Assignment 2

Erika Yazmin Blanco, Felipe Manzi, Olsa Reçica, Olta Berani

2026-02-03

Task 1

```
library(sf)
```

```
## Linking to GEOS 3.13.0, GDAL 3.8.5, PROJ 9.5.1; sf_use_s2() is TRUE
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.5.2
## v ggplot2   4.0.0     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.2
## v purrr     1.1.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(rnaturalearth)
library(rnaturalearthdata)
```

```
##
## Attaching package: 'rnaturalearthdata'
##
## The following object is masked from 'package:rnaturalearth':
##
##     countries110
```

In this section, we load the packages needed for the analysis. We use sf to work with spatial data, tidyverse for data manipulation and plotting, and the rnaturalearth packages to access Natural Earth spatial datasets.

```
countries <- ne_countries(
  scale = "medium",
  returnclass = "sf"
)
```

We load the world country boundaries from Natural Earth using the `ne_countries` function.
This dataset provides polygon geometries for countries and will be used as the base map for the analysis.

```
pop_places <- ne_download(
  scale = "medium",
  type = "populated_places",
  category = "cultural",
  returnclass = "sf"
)
```

```
## Reading 'ne_50m_populated_places.zip' from naturalearth...
```

We load the populated places dataset from Natural Earth. This dataset contains point locations of cities and settlements along with population information, which will be used to measure population distribution.

```
airports <- ne_download(
  scale = 10,
  type = "airports",
  category = "cultural",
  returnclass = "sf"
)
```

```
## Reading 'ne_10m_airports.zip' from naturalearth...
```

We load the airports dataset from Natural Earth.
This dataset contains point locations of major airports and will be used to compute distances between populated places and transportation infrastructure.

```
countries <- st_transform(countries, 3857)
pop_places <- st_transform(pop_places, 3857)
airports <- st_transform(airports, 3857)
```

We transform all spatial datasets to the same projected coordinate reference system.
Using a projected CRS allows distances to be calculated correctly and ensures consistency across all spatial operations.

Clean populated places

```
pop_places_clean <- pop_places %>%
  select(
    place_name = NAME,
    country_name = ADMONAME,
    population = POP_MAX,
    geometry
  ) %>%
  filter(!is.na(population))
```

We clean the populated places data by keeping only the relevant variables.
We also remove observations with missing population values so that the analysis is based only on valid population data.

Spatial join: populated places -> countries

```
pop_places_country <- st_join(
  pop_places_clean,
  countries,
  join = st_within
)
```

We spatially join populated places to country polygons.
This assigns each populated place to the country it falls within, which allows population to be aggregated at the country level.

Aggregate population at country level

```
country_population <- pop_places_country %>%
  st_drop_geometry() %>%
  group_by(admin) %>%
  summarise(
    total_population = sum(population, na.rm = TRUE),
    n_places = n()
  )
```

We aggregate the populated places data at the country level.
Total population is calculated by summing the population of all places within each country, and we also count how many populated places are included per country

Join results back to country polygons

```
countries_pop <- countries %>%
  left_join(country_population, by = "admin")
```

We join the country-level population totals back to the country polygons.
This allows population information to be linked with spatial boundaries for mapping and further analysis.

```
countries_pop %>%
  st_drop_geometry() %>%
  arrange(desc(total_population)) %>%
  select(admin, total_population, n_places) %>%
  head(10)
```

```
##                        admin total_population n_places
## 1                      China        206402291       96
## 2                      India        159122096       70
## 3  United States of America        134308467      103
## 4                     Brazil         73934145       43
## 5                      Japan         65779390       18
## 6                     Russia         47670747       78
```

```
## 7                   Mexico         43383240       25
## 8                 Pakistan         26416033        9
## 9                Indonesia         23765857       20
## 10               Argentina         20662269       18
```

