Lab 05-A Introduction to Tableau

RE 519 Data Analytics and Visualization | Autumn 2025

In this lab, we are going to use Tableau, a business intelligence and analytics platform that allows users to create interactive visualizations and dashboards to understand data. It provides free access to students. **Please try to download** Tableau for Students in advance.

The due day for each lab can be found on the course wesbite. The submission should include Rmd, html, and any other files required to rerun the codes. For Tableau, submit your **Tableau packaged workbook (.tbwx)** This will include any excel files that you attach so that when you share the file anyone with Tableau can open it!

From Lab 4, the use of any generative AI tool (ChatGPT, Copilot, etc.) is **allowed**. But, I still encourage you to write codes by yourself and use AI tools as a way to debug and explore new knowledge. More information about Academic Integrity and the Use of AI.

Lab 05-A: Introduction to Tableau

In this lab, we will use Seattle Airbnb listing data published by Inside Airbnb. Inside Airbnb is a mission driven project that provides data and advocacy about Airbnb's impact on residential communities. They published a report on Platform Failures: How Short-Term Rental Platforms like Airbnb fail to cooperate with cities and the need for strong regulations to protect housing which may be interesting to you.

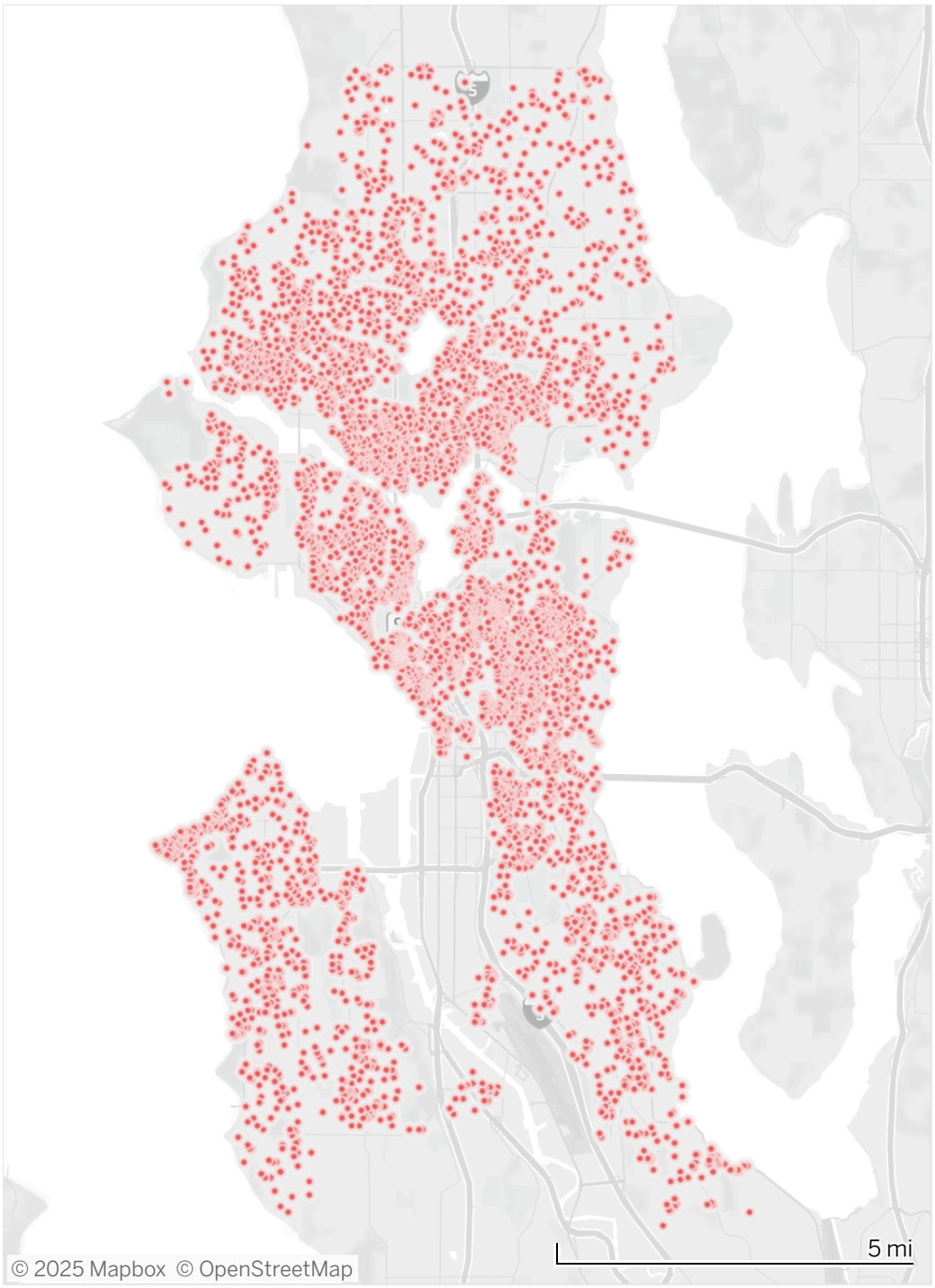
The data we will use was published on 21 June, 2025 and you can explore the data using their visualization tool. Please check the data dictionary below.

