Supporting Document for Thesis II

Codes

Joseph

Data Collection

1. Geoindicators Extraction (`gee.js`)

Purpose: Compute annual geospatial indicators (e.g., NDVI, NDBI, Land Surface Temperature) for each Swedish county using Google Earth Engine (GEE).

Key Steps:

1. **Initialization:** Load the Earth Engine API, authenticate, and define the study region by importing county boundary shapefiles.
2. **Satellite Band Selection:** Specify spectral bands or indices (e.g., `MODIS/006/MOD11A2` for LST, `LANDSAT/LC08/C01/T1_SR` for NDVI).
3. **Composite Creation:** For each year:
    - Filter image collections by date range.
    - Apply cloud masking functions.
    - Calculate mean or median composites per county polygon.
4. **Indicator Computation:** Derive additional indices, e.g., NDVI = (NIR - Red) / (NIR + Red).
5. **Aggregation:** Use `reduceRegion` with `ee.Reducer.mean()` and `ee.Reducer.stdDev()` over each county geometry.
6. **Export:** Write the annual statistics as CSV files to a specified Google Drive folder.

```
// ====================================================================
// ANNUAL GEOINDICATORS FOR SWEDEN (COUNTY LEVEL) - 2024
// Computes yearly averages for 2024, falling back to the most recent available year if neede
// Updated : 2025-05-11
// ====================================================================

// PARAMETERS
var targetYear   = '2024',
    fallbackYear = '2023',
    startDate    = targetYear   + '-01-01',
    endDate      = targetYear   + '-12-31',
    fbStart      = fallbackYear + '-01-01',
    fbEnd        = fallbackYear + '-12-31';

// 0. ADMINISTRATIVE UNITS
var counties = ee.FeatureCollection('projects/geodata-458113/assets/SWE_adm1');
var simplifiedCounties = counties.map(function(f) {
  return f.simplify({maxError: 100});
});

// ====================================================================
// 1. INDICATORS - YEARLY AVERAGES
// ====================================================================

// NIGHTTIME LIGHTS (VIIRS DNB Monthly)
// Dataset: NOAA/VIIRS/DNB/MONTHLY_V1/VCMCFG
// Definition: Average radiance (avg_rad) from nighttime lights
// Purpose: Proxy for human activity & economic development
// Period: January 1 - December 31 (annual)
// Resolution: ~500 m
// More info: https://developers.google.com/earth-engine/datasets/catalog/NOAA_VIIRS_DNB_MON
var viirsCol         = ee.ImageCollection('NOAA/VIIRS/DNB/MONTHLY_V1/VCMCFG')
                          .filterDate(startDate, endDate)
                          .select('avg_rad');
var viirsAvail       = viirsCol.size().gt(0);
var viirsFallbackCol = ee.ImageCollection('NOAA/VIIRS/DNB/MONTHLY_V1/VCMCFG')
                          .filterDate(fbStart, fbEnd)
                          .select('avg_rad');
var viirsImg         = ee.Image(ee.Algorithms.If(
                          viirsAvail,
                          viirsCol.mean(),
                          viirsFallbackCol.mean()
```

```
                        )).rename('VIIRS_avg_2024');
var viirsYear        = ee.String(ee.Algorithms.If(viirsAvail, targetYear, fallbackYear));

// URBAN COVER (Dynamic World)
// Dataset: GOOGLE/DYNAMICWORLD/V1
// Definition: Pixel-level classification; class 6 = built area
// Purpose: Proxy for built-up extent
// Period: January 1 - December 31 (annual)
// Resolution: 10 m
// More info: https://developers.google.com/earth-engine/datasets/catalog/GOOGLE_DYNAMICWORLD
var dwCol         = ee.ImageCollection('GOOGLE/DYNAMICWORLD/V1')
                          .filterDate(startDate, endDate)
                          .select('label');
var dwAvail       = dwCol.size().gt(0);
var dwFallbackCol = ee.ImageCollection('GOOGLE/DYNAMICWORLD/V1')
                          .filterDate(fbStart, fbEnd)
                          .select('label');
var urbanBinary   = ee.Image(ee.Algorithms.If(
                          dwAvail,
                          dwCol.mode().eq(6),
                          dwFallbackCol.mode().eq(6)
                        ));
var urbanImg      = urbanBinary.multiply(100).rename('Urban_pct_2024');
var urbanYear     = ee.String(ee.Algorithms.If(dwAvail, targetYear, fallbackYear));

// NDVI (MODIS Terra)
// Dataset: MODIS/006/MOD13A2
// Definition: NDVI at 1 km; scale factor ×0.0001
// Purpose: Vegetation greenness proxy
// Period: January 1 - December 31 (annual)
// Resolution: 1 km
// More info: https://developers.google.com/earth-engine/datasets/catalog/MODIS_006_MOD13A2
var ndviCol         = ee.ImageCollection('MODIS/006/MOD13A2')
                          .filterDate(startDate, endDate)
                          .select('NDVI');
var ndviAvail       = ndviCol.size().gt(0);
var ndviFallbackCol = ee.ImageCollection('MODIS/006/MOD13A2')
                          .filterDate(fbStart, fbEnd)
                          .select('NDVI');
var ndviImg         = ee.Image(ee.Algorithms.If(
                          ndviAvail,
                          ndviCol.mean(),
```

```
                          ndviFallbackCol.mean()
                        )).multiply(0.0001)
                          .rename('NDVI_avg_2024');
var ndviYear        = ee.String(ee.Algorithms.If(ndviAvail, targetYear, fallbackYear));

// LAND-SURFACE TEMPERATURE (MODIS)
// Dataset: MODIS/061/MOD11A2
// Definition: 8-day mean daytime LST at 1 km; scale factor ×0.02; convert to °C by subtract
// Purpose: Thermal anomaly proxy
// Period: January 1 - December 31 (annual)
// Resolution: 1 km
// More info: https://developers.google.com/earth-engine/datasets/catalog/MODIS_061_MOD11A2
var lstCol         = ee.ImageCollection('MODIS/061/MOD11A2')
                          .filterDate(startDate, endDate)
                          .select('LST_Day_1km');
var lstAvail       = lstCol.size().gt(0);
var lstFallbackCol = ee.ImageCollection('MODIS/061/MOD11A2')
                          .filterDate(fbStart, fbEnd)
