corresponding iso value is found in the developing column from the list_developed_developing.xlsx file located in the step4_train_and_tune_log_change/inputs folder. For all other rows, assign a value of 0.

   Group the dataset by the is_developing column, then use the summarise() function to create a new column count, which calculates the number of rows for each group.

   Create a new column share by dividing the count column by the total sum of count.

   Refer to the resulting dataset as real_share.

2. **Full Training Dataset Put in the Model**

   Define the developing_group list as: "CHL", "COL", "IDN", "KGZ", "PER", "PHL", "ALB", "BIH", "BLR", "MOZ", "SRB", "UZB", "VNM", "KEN", "LKA", "THA", "ECU".

   Use read.csv to read the following files: new_data_train_0_5deg.csv, new_data_valid_year_0_5deg.csv, new_data_valid_iso_0_5deg.csv, new_data_test_year_0_5deg.csv, and new_data_test_iso_0_5deg.csv, then combine them using bind_rows.

Create a new column, unit_gdp_af_sum_rescl, with values equal to the corresponding values from the column state_total_GDP.

Create a new column, original_order, with values equal to the row number, using the row_number() function.

Create a new column, is_developing, with a value of 1 if the corresponding iso value belongs to the developing_group list; otherwise, set the value to 0.

Group the dataset by the is_developing column, and create a new column, sample_count, with values equal to the number of rows in each group, using the n() function.

Ungroup the dataset.

Create a new column, sample_share, with values equal to sample_count / n(), which represents the proportion of samples in each group.

Use left_join to combine the current dataframe with the real_share dataframe obtained in the previous step, ensuring that the current dataframe is the base file.

Create a new column, normalized_weight, with values calculated as share / sample_share.

Exclude the columns sample_count, sample_share, count, and share.

Create a new column, import_weight, with values generated by applying the importance_weights(normalized_weight) function. This will signal to the random forest model that the normalized_weight column is to be used as a weight column.

Arrange the dataset in ascending order based on the original_order column created earlier.

Exclude the original_order column.

Refer the resulting dataframe as data_full.

3. **Training Algorithm:**

Define Cross-Validation:

- Set the random seed to 1234567 for reproducibility.
- Define the cross-validation splits using the function group_vfold_cv().
- Group by the iso column to ensure that data from the same country appears in only one fold. This is done by set group = "iso" in group_vfold_cv
- Set the number of folds to 5 by passing v = 5 to group_vfold_cv.
- Store the cross-validation splits in the folds variable.

Define the Random Forest Training Function:

- Define the target variable as GCP_share_0_5deg.
- Define the predictor variables as a list of columns: "pop_share", "CO2_bio_manuf_conbust_share", "CO2_bio_heavy_indus_share", "CO2_bio_tspt_share", "CO2_non_org_manuf_conbust_share", "CO2_non_org_heavy_indus_share", "CO2_non_org_tspt_share", "NPP_share", "NTL_urban_-

snow_free_period_share", "NTL_cropland_snow_free_period_share", "NTL_other_snow_free_period_share", "snow_ice_share", "water_share", "urban_share", "forest_share", "cropland_share", "mean_rug", "national_gdpc", "lag_NTL_urban_share", "lag_urban_share", "lag_cropland_share", "lag_NTL_other_share", "lag_NTL_cropland_share", "lag_CO2_bio_mc_share", "lag_CO2_nonorg_mc_share", "lag_CO2_bio_heavy_indus_share", "lag_CO2_non_org_heavy_indus_share", "lag_CO2_bio_tspt_share", "lag_CO2_non_org_tspt_share", "lag_pop_share", "lag_NPP_share", "import_weight"

- Create a formula for the model use: formula = as.formula(paste(target_var, "~", paste(predictor_vars, collapse = " + "))).

Defining Reasonable Hyperparameter Combinations:

- Print a message indicating that hyperparameter tuning is starting.
- **Define a Grid of Hyperparameters to Tune:** Due to memory limitations, it is not feasible to try every single possible combination of `mtry`, `trees`, and `min_n`, so we increase `mtry` and `min_n` with an increment of 3. As long as the values lie within a reasonable range, they should be sufficient. Avoid using values that are either too low or too high. Define the following hyperparameter ranges:
  - `mtry` (the number of variables to sample for each split) with possible values: {7, 10, 13, 16}
  - `trees` (the number of trees in the forest) with possible values: {750, 1000}
  - `min_n` (the minimum node size) with possible values: {10, 13, 16, 19}
- Use `expand.grid` to define combinations of `mtry`, `trees`, and `min_n`.
- Define a helper function `tune_hyperparameters()` to tune the hyperparameters.
- In the `tune_hyperparameters()` function, loop through each combination of hyperparameters, perform cross-validation, and calculate the performance metrics (MSE, R2) which will be defined below.
- Then save the best results based on the weighted R2 of annual changes (`wgt_chan_r2`) which will be defined below.

Calculate Performance Metrics for Each Combination of Hyperparameters:

- Initialize empty vectors for storing model evaluation metrics, including:
  - `mse_developed`, `mse_developing`, and `mse_all` for storing mean squared error (MSE).
  - `r2_developed`, `r2_developing`, and `r2_all` for storing R-squared values.
  - `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all` for storing annual changes R-squared values.
- Define the function `calculate_r2()` in the exact same way as it is defined in Section 31, Step 3.
- Define the function `calculate_mse()` in the exact same way as it is defined in Section 31, Step 3.

- Define the function calculate_chan_r2() in the exact same way as it is defined in Section 31, Step 3.
- Recall that the training countries are divided into five groups, creating five folds using the group_vfold_cv() function, stored in the folds file. Each fold consists of four groups used as the training set and one group used as the testing set. For each of the five folds, perform the following steps:
  - **Extract the Training and Testing Sets:** Extract the training and testing sets for the current fold using the analysis() and assessment() functions, respectively. To do this, use the following code: analysis <− as.data.frame(analysis(current_fold$splits[[i]])) and assessment <− as .data.frame(assessment(current_fold$splits[[i]]))
  - **Fit the Random Forest Model:**
    * **Define the Random Forest Model:** Use the rand_forest() function from the parsnip package with the parameters mtry, trees, and min_n, setting them to the current combination of hyperparameters in the loop.
    * **Set the Engine:** Use the set_engine() function from the parsnip package to specify the engine. Set the parameters as verbose = FALSE and seed = 1234567 to ensure reproducibility and suppress verbose output.
    * **Specify the Mode of the Model:** Use the set_mode("regression") function to indicate that this model is a regression model.
    * **Fit the Model:** Fit the model using the formula defined earlier, with the data argument set to the training set and weights argument set to the import_weight column from the training data.
    * **Complete Example Code:**
      ```
      fit <− rand_forest(mtry = params$mtry, trees = params$trees, min_n
       = params$min_n)%>%
      set_engine("ranger", verbose = FALSE, seed = 1234567)%>%
      set_mode("regression")%>%
       fit (formula, data = analysis, weights = import_weight)
      ```
  - **Predict on the Testing Set:** Use the predict command from the stats package to predict on the testing set. The code is: preds <− as. data.frame(predict(fit , assessment)), where fit refers to the fitted random forest model described above, and assessment is the testing set obtained for the current fold.
  - **Add the Prediction to the Testing Set:**
    * Begin with the assessment file.
    * Create a new column named pred_GCP_share_0_5deg and populate it with the predictions from the preds file.
    * Adjust the values in the pred_GCP_share_0_5deg column: set the value to 0 where the pop_share column equals 0; otherwise, retain the original values.
    * Refer to the updated file as assessment_with_preds.

* Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.