▶ **Click to view the full data dictionary**

We are going to build an interactive dashboard that visualizes key insights from our dataset. The goal is not only to display information but also to enable users to explore, filter, and compare different variables dynamically. After that, we will publish online through Tableau Public as well as your GitHub page. Also, we will continue working on dashboard in next lab, so do not worry if some functions you want do not show in this lab. The final results will be like this:

Seattle Airbnb Dashboard

Interative Dashboard to Show the Current Airbnb Market in Seattle, WA
Data Source: insideairbnb.com



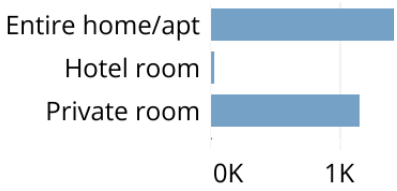
Neighbourhood Group

(All)

Average Price ($)

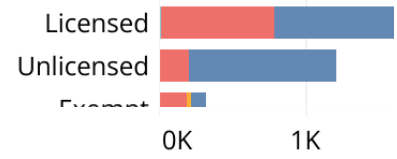
# of Room by Type

Airbnb hosts can list entire homes/a
hotel rooms.



Licenses

According to the law, specific short-
How many listings are unlicensed, fa
continue to advertise and profit fro

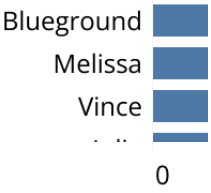


Room Type

Shared room Private r

Top Host

Some Airbnb hosts have multiple lis
apartment, or multiple apartments



The Tableau Workspace

Workspace. Source: Tableau Desktop and Web Authoring Help

- **A.** Workbook name. A workbook contains sheets. A sheet can be a worksheet, a dashboard, or a story. For more information, see Workbooks and Sheets.
- **B.** Cards and shelves - Drag fields to the cards and shelves in the workspace to add data to your view.
- **C.** Toolbar - Use the toolbar to access commands and analysis and navigation tools.
- **D.** View - This is the canvas in the workspace where you create a visualization (also referred to as a "viz").
- **E.** Click this icon to go to the Start page, where you can connect to data. For more information, see Start Page.
- **F.** Side Bar - In a worksheet, the side bar area contains the Data pane and the Analytics pane.
- **G.** Click this tab to go to the Data Source page and view your data. For more information, see the Data Source Page.
- **H.** Status bar - Displays information about the current view.
- **I.** Sheet tabs - Tabs represent each sheet in your workbook. This can include worksheets, dashboards, and stories. For more information, see Workbooks and Sheets.

Key Steps

Connect to Data

- In the **Connect pane**, click the file type (e.g., *Text File*).
- In the **Open** dialog box, select your dataset (e.g., listings.csv) and click **Open**.
- Change the data type in **data source page**.
- (Optional) Change column labels for readability (this won't rename your local file).
- Concepts: Dimensions and Measures
    - Dimensions contain qualitative values (such as names, dates, or geographical data).
    - Measures contain numeric, quantitative values that you can measure.
    - You can re-categorize a dimension as a measure or vice versa.

Global Formatting

- Adjust global font, font size, alignment, etc., under **Format** → **Workbook**.

Maps

- Drag **Latitude** → *Rows*, and **Longitude** → *Columns*.
- Customize map background: *Map* → *Map Layers* → *Style*.
- Add Tooltip: drag relevant fields (e.g., Name, Room Type, Price) to **Marks → Tooltip**.
- Rename the worksheet to *Map*.

Average Price

- Drag **Price** → *Columns* → right-click → **Measure → Average**.
- Add **Mark Labels** to show values.
- Hide the **X-axis** (right-click → *Show Header* off).
- Formatting numbers to currency in **default properties**.
- Rename to *Average Price*.