                          .select('LST_Day_1km');
var lstImg         = ee.Image(ee.Algorithms.If(
                          lstAvail,
                          lstCol.mean(),
                          lstFallbackCol.mean()
                        )).multiply(0.02)
                          .subtract(273.15)
                          .rename('LST_C_2024');
var lstYear        = ee.String(ee.Algorithms.If(lstAvail, targetYear, fallbackYear));

// PRECIPITATION TOTAL (CHIRPS)
// Dataset: UCSB-CHG/CHIRPS/DAILY
// Definition: Daily precipitation sum (mm)
// Purpose: Wet/dry anomaly proxy
// Period: January 1 - December 31 (annual)
// Resolution: ~5 km
// More info: https://developers.google.com/earth-engine/datasets/catalog/UCSB_CHG_CHIRPS_DA
var prcpCol         = ee.ImageCollection('UCSB-CHG/CHIRPS/DAILY')
                            .filterDate(startDate, endDate);
var prcpAvail       = prcpCol.size().gt(0);
var prcpFallbackCol = ee.ImageCollection('UCSB-CHG/CHIRPS/DAILY')
                            .filterDate(fbStart, fbEnd);
var prcpImg         = ee.Image(ee.Algorithms.If(
                            prcpAvail,
```

```
                             prcpCol.sum(),
                             prcpFallbackCol.sum()
                           )).rename('Precip_mm_2024');
var prcpYear        = ee.String(ee.Algorithms.If(prcpAvail, targetYear, fallbackYear));

// TROPOSPHERIC NO  (Sentinel-5P)
// Dataset: COPERNICUS/S5P/OFFL/L3_NO2
// Definition: Tropospheric NO  column density (µmol/m²)
// Purpose: Emissions proxy
// Period: January 1 - December 31 (annual)
// Resolution: ~7 km
// More info: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S5P_OFFL
var no2Col         = ee.ImageCollection('COPERNICUS/S5P/OFFL/L3_NO2')
                          .filterDate(startDate, endDate)
                          .select('tropospheric_NO2_column_number_density');
var no2Avail       = no2Col.size().gt(0);
var no2FallbackCol = ee.ImageCollection('COPERNICUS/S5P/OFFL/L3_NO2')
                          .filterDate(fbStart, fbEnd)
                          .select('tropospheric_NO2_column_number_density');
var no2Img         = ee.Image(ee.Algorithms.If(
                          no2Avail,
                          no2Col.mean(),
                          no2FallbackCol.mean()
                        )).rename('NO2_mol_m2_2024');
var no2Year        = ee.String(ee.Algorithms.If(no2Avail, targetYear, fallbackYear));

// SOIL-MOISTURE (GLDAS-2.1 NOAH)
// Dataset: NASA/GLDAS/V021/NOAH/G025/T3H
// Definition: Soil moisture top 10 cm (SoilMoi0_10cm_inst)
// Purpose: Surface soil-water proxy
// Period: January 1 - December 31 (annual)
// Resolution: ~25 km
// More info: https://developers.google.com/earth-engine/datasets/catalog/NASA_GLDAS_V021_NOA
var smCol         = ee.ImageCollection('NASA/GLDAS/V021/NOAH/G025/T3H')
                          .filterDate(startDate, endDate)
                          .select('SoilMoi0_10cm_inst');
var smAvail       = smCol.size().gt(0);
var smFallbackCol = ee.ImageCollection('NASA/GLDAS/V021/NOAH/G025/T3H')
                          .filterDate(fbStart, fbEnd)
                          .select('SoilMoi0_10cm_inst');
var smImg         = ee.Image(ee.Algorithms.If(
                          smAvail,
```

```
                          smCol.mean(),
                          smFallbackCol.mean()
                        )).rename('SoilM_m3m3_2024');
var smYear        = ee.String(ee.Algorithms.If(smAvail, targetYear, fallbackYear));

// ELEVATION & SLOPE (Copernicus GLO-30 Latest)
// Dataset: COPERNICUS/DEM/GLO30
// Definition: Digital elevation model; Slope derived from DEM
// Purpose: Terrain ruggedness proxy
// Epoch: 2024 (latest GLO-30 mosaic)
// Resolution: 30 m
// More info: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_DEM_GLO3
var demImg        = ee.ImageCollection('COPERNICUS/DEM/GLO30')
                          .select('DEM')
                          .mosaic()
                          .rename('Elevation_m_2024');
var slopeImg      = ee.Terrain.slope(demImg)
                          .rename('Slope_deg_2024');
var topoYear      = targetYear;

// ====================================================================
// 2. COMBINE & REDUCE
// ====================================================================
var multi = ee.Image.cat([
  viirsImg, urbanImg, ndviImg,
  lstImg, prcpImg, no2Img, smImg,
  demImg, slopeImg
]);

var countiesWithMeta = simplifiedCounties.map(function(f) {
  return f.set({
    'VIIRS_year' : viirsYear,
    'Urban_year' : urbanYear,
    'NDVI_year'  : ndviYear,
    'LST_year'   : lstYear,
    'Precip_year': prcpYear,
    'NO2_year'   : no2Year,
    'SoilM_year' : smYear,
    'Topo_year'  : topoYear
  });
});
```

```
var stats = multi.reduceRegions({
  collection: countiesWithMeta,
  reducer: ee.Reducer.mean(),
  scale: 1000,
  crs: 'EPSG:4326'
});

// ===================================================================
// 3. EXPORT & VISUALISE
// ===================================================================
Export.table.toDrive({
  collection : stats,
  description: 'Sweden_Annual_Geoindicators_2024',
  folder     : 'unsae',
  fileFormat : 'CSV'
});

Map.centerObject(simplifiedCounties);
Map.addLayer(viirsImg, {min:0, max:50}, 'VIIRS 2024');
Map.addLayer(urbanImg, {min:0, max:100}, 'Urban % 2024');
Map.addLayer(ndviImg, {min:0, max:0.8}, 'NDVI 2024');
Map.addLayer(lstImg, {min:-10, max:35}, 'LST 2024');
Map.addLayer(prcpImg, {}, 'Precip 2024');
Map.addLayer(no2Img, {}, 'NO2 2024');
Map.addLayer(smImg, {}, 'SoilM 2024');
Map.addLayer(demImg, {}, 'Elevation 2024');
Map.addLayer(slopeImg, {}, 'Slope 2024');
```

2. Direct Unemployment Estimates (`1_SCBdirect_estimate.py`)

Purpose: Retrieve and process direct unemployment counts per county from Statistics Sweden (SCB) API.