- **Calculate MSE:** Use the calculate_mse() function defined earlier to compute the mean squared error (MSE) for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_share_0_5deg column. For the predicted_values argument, use the pred_GCP_share_0_5deg column. Store the computed MSE values in the initialized empty vectors defined earlier: mse_developed, mse_developing, and mse_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
- **Adjust the assessment_with_preds File:**
  * Start with the assessment_with_preds file obtained in the previous steps.
  * Group the dataset by the iso and year columns.
  * Create a new column, pred_GCP_share_0_5deg_rescaled, with values calculated as pred_GCP_share_0_5deg / sum(pred_GCP_share_0_5deg) within each group.
  * Ungroup the dataframe.
  * Create a new column, pred_GCP_0_5deg, with values calculated as pred_GCP_share_0_5deg_rescaled * state_total_GDP.
  * Refer to the updated file as the new assessment_with_preds file.
  * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
- **Calculate R2 for log(GDP):** Use the calculate_r2() function defined earlier to compute the R2 for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_0_5deg column. For the predicted_values argument, use the pred_GCP_0_5deg column. Store the computed R2 values in the initialized empty vectors defined earlier: r2_developed, r2_developing, and r2_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
- **Adjust the assessment_with_preds File Again:**
  * Start with the updated assessment_with_preds file obtained in the previous steps.
  * Arrange the dataset in ascending order by the columns iso, cell_id, subcell_id, and year.
  * Group the dataset by the iso, cell_id, and subcell_id columns.
  * Create a new column, prev_year_pred, with values calculated as: ifelse (year − 1 %in% year, pred_GCP_0_5deg[match(year − 1, year)], NA).

  - Create a new column, `prev_year_true`, with values calculated as: ifelse (year − 1 %in% year, GCP_0_5deg[match(year − 1, year)], NA).
  - Ungroup the dataframe.
  - Keep only rows where the `prev_year_pred` and `prev_year_true` columns values are not `NA`.
  - Refer to the updated file as the new `assessment_with_preds` file.
  - Split the `assessment_with_preds` dataframe into two files: `developed` and `developing`, representing rows where the `is_developing` column has values `0` and `1`, respectively.
- **Calculate R2 for log(GDP in t) - log(GDP in t-1):** Use the `calculate_chan_r2()` function defined earlier to compute the R2 for the three new files: `developed`, `developing`, and `assessment_with_preds`. For the `true_values` argument, use the `GCP_0_5deg` column. For the `true_last` argument, use the `prev_year_true` column. For the `predicted_values` argument, use the `pred_GCP_0_5deg` column. For the `predicted_last` argument, use the `prev_year_pred` column. Store the computed R2 values in the initialized empty vectors defined earlier: `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all`, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.

- Under the previous loop step, we now have the following vectors for the current combination of hyperparameters: `mse_developed`, `mse_developing`, `mse_all`, `r2_developed`, `r2_developing`, `r2_all`, `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all`. Each vector contains five values corresponding to the five folds.
- Calculate the single value `wgt_chan_r2` as the weighted average of `mean(chan_r2_developed)` and `mean(chan_r2_developing)`, using the respective shares from the `real_share` dataframe for developed (`is_developing == 0`) and developing (`is_developing == 1`) groups.
- Return the `wgt_chan_r2` value.

Select the Best Hyperparameters:

- After the above steps, we obtain `wgt_chan_r2` value for each combination of hyperparameters.
- Identify the combination of hyperparameters that yields the highest `wgt_chan_r2` value.

Fit the Final Model:

- **Define the Final Random Forest Model:**
  - Now we use the best combination of hyperparameters to fit the final Random Forest model.
  - Use the `rand_forest()` function with the best values for `mtry`, `trees`, and `min_n`.

- Set the engine to "ranger" use set_engine() with the following additional configurations:
    - importance = "impurity" to compute feature importance.
    - verbose = TRUE to enable verbose output during training.
    - num.threads = 20 to utilize parallel processing with 20 threads.
    - seed = 1234567 to ensure reproducibility.
  - Set the mode of the model to "regression" using set_mode()
  - Refer to this as rf_model_final
- **Create the Workflow:**
  - Initialize a workflow object using workflow().
  - Use the recipe() function to create the recipe for the Random Forest model by specifying the formula as generated in the previous steps and setting the training dataset to data_full. The code is rf_recipe <- recipe(formula = formula, data = data_full).
  - Add the preprocessing recipe (rf_recipe) to the workflow using add_recipe(rf_recipe).
  - Add the defined random forest model (rf_model_final) to the workflow using add_model(rf_model_final).
  - Include case weights (import_weight column) in the workflow using add_case_weights(import_weight).
  - Refer to this as rf_workflow_final.
- **Fit the Model:**
  - Use the fit() function to train the workflow rf_workflow_final on the complete dataset data_full.
  - Save the resulting fitted workflow as rf_model9_good_grid_search_0_5deg.RData. This is the final trained random forest model that will be used to predict global 0.5-degree cell GDP.

33 Train the 0.25-degree Random Forest Model: Include Years 2012 to 2021

In this section, we build the algorithm to train the 0.25-degree models.

1. **Calculate the Share of Cells Representing Developed and Developing Countries in the Actual World**

   Read the new_predictors_put_in_model_0_25deg.RData file using the load() function.

   Adjust the iso column values: if the original iso value is Ala, change it to USA, and leave all other values unchanged.

   Convert the dataset into a data frame using as.data.frame.

Exclude the geom column.

Select unique combinations of cell_id, subcell_id, subcell_id_0_25 and iso columns using distinct(cell_id, subcell_id, subcell_id_0_25, iso).

Create a new column is_developing with a value of 1 if the corresponding iso value is found in the developing column from the list_developed_developing.xlsx file located in the step4_train_and_tune_log_change/inputs folder. For all other rows, assign a value of 0.

Group the dataset by the is_developing column, then use the summarise() function to create a new column count, which calculates the number of rows for each group.

Create a new column share by dividing the count column by the total sum of count.

Refer to the resulting dataset as real_share.

2. **Full Training Dataset Put in the Model**

Define the developing_group list as: "CHL", "COL", "IDN", "KGZ", "PER", "PHL", "ALB", "BIH", "BLR", "MOZ", "SRB", "UZB", "VNM", "KEN", "LKA", "THA", "ECU".

Use read.csv to read the following files: new_data_train_0_25deg.csv, new_data_valid_year_0_25deg.csv, new_data_valid_iso_0_25deg.csv, new_data_test_year_0_25deg.csv, and new_data_test_iso_0_25deg.csv, then combine them using bind_rows.

Create a new column, unit_gdp_af_sum_rescl, with values equal to the corresponding values from the column state_total_GDP.

Create a new column, original_order, with values equal to the row number, using the row_number() function.

Create a new column, is_developing, with a value of 1 if the corresponding iso value belongs to the developing_group list; otherwise, set the value to 0.

Group the dataset by the is_developing column, and create a new column, sample_count, with values equal to the number of rows in each group, using the n() function.

Ungroup the dataset.

Create a new column, sample_share, with values equal to sample_count / n(), which represents the proportion of samples in each group.

Use left_join to combine the current dataframe with the real_share dataframe obtained in the previous step, ensuring that the current dataframe is the base file.

Create a new column, normalized_weight, with values calculated as share / sample_share.

Exclude the columns sample_count, sample_share, count, and share.

Create a new column, import_weight, with values generated by applying the importance_weights(normalized_weight) function. This will signal to the random forest model that the normalized_weight column is to be used as a weight column.

Arrange the dataset in ascending order based on the original_order column created earlier.

Exclude the original_order column.

Refer the resulting dataframe as data_full.

3. **Training Algorithm:**

Define Cross-Validation:

- Set the random seed to 1234567 for reproducibility.
- Define the cross-validation splits using the function group_vfold_cv().
- Group by the iso column to ensure that data from the same country appears in only one fold. This is done by set group = "iso" in group_vfold_cv
- Set the number of folds to 5 by passing v = 5 to group_vfold_cv.
- Store the cross-validation splits in the folds variable.

