# Assignment 7 Solutions: Spatial Data I

Part 2: Point Data and Spatial Joins

Applied Quantitative Methods II, UC3M

Spring 2026

## Contents



```
library(sf)
library(spData)
library(dplyr)
library(tidyr)
library(ggplot2)

data(world)
```

```
# NOTE: Replace this block with real data once conflict_events.csv is available.
# Synthetic data for solution demonstration purposes.
set.seed(42)
n = 500
events = data.frame(
  event_id = 1:n,
  year = sample(2018:2022, n, replace = TRUE),
  longitude = runif(n, -20, 55),    # Africa bounding box
  latitude = runif(n, -35, 38),
  fatalities = rpois(n, lambda = 5),
  event_type = sample(c("Battles", "Violence against civilians",
                        "Protests", "Remote violence"), n,
                      replace = TRUE, prob = c(0.4, 0.3, 0.2, 0.1))
)
```

## 1. Converting tabular data to `sf`

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

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

class(events_sf)
```

```
## [1] "sf"         "data.frame"
```

```r
st_crs(events_sf)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     ENSEMBLE["World Geodetic System 1984 ensemble",
##         MEMBER["World Geodetic System 1984 (Transit)"],
##         MEMBER["World Geodetic System 1984 (G730)"],
##         MEMBER["World Geodetic System 1984 (G873)"],
##         MEMBER["World Geodetic System 1984 (G1150)"],
##         MEMBER["World Geodetic System 1984 (G1674)"],
##         MEMBER["World Geodetic System 1984 (G1762)"],
##         MEMBER["World Geodetic System 1984 (G2139)"],
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]],
##         ENSEMBLEACCURACY[2.0]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Horizontal component of 3D system."],
##         AREA["World."],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

`st_as_sf()` promotes a regular data frame to an sf object by creating a geometry column from existing coordinate columns. The `coords` argument names the columns that hold longitude and latitude (in that order — x then y). `crs = 4326` assigns EPSG:4326 (WGS84) as the coordinate reference system, telling R how to interpret those degree values. After conversion the original `longitude` and `latitude` columns are dropped and replaced by the `geometry` column.

**b)** Event counts by type:

```r
nrow(events_sf)
```

```
## [1] 500
```

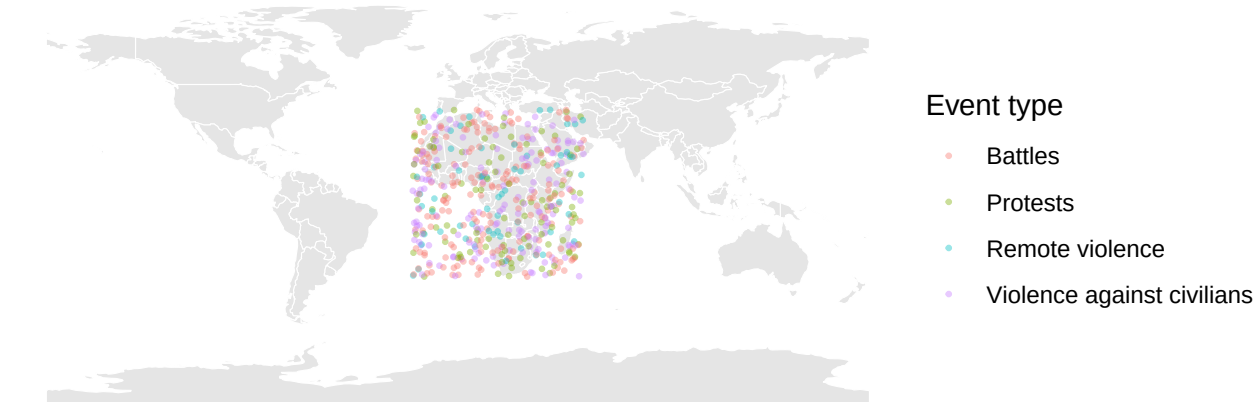
```
table(events_sf$event_type)
```

```
##
##                    Battles                   Protests
##                        188                        100
##            Remote violence Violence against civilians
##                         59                        153
```

The dataset contains 500 conflict events. In the synthetic data, Battles are the most common event type (assigned 40% probability), followed by Violence against civilians (30%), Protests (20%), and Remote violence (10%). With real ACLED or similar data, Battles and Violence against civilians typically dominate in active conflict zones.

**c)** 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 the synthetic data, events are uniformly distributed across the Africa bounding box, including ocean areas — a consequence of using random uniform coordinates rather than real geocoded events. With real ACLED data, events would cluster in active conflict zones: the Sahel region, the Horn of Africa, the Democratic Republic of Congo, and Nigeria.