Key Steps:

1. **Library Imports:** Use `requests`, `pandas`, and `json` for HTTP calls and data handling.
2. **API Authentication:** No key required; set headers for JSON content.
3. **Endpoint Definition:** Define URL and query parameters to request unemployment by county and year.

4. **Data Fetching:** Loop over target years, send POST requests, and collect JSON responses.
5. **Data Cleaning:** Normalize nested JSON into tabular form, rename columns for clarity, and filter out incomplete records.
6. **Output:** Save combined DataFrame as `direct_unemployment.csv` in the data folder.

```
import json
import requests
import pandas as pd
from io import StringIO
from pathlib import Path

# ------------------- 1. Load the Query Object and Table ID -------------------

config_path = Path("data/directEstimate.json")

with config_path.open("r", encoding="utf-8") as f:
    cfg = json.load(f)

payload = cfg["queryObj"]
table_id = cfg["tableIdForQuery"]

# ------------------- 2. Build the API URL -------------------

url = f"https://api.scb.se/OV0104/v1/doris/en/ssd/START/AM/AM0401/AM0401N/{table_id}"
headers = {"Content-Type": "application/json"}

# ------------------- 3. Fetch the Data -------------------

r = requests.post(url, json=payload, headers=headers)
r.raise_for_status()  # Raise an error if the request failed

# ------------------- 4. Parse According to Response Format -------------------

fmt = payload.get("response", {}).get("format", "").lower()

# Make sure 'data/' directory exists
data_dir = Path("data")
data_dir.mkdir(parents=True, exist_ok=True)

if fmt == "csv":
```

```
    # If the response is CSV format, read it into a DataFrame
    df = pd.read_csv(StringIO(r.text), sep=",")

    # Pick and rename the important columns
    df_selected = df.rename(columns={
        "region": "County",
        "Percent 2025K1": "Percent_2025K1",
        "Margin of error ± percent 2025K1": "Percent_2025K1_me"
    })[["County", "Percent_2025K1", "Percent_2025K1_me"]]

    # Clean the 'County' names: remove leading numbers and "county" word
    df_selected["County"] = (
        df_selected["County"]
        .str.replace(r"^\d+\s+", "", regex=True)     # Remove leading numbers and spaces
        .str.replace(r"\scounty$", "", regex=True)   # Remove " county" at the end
        .str.strip()                                 # Remove any leading/trailing whitespace
    )

    # Save the cleaned DataFrame to CSV
    output_filename = "direct_estimates.csv"
    out_csv = data_dir / output_filename
    df_selected.to_csv(out_csv, index=False)
    print(f" Data saved to {out_csv.resolve()}")

elif fmt == "px":
    # If the response is PX format, save it as a PX file
    output_filename = f"{table_id}.px"
    out_px = data_dir / output_filename
    out_px.write_text(r.text, encoding="utf-8")
    print(f" PC-Axis file saved to {out_px.resolve()}")

    # (Optional) Code example to read PX file later:
    # import pxpy
    # table = pxpy.read_pxd(str(out_px))
    # df = table.to_dataframe()
    # print(df.head())

else:
    # If the format is unknown, print the raw response
    print(" Response format not recognized. Here is the raw text:")
    print(r.text)
```

3. Population Density Data (2_SCBpopDensity.py)

Purpose: Calculate population density per county by combining population counts and land area from SCB.

Key Steps:

1. **Imports:** `requests`, `pandas`, and `numpy`.
2. **Population Query:** Request `BE/BE0101` table for population totals by county.
3. **Area Query:** Request `KM/KT0103` table for land area in square kilometers.
4. **Merging Data:** Join population and area tables on county codes.
5. **Density Calculation:** Compute `density = population / area` (persons per km^2).
6. **Data Validation:** Check for zero or missing areas to avoid division errors.
7. **Output:** Export `population_density.csv` for downstream merging.

```python
import json
import requests
import pandas as pd
import re
from io import StringIO
from pathlib import Path

# 1. Load the query object and table ID from disk
config_path = Path("data/popdensity.json")
with config_path.open("r", encoding="utf-8") as f:
    cfg = json.load(f)

payload = cfg["queryObj"]
table_id = cfg["tableIdForQuery"]

# 2. Build the URL dynamically
url = f"https://api.scb.se/OV0104/v1/doris/en/ssd/START/BE/BE0101/BE0101C/{table_id}"

headers = {"Content-Type": "application/json"}

# 3. Fetch the data
r = requests.post(url, json=payload, headers=headers)
r.raise_for_status()

# 4. Parse according to response format
```

```
fmt = payload.get("response", {}).get("format", "").lower()

# 5. Handle CSV response
if fmt == "csv":
    df = pd.read_csv(StringIO(r.text), sep=",")

    # Pick and rename columns appropriately
    df_selected = df.rename(columns={
        "region": "County",
        "Population density per sq. km 2024": "PopDensity_2024"
    })[["County", "PopDensity_2024"]]

    # Clean 'County' names
    df_selected["County"] = (
        df_selected["County"]
        .str.replace(r"^\d+\s+", "", regex=True)     # remove leading numbers
        .str.replace(r"\scounty$", "", regex=True, flags=re.IGNORECASE)   # remove "county" a
        .str.strip()
    )

    # Ensure 'data/' directory exists
    data_dir = Path("data")
    data_dir.mkdir(parents=True, exist_ok=True)

    # Save the cleaned DataFrame
    output_filename = "popdensity.csv"
    out_csv = data_dir / output_filename
    df_selected.to_csv(out_csv, index=False)
    print(f"Data saved to {out_csv.resolve()}")

elif fmt == "px":
    data_dir = Path("data")
    data_dir.mkdir(parents=True, exist_ok=True)

    output_filename = f"{table_id}.px"
    out_px = data_dir / output_filename
    out_px.write_text(r.text, encoding="utf-8")
    print(f"PC-Axis file saved to {out_px.resolve()}")

else:
    print("Response format not recognized. Here's the raw text:")
    print(r.text)
```

Data Preprocessing

4. Load Libraries (`R/1_load_libraries.R`)

Purpose: Ensure all required R packages are installed and loaded for spatial and statistical analysis.