Define the Random Forest Training Function:

- Define the target variable as GCP_share_0_25deg.
- Define the predictor variables as a list of columns: "pop_share", "CO2_bio_manuf_conbust_share", "CO2_bio_heavy_indus_share", "CO2_bio_tspt_share", "CO2_non_org_manuf_conbust_share", "CO2_non_org_heavy_indus_share", "CO2_non_org_tspt_share", "NPP_share", "NTL_urban_snow_free_period_share", "NTL_cropland_snow_free_period_share", "NTL_other_snow_free_period_share", "snow_ice_share", "water_share", "urban_share", "forest_share", "cropland_share", "mean_rug", "national_gdpc", "lag_NTL_urban_share", "lag_urban_share", "lag_cropland_share", "lag_NTL_other_share", "lag_NTL_cropland_share", "lag_CO2_bio_mc_share", "lag_CO2_nonorg_mc_share", "lag_CO2_bio_heavy_indus_share", "lag_CO2_non_org_heavy_indus_share", "lag_CO2_bio_tspt_share", "lag_CO2_non_org_tspt_share", "lag_pop_share", "lag_NPP_share", "import_weight"
- Create a formula for the model use: formula = as.formula(paste(target_var, "~", paste(predictor_vars, collapse = " + "))).

Defining Reasonable Hyperparameter Combinations:

- Print a message indicating that hyperparameter tuning is starting.
- **Define a Grid of Hyperparameters to Tune:** Due to memory limitations, it is not feasible to try every single possible combination of mtry, trees, and min_n, so we increase mtry and min_n with an increment of 3. As long as the values lie within a reasonable range, they should be sufficient. Avoid using values that are either too low or too high. Define the following hyperparameter ranges:
  - mtry (the number of variables to sample for each split) with possible values: {7, 10, 13, 16}
  - trees (the number of trees in the forest) with possible values: {1000}

  - min_n (the minimum node size) with possible values: {10, 13, 16, 19}
- Use expand.grid to define combinations of mtry, trees, and min_n.
- Define a helper function tune_hyperparameters() to tune the hyperparameters.
- In the tune_hyperparameters() function, loop through each combination of hyperparameters, perform cross-validation, and calculate the performance metrics (MSE, R2) which will be defined below.
- Then save the best results based on the weighted R2 of annual changes (wgt_chan_r2) which will be defined below.

Calculate Performance Metrics for Each Combination of Hyperparameters:

- Initialize empty vectors for storing model evaluation metrics, including:
  - mse_developed, mse_developing, and mse_all for storing mean squared error (MSE).
  - r2_developed, r2_developing, and r2_all for storing R-squared values.
  - chan_r2_developed, chan_r2_developing, and chan_r2_all for storing annual changes R-squared values.
- Define the function calculate_r2() in the exact same way as it is defined in Section 31, Step 3.
- Define the function calculate_mse() in the exact same way as it is defined in Section 31, Step 3.
- Define the function calculate_chan_r2() in the exact same way as it is defined in Section 31, Step 3.
- Recall that the training countries are divided into five groups, creating five folds using the group_vfold_cv() function, stored in the folds file. Each fold consists of four groups used as the training set and one group used as the testing set. For each of the five folds, perform the following steps:
  - **Extract the Training and Testing Sets:** Extract the training and testing sets for the current fold using the analysis() and assessment() functions, respectively. To do this, use the following code: analysis <− as.data.frame(analysis(current_fold$splits [[i]])) and assessment <− as .data.frame(assessment(current_fold$splits[[i]]))
  - **Fit the Random Forest Model:**
    - **Define the Random Forest Model:** Use the rand_forest() function from the parsnip package with the parameters mtry, trees, and min_n, setting them to the current combination of hyperparameters in the loop.
    - **Set the Engine:** Use the set_engine() function from the parsnip package to specify the engine. Set the parameters as verbose = FALSE and seed = 1234567 to ensure reproducibility and suppress verbose output.
    - **Specify the Mode of the Model:** Use the set_mode("regression") function to indicate that this model is a regression model.

* **Fit the Model:** Fit the model using the formula defined earlier, with the data argument set to the training set and weights argument set to the import_weight column from the training data.
* **Complete Example Code:**

```
fit <- rand_forest(mtry = params$mtry, trees = params$trees, min_n
 = params$min_n)%>%
set_engine("ranger", verbose = FALSE, seed = 1234567)%>%
set_mode("regression")%>%
fit (formula, data = analysis, weights = import_weight)
```

- **Predict on the Testing Set:** Use the predict command from the stats package to predict on the testing set. The code is: preds <- as.data.frame(predict(fit, assessment)), where fit refers to the fitted random forest model described above, and assessment is the testing set obtained for the current fold.
- **Add the Prediction to the Testing Set:**
  * Begin with the assessment file.
  * Create a new column named pred_GCP_share_0_25deg and populate it with the predictions from the preds file.
  * Adjust the values in the pred_GCP_share_0_25deg column: set the value to 0 where the pop_share column equals 0; otherwise, retain the original values.
  * Refer to the updated file as assessment_with_preds.
  * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
- **Calculate MSE:** Use the calculate_mse() function defined earlier to compute the mean squared error (MSE) for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_share_0_25deg column. For the predicted_values argument, use the pred_GCP_share_0_25deg column. Store the computed MSE values in the initialized empty vectors defined earlier: mse_developed, mse_developing, and mse_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
- **Adjust the assessment_with_preds File:**
  * Start with the assessment_with_preds file obtained in the previous steps.
  * Group the dataset by the iso and year columns.
  * Create a new column, pred_GCP_share_0_25deg_rescaled, with values calculated as pred_GCP_share_0_25deg / sum(pred_GCP_share_0_25deg) within each group.
  * Ungroup the dataframe.

    * Create a new column, pred_GCP_0_25deg, with values calculated as pred_GCP_share_0_25deg_rescaled * state_total_GDP.
    * Refer to the updated file as the new assessment_with_preds file.
    * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
  - **Calculate R2 for log(GDP):** Use the calculate_r2() function defined earlier to compute the R2 for the three files: developed, developing, and assessment_with_preds. For the true_values argument of the function, use the GCP_0_25deg column. For the predicted_values argument, use the pred_GCP_0_25deg column. Store the computed R2 values in the initialized empty vectors defined earlier: r2_developed, r2_developing, and r2_all, in the i-th position of each vector. Recall that since there are 5 folds, each vector will contain five values.
  - **Adjust the** assessment_with_preds **File Again:**
    * Start with the updated assessment_with_preds file obtained in the previous steps.
    * Arrange the dataset in ascending order by the columns iso, cell_id, subcell_id, subcell_id_0_25, and year.
    * Group the dataset by the iso, cell_id, subcell_id, and subcell_id_0_25 columns.
    * Create a new column, prev_year_pred, with values calculated as: ifelse (year − 1 %in% year, pred_GCP_0_25deg[match(year − 1, year)], NA).
    * Create a new column, prev_year_true, with values calculated as: ifelse (year − 1 %in% year, GCP_0_25deg[match(year − 1, year)], NA).
    * Ungroup the dataframe.
    * Keep only rows where the prev_year_pred and prev_year_true columns values are not NA.
    * Refer to the updated file as the new assessment_with_preds file.
    * Split the assessment_with_preds dataframe into two files: developed and developing, representing rows where the is_developing column has values 0 and 1, respectively.
  - **Calculate R2 for log(GDP in t) - log(GDP in t-1):** Use the calculate_chan_r2() function defined earlier to compute the R2 for the three new files: developed, developing, and assessment_with_preds. For the true_values argument, use the GCP_0_25deg column. For the true_last argument, use the prev_year_true column. For the predicted_values argument, use the pred_GCP_0_25deg column. For the predicted_last argument, use the prev_year_pred column. Store the computed R2 values in the initialized empty vectors defined earlier: chan_r2_developed, chan_r2_developing, and chan_r2_all, in the i-th position of each

vector. Recall that since there are 5 folds, each vector will contain five values.

- Under the previous loop step, we now have the following vectors for the current combination of hyperparameters: `mse_developed`, `mse_developing`, `mse_all`, `r2_developed`, `r2_developing`, `r2_all`, `chan_r2_developed`, `chan_r2_developing`, and `chan_r2_all`. Each vector contains five values corresponding to the five folds.
- Calculate the single value `wgt_chan_r2` as the weighted average of `mean(chan_r2_developed)` and `mean(chan_r2_developing)`, using the respective shares from the `real_share` dataframe for developed (`is_developing == 0`) and developing (`is_developing == 1`) groups.
- Return the `wgt_chan_r2` value.

Select the Best Hyperparameters:

- After the above steps, we obtain `wgt_chan_r2` value for each combination of hyperparameters.
- Identify the combination of hyperparameters that yields the highest `wgt_chan_r2` value.

