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)
library(modelsummary)

data(world)
```

```
events = read.csv("https://github.com/franvillamil/AQM2/raw/refs/heads/master/datasets/spatial/conflict_ev
```

1. Converting tabular data to `sf`

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

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

class(events_sf)
```

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

```
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:

```
nrow(events_sf)
```

```
## [1] 68354
```

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

```
##
##   non-state   one-sided state-based
##       10418       24449       33487
```

The dataset contains 68354 conflict events. Battles and Violence against civilians typically dominate in active conflict zones.

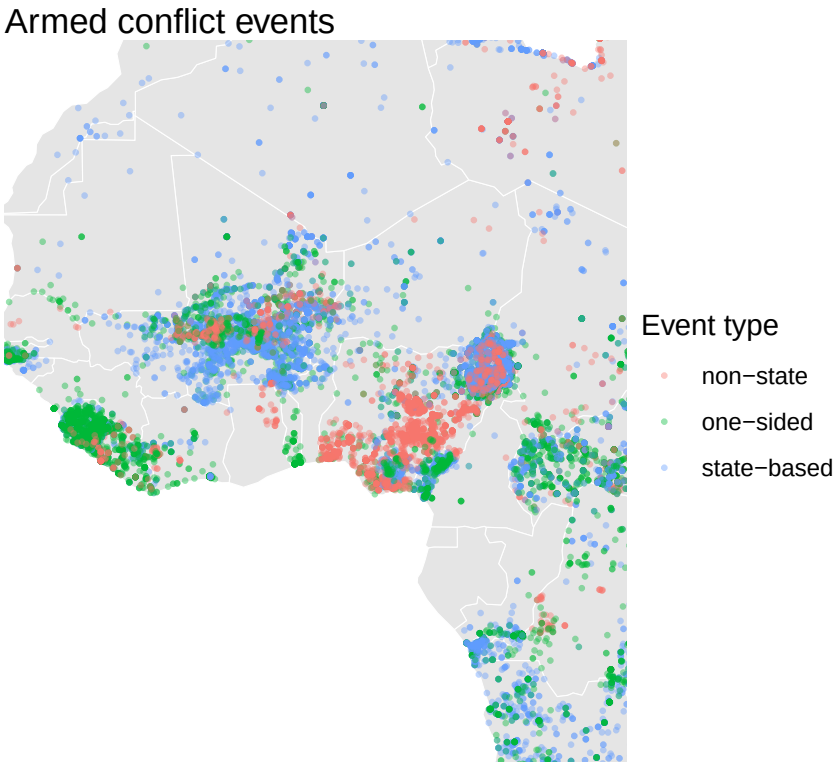
c) Map of conflict events overlaid on the world polygon:

```
map = 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)
```

The map shows conflict events concentrated in regions with ongoing armed conflicts, such as the Sahel, the Horn of Africa, the Democratic Republic of Congo, and Nigeria.

Note: You can also limit the map extent using `coord_sf(xlim, ylim)` to zoom in on a specific region:

```
map +
  coord_sf(
    xlim = c(-15, 20), # from 15 W to 20 E
    ylim = c(-10, 30)  # from 10 S to 30 N
  )
```




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

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
```

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

nrow(events_joined)
```

```
## [1] 68354
```

```
nrow(events_sf)
```

```
## [1] 68354
```

`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:

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

```
## [1] 1576
```

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

```
## [1] 0.023
```

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; (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:

```
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 Democratic Republic of the Congo    9057           191828
##  2 Nigeria                             7166            69798
##  3 Somalia                             6682            58310
##  4 Ethiopia                            5456           385694
##  5 Algeria                             4006            20767
##  6 Sudan                               3620            65962
##  7 Burundi                             3067           124041
```