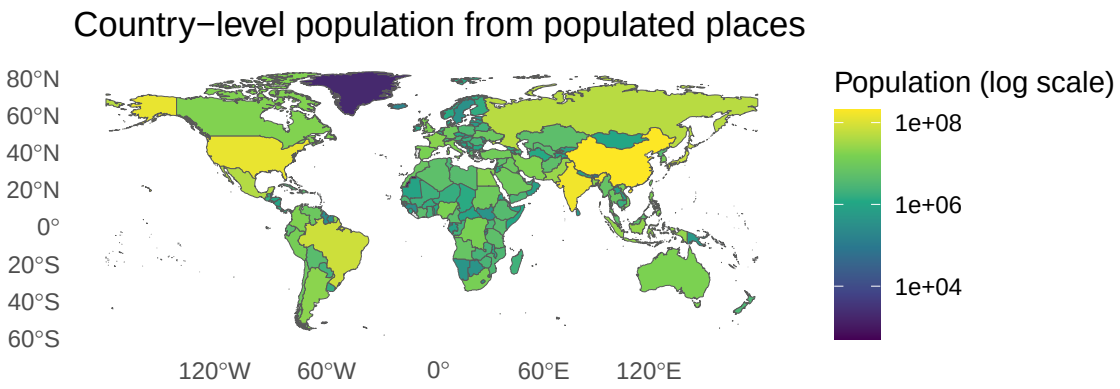
We display the countries with the highest total population.
This provides a quick check of the aggregated population values before moving on to visualisation.

1 Map of population by country

```
countries_pop_plot <- st_transform(countries_pop, 4326)
```

We transform the country-level dataset back to a geographic coordinate system. This step is done to prepare the data for mapping and visualisation.

```
countries_pop_plot %>%
  filter(admin != "Antarctica") %>%
  ggplot() +
  geom_sf(aes(fill = total_population), linewidth = 0.1) +
  scale_fill_viridis_c(
    trans = "log10",
    na.value = "grey90"
  ) +
  theme_minimal() +
  theme(
    panel.grid = element_blank()
  ) +
  labs(
    title = "Country-level population from populated places",
    fill = "Population (log scale)"
  )
```



We create a world map showing total population by country.
Countries are coloured by total population using a logarithmic scale to make differences across countries easier to see.

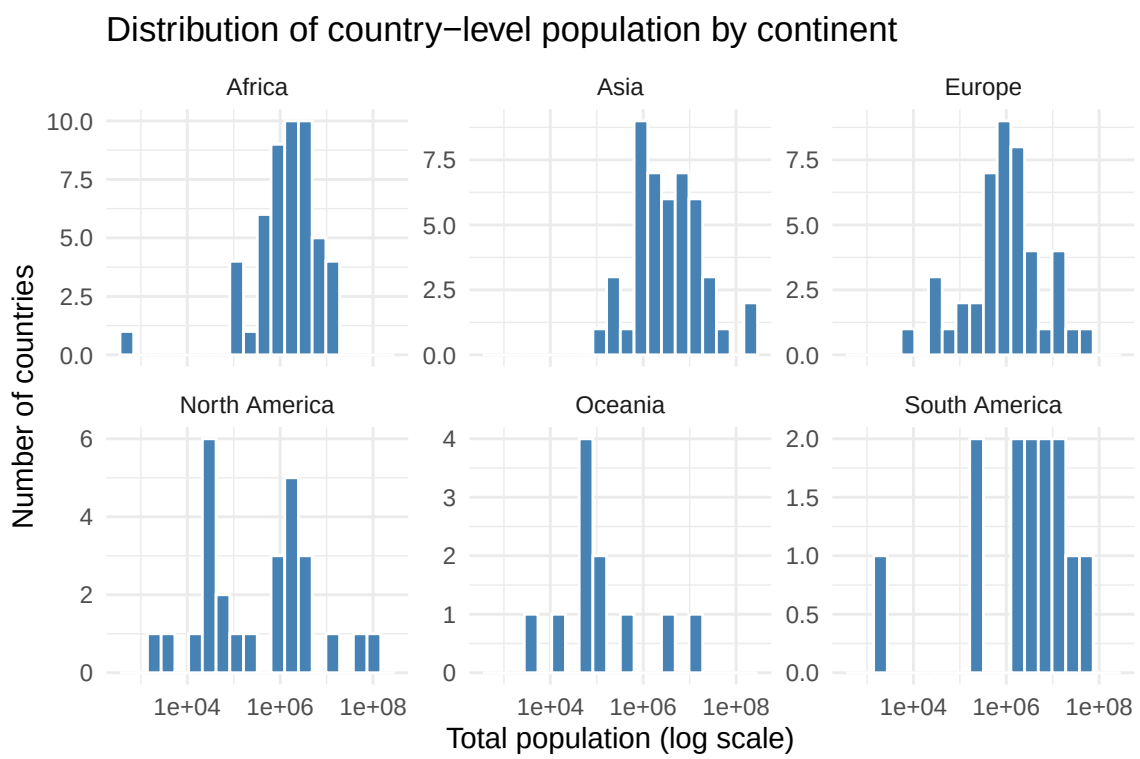
2 Histogram of country population by continent

```
pop_by_continent <- countries_pop %>%
  st_drop_geometry() %>%
  filter(!is.na(total_population), !is.na(continent))
```