Key Steps:

1. **Install Missing Packages:** Check for packages (`tidyverse`, `sf`, `spdep`, `INLA`, `gt`, `leaflet`, etc.) and install if absent.
2. **Library Loading:** Load each package into the R session.
3. **Version Logging:** Print package versions to console for reproducibility.

```r
# 1_load_libraries.R
# ----------------------------------------------------------
# 1. Library Loader
# ----------------------------------------------------------

# ----------------------------------------------------------
# 2. Required Packages
# ----------------------------------------------------------
# Vector of packages needed for this project
# Include all libraries used across preprocessing, visualization, and SAE scripts
target_pkgs <- c(
  "sf",          # spatial data handling
  "tidyverse",   # data manipulation & visualization
  "emdi",        # small area estimation
  "tmap",        # thematic mapping
  "glmnet",      # regularized regression
  "here",        # project-relative file paths
  "conflicted", # manage function conflicts
  "spdep",       # spatial dependence matrices and tests
  "corrplot"     # correlation plots
)

# ----------------------------------------------------------
# 3. Install Missing Packages
# ----------------------------------------------------------
toinstall <- setdiff(target_pkgs, rownames(installed.packages()))
if (length(toinstall) > 0) {
  message("Installing missing packages: ", paste(toinstall, collapse = ", "))
```

```
  install.packages(toinstall)
}

# ----------------------------------------------------------
# 4. Load Packages
# ----------------------------------------------------------
for (pkg in target_pkgs) {
  message("Loading package: ", pkg)
  library(pkg, character.only = TRUE)
}

# ----------------------------------------------------------
# 5. Resolve Name Conflicts
# ----------------------------------------------------------
# Ensure dplyr's filter and lag take precedence
conflict_prefer("filter", "dplyr")
conflict_prefer("lag",    "dplyr")
message("Function conflicts resolved: filter and lag from dplyr preferred.")

# ----------------------------------------------------------
# 6. Set Seed
# ----------------------------------------------------------
# For reproducibility of random operations
set.seed(2)
message("Random seed set to 2.")

# ----------------------------------------------------------
# 7. Data Path Helper
# ----------------------------------------------------------
# Usage: data_path("raw", "myfile.csv") -> <project_root>/data/raw/myfile.csv
data_path <- function(...) {
  here::here("data", ...)
}
message("Helper 'data_path()' defined for project-relative data paths.")
```

5. Sweden Data Preprocessing (`R/2_sweden_preprocess.R`)

Purpose: Read, clean, and merge spatial and tabular data to create the analysis-ready dataset.

Key Steps:

1. **Read Shapefiles:** Use `st_read` to import county boundaries (e.g., `Sweden_Counties.shp`).
2. **Import CSVs:** Load `direct_unemployment.csv` and `population_density.csv`.
3. **Data Cleaning:** Standardize county code formats, handle missing or NA values (e.g., impute or remove), and ensure coordinate reference systems match.
4. **Join Datasets:** Merge shapefile with unemployment and density tables by county code.
5. **Add Geoindicators:** Read GEE output CSVs and merge on county and year.
6. **Final Checks:** Validate geometry integrity (`st_is_valid`), and export as an RDS object (`Sweden_data.rds`).

```
# ----------------------------------------------------------
# Sweden Preprocess (Revised)
# ----------------------------------------------------------

# ----------------------------------------------------------
# 1. Source Library Loader
# ----------------------------------------------------------
source(here::here("R", "1_load_libraries.R"))

# ----------------------------------------------------------
# 2. Define Paths & Helper Functions
# ----------------------------------------------------------
paths <- list(
  shapefile   = data_path("SWE_adm", "SWE_adm1.shp"),
  direct_est  = data_path("direct_estimates.csv"),
  geodata     = data_path("geodata.csv"),
  popdensity  = data_path("popdensity.csv"),
  vacancies   = data_path("vacancies.csv")
)

# County name recoding (fix special characters)
name_map <- c("Orebro" = "Örebro")
recode_county <- function(x) {
  factor(dplyr::recode(as.character(x), !!!name_map))
}

# CSV reader with status messages
read_data <- function(path) {
  message("Reading: ", path)
  readr::read_csv(path, show_col_types = FALSE)
}
```

```r
# -----------------------------------------------------------
# 3. Load & Clean County Shapefile
# -----------------------------------------------------------
sweden_shape <-
  sf::st_read(paths$shapefile, quiet = TRUE) %>%
  sf::st_make_valid() %>%
  sf::st_transform(4326) %>%
  dplyr::select(NAME_1) %>%
  dplyr::mutate(NAME_1 = recode_county(NAME_1)) %>%
  dplyr::arrange(NAME_1)
message("Shapefile loaded: ", nrow(sweden_shape), " polygons")

# -----------------------------------------------------------
# 4. Read Direct Estimates & Compute Variance
# -----------------------------------------------------------
direct_est <-
  read_data(paths$direct_est) %>%
  dplyr::mutate(
    County          = recode_county(County),
    Percent         = na_if(Percent_2025K1, "..") %>% as.numeric() / 100,
    SE95            = na_if(Percent_2025K1_me, "..") %>% as.numeric() / 100,
    standard_error  = SE95 / 1.96,
    var_est         = standard_error^2,
    eff_sample_size = (Percent / standard_error)^2
  ) %>%
  dplyr::select(County, Percent, standard_error, var_est, eff_sample_size)
message("Direct estimates processed: ", nrow(direct_est), " records")

# -----------------------------------------------------------
# 5. Read & Clean Covariates
# -----------------------------------------------------------
# a) Geospatial covariates: select annual indicators (excluding precipitation)
geo_data <-
  read_data(paths$geodata) %>%
  dplyr::select(
    County         = NAME_1,
    VIIRS_avg      = VIIRS_avg_2024,
    Urban_pct      = Urban_pct_2024,
    NDVI_avg       = NDVI_avg_2024,
    LST_C          = LST_C_2024,
    NO2_mol_m2     = NO2_mol_m2_2024,
    SoilMoisture   = SoilM_m3m3_2024,
```

```r
    Elevation_m    = Elevation_m_2024,
    Slope_deg      = Slope_deg_2024
  ) %>%
  dplyr::mutate(
    County = recode_county(County),
    across(-County, as.numeric)
  )
message("Geospatial data loaded: ", nrow(geo_data), " rows with selected indicators")

# b) Population density
pop_density <-
  read_data(paths$popdensity) %>%
  dplyr::rename(
    County         = County,
    PopDensity     = PopDensity_2024
  ) %>%
  dplyr::mutate(
    County = recode_county(County),
    PopDensity = as.numeric(PopDensity)
  )
message("Population density loaded: ", nrow(pop_density), " rows")

# c) Job vacancies (latest period)
vacancies <-
  read_data(paths$vacancies) %>%
  dplyr::filter(Period == max(Period, na.rm = TRUE)) %>%
  dplyr::mutate(
    County      = stringr::str_remove(Län, " län$"),           # drop ' län'
    County      = stringr::str_remove(County, "s$")           # remove trailing 's'
    %>% stringr::str_to_title()                                  # title case
    %>% recode_county(),
    Vacancy_New = as.numeric(`Nya lediga jobb`)
  ) %>%
  dplyr::select(County, Vacancy_New)
message("Vacancies loaded: ", nrow(vacancies), " rows")

# ------------------------------------------------------------
# 6. Merge All Data
# ------------------------------------------------------------
northern_list <- c("Norrbotten", "Västerbotten", "Jämtland", "Västernorrland", "Gävleborg")
combined_data <-
  direct_est %>%
```

```r
  dplyr::left_join(geo_data,    by = "County") %>%
  dplyr::left_join(pop_density, by = "County") %>%
  dplyr::left_join(vacancies,   by = "County") %>%
  dplyr::mutate(
    Northern = factor(
      ifelse(as.character(County) %in% northern_list, "North", "South"),
      levels = c("South", "North")
    )
  )
message("Data merged: ", nrow(combined_data), " rows with all covariates")

# -----------------------------------------------------------
# 7. Prepare Spatial Join for Mapping
# -----------------------------------------------------------
sweden_map_data <-
  sweden_shape %>%
  dplyr::left_join(combined_data, by = c("NAME_1" = "County"))
missing_count <- sum(is.na(sweden_map_data$Percent))
if (missing_count > 0) {
  warning(missing_count, " features missing data after join")
}
message("Spatial join complete: ", nrow(sweden_map_data), " features")

# -----------------------------------------------------------
# 8. Save Processed Data
# -----------------------------------------------------------
dir.create(here::here("data"), showWarnings = FALSE)
save(
  sweden_shape,
  combined_data,
  sweden_map_data,
  file = data_path("processed_sweden.RData")
)
message("Processed data saved to ", data_path("processed_sweden.RData"))
```

6. Mapping Direct Estimates (`R/3_visualization.R`)

Purpose: Generate both static and interactive maps to visualize direct unemployment estimates.

Key Steps:

1. **Load Data:** Read `Sweden_data.rds`.
2. **Static Maps:** Use `ggplot2`:
   - Create choropleth layers with `geom_sf`.
   - Customize fill scales (`scale_fill_viridis_c`), legends, and titles.
3. **Interactive Maps:** Use `leaflet`:
   - Convert sf object to `leaflet` object.
   - Add polygons with `addPolygons`, tooltips for county names and values.
   - Integrate base maps (e.g., CartoDB.Positron).
4. **Save Outputs:** Export static maps as PNG and interactive HTML widgets.

```r
# ----------------------------------------------------------
# 3_visualization.R (Revised)
# ----------------------------------------------------------

# 1. Source Libraries & Data
# ----------------------------------------------------------
library(here)
source(here("R", "1_load_libraries.R"))     # Loads sf, tidyverse, tmap, etc.
source(here("R", "2_sweden_preprocess.R"))  # Loads sweden_map_data

# 2. Prepare Map Data
# ----------------------------------------------------------
# Format percent labels
sweden_map_data <- sweden_map_data %>%
  dplyr::mutate(
    Percent_label = scales::label_percent(accuracy = 0.1)(Percent)
  )

# 3. Ensure Output Directories
# ----------------------------------------------------------
out_html <- here("outputs", "html")
out_img  <- here("outputs", "img")
dir.create(out_html, recursive = TRUE, showWarnings = FALSE)
dir.create(out_img,  recursive = TRUE, showWarnings = FALSE)

# 4. Interactive Map with tmap + Leaflet
# ----------------------------------------------------------
tmap::tmap_mode("view")
```

```
tm_direct <-
  tm_shape(sweden_map_data) +
  tm_polygons(
    col        = "Percent",
    palette    = "Blues",
    border.col = "grey20",
    alpha      = 0.7,
    title      = "Unemployment Rate"
  ) +
  tm_text(
    text           = "Percent_label",
    size           = "AREA",
    remove.overlap = TRUE,
    bg.color       = "white",
    bg.alpha       = 0.5
  ) +
  tm_layout(
    main.title      = "Direct Unemployment Estimates by County",
    main.title.size = 1.2,
    legend.outside  = TRUE,
    frame           = FALSE
  )
print(tm_direct)

leaflet_map <- tmap_leaflet(tm_direct)
htmlwidgets::saveWidget(
  widget        = leaflet_map,
  file          = here(out_html, "sweden_direct_map.html"),
  selfcontained = TRUE
)

# 5. Static Thematic Map (tmap → PNG)
# -----------------------------------------------------------
tmap::tmap_mode("plot")
tmap::tmap_save(
  tm_direct,
  filename = here(out_img, "sweden_direct_map_tmap.png"),
  dpi      = 300,
  width    = 8, height = 6, units = "in"
)

# 6. Static Map with ggplot2
```

```r
# ----------------------------------------------------------
sweden_proj <- sf::st_transform(sweden_map_data, crs = 3006)
static_map <- ggplot2::ggplot() +
  ggplot2::geom_sf(
    data  = sweden_proj,
    aes(fill = Percent),
    color = "grey20",
    alpha = 0.8
  ) +
  ggplot2::geom_sf_label(
    data          = sweden_proj,
    aes(label = Percent_label),
    size          = 3,
    label.padding = grid::unit(0.15, "lines"),
    fill          = "white"
  ) +
  scale_fill_viridis_c(name = "Unemployment (%)") +
  theme_minimal() +
  labs(
    title    = "Direct Unemployment Estimates by County",
    subtitle = "Sweden, 2025"
  ) +
  theme(
    panel.grid      = element_blank(),
    legend.position = "right"
  )
print(static_map)

ggplot2::ggsave(
  filename = here(out_img, "sweden_direct_map_ggplot.png"),
  plot     = static_map,
  dpi      = 300,
  width    = 8, height = 6, units = "in"
)

# 7. Save Visualization Objects
# ----------------------------------------------------------
dir.create(here("data"), recursive = TRUE, showWarnings = FALSE)
save(
  tm_direct,
  static_map,
  file = data_path("visual.RData")
```

```
)
message("Visualization outputs saved under 'outputs/' and objects to ", data_path("visual.RDa


# # ----------------------------------------------------------
# # 3_visualization.R (Updated)
# # ----------------------------------------------------------

# # ----------------------------------------------------------
# # 1. Source Libraries and Data
# # ----------------------------------------------------------
# library(here)
# source(here("R", "1_load_libraries.R"))    # Loads tidyverse, tmap, sf, etc.
# source(here("R", "2_sweden_preprocess.R")) # Loads sweden_map_data

# # ----------------------------------------------------------
# # 2. Prepare Map Data
# # ----------------------------------------------------------
# # Create formatted labels using updated 'Percent'
# sweden_map_data <- sweden_map_data %>%
#   dplyr::mutate(
#     Percent_label = scales::label_percent(accuracy = 0.1)(Percent)
#   )

# # Ensure outputs directory exists for exported files
# output_dir <- here("outputs")
# if (!dir.exists(output_dir)) dir.create(output_dir, recursive = TRUE)

# # ----------------------------------------------------------
# # 3. Interactive Map (tmap + Leaflet)
# # ----------------------------------------------------------
# tmap::tmap_mode("view")

# # Build and display interactive map with 'Percent'
# tm_direct <-
#   tm_shape(sweden_map_data) +
#   tm_polygons(
#     col        = "Percent",
#     palette    = "Blues",
#     border.col = "grey20",
#     alpha      = 0.7,
#     title      = "Unemployment Rate"
```

```
#   ) +
#   tm_text(
#     text           = "Percent_label",
#     size           = "AREA",
#     remove.overlap = TRUE,
#     bg.color       = "white",
#     bg.alpha       = 0.5
#   ) +
#   tm_layout(
#     main.title      = "Direct Unemployment Estimates by County",
#     main.title.size = 1.2,
#     legend.outside  = TRUE,
#     frame           = FALSE
#   )
# print(tm_direct)

# # Save interactive map as standalone HTML
# leaflet_map <- tmap_leaflet(tm_direct)
# htmlwidgets::saveWidget(
#   leaflet_map,
#   file          = file.path(output_dir, "sweden_direct_map.html"),
#   selfcontained = TRUE
# )

# # ---------------------------------------------------------
# # 4. Export Static Tmap Map as PNG
# # ---------------------------------------------------------
# tmap::tmap_mode("plot")
# tmap::tmap_save(
#   tm_direct,
#   filename = file.path(output_dir, "sweden_direct_map.png"),
#   dpi      = 300,
#   width    = 8,
#   height   = 6,
#   units    = "in"
# )

# # ---------------------------------------------------------
# # 5. Static Map (ggplot2)
# # ---------------------------------------------------------
# # Reproject for accurate spatial labeling
# sweden_proj <- sf::st_transform(sweden_map_data, crs = 3006)
```

```
# static_map <- ggplot2::ggplot() +
#   ggplot2::geom_sf(
#     data  = sweden_proj,
#     aes(fill = Percent),
#     color = "grey20",
#     alpha = 0.8
#   ) +
#   ggplot2::geom_sf_label(
#     data          = sweden_proj,
#     aes(label = Percent_label),
#     size          = 3,
#     label.padding = grid::unit(0.15, "lines"),
#     fill          = "white"
#   ) +
#   scale_fill_viridis_c(name = "Unemployment (%)") +
#   theme_minimal() +
#   labs(
#     title    = "Direct Unemployment Estimates by County",
#     subtitle = "Sweden, 2025"
#   ) +
#   theme(
#     panel.grid      = element_blank(),
#     legend.position = "right"
#   )
# print(static_map)

# # Export static ggplot map as high-resolution PNG
# ggsave(
#   filename = file.path(output_dir, "sweden_direct_map_static.png"),
#   plot     = static_map,
#   dpi      = 300,
#   width    = 8,
#   height   = 6,
#   units    = "in"
# )

# # ---------------------------------------------------------
# # 6. Save Visualization Objects
# # ---------------------------------------------------------
# save(
#   tm_direct,
#   static_map,
```

```
#   file = here("data", "visual.RData")
# )
# message("Visualization scripts updated to use 'Percent' variable")
```

Small-Area Estimation

7. Compute Small-Area Estimates (`R/test4.R`)

Purpose: Fit Fay–Herriot small-area models to improve precision of county-level unemployment estimates.