Number of Room by Type

- Drag **Room Type** → *Rows*.

- Drag **Name** → *Columns* → **Count** (right-click → *Measure* → *Count*).
- Hide the **X-axis** and label each bar.
- Rename worksheet and add short descriptive text in text box.

Licenses

- Create a **Group** for license categories (e.g., *Licensed / Unlicensed / Exempt*).
- Drag **License Categories** → *Rows* and **Name** → *Columns* → *Count*.
- Drag **Room Type** → *Color* in the Marks card.
- Customize the **color palette**.
- Rename worksheet and add notes.

Top Hosts

- Drag **Host Name** → *Rows*.
- Drag **Host ID** → *Columns* → *Count*.
- Sort descending by count.
- Add mark labels and rename the worksheet.

Dashboard

- Click **New Dashboard (+)**.
- Set min and max dashboard size.
- Drag your worksheets into the layout.
- Use **Text Box** to add title, description, and data source citation.
- Apply **consistent background color** and **font**.

Interactivity

- Right-click filters (e.g., *Neighbourhood Group*) → **Show Filter**.
- Change filter type to dropdown or slider as needed.
- Make filters **affect all worksheets** (*Apply to Worksheets → All Using This Data Source*).
- Test interactivity — selecting a neighborhood should update all charts.

Publish

- Save to Tableau Public
  1. Click **File → Save to Tableau Public**
  2. Log in with your Tableau Public account (create one if needed).
  3. Wait for the upload to finish — Tableau will open your dashboard in the browser.
- Embed to your webpage
  1. On your Tableau Public page, click **Share** (top right).
  2. Copy the **Embed Code**.
  3. Paste it into your website's HTML or Markdown.
- Save as Packaged Workbook (.twbx)
  1. Go to **File → Export Packaged Workbook**
  2. Name your file (e.g., Seattle_Airbnb_Dashboard.twbx).
  3. Click **Save**.

Hierarchy

- Drag **neighbourhood group** and **neighbourhood** to *Rows* — order matters (group first, then neighborhood).
- Drag **Name** to *Columns* → right-click → *Measure* → *Count*.
- Sort descending if needed.

Worksheet, Dashboard, Story

- A worksheet contains a single view along with shelves, cards, legends, and the Data and Analytics panes in its sidebar. For details on the worksheet workspace, see The Tableau Workspace.
- A dashboard is a collection of views from multiple worksheets. The Dashboard and Layout panes are available in its sidebar. For more details about creating dashboards, see Dashboards.
  - What is changed in the dashboard will affect the worksheet
- A story contains a sequence of worksheets or dashboards that work together to convey information. The Story and Layout panes are available in its side bar. For more details about creating stories, see Stories
  - What's changed in the story will not affect the worksheet or dashboard

📚 TODO: Tableau

14 points

Create visualizations, at least 5 worksheets, using 2022 5-year estimate data from American Community Survey (ACS). The dataset is about **housing affordability** in King County, WA. You can download the data here. The `geojson` (What is it?) data also include geospatial column (geometry).You can check the data dictionary below.

If you have a strong interest in using other datasets for this assignment, you are free to do that.

Calculation field. You can build your own variables using existing variables, such as defining a affordability index. This page may be helpful to you.

Arrange these sheets into a dashboard. Make sure that any filters that you apply on your dashboard work on all (relevant) sheets. If you want, play around with creating multiple story points.

Submit your **Tableau packaged workbook (.tbwx)** This will include any data files that you attach so that when you share the file anyone with Tableau can open it!

Raw ACS Variables Dictionary

Variable	ACS Code	Description
med_income	B19013_001	Median household income (past 12 months)
gini	B19083_001	Gini Index of Income Inequality
med_rent	B25064_001	Median gross rent (monthly)
med_home_value	B25077_001	Median home value (owner-occupied units)
total_pop	B01003_001	Total population
poverty_rate	B17001_002	Individuals below poverty line (numerator)
total_poverty	B17001_001	Total population used for poverty calculation
bachelors	B15003_022	Population with a bachelor's degree
graduate	B15003_023	Population with a master's degree or higher
total_edu	B15003_001	Total education population (age ≥25)
white	B03002_003	White alone

black	B03002_004	Black or African American alone
asian	B03002_006	Asian alone
hispanic	B03002_012	Hispanic or Latino (any race)
total_race	B03002_001	Total population by race
unemployed	B23025_005	Unemployed population (age 16+)
labor_force	B23025_003	Civilian labor force (age 16+)