We prepare the country-level population data for comparison across continents.
Countries with missing population or continent information are removed before creating continent-level summaries and plots.

```
pop_by_continent <- countries_pop %>%
  st_drop_geometry() %>%
  filter(
    !is.na(total_population),
    !is.na(continent),
    !continent %in% c("Antarctica", "Seven seas (open ocean)")
  )

ggplot(pop_by_continent, aes(x = total_population)) +
  geom_histogram(bins = 20, fill = "steelblue", color = "white") +
  scale_x_log10() +
  facet_wrap(~ continent, scales = "free_y") +
  theme_minimal() +
```

```
  labs(
    title = "Distribution of country-level population by continent",
    x = "Total population (log scale)",
    y = "Number of countries"
  )
```




```
  theme(
  panel.grid = element_blank()
)
```

```
## <theme> List of 1
##  $ panel.grid: <ggplot2::element_blank>
##  @ complete: logi FALSE
##  @ validate: logi TRUE
```

We plot a histogram of country-level population grouped by continent.Using a logarithmic scale helps visualise differences in population size across countries, while faceting allows comparison between continents.

3 Histogram of country-level average distance to airports by continent

```
dist_matrix <- st_distance(pop_places_clean, airports)

pop_places_dist <- pop_places_clean %>%
  mutate(
    min_dist_airport_km = apply(dist_matrix, 1, min) / 1000
```