```
##  8 Mali                          2457           16609
##  9 Rwanda                        2452          670497
## 10 South Africa                  2388            4711
```

The top 10 countries by event count are shown above. The ranking reflects documented conflict hotspots in the data.

3. Choropleth of conflict intensity

a) Join event counts back to the world polygon:

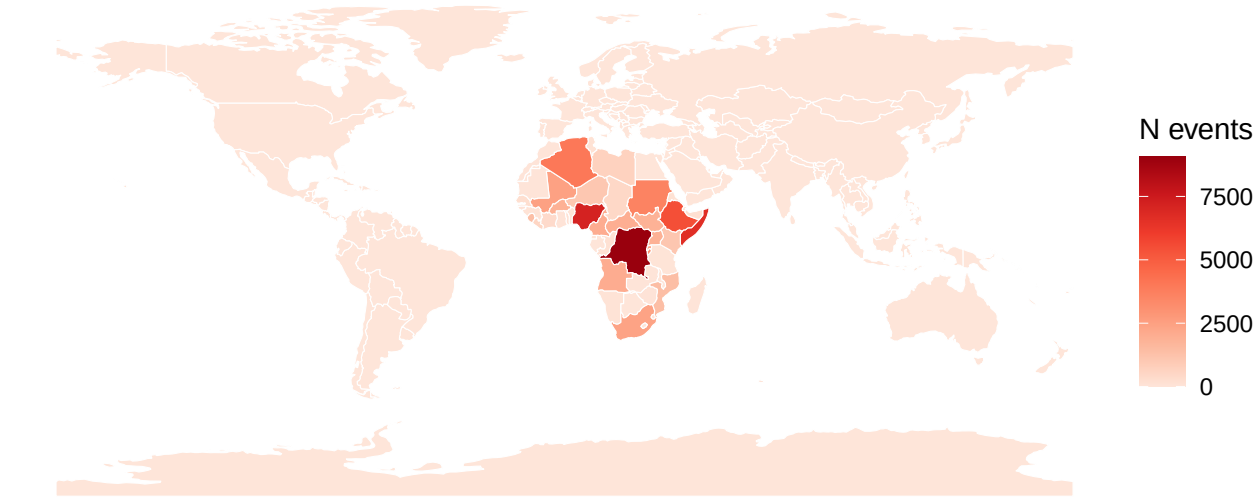
```
events_by_country_df = st_drop_geometry(events_by_country)

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")
```

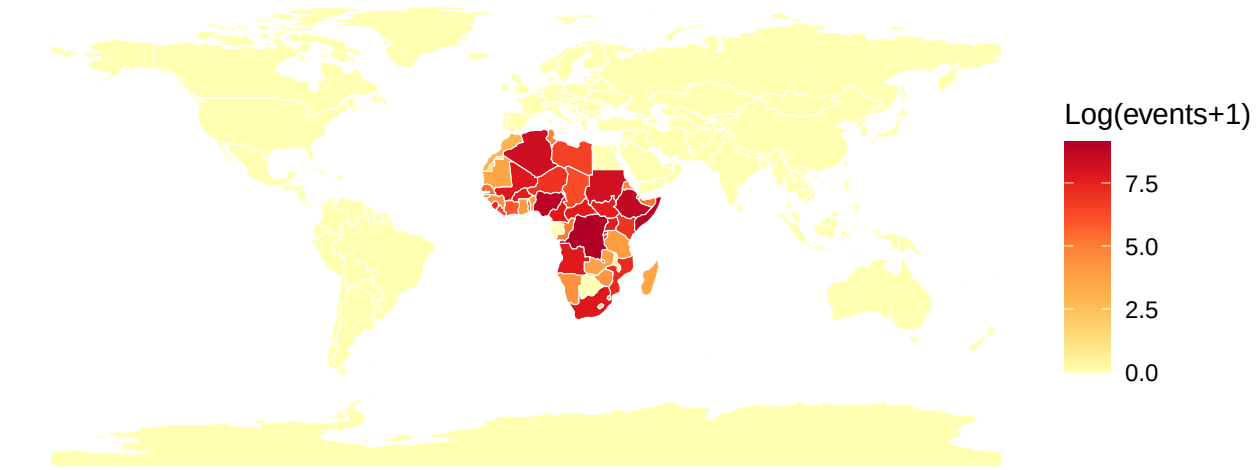


```
ggsave("conflict_by_country.pdf", width = 10, height = 5)
```

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.

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)")
```



