Applied Quantitative Methods for the Social Sciences II
MA in Social Sciences, UC3M, Spring 2026

Assignment 7: Spatial Data I

Instructions:

- **Deadline**: **[DEADLINE - TBD], before class**
- Submit your work in a separate folder in your GitHub repository
  - You can include only the R file or additional ones (e.g. pdf with results)
- **Always use comments** in your R code – and use them to answer questions
- You are encouraged to work together, but each person must submit their own code
- Plan is to start Part 1 in class and complete Part 2 at home
- I'll upload a solution file to the website after next class

Contents

1 Part 1: In-Class (Exploring Spatial Data with `sf`)

In this lab we work with the `world` dataset from the `spData` package, which contains country polygons for the entire world along with socioeconomic attributes. This dataset is ideal for a first encounter with `sf` objects: it loads directly into R (no file download needed), uses real data, and illustrates the key operations you will use throughout the spatial analysis part of the course.

Load the required packages and data with:

```
library(sf)
library(spData)
data(world)
```

Key variables in `world`:

- `name_long` — country name (long form)
- `continent` — continent
- `area_km2` — country area in km^2
- `pop` — population estimate
- `gdpPercap` — GDP per capita
- `lifeExp` — life expectancy
- `geom` — geometry column (sf)

1.1 Inspecting an `sf` object

a) Load the `world` dataset and inspect its structure. Run `class(world)`, `names(world)`, and `nrow(world)`. In a comment, describe what makes an `sf` object different from a regular R data frame. What is the geometry column, and how is it stored?

b) Check the coordinate reference system (CRS) with `st_crs(world)`. What EPSG code does the dataset use? In a comment, explain what WGS84 means and why it is the standard CRS for global geographic data.

c) Use `st_geometry_type(world)` and `unique(st_geometry_type(world))` to inspect the geometry type. In a comment, explain what a MULTIPOLYGON is and give two concrete examples of countries that would require multiple polygons to represent their territory.

d) Produce a quick map of GDP per capita using base R graphics:

```
pdf("world_gdp_base.pdf")
plot(world["gdpPercap"])
dev.off()
```

In a comment, describe what you see. Which regions appear wealthiest and which poorest?

1.2 Attribute operations

A key feature of sf objects is that standard dplyr verbs work on them directly, operating on the attribute table while automatically carrying the geometry column along.

a) Using filter(), create a subset of world containing only African countries. Call it africa. How many African countries are in the dataset? Plot africa["gdpPercap"] using base graphics. In a comment, note whether the country count matches your expectations.

b) Add a new variable pop_millions equal to population divided by 1,000,000 using mutate(). Then compute the average GDP per capita by continent using group_by() and summarise():

```
library(dplyr)

world = world %>%
  mutate(pop_millions = pop / 1e6)

gdp_by_continent = world %>%
  group_by(continent) %>%
  summarise(mean_gdpPercap = mean(gdpPercap, na.rm = TRUE))

print(gdp_by_continent)
```

In a comment, note that summarise() on a grouped sf object *unions* the geometries by group and keeps the geometry column. To get a plain data frame without geometry, use st_drop_geometry() first.

c) Sort the African countries by GDP per capita (descending) using arrange(). Print the top 5 rows with name_long and gdpPercap. Name the five countries in a comment.

1.3 Simple visualization with ggplot2

The geom_sf() layer in ggplot2 allows you to plot sf objects using the standard grammar of graphics.

a) Make a choropleth map of the world colored by gdpPercap:

```
library(ggplot2)

ggplot(world) +
  geom_sf(aes(fill = gdpPercap)) +
  scale_fill_viridis_c(option = "plasma", na.value = "grey80",
                       name = "GDP per capita") +
  theme_void() +
  labs(title = "GDP per capita by country")
ggsave("world_gdp.pdf", width = 10, height = 5)
```

In a comment, describe the geographic pattern. Which regions appear wealthiest? Which appear poorest?

b) Make the same map restricted to the `africa` object. Use `scale_fill_viridis_c()` with `option = "magma"` and save as `africa_gdp.pdf`. Describe the variation in GDP per capita across African countries.

c) Improve the Africa map by adding white country borders: modify `geom_sf()` to include `color = "white"` and `linewidth = 0.3`. Save as `africa_gdp_borders.pdf`. In a comment, explain how the border layer improves readability.

2 Part 2: Take-Home (Point Data and Spatial Joins)

A key task in spatial data analysis is combining point data (events or observations with coordinates) with polygon data (regions or countries). We do this using *spatial joins*: each point is assigned the attributes of the polygon it falls within. In this part we work with geo-coded armed conflict event data.