Fit the Final Model:

- **Define the Final Random Forest Model:**
  - Now we use the best combination of hyperparameters to fit the final Random Forest model.
  - Use the `rand_forest()` function with the best values for `mtry`, `trees`, and `min_n`.
  - Set the engine to `"ranger"` use `set_engine()` with the following additional configurations:
    * `importance = "impurity"` to compute feature importance.
    * `verbose = TRUE` to enable verbose output during training.
    * `num.threads = 20` to utilize parallel processing with 20 threads.
    * `seed = 1234567` to ensure reproducibility.
  - Set the mode of the model to `"regression"` using `set_mode()`
  - Refer to this as `rf_model_final`
- **Create the Workflow:**
  - Initialize a workflow object using `workflow()`.
  - Use the `recipe()` function to create the recipe for the Random Forest model by specifying the `formula` as generated in the previous steps and setting the training dataset to `data_full`. The code is `rf_recipe <- recipe(formula = formula, data = data_full)`.
  - Add the preprocessing recipe (`rf_recipe`) to the workflow using `add_recipe(rf_recipe)`.
  - Add the defined random forest model (`rf_model_final`) to the workflow using `add_model(rf_model_final)`.

- Include case weights (import_weight column) in the workflow using add_case_weights(import_weight).
  - Refer to this as rf_workflow_final.
- **Fit the Model:**
  - Use the fit() function to train the workflow rf_workflow_final on the complete dataset data_full.
  - Save the resulting fitted workflow as rf_model9_good_grid_search_0_25deg.RData. This is the final trained random forest model that will be used to predict global 0.25-degree cell GDP.

34 Predict 1-degree Cell GDP Around the World

1. **Load the Random Forest Model:** Read the file rf_model9_good_grid_search_1deg.RData obtained in Section 31 using load().

2. **Prepare the Predictors Dataset:**

   Read the predictors' dataset new_predictors_put_in_model_1deg.RData obtained in Section 29.

   Adjust the iso column values to USA for rows where the original iso column values are Ala, leaving others unchanged.

   Apply left_join() to combine this dataframe with the rgdp_total_af_sum_rescl.csv file from Section 19, ensuring the current dataframe remains the base file.

   Refer to the resulting file as predict_data_complete.

3. **Predict 1-Degree Cell GDP Share:**

   Use the predict() function from the stats package to predict 1-degree cell GDP share. The code is:
   predictions_model <− as.data.frame(predict(object = rf_model9_good_grid_search_1deg , new_data = predict_data_complete)).

   For cells in the training sample, use out-of-bag predictions instead of the predict() function. Obtain these predictions using: pred_train_sam <− as.data.frame(rf_model_goodfitfitfitpredictions).

4. **Organize Training Sample Data and Predictions:**

   Read the following files using read.csv: new_data_train_1deg.csv, new_data_valid_year_1deg.csv, new_data_valid_iso_1deg.csv, new_data_test_year_1deg.csv, and new_data_test_iso_1deg.csv.

   Combine these files using bind_rows(), ensuring the row order matches the data_full file used to train the model in Section 31 (as the pred_train_sam file lacks row identifiers).

   Rename the iso column to id.

Create a new column pred_GCP_share_1deg with values assigned from the corresponding row's first column values in the pred_train_sam file.

Select only the columns: cell_id, id, year, and pred_GCP_share_1deg.

Convert the cell_id column to characters using as.character().

Adjust the id column values: if the first four characters are USA_, replace them with characters 5 and 6; otherwise, retain the original values.

Refer to the resulting file as data_full.

5. **Organize World Predictions:**

Start with the predict_data_complete file.

Apply left_join() to combine it with data_full, ensuring the current dataframe remains the base file.

Create a new column pred_model with values assigned from the corresponding row's first column values in the predictions_model file.

Adjust the pred_GCP_share_1deg column: if its value is not missing, retain it; otherwise, assign the value from pred_model. This step ensures that for cells in the training sample, the predictions are based on out-of-bag predictions.

Convert the file to a dataframe using as.data.frame.

Remove the pred_model column.

Set the pred_GCP_share_1deg column to 0 for rows where pop_share equals 0 ; retain other values as is.

Group the dataset by id and year.

Create a new column pred_GCP_share_1deg_rescaled with values equal to pred_GCP_share_1deg / sum(pred_GCP_share_1deg) within each group.

Remove the grouping.

Create a new column pred_GCP_1deg with values equal to pred_GCP_share_1deg_rescaled * unit_gdp_af_sum_rescl.

Save the resulting file as predict_data_results_1deg_with_prov_boundary.RData.

Group the dataset by iso, year, and cell_id.

Create a new column pred_GCP_1deg_no_prov_bound with values equal to sum(pred_GCP_1deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, iso, year, pred_GCP_1deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_1deg_without_prov_boundary.RData.

6. **Intersect Country Geometry with 1-degree Grids:**

Apply the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection.

Set the input layer to world_poly.gpkg, obtained in Section 17.

Set the overlay layer to just_grid_1degree.gpkg, obtained in Section 18.

Save the output as country_1deg_intersected.gpkg.

Select only the columns cell_id, iso, and geom.

Adjust the iso column values to USA for rows with the original iso values equal to Ala; otherwise, retain the original values.

Refer to the resulting file as deg1_geometry.

7. **Population for Polygons Predicted by 1-degree Model:**

   Load the land_pop_extracted_region_level_1deg.RData file obtained in Section 20, Step 1.

   Select rows where the year column values are less than or equal to 2021.

   Convert the dataset to a dataframe using as.data.frame.

   Select the columns cell_id, id, iso, year, and pop.

   Adjust the pop column values to floor(pop).

   Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

   Save the resulting file as pop.

8. **Land Area for Polygons Predicted by 1-degree Model:**

   Load the lc_full_1deg.RData file obtained in Section 21.

   Select rows where the year column values are less than or equal to 2021.

   Convert the dataset to a dataframe using as.data.frame.

   Select the columns: cell_id, id, iso, year, water, barren, snow_ice, urban, dense_forest, open_forest, forest_cropland, herbaceous, cropland, shrub, and herbaceous_cropland.

   Replace any NA values with 0.

   Create a new column land_area_km2 with values equal to: barren + snow_ice + urban + dense_forest + open_forest + forest_cropland + herbaceous + cropland + shrub + herbaceous_cropland.

   Select the columns cell_id, id, iso, year, and land_area_km2.

   Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

   Save the resulting file as land_area.

9. **Population in 1degree-country Intersected Geometry:**

National Population:

- Read the rgdp_total_af_sum_rescl.csv file obtained from Section 19 using read.csv.
- Convert the file to a dataframe using as.data.frame.
- Select only the columns iso, year, and national_population.
- Select distinct rows based on the combination of the columns iso, year, and national_population, while retaining all other columns using .keep_all = TRUE.
- Filter rows where the year column value is less than or equal to 2021.
- Refer to the resulting file as national_population.

Start from the land_pop_extracted_region_level_1deg.RData file obtained in Section 20.

Filter rows where the year column value is less than or equal to 2021.

Adjust the iso column values to USA for rows where the original iso value is Ala; otherwise, retain the original values.

Convert the file to a dataframe using as.data.frame.

Select only the columns cell_id, id, iso, year, and pop.

Apply the left_join function to combine the current dataframe with the land_area file.

Adjust the pop column value to 0 if the land_area_km2 column value is 0; otherwise, retain the original values.

Remove rows with missing values using na.omit.

Group the dataset by the columns year, iso, and cell_id.

Create a new column pop_cell with values equal to the sum(pop) within each group.

Select distinct rows based on the combination of the columns year, iso, and cell_id, while retaining all other columns using .keep_all = TRUE.

Ungroup the dataset.

Select only the columns cell_id, iso, year, and pop_cell.

Apply left_join to combine the current dataframe with the national_population file, ensuring the current dataframe is the base file.

Group the dataset by iso and year.

Create a new column pop_cell_rescaled:

- Set the value to the pop_cell column if the national_population column value is missing.
- Otherwise, calculate it as pop_cell * national_population / sum(pop_cell) within each group.
- Apply the floor() function to ensure integer values.