Derived Indicator Dictionary

Derived Variable	Formula	Interpretation
poverty_pct	poverty_rate / total_poverty	Share of population below poverty line
college_pct	(bachelors + graduate) / total_edu	Share of population with ≥ bachelor's degree
white_pct	white / total_race	White population share
black_pct	black / total_race	Black or African American share
asian_pct	asian / total_race	Asian population share
hispanic_pct	hispanic / total_race	Hispanic or Latino share
unemployment_rate	unemployed / labor_force	Civilian unemployment rate

---

Lab 05-B: Spatial Data and Mapping using R

Recommended Readings:

Chapter 17 and 18, Modern Data Science with R. Baumer et al. 2024.

In section B, we will use `sf`, a designated package for spatial data that integrates with `tidyverse`. These will have the class `sf`. We will mainly treat geospatial information as an additional attribute for observation (rows). Instead of focusing on one product, we will test some *small* tasks to see how to manipulate spatial data. This is just a quick overview of spatial data processing. If you would like to do more spatial analysis, we'd like to recommend to use

ArcGIS Proor QGIS. Some GIS course at UW:

- RE 497/597: Real Estate Data Modeling
- URBAN 404/504: Introduction to Geographic Information Systems
- URBAN 422/522: Urban and Regional Geospatial Analysis
- GEOG 561: Urban Geographic Information Systems
- SEFS 520: Geographic Information Systems in Forest Resources

```
# install.packages("sf") # you only need to install once
library(sf)
library(tidyverse)
```

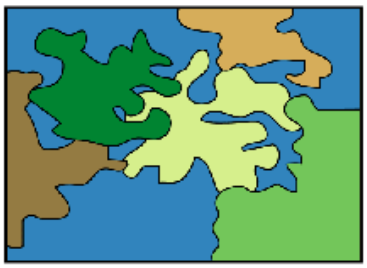
We will still use Seattle Airbnb (point, in CSV) and King County affordability data (polygon, in GeoJson), and three new:

- Light Rail Stations from Seattle GeoData Platform (points, in Geodatabase)
- Transit Routes for King County Metro from King County GIS Open Data Platform (lines, in Shapefile)
- Urban Centers Villages and Manufacturing Industrial Centers from Seattle GeoData Platform (polygon, in GeoJson)

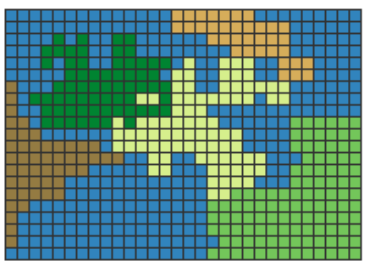
Spatial Data Types

In general, spatial data includes **two** types (we mentioned them in lecture 2):

- **Vector data** represent discrete features such as points, lines, and polygons.
- **Raster data** represent continuous surfaces such as elevation, temperature, or satellite imagery. Raster data will not be covered in this class. It is mainly used in remote sensing, oceanography,and atmospheric sciences. In R, `terra` is the most commonly used package for raster data.



Polygon features



Raster polygon features

Polygon in Vector and Raster Data. Source: Esri ArcMap Help

Vector - Points

Let's get `light_rail_stations.gdb` data, which is stored in a ESRI File Geodatabase. Such geodatabase could have multiple layers (bus line + bus stop + service area, for example). We started from check the list of layer using `st_layer()`.

```
st_layers("light_rail_stations.gdb")
```

```
## Driver: OpenFileGDB
## Available layers:
##            layer_name geometry_type features fields
## 1 Light_Rail_Stations         Point       58     13
##                               crs_name
## 1 NAD83(HARN) / Washington North (ftUS)
```

Only one layer inside called `Light_Rail_Stations` and we want to read it using `st_read()`:

```
lrstations <- st_read("light_rail_stations.gdb", layer = "Light_Rail_Stations")
```

```
## Reading layer `Light_Rail_Stations' from data source
##   `/Users/yohaoyu/Repo/data-analytics-visualization/labs/lab-05/light_rail_stations.gdb'
##   using driver `OpenFileGDB'
## Simple feature collection with 58 features and 13 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 1241447 ymin: 91347.08 xmax: 1325921 ymax: 300928.9
## Projected CRS: NAD83(HARN) / Washington North (ftUS)
```

We saw some meta information will show up:

1. File locations
2. Number of simple features (rows) and fields (columns)
3. Geometry type (point, line, and polygon)
4. Dimension - 2D in this data;
5. Bounding box
6. Geographic/Projected CRS (Coordinate Reference System)

Many time, we may only have the latitude and longitude in a CSV file, like the Airbnb dataset.