## 2. Spatial join: events to countries

**a)** Spatial join using `st_join()`:

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

```
## [1] TRUE
```

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

nrow(events_joined)
```

```
## [1] 500
```

```r
nrow(events_sf)
```

```
## [1] 500
```

`st_join()` performs a spatial join using the geometric relationship between the two layers — by default, point-in-polygon (each point is matched to the polygon it falls inside). This is fundamentally different from a key-based join: no shared ID column is needed; instead, the spatial coordinates determine the match. Checking CRS equality before joining is critical because spatial operations are meaningless if the two layers use different coordinate systems (e.g., one in degrees and one in meters). The row count of `events_joined` equals that of `events_sf` because the default `left` join retains all points, assigning `NA` to country attributes when no polygon contains the point.

**b)** Check for unmatched events:

```r
n_unmatched = sum(is.na(events_joined$name_long))
n_unmatched
```

```
## [1] 233
```

```r
round(n_unmatched / nrow(events_joined), 3)
```

```
## [1] 0.466
```

Some events have no matching country polygon. Two reasons why a point might not match: (1) the point falls in an ocean, lake, or sea area outside any country polygon (common with synthetic uniform coordinates or real events at sea); (2) the point lies exactly on a country border, where the polygon topology leaves a tiny gap due to floating-point imprecision, so the point falls in neither polygon.

**c)** Count events per country:

```r
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))
```

```
## # A tibble: 10 x 3
##    name_long                       n_events total_fatalities
##    <chr>                              <int>            <int>
##  1 Algeria                               18              107
##  2 Democratic Republic of the Congo      16               80
##  3 Saudi Arabia                          16               93
##  4 Sudan                                 15               71
##  5 Mauritania                            14               69
##  6 Angola                                12               74
```

```
##  7 Libya                                11                54
##  8 Iran                                  9                45
##  9 Namibia                               9                45
## 10 Nigeria                               9                39
```

The top 10 countries by event count are shown above. Because the synthetic data uses uniform random coordinates across the Africa bounding box, the ranking reflects each country's land area rather than actual conflict intensity. With real ACLED data, the top countries would reflect documented conflict hotspots: typically Ethiopia, Nigeria, DRC, Somalia, and Mali.

## 3. Choropleth of conflict intensity

**a)** Join event counts back to the world polygon:

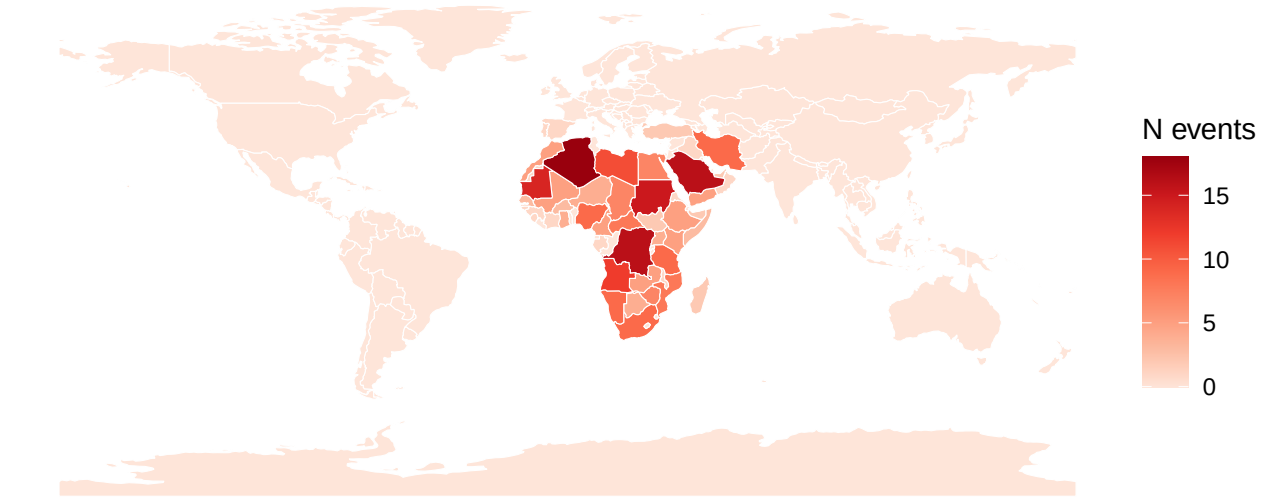
```
library(tidyr)