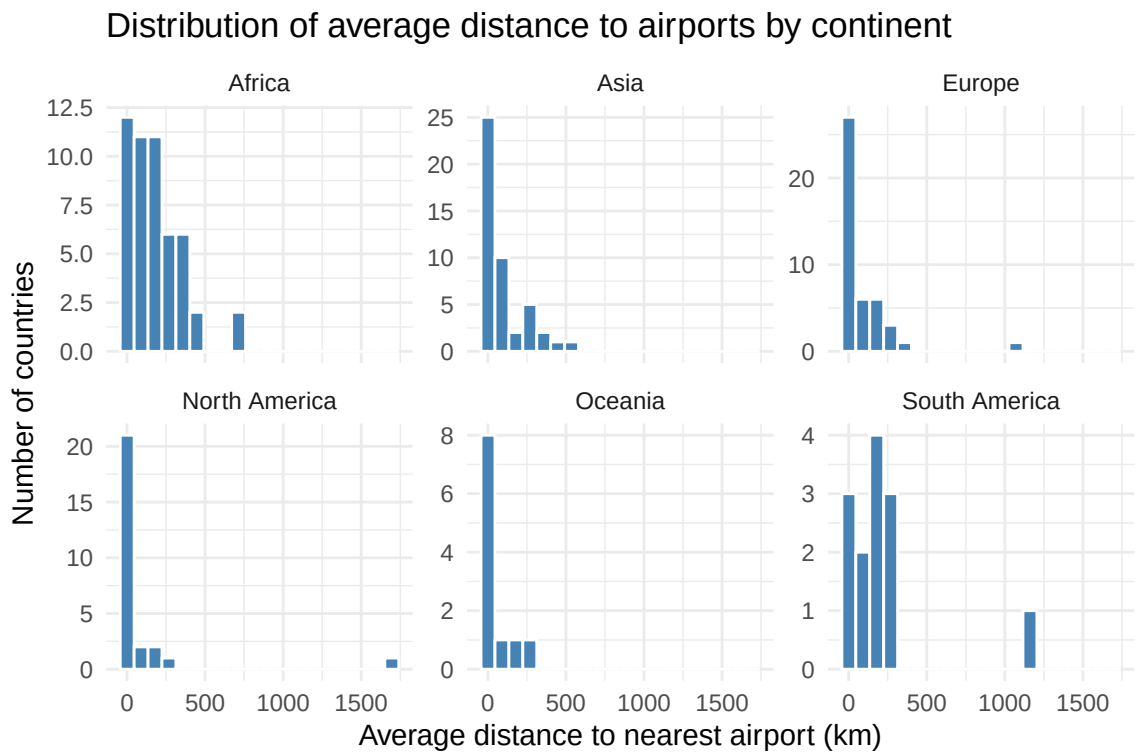
```
  )

pop_places_dist_country <- st_join(
  pop_places_dist,
  countries,
  join = st_within
)

country_dist <- pop_places_dist_country %>%
  st_drop_geometry() %>%
  group_by(admin, continent) %>%
  summarise(
    avg_dist_airport_km = mean(min_dist_airport_km, na.rm = TRUE),
    .groups = "drop"
  )

countries_final <- countries_pop %>%
  left_join(country_dist, by = c("admin", "continent"))
```

We compute the distance from each populated place to the nearest airport using spatial distance calculations. Distances are converted to kilometres and then averaged at the country level.
This produces a country-level measure of average distance to airports, which is later used to compare accessibility across continents.

```
dist_by_continent <- countries_final %>%
  st_drop_geometry() %>%
  filter(
    !is.na(avg_dist_airport_km),
    !is.na(continent),
    !continent %in% c("Antarctica", "Seven seas (open ocean)")
  )
```

We prepare the country-level distance data for analysis by continent.
Countries with missing distance or continent information are removed so that the comparison across continents is based on valid observation

```
ggplot(dist_by_continent, aes(x = avg_dist_airport_km)) +
  geom_histogram(bins = 20, fill = "steelblue", color = "white") +
  facet_wrap(~ continent, scales = "free_y") +
  theme_minimal() +
  labs(
    title = "Distribution of average distance to airports by continent",
    x = "Average distance to nearest airport (km)",
    y = "Number of countries"
  )
```



We plot a histogram of the average distance to the nearest airport at the country level. The data are grouped by continent to compare differences in accessibility across regions.

Task 2

```
library(sf)
library(tidyverse)
library(readxl)
library(rnaturalearth)
library(osmdata)
```

```
## Data (c) OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
```