So, we read it as a normal dataframe and convert it to geometry column.

```
airbnb <- read_csv("airbnb_sea_listings.csv")
airbnb <- st_as_sf(
  airbnb,
  coords = c("longitude", "latitude"),
  crs = 4326 # we will set the coordinate reference system 4326
)
```

We can visualize spatial data by using `ggplot2` by add layer called `geom_sf()`:

```
ggplot(airbnb) +
  geom_sf(size = 0.2) +
  theme_void()
```

Vector - Line

`Shapefile` is a geospatial vector data format developed by Esri. The annoying point of `Shapefile` is a set of at least 3–7 separate files that must all stay together.

```r
kcm_routes <- st_read("kcm_transit_routes/kcm_transit_routes.shp")
```

```
## Reading layer `kcm_transit_routes' from data source
##   `/Users/yohaoyu/Repo/data-analytics-visualization/labs/lab-05/kcm_transit_routes/kcm_transit_route
##   using driver `ESRI Shapefile'
## Simple feature collection with 149 features and 9 fields
## Geometry type: MULTILINESTRING
## Dimension:     XY
## Bounding box:  xmin: 1225211 ymin: 70794.9 xmax: 1423864 ymax: 320456.3
## Projected CRS: NAD83(HARN) / Washington North (ftUS)
```

Vector - Polygon

GeoJSON is a text-based format for encoding geographic data structures like points, lines, and polygons, built on the standard JSON format.

```r
king_affordability <- st_read("king_affordability_2022.geojson")
```

```
## Reading layer `king_map_2022' from data source
##   `/Users/yohaoyu/Repo/data-analytics-visualization/labs/lab-05/king_affordability_2022.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 495 features and 29 fields (with 1 geometry empty)
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -122.5281 ymin: 47.0844 xmax: -121.0659 ymax: 47.78058
## Geodetic CRS:  NAD83
```

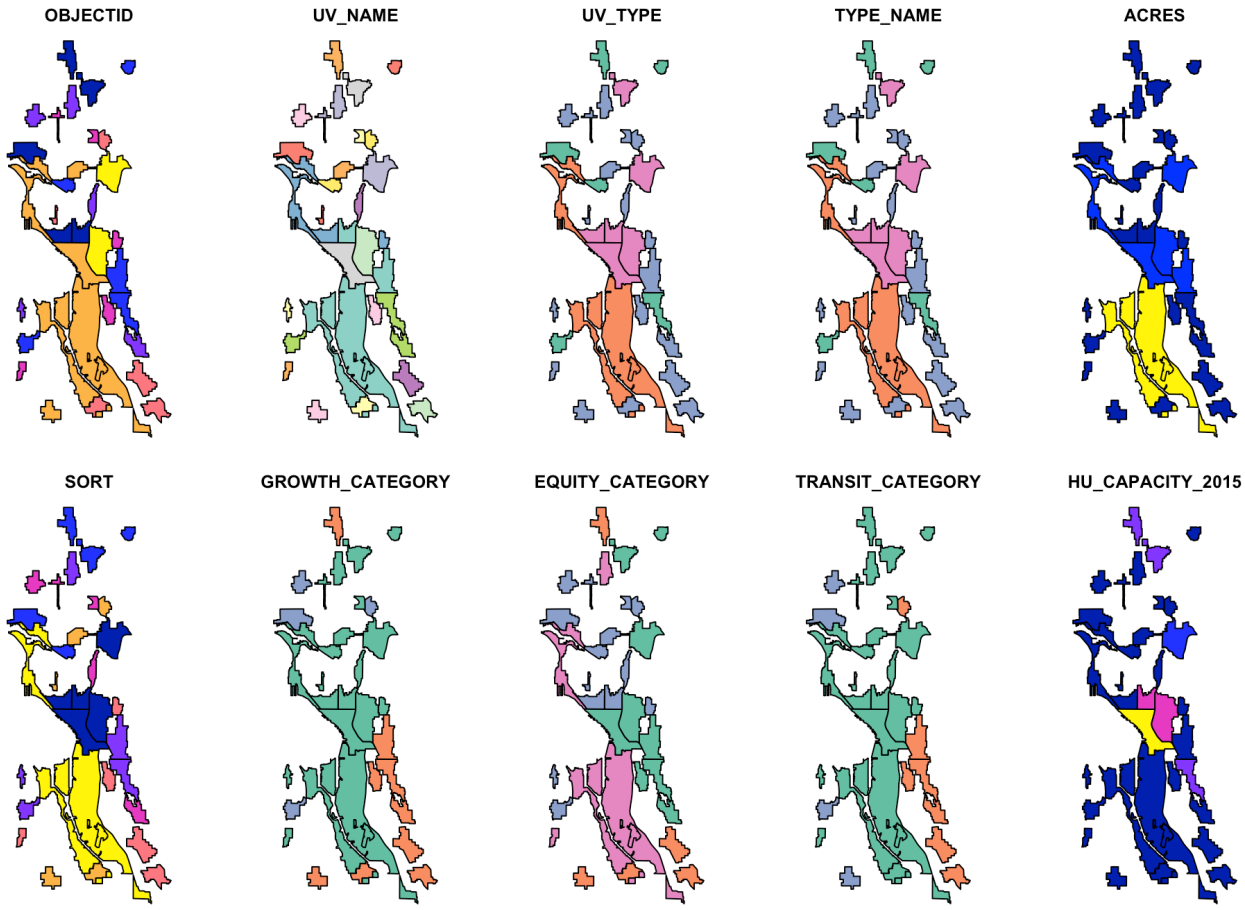
```r
urban_village <- st_read("urban_center_villages_manufacturing_industrial_centers.geojson")
```

```
## Reading layer `Urban_Centers_Villages_and_Manufacturing_Industrial_Centers' from data source `/Users
##   using driver `GeoJSON'
## Simple feature collection with 32 features and 31 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -122.3983 ymin: 47.50724 xmax: -122.2592 ymax: 47.73413
## Geodetic CRS:  WGS 84
```