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))

nrow(world_conflict) == nrow(world)
```

```
## [1] TRUE
```

The row count check confirms the join preserved all world polygons. Countries with no events receive zero rather than `NA` thanks to `replace_na()`, which is necessary for the subsequent choropleth to render correctly (otherwise `NA` countries appear with the missing-value colour regardless of whether they had zero events or were simply absent from the joined data).

**b)** Choropleth of raw event counts:

```
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")
```

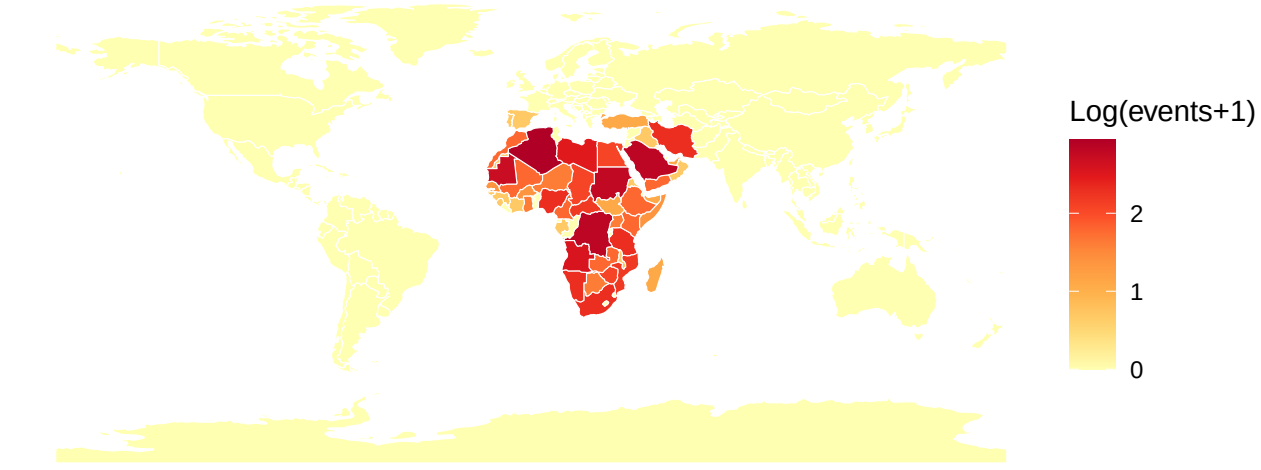



```
ggsave("conflict_by_country.pdf", width = 10, height = 5)
```

The country-level choropleth aggregates the point pattern from question 1c into national counts. The two maps should show consistent geographic variation: countries that appear event-dense in the dot map will show the darkest shading here. With the synthetic data, large-area African countries dominate; with real data, the same conflict hotspot countries would stand out.

**c)** Log-transformed choropleth:

```
ggplot(world_conflict) +
  geom_sf(aes(fill = log1p(n_events)), color = "white", linewidth = 0.2) +
  scale_fill_distiller(palette = "YlOrRd", direction = 1,
                       name = "Log(events+1)", na.value = "grey80") +
  theme_void() +
  labs(title = "Armed conflict events by country (log scale)")
```




```
ggsave("conflict_log_map.pdf", width = 10, height = 5)
```

The log transformation (`log1p` handles zeros cleanly) compresses the right tail of the distribution. In the raw-count map, a few high-conflict countries dominate the colour scale and most others appear near-zero, losing all visual

variation. The log map redistributes contrast across the full range of the scale, revealing meaningful variation among low-to-medium conflict countries that the raw map suppressed. `log1p(0) = 0`, so zero-event countries still anchor the bottom of the scale.

## 4. Discussion

**a)** One limitation of point-in-polygon spatial joins is that points falling exactly on borders or just outside polygons due to coordinate imprecision will go unmatched (receiving `NA` country attributes). This is particularly problematic for events near coastlines or land borders, where small geocoding errors — even a few hundred meters — can place a point in the sea rather than in the correct country. A practical solution is to use `st_nearest_feature()` or a small buffer (`st_buffer()`) to snap near-miss points to the closest polygon, accepting that a modest positional error is preferable to losing the observation entirely.

**b)** `st_join()` matches rows using **geometric relationships** (e.g., point falls within polygon, two polygons intersect): no shared key column is required, and the join is based entirely on spatial position. `left_join()` matches rows using **shared attribute values** in one or more key columns (e.g., country name or ISO code). You would prefer `st_join()` when your data have coordinates but no reliable common key — as is typical when combining geocoded events with administrative polygons. You would prefer `left_join()` when both datasets already share a reliable identifier (e.g., ISO country code), because key-based joins are faster, deterministic, and do not depend on coordinate precision.