Adjust the pop_cell_rescaled column value to 0 for rows where pop_cell is 0; otherwise, retain its current value.

Ungroup the dataset.

Apply left_join to combine the current dataframe with deg1_geometry, ensuring the current dataframe is the base file.

Save the resulting file as pop_cell_1deg.RData.

10. **Finalize World Predictions:**

Load the predict_data_results_1deg_with_prov_boundary.RData file.

Select the columns cell_id, id, iso, year, unit_gdp_af_sum_rescl, pred_GCP_share_1deg, pred_GCP_share_1deg_rescaled, pred_GCP_1deg, and geom.

Apply left_join to combine the current dataframe with the pop file. Ensure the current dataframe is the base file.

Apply left_join to combine the current dataframe with the land_area file. Ensure the current dataframe is the base file.

Create a new column pop_density_km2 with values equal to 0 if the land_area_km2 value is 0; otherwise, it is pop/land_area_km2.

Convert the dataframe to an sf object using st_as_sf.

Omit any rows with NA values using na.omit().

Create a new column pred_GCP_share_1deg with values equal to 0 if pop_density_km2 equals 0; otherwise, retain the original values.

Create a new column is_censored with values equal to 1 if pop_density_km2 equals 0 ; otherwise, set the value to 0.

Group the dataset by id and year.

Create a new column pred_GCP_share_1deg_rescaled with values equal to 0 if pred_GCP_share_1deg equals 0; otherwise, calculate it as pred_GCP_share_1deg/sum(pred_GCP_share_1deg) within each group.

Ungroup the dataset.

Create a new column pred_GCP_1deg with values equal to pred_GCP_share_1deg_rescaled * unit_gdp_af_sum_rescl.

Group the dataset by iso, year, and cell_id.

Create a new column is_cell_censored with values equal to 1 if any value in the is_censored column within the group equals 1; otherwise, set it to 0.

Create a new column pred_GCP_1deg_no_prov_bound with values equal to sum(pred_GCP_1deg) within each group.

Ungroup the dataset.

Convert the dataset to a dataframe using as.data.frame().

Select the columns cell_id, iso, year, pred_GCP_1deg_no_prov_bound, and is_cell_censored.

Select distinct rows based on the combination of iso, year, and cell_id, while retaining all other columns using .keep_all = TRUE.

Rename the pred_GCP_1deg_no_prov_bound column to predicted_GCP.

Select the columns cell_id, iso, year, predicted_GCP, and is_cell_censored.

Create a new column method with values equal to post-adjust zero GDP for pop density = 0.

Create a new column cell_size with values equal to 1-deg by 1-deg.

Apply left_join to combine the current dataframe with the deg1_geometry file, ensuring the current dataframe is the base file.

Apply left_join to combine the current dataframe with the pop_cell_1deg file, ensuring the current dataframe is the base file.

Adjust the predicted_GCP values to 0 if the pop_cell_rescaled column equals 0 ; otherwise, retain the original values.

Create a new column cell_GDPC with values equal to 0 if pop_cell_rescaled equals 0 ; otherwise, calculate it as predicted_GCP/pop_cell_rescaled.

Exclude the pop_cell column.

Rename the pop_cell_rescaled column to pop_cell.

Save the resulting file as GDPC_1deg_postadjust_pop_dens_no_extra_adjust.RData.

Apply left_join to combine the current dataframe with the following file:

- Read the just_grid_1deg_with_lon_lat.csv file provided in the step5_predict_and_post_adjustments_log_change/outputs folder.
- Adjust the cell_id column to characters using as.character().

Convert the dataset to a dataframe using as.data.frame().

Exclude the geom column.

Save the resulting file as GDPC_1deg_postadjust_pop_dens_no_extra_adjust.csv, ensuring that the output excludes row names by setting the parameter row.names = FALSE.

35 Predict 0.5-degree Cell GDP Around the World

1. **Load the Random Forest Model:** Read the file rf_model9_good_grid_search_0_5deg.RData obtained in Section 32 using load().

2. **Prepare the Predictors Dataset:**

Read the predictors' dataset new_predictors_put_in_model_0_5deg.RData obtained in Section 29.

Adjust the iso column values to USA for rows where the original iso column values are Ala, leaving others unchanged.

Apply left_join() to combine this dataframe with the rgdp_total_af_sum_rescl.csv file from Section 19, ensuring the current dataframe remains the base file.

Refer to the resulting file as predict_data_complete.

3. **Predict 0.5-Degree Cell GDP Share:**

   Use the predict() function from the stats package to predict 0.5-degree cell GDP share. The code is: predictions_model <− as.data.frame(predict(object = rf_model9_good_grid_search_0_5deg , new_data = predict_data_complete)).

   For cells in the training sample, use out-of-bag predictions instead of the predict() function. Obtain these predictions using: pred_train_sam <− as.data.frame(rf_model_goodfitfitfitpredictions).

4. **Organize Training Sample Data and Predictions:**

   Read the following files using read.csv: new_data_train_0_5deg.csv, new_data_valid_year_0_5deg.csv, new_data_valid_iso_0_5deg.csv, new_data_test_year_0_5deg.csv, and new_data_test_iso_0_5deg.csv.

   Combine these files using bind_rows(), ensuring the row order matches the data_full file used to train the model in Section 32 (as the pred_train_sam file lacks row identifiers).

   Rename the iso column to id.

   Create a new column pred_GCP_share_0_5deg with values assigned from the corresponding row's first column values in the pred_train_sam file.

   Select only the columns: cell_id, subcell_id, id, year, and pred_GCP_share_0_5deg.

   Convert the cell_id column to characters using as.character().

   Adjust the id column values: if the first four characters are USA_, replace them with characters 5 and 6; otherwise, retain the original values.

   Refer to the resulting file as data_full.

5. **Organize World Predictions:**

   Start with the predict_data_complete file.

   Apply left_join() to combine it with data_full, ensuring the current dataframe remains the base file.

   Create a new column pred_model with values assigned from the corresponding row's first column values in the predictions_model file.

Adjust the pred_GCP_share_0_5deg column: if its value is not missing, retain it; otherwise, assign the value from pred_model. This step ensures that for cells in the training sample, the predictions are based on out-of-bag predictions.

Convert the file to a dataframe using as.data.frame.

Remove the pred_model column.

Set the pred_GCP_share_0_5deg column to 0 for rows where pop_share equals 0 ; retain other values as is.

Group the dataset by id and year.

Create a new column pred_GCP_share_0_5deg_rescaled with values equal to pred_GCP_share_0_5deg / sum(pred_GCP_share_0_5deg) within each group.

Remove the grouping.

Create a new column pred_GCP_0_5deg with values equal to pred_GCP_share_0_5deg_rescaled * unit_gdp_af_sum_rescl.

Save the resulting file as predict_data_results_0_5deg_with_prov_boundary.RData.

Group the dataset by iso, year, cell_id, and subcell_id.

Create a new column pred_GCP_0_5deg_no_prov_bound with values equal to sum(pred_GCP_0_5deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, subcell_id, iso, year, pred_GCP_0_5deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_0_5deg_without_prov_boundary.RData.

Group the dataset by iso, year, and cell_id.

Create a new column pred_GCP_1deg_no_prov_bound with values equal to sum(pred_GCP_0_5deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, iso, year, pred_GCP_1deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_1deg_from_0_5deg_without_prov_boundary.RData.

6. **Intersect Country Geometry with 0.5-degree Grids:**

Apply the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection.

Set the input layer to world_poly.gpkg, obtained in Section 17.

Set the overlay layer to just_grid_0_5degree.gpkg, obtained in Section 18.

Save the output as country_0_5deg_intersected.gpkg.

Select only the columns cell_id, subcell_id, iso, and geom.

Adjust the iso column values to USA for rows with the original iso values equal to Ala; otherwise, retain the original values.

Refer to the resulting file as deg0_5_geometry.

7. **Population for Polygons Predicted by 0.5-degree Model:**

Load the land_pop_extracted_region_level_0_5deg.RData file obtained in Section 20.