We can also use `plot` function to generate plots instead of `ggplot2`. It will produce one map for each attribute.

```r
plot(urban_village)
```

```
## Warning: plotting the first 10 out of 31 attributes; use max.plot = 31 to plot
## all
```



You may find that both Airbnb and Urban Village are closed to the shape of City of Seattle, but they are different! Because they are using different Coordinate Reference Systems: WGS 84 and NAD83(HARN) / Washington North (ftUS).

Coordinate Reference System (CRS)

Why is Greenland huge on the world map, even larger than the whole of Africa? The True Size of… gives us a way to compare the true size of different countries.

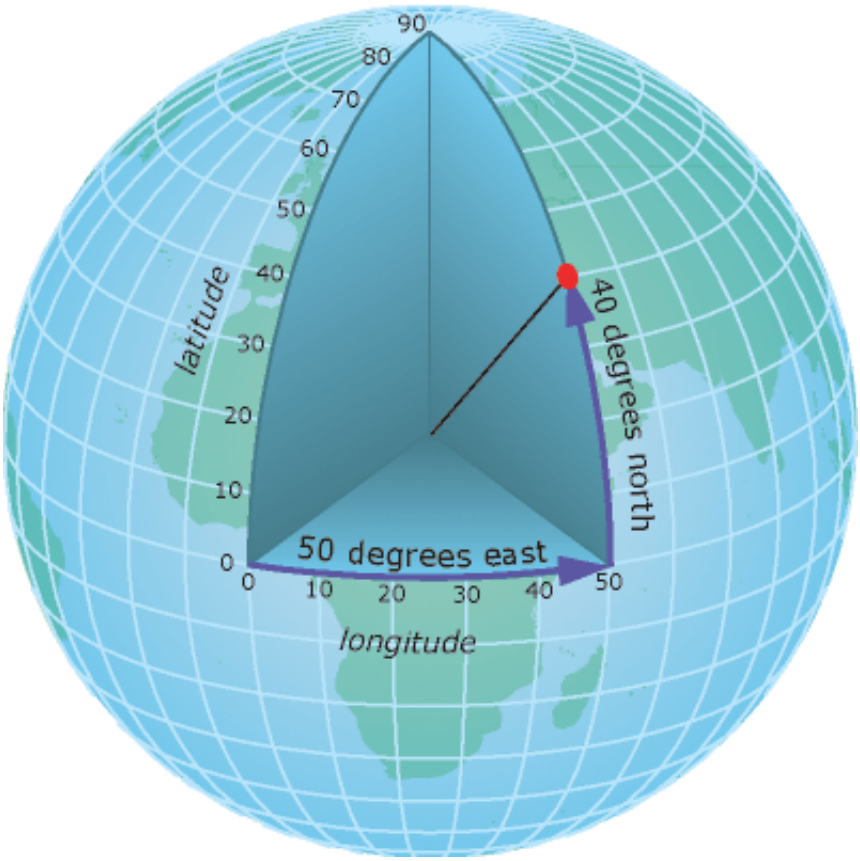
The Earth is a 3-D sphere (almost), and we have to use a way to flatten it onto a 2-D object when conducting visualizations. Here is a two-step process:

1. Know the location of a place on 3-D Earth using **Geographic Coordinate Systems**
2. Flatten it onto a 2-D map using **Projected Coordinate System**

Geographic Coordinate Systems (GCS)

Geographic Coordinate Systems (GCS) use degrees of **latitude** and **longitude** and sometimes also a height value to describe a location on the earth's surface. The most popular is called WGS 84 (EPSG: 4326). In the US, we use NAD83(HARN) (EPSG:4152) is the most common one designed for North America, which is also the official Geographic Coordinate Systems for Washington State (RCW 58.20.110).

It's worth to know the meaning of **EPSG** (European Petroleum Survey Group). They create a public database containing standardized definitions for different coordinate reference system (and more). We often use the unique numeric code (EPSG code) to refer a coordinate reference system.



A geographic coordinate system locates latitude and longitude location using angles. Thus the spacing of each line of latitude moving north and south is not uniform. Source: ESRI

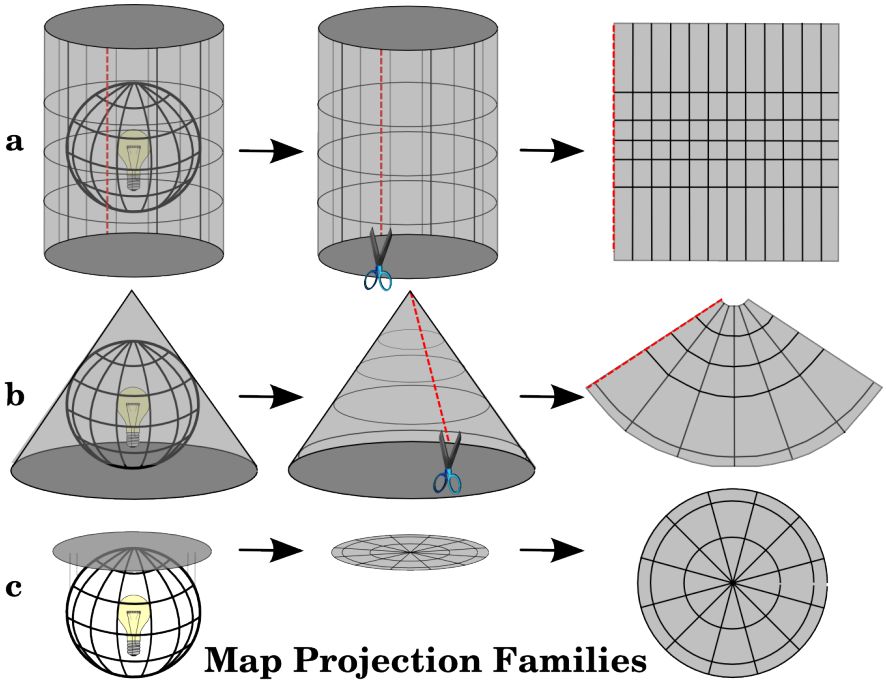
Let's explore the latitude and longitude of Gould Hall under different Geographic Coordinate Systems using epsg.io.

- Under WGS 84 EPSG:4326: -122.312776 47.654982
- Under NAD83(HARN) EPSG:4152: -122.312776 47.654982
- Under Beijing 1954 EPSG:4214: -122.31277 47.654981

Even when coordinate numbers look identical, different GCS define slightly different positions on Earth with real spatial differences, sometimes a few feet or even hundreds of meters, depending on the system.

Projected Coordinate System (PCS)

Having latitude and longitude with a GCS, we only have angles. Projected Coordinate System (PCS) helps use transform latitude and longitude to X and Y on a 2-D map. The process is called projection.