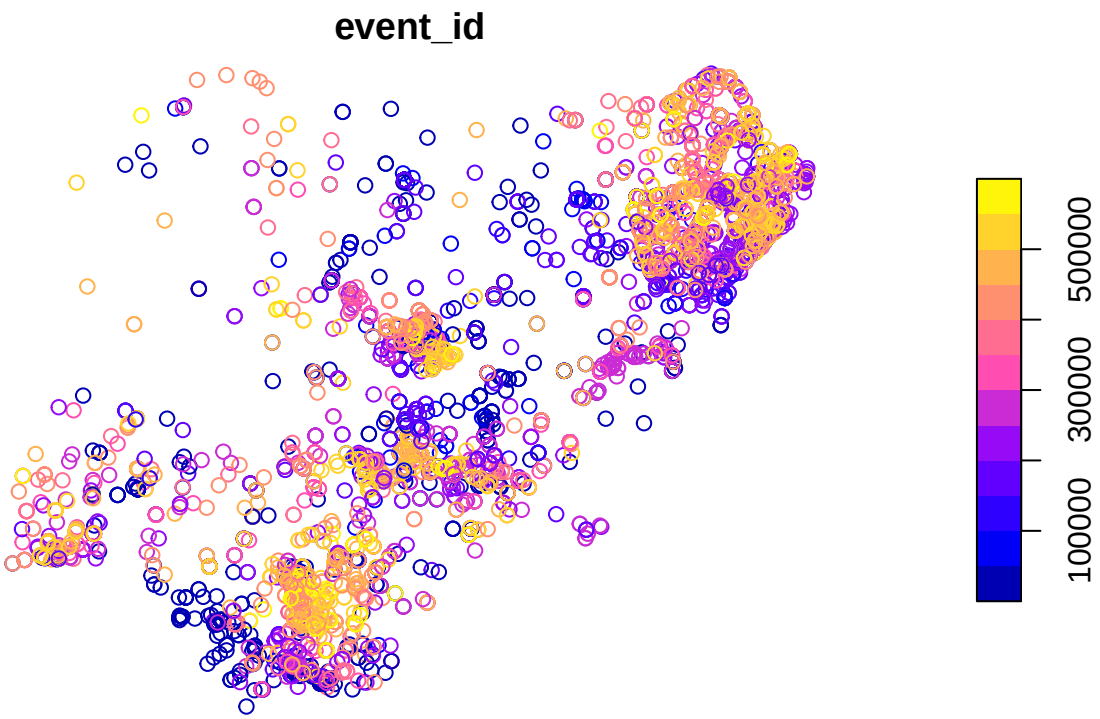

```
ggsave("conflict_log_map.pdf", width = 10, height = 5)
```

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. Bonus: Distance to capital in Nigeria

a-c) Filter Nigeria events and create the Abuja reference point:

```
# Filter events in Nigeria using spatial join
nigeria = events_sf %>%
  st_join(world[, c("name_long")]) %>%
  filter(name_long == "Nigeria")

# Quick check
plot(nigeria[1])
```



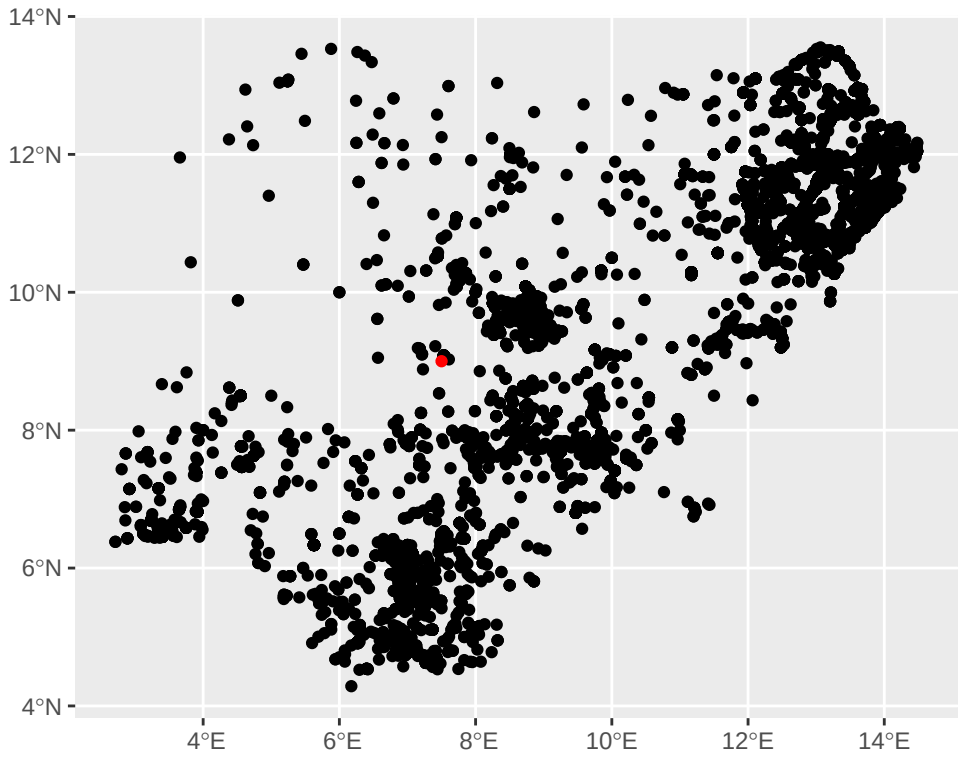

```
# Create Abuja point (approximately 9 N, 7.5 E)
abuja = data.frame(city = "abuja", lon = 7.5, lat = 9) %>%
  st_as_sf(coords = c("lon", "lat"), crs = 4326)

# Verify CRS match
st_crs(nigeria)$epsg
```

```
## [1] 4326
```

```
st_crs(abuja)$epsg
```

```
## [1] 4326
```

```
ggplot(nigeria) +
  geom_sf() +
  geom_sf(data = abuja, color = "red")
```