Key Steps:

1. **Load Data:** Read merged `Sweden_data.rds`.
2. **Exploratory Analysis:** Compute Moran's I (`spdep`) for spatial autocorrelation and VIF for collinearity checks.
3. **Model Fitting:**
   - **Untransformed Models:** Apply `lme` or `inla` for Fay–Herriot on the raw unemployment rates.
   - **Transformed Models:** Log-transform rates, refit models, and back-transform estimates.
4. **Diagnostics:** Plot residual vs. fitted values, Q-Q plots, and calculate MSE and CV for each area.
5. **Result Tables:** Use `gt` to create publication-ready tables of fixed effects coefficients and error metrics.
6. **Mapping SAE:** Add model-based estimates and CV layers to the sf object, and export final shapefile (`SAE_Results.shp`).

```
# ----------------------------------------------------------
# SAE Modeling
# ----------------------------------------------------------

# ----------------------------------------------------------
# 1. Load Libraries and Preprocessing
# ----------------------------------------------------------
library(here)

# Source shared scripts
```

```r
source(here("R", "1_load_libraries.R"))    # common packages
source(here("R", "2_sweden_preprocess.R")) # defines data

#  Setup Output Directory
out_dir <- file.path(here("outputs"), "sae")
dir.create(out_dir, recursive = TRUE, showWarnings = FALSE)

# ---------------------------------------------------------
# 2. Spatial Diagnostics
# ---------------------------------------------------------
# Ensure fh_data ordered by domain
fh_data <- combined_data %>% arrange(County)
# Build spatial weights
nb  <- poly2nb(sweden_shape, row.names = sweden_shape$NAME_1)
W   <- nb2mat(nb, style = "W", zero.policy = TRUE)
idx <- which(!is.na(fh_data$Percent))
# Moran's I test
spatialcor.tests(
  direct    = fh_data$Percent[idx],
  corMatrix = W[idx, idx]
)

# ---------------------------------------------------------
# 3. Data Transformation
# ---------------------------------------------------------
# Convert and log-transform skewed covariates
fh_data <- combined_data %>%
  mutate(
    Percent         = as.numeric(Percent),
    var_est         = as.numeric(var_est),
    Elevation_log   = log(Elevation_m),
    LST_log         = log(LST_C),
    SoilMoist_log   = log(SoilMoisture),
    PopDensity_log  = log(PopDensity),
    Vacancy_log     = log(Vacancy_New)
  )
# Covariate names (use log-transformed variables where applied)
covariates <- c(
  "Elevation_log",    # log(Elevation_m)
  "LST_log",          # log(LST_C)
  "SoilMoist_log",    # log(SoilMoisture)
  "PopDensity_log",   # log(PopDensity)
```

```r
  "Vacancy_log",      # log(Vacancy_New)
  "NDVI_avg",         # Vegetation index
  "NO2_mol_m2",       # NO2 density (mol/m²)
  "Slope_deg",        # Terrain slope (°)
  "Urban_pct",        # Urban cover (%)
  "VIIRS_avg",        # Night-time lights
  "Northern"          # Regional factor
)

# ----------------------------------------------------------
# 4. Correlation Matrix
# ----------------------------------------------------------
# 4.1 Prepare data and rename for readable labels
corr_vars <- c("Percent", covariates[covariates != "Northern"])
corr_data <- fh_data %>%
  select(all_of(corr_vars)) %>%
  rename(
    `Unemp. Rate`       = Percent,
    `Elevation (log)`   = Elevation_log,
    `LST (log)`         = LST_log,
    `NDVI`              = NDVI_avg,
    `NO2`               = NO2_mol_m2,
    `Slope`             = Slope_deg,
    `Soil Moist. (log)` = SoilMoist_log,
    `Urban (%)`         = Urban_pct,
    `Night Lights`      = VIIRS_avg,
    `Pop Density (log)` = PopDensity_log,
    `Vacancy (log)`     = Vacancy_log
  )

# 4.2 Compute correlation matrix
corr_mat <- cor(corr_data, use = "pairwise.complete.obs")

# 4.3 Plot correlation matrix with original order and labels
out_dir_corr <- file.path(here("outputs"), "sae")
if (!dir.exists(out_dir_corr)) dir.create(out_dir_corr, recursive = TRUE)
png(file.path(out_dir_corr, "correlation_matrix.png"), width = 2000, height = 2000, res = 20
corrplot(
  corr_mat,
  method      = "color",
  type        = "upper",
  order       = "original",
```

```
  addCoef.col = "black",
  tl.col      = "black",
  tl.cex      = 0.8,
  tl.srt      = 45
)
dev.off()

# ----------------------------------------------------------
# 5. Full Fay-Herriot Model Fits
# ----------------------------------------------------------

fh_data <- fh_data %>%
  dplyr::mutate(
    Percent   = as.numeric(Percent),
    var_est   = as.numeric(var_est)
  ) %>%
  as.data.frame()
# Define full formula
formula_full <- as.formula(paste("Percent ~", paste(covariates, collapse = " + ")))

# Initial untransformed model (REML)
fh_initial <- emdi::fh(
  fixed          = formula_full,
  vardir         = "var_est",
  combined_data = fh_data,
  domains        = "County",
  method         = "reml",
  interval       = c(0, 100),
  B              = 1000,
  MSE            = TRUE
)

summary(fh_initial)

# Transformed initial models
fh_arc_init <- emdi::fh(
  fixed                = formula_full,
  vardir               = "var_est",
  combined_data        = fh_data,
  domains              = "County",
  transformation       = "arcsin",
  backtransformation = "bc",
```

```
  eff_smpsize         = "eff_sample_size",
  mse_type            = "boot",
  B                   = 1000,
  MSE                 = TRUE
)

summary(fh_arc_init)

fh_log_init <- emdi::fh(
  fixed               = formula_full,
  vardir              = "var_est",
  combined_data       = fh_data,
  domains             = "County",