Select rows where the year column values are less than or equal to 2021.

Convert the dataset to a dataframe using as.data.frame.

Select the columns cell_id, subcell_id, id, iso, year, and pop.

Adjust the pop column values to floor(pop).

Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

Save the resulting file as pop.

8. **Land Area for Polygons Predicted by 0.5-degree Model:**

Load the lc_full_0_5deg.RData file obtained in Section 21.

Select rows where the year column values are less than or equal to 2021.

Convert the dataset to a dataframe using as.data.frame.

Select the columns: cell_id, subcell_id, id, iso, year, water, barren, snow_ice, urban, dense_forest, open_forest, forest_cropland, herbaceous, cropland, shrub, and herbaceous_cropland.

Replace any NA values with 0.

Create a new column land_area_km2 with values equal to: barren + snow_ice + urban + dense_forest + open_forest + forest_cropland + herbaceous + cropland + shrub + herbaceous_cropland.

Select the columns cell_id, subcell_id, id, iso, year, and land_area_km2.

Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

Save the resulting file as land_area.

9. **Population in 0.5degree-country Intersected Geometry:**

National Population:

- Read the rgdp_total_af_sum_rescl.csv file obtained from Section 19 using read.csv.
- Convert the file to a dataframe using as.data.frame.
- Select only the columns iso, year, and national_population.

- Select distinct rows based on the combination of the columns iso, year, and national_population, while retaining all other columns using .keep_all = TRUE.
- Filter rows where the year column value is less than or equal to 2021.
- Refer to the resulting file as national_population.

Start from the land_pop_extracted_region_level_0_5deg.RData file obtained in Section 20.

Filter rows where the year column value is less than or equal to 2021.

Adjust the iso column values to USA for rows where the original iso value is Ala; otherwise, retain the original values.

Convert the file to a dataframe using as.data.frame.

Select only the columns cell_id, subcell_id, id, iso, year, and pop.

Apply the left_join function to combine the current dataframe with the land_area file.

Adjust the pop column value to 0 if the land_area_km2 column value is 0; otherwise, retain the original values.

Remove rows with missing values using na.omit.

Group the dataset by the columns year, iso, cell_id, and subcell_id.

Create a new column pop_cell with values equal to the sum(pop) within each group.

Select distinct rows based on the combination of the columns year, iso, cell_id, and subcell_id, while retaining all other columns using .keep_all = TRUE.

Ungroup the dataset.

Select only the columns cell_id, subcell_id, iso, year, and pop_cell.

Apply left_join to combine the current dataframe with the national_population file, ensuring the current dataframe is the base file.

Group the dataset by iso and year.

Create a new column pop_cell_rescaled:

- Set the value to the pop_cell column if the national_population column value is missing.
- Otherwise, calculate it as pop_cell * national_population / sum(pop_cell) within each group.
- Apply the floor() function to ensure integer values.

Adjust the pop_cell_rescaled column value to 0 for rows where pop_cell is 0; otherwise, retain its current value.

Ungroup the dataset.

Apply left_join to combine the current dataframe with deg0_5_geometry, ensuring the current dataframe is the base file.

Save the resulting file as pop_cell_0_5deg.RData.

10. **Finalize World Predictions:**

    Load the predict_data_results_0_5deg_with_prov_boundary.RData file.

    Select the columns cell_id, subcell_id, id, iso, year, unit_gdp_af_sum_rescl, pred_GCP_share_0_5deg, pred_GCP_share_0_5deg_rescaled, pred_GCP_0_5deg, and geom.

    Apply left_join to combine the current dataframe with the pop file. Ensure the current dataframe is the base file.

    Apply left_join to combine the current dataframe with the land_area file. Ensure the current dataframe is the base file.

    Create a new column pop_density_km2 with values equal to 0 if the land_area_km2 value is 0; otherwise, it is pop/land_area_km2.

    Convert the dataframe to an sf object using st_as_sf.

    Omit any rows with NA values using na.omit().

    Create a new column pred_GCP_share_0_5deg with values equal to 0 if pop_density_km2 equals 0; otherwise, retain the original values.

    Create a new column is_censored with values equal to 1 if pop_density_km2 equals 0; otherwise, set the value to 0.

    Group the dataset by id and year.

    Create a new column pred_GCP_share_0_5deg_rescaled with values equal to 0 if pred_GCP_share_0_5deg equals 0; otherwise, calculate it as pred_GCP_share_0_5deg/sum(pred_GCP_share_0_5deg) within each group.

    Ungroup the dataset.

    Create a new column pred_GCP_0_5deg with values equal to pred_GCP_share_0_5deg_rescaled * unit_gdp_af_sum_rescl.

    Group the dataset by iso, year, cell_id, and subcell_id.

    Create a new column is_cell_censored with values equal to 1 if any value in the is_censored column within the group equals 1; otherwise, set it to 0.

    Create a new column pred_GCP_0_5deg_no_prov_bound with values equal to sum(pred_GCP_0_5deg) within each group.

    Ungroup the dataset.

    Convert the dataset to a dataframe using as.data.frame().

    Select the columns cell_id, subcell_id, iso, year, pred_GCP_0_5deg_no_prov_bound, and is_cell_censored.

    Select distinct rows based on the combination of iso, year, cell_id, and subcell_id, while retaining all other columns using .keep_all = TRUE.

    Rename the pred_GCP_0_5deg_no_prov_bound column to predicted_GCP.

Select the columns cell_id, subcell_id, iso, year, predicted_GCP, and is_cell_censored.

Create a new column method with values equal to post-adjust zero GDP for pop density = 0.

Create a new column cell_size with values equal to 0.5-deg by 0.5-deg.

Apply left_join to combine the current dataframe with the deg0_5_geometry file, ensuring the current dataframe is the base file.

Apply left_join to combine the current dataframe with the pop_cell_0_5deg file, ensuring the current dataframe is the base file.

Adjust the predicted_GCP values to 0 if the pop_cell_rescaled column equals 0 ; otherwise, retain the original values.

Create a new column cell_GDPC with values equal to 0 if pop_cell_rescaled equals 0 ; otherwise, calculate it as predicted_GCP/pop_cell_rescaled.

Exclude the pop_cell column.

Rename the pop_cell_rescaled column to pop_cell.

Save the resulting file as GDPC_0_5deg_postadjust_pop_dens_no_extra_adjust.RData.

Apply left_join to combine the current dataframe with the following file:

- Read the just_grid_0_5deg_with_lon_lat.csv file provided in the step5_predict_and_post_adjustments_log_change/outputs folder.
- Adjust the cell_id column to characters using as.character().

Convert the dataset to a dataframe using as.data.frame().

Exclude the geom column.

Save the resulting file as GDPC_0_5deg_postadjust_pop_dens_no_extra_adjust.csv, ensuring that the output excludes row names by setting the parameter row.names = FALSE.

36 Predict 0.25-degree Cell GDP Around the World

1. **Load the Random Forest Model:** Read the file rf_model9_good_grid_search_0_25deg.RData obtained in Section 33 using load().

2. **Prepare the Predictors Dataset:**

Read the predictors' dataset new_predictors_put_in_model_0_25deg.RData obtained in Section 29.

Adjust the iso column values to USA for rows where the original iso column values are Ala, leaving others unchanged.

Apply left_join() to combine this dataframe with the rgdp_total_af_sum_rescl.csv file from Section 19, ensuring the current dataframe remains the base file.

Refer to the resulting file as `predict_data_complete`.

3. **Predict 0.25-Degree Cell GDP Share:**

   Use the `predict()` function from the `stats` package to predict 0.25-degree cell GDP share. The code is: predictions_model <− as.data.frame(predict(object = rf_model9_good_grid_search_0_25de , new_data = predict_data_complete)).

   For cells in the training sample, use out-of-bag predictions instead of the `predict( )` function. Obtain these predictions using: pred_train_sam <− as.data.frame(rf_model_goodfitfitfitpredictions).

4. **Organize Training Sample Data and Predictions:**

   Read the following files using `read.csv`: `new_data_train_0_25deg.csv`, `new_data_valid_year_0_25deg.csv`, `new_data_valid_iso_0_25deg.csv`, `new_data_test_year_0_25deg.csv`, and `new_data_test_iso_0_25deg.csv`.