d) Transform to UTM projection and calculate distance:

```
# Transform to UTM zone 32N (EPSG:32632, appropriate for Nigeria)
nigeria_m = st_transform(nigeria, 32632)
abuja_m = st_transform(abuja, 32632)

# Test distance calculation
st_distance(nigeria_m[1:3,], abuja_m)
```

```
## Units: [m]
##          [,1]
## [1,] 533380.5
## [2,] 714912.4
## [3,] 825617.4
```

```
# Calculate distance for all events (in meters)
nigeria$dist_abuja = as.numeric(st_distance(nigeria_m, abuja_m))
```

e-f) Run linear models with fatalities as outcome and distance from Abuja as predictor:

```
# Create transformed variables
nigeria = nigeria %>%
  mutate(log_fatalities = log(fatalities + 1),
    dist_abuja_log_km = log((dist_abuja + 1) / 1000))

# Models
m1 = lm(fatalities ~ dist_abuja, data = nigeria)
m2 = lm(log_fatalities ~ dist_abuja_log_km, data = nigeria)
m3 = lm(log_fatalities ~ dist_abuja_log_km + event_type,
```

```
    data = nigeria)
m4 = lm(log_fatalities ~ dist_abuja_log_km * event_type,
    data = nigeria)

modelsummary(list(m1, m2, m3, m4), stars = TRUE, output = "markdown")
```

	Model 1	Model 2	Model 3	Model 4
(Intercept)	11.100***	1.664***	1.060***	2.146***
	(1.316)	(0.138)	(0.160)	(0.232)
dist_abuja	0.000			
	(0.000)			
dist_abuja_log_km		-0.026	0.098***	-0.095*
		(0.022)	(0.028)	(0.041)
event_typeone-sided			-0.138***	-1.926***
			(0.039)	(0.389)
event_typestate-based			-0.288***	-2.970***
			(0.038)	(0.451)
dist_abuja_log_km × event_typeone-sided				0.304***
				(0.064)
dist_abuja_log_km × event_typestate-based				0.440***
				(0.073)
Num.Obs.	7166	7166	7166	7166
R2	0.000	0.000	0.009	0.015
R2 Adj.	0.000	0.000	0.008	0.014
AIC	74218.2	21815.6	21759.2	21719.2
BIC	74238.9	21836.2	21793.6	21767.3
Log.Lik.	-37106.110	-10904.791	-10874.620	-10852.591
F	1.254	1.356	20.643	21.290
RMSE	42.91	1.11	1.10	1.10

Note: + p < 0.1, * p < 0.05, ** p < 0.01, *** p < 0.001

Model 1 uses raw fatalities and raw distance; Models 2-4 use log-transformed fatalities and log distance in kilometers. Model 3 adds event type as a control, and Model 4 includes the interaction between distance and event type. The results show whether events further from Abuja (the national capital) tend to be more deadly, which could reflect lower state capacity in peripheral areas.

5. 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.