Data Loading

First, we load the market coordinates and price data.

```
## Coordinates
coords <- read_excel(
  "/Users/yaz/Desktop/BSE/Term 2/Geospatial DS/Assignment 2/MktCoords.xlsx"
)

head(coords)
```

```
## # A tibble: 6 x 5
```

```
##   ctrycode mktcode market     longitude latitude
##   <chr>      <dbl> <chr>          <dbl>    <dbl>
## 1 AO           200 Luanda         13.2     -8.84
## 2 BJ           401 Cotonou         2.43     6.37
## 3 BJ           402 Malanville      3.39    11.9
## 4 BJ           403 Natitingou      1.39    10.3
## 5 BJ           404 Parakou         2.62     9.35
## 6 BW           110 Gaborone       25.9    -24.7
```

```
## Prices

prices <- read_excel(
  "/Users/yaz/Desktop/BSE/Term 2/Geospatial DS/Assignment 2/PriceMaster4GAMS.xlsx"
)

head(prices)
```

```
## # A tibble: 6 x 148
##   mktcode country market  crop     `1`    `2`    `3`    `4`    `5`    `6`    `7`
##     <dbl> <chr>   <chr>   <chr>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1     200 Angola  Luanda  Maize NA     NA     NA     NA     NA     NA     NA
## 2     401 Benin   Cotonou Maize  0.250  0.250  0.263  0.277  0.299  0.300  0.311
## 3     401 Benin   Cotonou Rice   0.480  0.480  0.472  0.467  0.481  0.504  0.532
## 4     402 Benin   Malanv~ Maize  0.169  0.176  0.189  0.203  0.216  0.205  0.266
## 5     402 Benin   Malanv~ Sorg~  0.182  0.189  0.209  0.216  0.258  0.300  0.319
## 6     403 Benin   Natiti~ Maize  0.189  0.216  0.216  0.243  0.258  0.270  0.311
## # i 137 more variables: `8` <dbl>, `9` <dbl>, `10` <dbl>, `11` <dbl>,
## #   `12` <dbl>, `13` <dbl>, `14` <dbl>, `15` <dbl>, `16` <dbl>, `17` <dbl>,
## #   `18` <dbl>, `19` <dbl>, `20` <dbl>, `21` <dbl>, `22` <dbl>, `23` <dbl>,
## #   `24` <dbl>, `25` <dbl>, `26` <dbl>, `27` <dbl>, `28` <dbl>, `29` <dbl>,
## #   `30` <dbl>, `31` <dbl>, `32` <dbl>, `33` <dbl>, `34` <dbl>, `35` <dbl>,
## #   `36` <dbl>, `37` <dbl>, `38` <dbl>, `39` <dbl>, `40` <dbl>, `41` <dbl>,
## #   `42` <dbl>, `43` <dbl>, `44` <dbl>, `45` <dbl>, `46` <dbl>, `47` <dbl>, ...
```

Data Processing

We convert the market data into an `sf` object and calculate average prices per market.

```
## Create market points
markets_sf <- st_as_sf(
    coords,
    coords = c("longitude", "latitude"),
    crs = 4326
)

## Compute average prices
prices_long <- prices |>
    pivot_longer(
        cols = `1`:`144`,
        names_to = "time",
        values_to = "price"
    )
```

```
### Average per market
avg_prices <- prices_long |>
    group_by(mktcode) |>
    summarise(
        avg_price = mean(price, na.rm = TRUE)
    )

### Merge prices with spatial data
markets_sf <- markets_sf |>
    left_join(avg_prices, by = "mktcode")
```

Spatial Features

We download and process the coastline, roads, and airport data.

```
## Coast, roads and airport info

africa <- ne_countries(
    continent = "Africa",
    returnclass = "sf"
)

### Coast
coast <- ne_download(
    scale = "medium",
    type = "coastline",
    category = "physical",
    returnclass = "sf"
)
```

```
## Reading 'ne_50m_coastline.zip' from naturalearth...
```

```
### Roads
roads <- ne_download(
    scale = 10,
    type = "roads",
    category = "cultural",
    returnclass = "sf"
)
```

```
## Reading 'ne_10m_roads.zip' from naturalearth...
```

```
### Airports
# Using Natural Earth to avoid Overpass timeouts
airports <- ne_download(
    scale = 10,
    type = "airports",
    category = "cultural",
    returnclass = "sf"
)
```

```
## Reading 'ne_10m_airports.zip' from naturalearth...
```

```
## Cropping layers to Africa
sf_use_s2(FALSE)
```

```
## Spherical geometry (s2) switched off
```

```
africa_valid <- st_make_valid(africa)

coast <- st_intersection(coast, africa_valid)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they
## are planar
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