Download the conflict events data from the course repository:

- github.com/franvillamil/AQM2/tree/master/datasets/spatial

Load it with `events = read.csv("conflict_events.csv")`. Key variables:

- `event_id` — unique event identifier
- `year` — year of the event
- `country` — country name (character; may not match `world` exactly)
- `longitude`, `latitude` — geographic coordinates (WGS84)
- `fatalities` — estimated number of fatalities
- `event_type` — type of event (e.g., battles, violence against civilians)

2.1 Converting tabular data to `sf`

a) Convert the `events` data frame to an `sf` object using:

```
events_sf = st_as_sf(events,
                     coords = c("longitude", "latitude"),
                     crs = 4326)
```

Run `class(events_sf)` and `st_crs(events_sf)` to verify it worked. In a comment, explain what `st_as_sf()` does: what does the `coords` argument specify, and what does `crs = 4326` mean?

b) How many events are in the dataset? Use `nrow()` and `table(events_sf$event_type)` to show the count by event type. In a comment, which event type is most common?

c) Make a map of conflict events overlaid on the world polygon:

```
ggplot() +
  geom_sf(data = world, fill = "grey90", color = "white", linewidth = 0.2) +
  geom_sf(data = events_sf, aes(color = event_type),
          size = 0.5, alpha = 0.4) +
  theme_void() +
  labs(title = "Armed conflict events", color = "Event type")
ggsave("conflict_events_map.pdf", width = 10, height = 5)
```

In a comment, describe the geographic pattern. In which regions are conflict events most concentrated?

2.2 Spatial join: events to countries

a) Use `st_join()` to assign country attributes to each conflict event:

```
# Verify both objects share the same CRS before joining
st_crs(events_sf) == st_crs(world)

events_joined = st_join(events_sf, world[, c("name_long", "continent", "
   gdpPercap")])
```

Run `nrow(events_joined)` and verify it equals `nrow(events_sf)`. In a comment, explain what `st_join()` is doing: how does it determine which country polygon each event point falls within? Why is checking the CRS before joining important?

b) Some events may not match any country polygon (e.g., events at sea, on islands, or exactly on a border). Check with `sum(is.na(events_joined$name_long))`. What fraction of events has no matching country? In a comment, list two possible reasons why a point might not match any polygon.

c) Count the number of events per country:

```
library(dplyr)

events_by_country = events_joined %>%
  filter(!is.na(name_long)) %>%
  group_by(name_long) %>%
  summarise(n_events = n(),
            total_fatalities = sum(fatalities, na.rm = TRUE)) %>%
  arrange(desc(n_events))

print(head(st_drop_geometry(events_by_country), 10))
```

Report the top 10 countries by number of events. In a comment, are the results consistent with your knowledge of contemporary armed conflicts?

2.3 Choropleth of conflict intensity

a) Join the event counts back to the world polygon data. Note that `events_by_country` is still an `sf` object, so we first drop its geometry before joining:

```
library(tidyr)

events_by_country_df = st_drop_geometry(events_by_country)

world_conflict = world %>%
  left_join(events_by_country_df, by = "name_long") %>%
  mutate(n_events = replace_na(n_events, 0),
         total_fatalities = replace_na(total_fatalities, 0))
```

```
```

Verify that `nrow(world_conflict) == nrow(world)`.

b) Make a choropleth map of conflict event counts by country:

```
ggplot(world_conflict) +
  geom_sf(aes(fill = n_events), color = "white", linewidth = 0.2) +
  scale_fill_distiller(palette = "Reds", direction = 1,
                       name = "N events", na.value = "grey80") +
  theme_void() +
  labs(title = "Armed conflict events by country")
ggsave("conflict_by_country.pdf", width = 10, height = 5)
```

In a comment, describe the map. Does the geographic pattern match the event-level map from question 2.1c?

c) Make a second map using log-transformed counts: use `log1p(n_events)` as the fill variable (so countries with zero events are handled). Use `scale_fill_distiller()` with `palette = "YlOrRd"`, `direction = 1`, and `name = "Log(events+1)"`. Save as `conflict_log_map.pdf`. In a comment, explain why the log transformation is useful and what it reveals that the raw count map did not.

2.4 Discussion

Answer the following questions as comments in your R script. Each answer should be 2–3 sentences.

a) Discuss one limitation of the spatial join approach used in this assignment. For example: what happens to events that fall exactly on the border between two countries? How might you handle events that fall just outside a polygon due to small coordinate imprecisions?

b) What is the difference between `st_join()` and `left_join()`? What information does each use to match rows, and when would you prefer one over the other?

3 Data Sources

- World polygons: `world` dataset in the `spData` R package (`library(spData); data(world)`)
- Conflict events: `github.com/franvillamil/AQM2/tree/master/datasets/spatial`

4 Submission

Commit your file to your GitHub repository before the deadline. Put it in a separate folder, e.g. `assignment7`. Make sure your repository is public so I can access it.

Your R script should:

- Be well-organized with clear section headers (using comments)
- Include all code needed to reproduce your analysis
- Include your answers and interpretations as comments
- Save any plots to files (using `pdf()`/`dev.off()` or `ggsave()`)
- Run without errors from top to bottom