  transformation      = "log",
  backtransformation = "bc_crude",
  eff_smpsize         = "eff_sample_size",
  mse_type            = "analytical",
  B                   = 1000,
  MSE                 = TRUE
)

summary(fh_log_init)

fh_logit_init <- emdi::fh(
  fixed               = formula_full,
  vardir              = "var_est",
  combined_data       = fh_data,
  domains             = "County",
  transformation      = "logit",
  backtransformation = "bc",
  eff_smpsize         = "eff_sample_size",
  mse_type            = "boot",
  B                   = 1000,
  MSE                 = TRUE
)

summary(fh_logit_init)

# ----------------------------------------------------------
# 6. Reduced Models Fay-Herriot Model
# ----------------------------------------------------------
# Fit full-ML for BIC selection
```

```r
fh_full_ml <- emdi::fh(
  fixed         = formula_full,
  vardir        = "var_est",
  combined_data = fh_data,
  domains       = "County",
  method        = "ml",
  B             = 1000,
  MSE           = TRUE
)
# Stepwise backward BIC
fh_step <- emdi::step(
  object    = fh_full_ml,
  criteria  = "BIC",
  direction = "backward",
  B         = 1000,
  MSE       = TRUE
)
# Extract reduced formula
formula_step <- fh_step$fixed

# Refit reduced models under REML
fh_red_init  <- emdi::fh(
  fixed = formula_step, vardir = "var_est", combined_data = fh_data,
  domains = "County", method = "reml", B = 1000, MSE = TRUE
)
fh_red_arc   <- update(fh_arc_init,   fixed = formula_step)
fh_red_log   <- update(fh_log_init,   fixed = formula_step)
fh_red_logit <- update(fh_logit_init, fixed = formula_step)

# -----------------------------------------------------------
# 7. Mapping, Comparison, and Diagnostics
# -----------------------------------------------------------
# Assemble models
models <- list(
  Initial     = fh_initial,
  ArcsinInit  = fh_arc_init,
  LogInit     = fh_log_init,
  LogitInit   = fh_logit_init,
  Stepwise    = fh_step,
  RedInitial  = fh_red_init,
  RedArcsin   = fh_red_arc,
  RedLog      = fh_red_log,
```

```
  RedLogit    = fh_red_logit
)

# Helper to rename estimate column
extract_FH <- function(df) {
  if ("FH" %in% names(df)) dplyr::rename(df, FH_est = FH)
  else if ("prediction" %in% names(df)) dplyr::rename(df, FH_est = prediction)
  else df
}

# Loop through models and create outputs
purrr::iwalk(models, function(mod, nm) {
  # 7.1 Prepare map data quietly
  md <- suppressMessages(
    map_plot(
      object     = mod,
      map_obj    = sweden_shape,
      map_dom_id = "NAME_1",
      indicator  = "FH",
      MSE        = TRUE,
      CV         = TRUE,
      return_data= TRUE
    )
  ) %>% extract_FH()

  # SAE Map
  ggsave(
    file.path(out_dir, paste0("sae_map_", nm, ".png")),
    ggplot(md) +
      geom_sf(aes(fill = FH_est), color = "grey20", alpha = 0.8) +
      scale_fill_viridis_c() + theme_minimal() +
      labs(title = paste("SAE Estimates -", nm), fill = "Estimate"),
    width  = 8, height = 6, dpi = 300
  )

  # Comparison Plot
  compare_path <- file.path(out_dir, paste0("compare_", nm, ".png"))
  png(compare_path, width = 800, height = 600)
  compare_plot(
    object        = mod,
    combined_data = fh_data,
    indicator     = "FH",
```

```
    MSE            = TRUE,
    CV             = TRUE
  )
  dev.off()
})

# # Collect models
# models <- list(
#   Initial     = fh_initial,
#   ArcsinInit  = fh_arc_init,
#   LogInit     = fh_log_init,
#   LogitInit   = fh_logit_init,
#   Stepwise    = fh_step,
#   RedInitial  = fh_red_init,
#   RedArcsin   = fh_red_arc,
#   RedLog      = fh_red_log,
#   RedLogit    = fh_red_logit
# )
#
# out_dir <- here("outputs")
# for (nm in names(models)) {
#   # SAE map
#   md <- map_plot(
#     models[[nm]], map_obj = sweden_shape,
#     map_dom_id = "NAME_1", indicator = "FH", MSE = TRUE, CV = TRUE,
#     return_data = TRUE
#   ) %>% rename(FH_est = FH)
#
#   ggsave(
#     filename = file.path(out_dir, paste0("sae_map_", nm, ".png")),
#     plot     = ggplot(md) +
#                geom_sf(aes(fill = FH_est), color = "grey20", alpha = 0.8) +
#                scale_fill_viridis_c() + theme_minimal() +
#                labs(title = paste("SAE Estimates -", nm), fill = "Est"),
#     width    = 8, height = 6, dpi = 300
#   )
#
#   # Compare plot
#   png(file.path(out_dir, paste0("compare_", nm, ".png")), width = 800, height = 600)
#   compare_plot(models[[nm]], CV = TRUE, MSE = TRUE)
#   dev.off()
# }
```

```r
# ------------------------------------------------------------
# 8. Coefficient Extraction and Saving
# ------------------------------------------------------------
extract_coefs <- function(obj) {
  sm <- summary(obj)
  df <- as.data.frame(sm$model$coefficients, stringsAsFactors = FALSE)
  df$term <- rownames(df)
  df %>% rename(
    estimate  = coefficients,
    std.error = std.error,
    t.value   = t.value,
    p.value   = p.value
  ) %>% select(term, estimate, std.error, t.value, p.value)
}
coefs <- purrr::imap_dfr(models, ~ extract_coefs(.x) %>% mutate(model = .y))

# Save outputs
dir.create(out_dir, showWarnings = FALSE)
saveRDS(models, file = file.path(out_dir, "fh_models.rds"))
write.csv(coefs, file = file.path(out_dir, "fh_coefficients.csv"), row.names = FALSE)

message("test4.R optimized and styled with comment banners.")

## Export final model

unemp = estimators(fh_red_init, MSE = T, CV = T) %>%
  as.data.frame() %>%
  left_join(sweden_shape, by = c("Domain" = "NAME_1"))

saveRDS(unemp, file = file.path(out_dir, "unemp.rds"))
```