```
roads <- st_intersection(roads, africa_valid)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they
## are planar
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

```
airports <- st_intersection(airports, africa_valid)
```

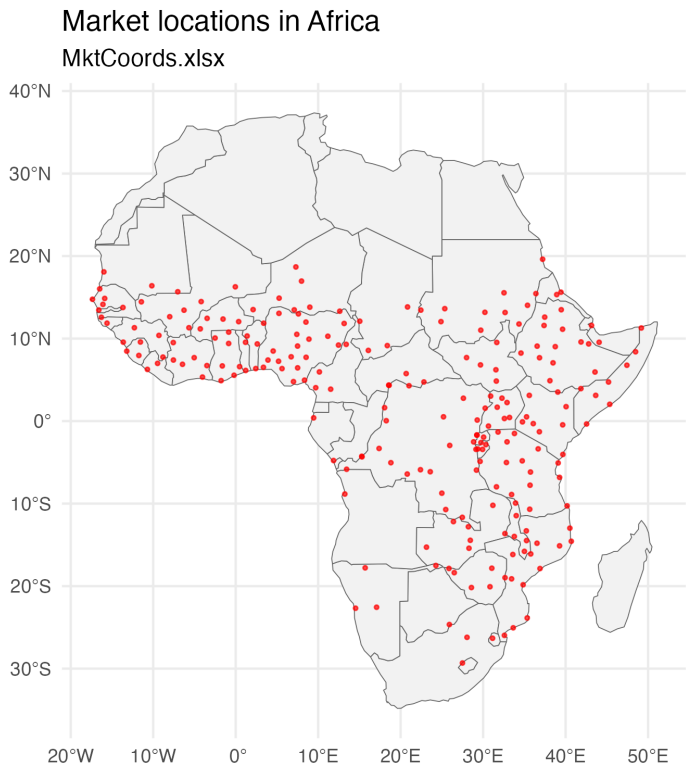
```
## although coordinates are longitude/latitude, st_intersection assumes that they
## are planar
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

Market Locations Map

Here we visualize the market locations across Africa.

```
ggplot() +
    geom_sf(data = africa, fill = "gray95", color = "gray40", linewidth = 0.2) +
    geom_sf(data = markets_sf, color = "red", size = 0.6, alpha = 0.7) +
    labs(title = "Market locations in Africa", subtitle = "MktCoords.xlsx") +
    theme_minimal()
```



Distance Calculation

We project all layers to a metric CRS (Web Mercator) and compute the minimum distance from each market to the nearest feature.

```
## Project to meters
crs_m <- 3857

markets_p <- st_transform(markets_sf, crs_m)
coast_p <- st_transform(coast, crs_m)
roads_p <- st_transform(roads, crs_m)
airports_p <- st_transform(airports, crs_m)

