Maps

What We Have Done So Far

- Worked with:
  - dplyr
  - ggplot2

  } we can analyze many data sets, whether finding sub datasets or make simple visualizations: bar charts, density plots, histograms, etc.

Approaching Geographic Data

- there are numerous ways to deal with geographic data
  - ↳ Ex: Maps
- Some packages include:
  - maps - data sets of maps and functions for drawing
  - rnaturalearth - interactive w/ Natural Earth (public domain)
  - sf - classes & functions for vector data
  - leaflet - interactive
  - tmap - static / interactive maps

Mainly be covering: Basic, Simple Features, Leaflet

Basic Maps

Libraries:
- ggplot2
- map

NOTE*: this approach is very LIMITED and does not produce $\boxed{\text{high-quality}}$ maps

↳ (ggplot 2, sf, rnaturalearth are better alternatives)

The legacy approach uses the ggplot2 structure and uses maps as visual plots (histograms, bar charts, scatterplots)
↳ Why is this considered "legacy"?

- Usually ggplot2 is not meant for maps and detailed geographical/image visualizations
  ↳ because of how ggplot2 creates ~~[illegible]~~ visualizations, not optimal for maps

Process Using Maps + ggplot2

1. Custom Location Data
   - Usually maps have latitude and longitude, so to make "markers" or data, you would need to have custom data, via (~~ha~~ x = long, y = lat)

Ex:
```
library(tidyverse)   → to filter data
library(maps)

storms75 <- filter(storms, year == 1975)
```
storms → data set
1975 → only entries that are in the year 1975

2. Create/Load a World Map
   - This is the heart of our visualization, the "world" is a data frame of long, lat coordinates

Ex:
```
world_map = map_data("world")
```
map_data → from "maps" library

3. Combine the visualizations

For readability purposes, lets assign a variable as our world map, makes combining visualitations easier

Ex:
```
gg-world = ggplot() +
    geom_polygon(data = world-map,
            aes(x = long, y = lat, group = group),
            fill = "gray95", colour = "gray70",
            linewidth = 0.2) +
  theme_bw()
```

Note*: Calling gg-world should display a world map

Finally, to combine everything:

```
gg-world+
  geom_point(data = storms75,
        aes(x = long, y = lat, color = name))
```

Simple Features Maps

Libraries:
- sf
- rnaturalearth

Why "sf"?
- Specific functions TAILORED for handling geospatial data
- Better performance than Basic Maps structure
- Tabular data (easy to use data frame operations)

Basic World Maps:
Using "rnaturalearth":

- ne_coastline() $\rightarrow$ world coastline map
- ne_countries() $\rightarrow$ world country polygons

World Coastline Map:

```
world_coast = ne_coastline(scale = "medium",
                           returnclass = "sf")
```

ne_coastline function:
Scale: numeric or string - returns scale of map
Numeric: 10, 50, 110
String: "Small", "medium", "large"

returnclass: string determining the spatial object to return

String: "sf" (Simple feature)
"sv" (Spatial Vector)

Usually for clarity, it is recommended to have theme(panel.background = element_blank()) (clears the background)

- We can use coord_sf() to zoom into a specific region

coord_sf function:
xlim: vector (range of ~~latitude~~ longitude)
ylim: vector (range of latitude)

What if we only wanted a specific continent?

• it can be too tedious sometimes to use coord_sf, hence in the ne_countries function:

ne_countries function:
- Scale: (same as ne_coastline)
- type: string - country type (countries, map_units, sovereignty, tiny_countries)
- Continent: vector (char) - continent names
- Country: vector (char) - country names
- geounit: vector (char) - geounit names
- sovereignty: vector (char) - soverignty names
- returnclass: (same as ne_coastline)

Ex: (Same from "Basic Maps")

Recall "Storms 75"

```
north_america = ne_countries(continent = "North
                             America",
                             returnclass = "sf")

ggplot(data = north_america) +
    geom_sf() +
    geom_point(data = storms75,
               aes(x = long, y = lat, color = name)) +
    theme(panel.background = element_blank())
```

Multiple Maps:
we can also create multiple maps by using facet_wrap(~ separator)

* All you need to do is add that line to your ggplot function!

Leaflet Maps

Libraries:
leaflet → drawing of maps
sf

What is leaflet?
- Open source javascript library solely for interactive maps

Basic World Map:
```
leaflet() |>
  addTiles()
```
} This output might look strange, displaying multiple worlds, the key is to $\boxed{\text{zoom in}}$

Zooming:
Ex:
```
leaflet() |>
    addTiles() |>
    setView(lng=-80.19, lat=25.76, zoom=3)
```
→ Allows you to go to a specific coordinate and zoom set around said (long, lat)

Adding Markers:
- You can add markers, perhaps to indicate locations using addMarkers()

addMarkers function:
lng → vector (numeric) - longitudes (can be infered from data frames)
lat → vector (numeric) - latitudes
popup → vector (char) → basically a description

Add Provider Tiles
- basically think about provider tiles as a "theme"

addProviderTiles function:
    map: map
    provider: string (name of provider)

Ex (Basic Map Storms):

Recall "storms75" data

- Before we make the leaflet, we need to set our colors to a rainbow scheme

```
count_storms75 = storms75 |>
        distinct(name)|>
        nrow()

pal <- colorFactor(
    palette = rainbow(n=count_storms75),
    domain = storms75$name
  )
```

- Now our leaflet:

```
storms75 |>
   leaflet() |>
   addProviderTiles('CartoDB') |>
   setView(lng= -80.19, lat = 25.76, zoom = 3) |>
   addCircleMarkers(
       lng = ~long,
       lat = ~lat, radius = 2,
       color = ~pal(name))
```

Adding a Legend:

- Adding a legend in leaflet allows the view to understand colors, symbols, and representations you might encode
  - ↳ more control than ggplot legends

addLegend function:

position: String (position of the legend)
color: Vector (HTML colors, assuming no "pal")
labels: vector
title: String (title of legend
opacity: numeric
group: group name of a leaflet layer group (ties legend to leaflet layer group)

Summary

- So far in the class we learned about data visualizations:

$$\begin{tikzcd}
 & & \text{data visualizations:} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Histograms} & \text{Line Graphs} & \text{Scatterplots} & \text{Bar Charts}
\end{tikzcd}$$

MAPS are another such visualiqation and while we CAN use ggplot2 and maps, it simply is not computationally efficient (might even cause crashes/ memory overloading)

(SF means "simple features")

• Instead we can use 2 other methods:

$$\begin{tikzcd}
 & \text{can use} \arrow[dl] \arrow[dr] & \\
\text{SF: rnaturalearth, sf} & & \text{Leaflets: leaflet, sf}
\end{tikzcd}$$

• Both methods are valid and depend on the overall [illegible] utility that you need when making a map

tidyverse → data frame cleaning

General Process

$$\begin{tikzcd}
 & & \text{Leaflet} \\
\text{Data Frame (make sure to have latitude and longitude (location data))} \arrow[r] & \text{TidyVerse data cleaning} \arrow[ur] \arrow[r] & \text{SF}
\end{tikzcd}$$

} Just like ggplot visualizatio