   Combine these files using `bind_rows()`, ensuring the row order matches the `data_full` file used to train the model in Section 33 (as the `pred_train_sam` file lacks row identifiers).

   Rename the `iso` column to `id`.

   Create a new column `pred_GCP_share_0_25deg` with values assigned from the corresponding row's first column values in the `pred_train_sam` file.

   Select only the columns: `cell_id`, `subcell_id`, `subcell_id_0_25`, `id`, `year`, and `pred_GCP_share_0_25deg`.

   Convert the `cell_id` column to characters using `as.character()`.

   Adjust the `id` column values: if the first four characters are `USA_`, replace them with characters 5 and 6; otherwise, retain the original values.

   Refer to the resulting file as `data_full`.

5. **Organize World Predictions:**

   Start with the `predict_data_complete` file.

   Apply `left_join()` to combine it with `data_full`, ensuring the current dataframe remains the base file.

   Create a new column `pred_model` with values assigned from the corresponding row's first column values in the `predictions_model` file.

   Adjust the `pred_GCP_share_0_25deg` column: if its value is not missing, retain it; otherwise, assign the value from `pred_model`. This step ensures that for cells in the training sample, the predictions are based on out-of-bag predictions.

   Convert the file to a dataframe using `as.data.frame`.

   Remove the `pred_model` column.

Set the pred_GCP_share_0_25deg column to 0 for rows where pop_share equals 0 ; retain other values as is.

Group the dataset by id and year.

Create a new column pred_GCP_share_0_25deg_rescaled with values equal to pred_GCP_share_0_25deg / sum(pred_GCP_share_0_25deg) within each group.

Remove the grouping.

Create a new column pred_GCP_0_25deg with values equal to pred_GCP_share_0_25deg_rescaled * unit_gdp_af_sum_rescl.

Save the resulting file as predict_data_results_0_25deg_with_prov_boundary.RData.

Group the dataset by iso, year, cell_id, subcell_id, subcell_id_0_25.

Create a new column pred_GCP_0_25deg_no_prov_bound with values equal to sum(pred_GCP_0_25deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, subcell_id, subcell_id_0_25, iso, year, pred_GCP_0_25deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_0_25deg_without_prov_boundary.RData.

Group the dataset by iso, year, and cell_id.

Create a new column pred_GCP_1deg_no_prov_bound with values equal to sum(pred_GCP_0_25deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, iso, year, pred_GCP_1deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_1deg_from_0_25deg_without_prov_boundary.RData.

Group the dataset by iso, year, cell_id, and subcell_id.

Create a new column pred_GCP_0_5deg_no_prov_bound with values equal to sum(pred_GCP_0_25deg) within each group.

Ungroup the dataset.

Select only the columns: cell_id, subcell_id, iso, year, pred_GCP_0_5deg_no_prov_bound, country_total_GDP, national_population, and geom.

Save the resulting file as predict_data_results_0_5deg_from_0_25deg_without_prov_boundary.RData.

6. **Intersect Country Geometry with 0.25-degree Grids:**

Apply the qgis_run_algorithm function from the qgisprocess package with the algorithm native:intersection.

Set the input layer to world_poly.gpkg, obtained in Section 17.

Set the overlay layer to just_grid_0_25degree.gpkg, obtained in Section 18.

Save the output as country_0_25deg_intersected.gpkg.

Select only the columns cell_id, subcell_id, subcell_id_0_25, iso, and geom.

Adjust the iso column values to USA for rows with the original iso values equal to Ala; otherwise, retain the original values.

Refer to the resulting file as deg0_25_geometry.

7. **Population for Polygons Predicted by 0.25-degree Model:**

   Load the land_pop_extracted_region_level_0_25deg.RData file obtained in Section 20.

   Select rows where the year column values are less than or equal to 2021.

   Convert the dataset to a dataframe using as.data.frame.

   Select the columns cell_id, subcell_id, subcell_id_0_25, id, iso, year, and pop.

   Adjust the pop column values to floor(pop).

   Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

   Save the resulting file as pop.

8. **Land Area for Polygons Predicted by 0.25-degree Model:**

   Load the lc_full_0_25deg.RData file obtained in Section 21.

   Select rows where the year column values are less than or equal to 2021.

   Convert the dataset to a dataframe using as.data.frame.

   Select the columns: cell_id, subcell_id, subcell_id_0_25, id, iso, year, water, barren, snow_ice, urban, dense_forest, open_forest, forest_cropland, herbaceous, cropland, shrub, and herbaceous_cropland.

   Replace any NA values with 0.

   Create a new column land_area_km2 with values equal to: barren + snow_ice + urban + dense_forest + open_forest + forest_cropland + herbaceous + cropland + shrub + herbaceous_cropland.

   Select the columns cell_id, subcell_id, subcell_id_0_25, id, iso, year, and land_area_km2.

   Modify the iso column values such that if the value is "Ala", it is replaced with "USA", while other values remain unchanged.

   Save the resulting file as land_area.

9. **Population in 0.25degree-country Intersected Geometry:**

   National Population:

- Read the rgdp_total_af_sum_rescl.csv file obtained from Section 19 using read.csv.
- Convert the file to a dataframe using as.data.frame.
- Select only the columns iso, year, and national_population.
- Select distinct rows based on the combination of the columns iso, year, and national_population, while retaining all other columns using .keep_all = TRUE.
- Filter rows where the year column value is less than or equal to 2021.
- Refer to the resulting file as national_population.

Start from the land_pop_extracted_region_level_0_25deg.RData file obtained in Section 20.

Filter rows where the year column value is less than or equal to 2021.

Adjust the iso column values to USA for rows where the original iso value is Ala; otherwise, retain the original values.

Convert the file to a dataframe using as.data.frame.

Select only the columns cell_id, subcell_id, subcell_id_0_25, id, iso, year, and pop.

Apply the left_join function to combine the current dataframe with the land_area file.

Adjust the pop column value to 0 if the land_area_km2 column value is 0; otherwise, retain the original values.

Remove rows with missing values using na.omit.

Group the dataset by the columns year, iso, cell_id, subcell_id, and subcell_id_0_25.

Create a new column pop_cell with values equal to the sum(pop) within each group.

Select distinct rows based on the combination of the columns year, iso, cell_id, subcell_id, and subcell_id_0_25, while retaining all other columns using .keep_all = TRUE.

Ungroup the dataset.

Select only the columns cell_id, subcell_id, subcell_id_0_25, iso, year, and pop_cell.

Apply left_join to combine the current dataframe with the national_population file, ensuring the current dataframe is the base file.

Group the dataset by iso and year.

Create a new column pop_cell_rescaled:

- Set the value to the pop_cell column if the national_population column value is missing.
- Otherwise, calculate it as pop_cell * national_population / sum(pop_cell) within each group.

- Apply the `floor()` function to ensure integer values.

Adjust the `pop_cell_rescaled` column value to `0` for rows where `pop_cell` is `0`; otherwise, retain its current value.

Ungroup the dataset.

Apply `left_join` to combine the current dataframe with `deg0_25_geometry`, ensuring the current dataframe is the base file.

Save the resulting file as `pop_cell_0_25deg.RData`.

10. **Finalize World Predictions:**

Load the `predict_data_results_0_25deg_with_prov_boundary.RData` file.

Select the columns `cell_id`, `subcell_id`, `subcell_id_0_25`, `id`, `iso`, `year`, `unit_gdp_af_sum_rescl`, `pred_GCP_share_0_25deg`, `pred_GCP_share_0_25deg_rescaled`, `pred_GCP_0_25deg`, and `geom`.

Apply `left_join` to combine the current dataframe with the `pop` file. Ensure the current dataframe is the base file.

Apply `left_join` to combine the current dataframe with the `land_area` file. Ensure the current dataframe is the base file.

Create a new column `pop_density_km2` with values equal to `0` if the `land_area_km2` value is `0`; otherwise, it is `pop/land_area_km2`.

Convert the dataframe to an `sf` object using `st_as_sf`.

Omit any rows with `NA` values using `na.omit()`.