## Computing minimum distances

### Coast
d_coast <- st_distance(markets_p, coast_p)
min_coast <- apply(d_coast, 1, min)

### Roads
d_road <- st_distance(markets_p, roads_p)
min_road <- apply(d_road, 1, min)

### Airports
d_air <- st_distance(markets_p, airports_p)
min_air <- apply(d_air, 1, min)

### Store distances (converted to km)
```

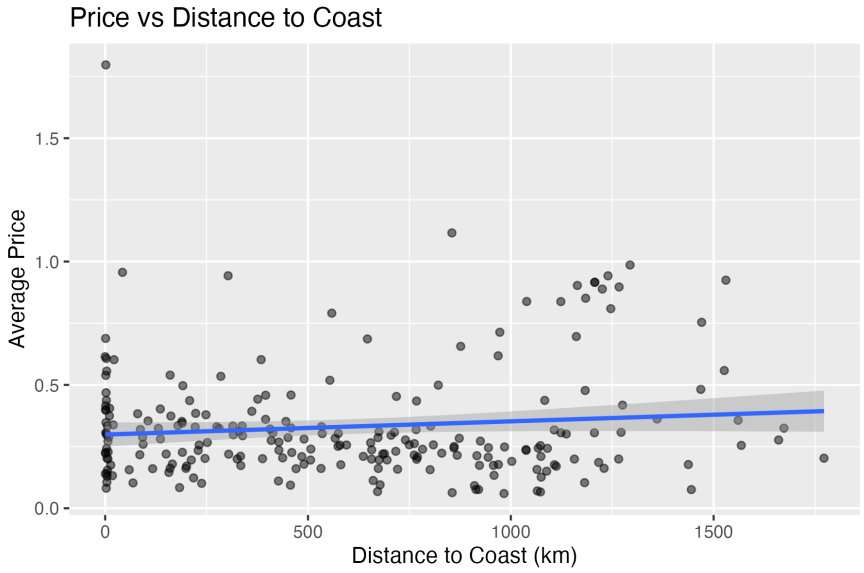
```
markets_sf$dist_coast_km <- as.numeric(min_coast) / 1000
markets_sf$dist_road_km <- as.numeric(min_road) / 1000
markets_sf$dist_air_km <- as.numeric(min_air) / 1000
```

Scatter Plots

We now generate the scatter plots to analyze the relationship between price and distance.

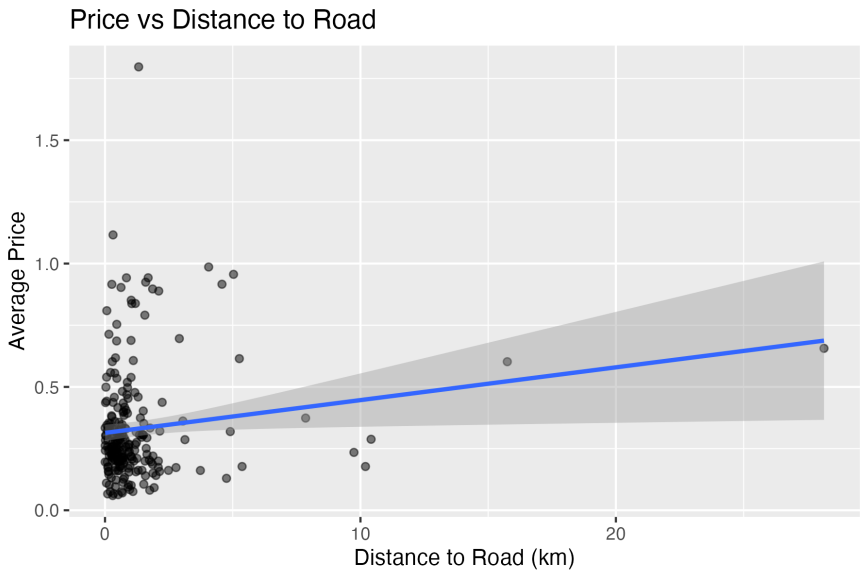
Price vs Coast

```
ggplot(
    markets_sf,
    aes(dist_coast_km, avg_price)
) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm") +
    labs(
        x = "Distance to Coast (km)",
        y = "Average Price",
        title = "Price vs Distance to Coast"
    ) +
    theme_minimal() +
    theme(
    text = element_text(size = 10),
    plot.title = element_text(size = 12)
    )
```



Price vs Roads

```
ggplot(
    markets_sf,
    aes(dist_road_km, avg_price)
) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm") +
    labs(
        x = "Distance to Road (km)",
        y = "Average Price",
        title = "Price vs Distance to Road"
    ) +
    theme_minimal() +
    theme(
    text = element_text(size = 10),
    plot.title = element_text(size = 12)
    )
```



Price vs Airports

```
ggplot(
    markets_sf,
    aes(dist_air_km, avg_price)
) +
    geom_point(alpha = 0.5) +
    geom_smooth(method = "lm") +
    labs(
        x = "Distance to Airport (km)",
        y = "Average Price",
        title = "Price vs Distance to Airport"
    ) +
    theme_minimal() +
    theme(
```

```
    text = element_text(size = 10),
    plot.title = element_text(size = 12)
    )
```