Create a new column `pred_GCP_share_0_25deg` with values equal to `0` if `pop_density_km2` equals `0`; otherwise, retain the original values.

Create a new column `is_censored` with values equal to `1` if `pop_density_km2` equals `0`; otherwise, set the value to `0`.

Group the dataset by `id` and `year`.

Create a new column `pred_GCP_share_0_25deg_rescaled` with values equal to `0` if `pred_GCP_share_0_25deg` equals `0`; otherwise, calculate it as `pred_GCP_share_0_25deg/sum(pred_GCP_share_0_25deg)` within each group.

Ungroup the dataset.

Create a new column `pred_GCP_0_25deg` with values equal to `pred_GCP_share_0_25deg_rescaled * unit_gdp_af_sum_rescl`.

Group the dataset by `iso`, `year`, `cell_id`, `subcell_id`, and `subcell_id_0_25`.

Create a new column `is_cell_censored` with values equal to `1` if any value in the `is_censored` column within the group equals `1`; otherwise, set it to `0`.

Create a new column `pred_GCP_0_25deg_no_prov_bound` with values equal to `sum(pred_GCP_0_25deg)` within each group.

Ungroup the dataset.

Convert the dataset to a dataframe using as.data.frame().

Select the columns cell_id, subcell_id, subcell_id_0_25, iso, year, pred_GCP_0_25deg_no_prov_bound, and is_cell_censored.

Select distinct rows based on the combination of iso, year, cell_id, and subcell_id, while retaining all other columns using .keep_all = TRUE.

Rename the pred_GCP_0_25deg_no_prov_bound column to predicted_GCP.

Select the columns cell_id, subcell_id, subcell_id_0_25, iso, year, predicted_GCP, and is_cell_censored.

Create a new column method with values equal to post-adjust zero GDP for pop density = 0.

Create a new column cell_size with values equal to 0.25-deg by 0.25-deg.

Apply left_join to combine the current dataframe with the deg0_25_geometry file, ensuring the current dataframe is the base file.

Apply left_join to combine the current dataframe with the pop_cell_0_25deg file, ensuring the current dataframe is the base file.

Adjust the predicted_GCP values to 0 if the pop_cell_rescaled column equals 0; otherwise, retain the original values.

Create a new column cell_GDPC with values equal to 0 if pop_cell_rescaled equals 0; otherwise, calculate it as predicted_GCP/pop_cell_rescaled.

Exclude the pop_cell column.

Rename the pop_cell_rescaled column to pop_cell.

Save the resulting file as GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.RData.

Apply left_join to combine the current dataframe with the following file:

- Read the just_grid_0_25deg_with_lon_lat.csv file provided in the step5_predict_and_post_adjustments_log_change/outputs folder.
- Adjust the cell_id column to characters using as.character().

Convert the dataset to a dataframe using as.data.frame().

Exclude the geom column.

Save the resulting file as GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.csv, ensuring that the output excludes row names by setting the parameter row.names = FALSE.

37 Transfer the Predicted GDP Values to Other Units and Create the Final Results:

Recall that in Section 34, 35, and 36, we obtain the GDPC_1deg_postadjust_pop_dens_no_extra_adjust.RData, GDPC_1deg_postadjust_pop_dens_no_extra_adjust.csv, GD

PC_0_5deg_postadjust_pop_dens_no_extra_adjust.RData, GDPC_0_5deg_postadjust_pop_dens_no_extra_adjust.csv, GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.RData, and GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.csv. They are in units constant 2017 USD. Here in this section, we can change the units to current USD, current PPP-adjusted international dollar, and constant 2017 PPP-adjusted international dollars by rescaling the values. We then consolidate all the values in various units into a single comprehensive file, which serves as our final result.

1. Read the file national_gdp_current_USD.csv, national_gdp_current_PPP.csv, national_gdp_const_2017_USD.csv, and national_gdp_const_2017_PPP.csv obtained in Section 14 use read.csv

2. **Scalar for Converting Constant 2017 USD to Current USD:**
   - Start with the national_gdp_const_2017_USD.csv file.
   - Select only the columns iso, year, and rgdp_total.
   - Rename the rgdp_total column to const_USD.
   - Apply left_join to combine the current dataframe with the following dataframe, ensuring the current dataframe is the base file:
     - For the national_gdp_current_USD.csv file, select only the columns iso, year, and rgdp_total.
     - Rename the rgdp_total column to curt_USD.
   - Create a new column const_to_curt_USD_idx with values calculated as curt_USD / const_USD.
   - Select only the columns iso, year, and const_to_curt_USD_idx.
   - Refer the resulting file as const_to_curt_USD

3. **Scalar for Converting Constant 2017 USD to Current PPP-adjusted International Dollars:**
   - Start with the national_gdp_const_2017_USD.csv file.
   - Select only the columns iso, year, and rgdp_total.
   - Rename the rgdp_total column to const_USD.
   - Apply left_join to combine the current dataframe with the following dataframe, ensuring the current dataframe is the base file:
     - For the national_gdp_current_PPP.csv file, select only the columns iso, year, and rgdp_total.
     - Rename the rgdp_total column to curt_PPP.
   - Create a new column const_to_curt_PPP_idx with values calculated as curt_PPP/const_USD.
   - Select only the columns iso, year, and const_to_curt_PPP_idx.

- Refer the resulting file as const_to_curt_PPP

4. **Scalar for Converting Constant 2017 USD to COnstant 2017 PPP-adjusted International Dollars:**

   - Start with the national_gdp_const_2017_USD.csv file.
   - Select only the columns iso, year, and rgdp_total.
   - Rename the rgdp_total column to const_USD.
   - Apply left_join to combine the current dataframe with the following dataframe, ensuring the current dataframe is the base file:
     - For the national_gdp_const_2017_PPP.csv file, select only the columns iso, year, and rgdp_total.
     - Rename the rgdp_total column to const_PPP.
   - Create a new column const_to_const_PPP_idx with values calculated as const_PPP/const_USD.
   - Select only the columns iso, year, and const_to_const_PPP_idx.
   - Refer the resulting file as const_to_const_PPP

5. **Final Results:** For each of the following files: GDPC_1deg_postadjust_pop_dens_no_extra_adjust.RData, GDPC_1deg_postadjust_pop_dens_no_extra_adjust.csv, GDPC_0_5deg_postadjust_pop_dens_no_extra_adjust.RData, GDPC_0_5deg_postadjust_pop_dens_no_extra_adjust.csv, GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.RData, and GDPC_0_25deg_postadjust_pop_dens_no_extra_adjust.csv, perform the following steps:

   - Read the file.
   - Convert the dataset into a dataframe using as.data.frame.
   - Apply left_join to combine the current dataframe with const_to_curt_USD, ensuring the current dataframe is the base file.
   - Apply left_join to combine the current dataframe with const_to_curt_PPP, ensuring the current dataframe is the base file.
   - Apply left_join to combine the current dataframe with const_to_const_PPP, ensuring the current dataframe is the base file.
   - Rename the predicted_GCP column to predicted_GCP_const_2017_USD and the cell_GDPC column to cell_GDPC_const_2017_USD.
   - Create the following new columns:
     - `predicted_GCP_current_USD` = `predicted_GCP_const_2017_USD * const_to_curt_USD_idx`
     - `predicted_GCP_const_2017_PPP` = `predicted_GCP_const_2017_USD * const_to_const_PPP_idx`

  - `predicted_GCP_current_PPP` = `predicted_GCP_const_2017_USD * const_to_curt_PPP_idx`
  - `cell_GDPC_current_USD` = `cell_GDPC_const_2017_USD * const_to_curt_USD_idx`
  - `cell_GDPC_const_2017_PPP` = `cell_GDPC_const_2017_USD * const_to_const_PPP_idx`
  - `cell_GDPC_current_PPP` = `cell_GDPC_const_2017_USD * const_to_curt_PPP_idx`
- Exclude the columns const_to_curt_USD_idx, const_to_curt_PPP_idx, and const_to_const_PPP_idx.
- Save the file as final_GDPC_xdeg_postadjust_pop_dens_no_extra_adjust.csv or RData, depending on the original file format.