The Three Families of Map Projections. (a) Cylindrical, (b) Conical, or (c) Azimuthal. Source: QGIS Documentation

We can say **Projected Coordinate System = Geographic Coordinate System + Projection Methods + Projection Parameters**. We can categorize projections methods by what they preserve, for example:

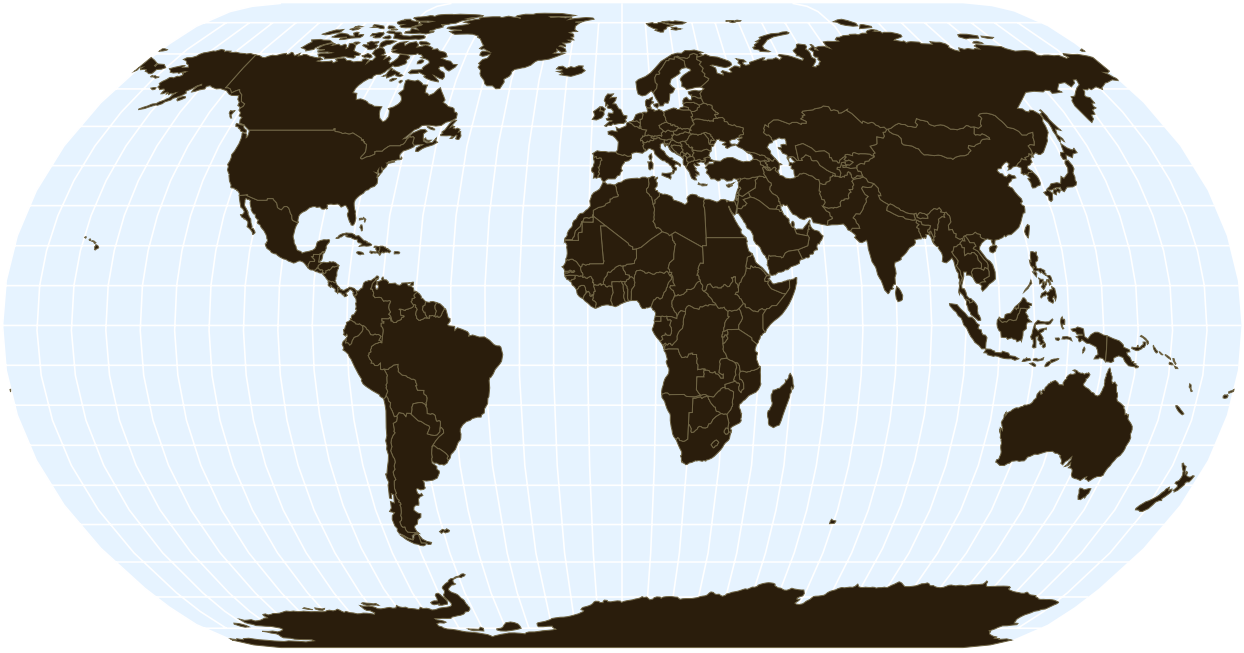
- Azimuthal Projection: Preserve Distance from Center
    - e.g. Azimuthal Equidistant

- Equal Area Projection: Preserve Proportional Area
    - e.g. Albers Equal Area Conic
- Conformal Projection: Preserve Local Angles and Shape
    - e.g. Mercator, Lambert Conformal Conic (WA Official)

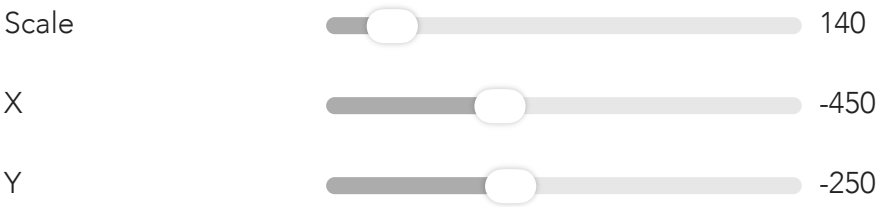
Let's explore the position of Gould Hall under different Coordinate Reference Systems using epsg.io.

- Under NAD83(HARN) EPSG:4152: -122.312776 47.654982
- Under NAD83(HARN)/Washington North (ftUS) EPSG:2926: 1251144.046 203467.392

Vega-Lite Cartographic Projections by Jeffrey Heer at University of Washington



Type naturalEarth1



In Practice

In our practice, you don't have to understand all of CRS, we just need to remember to use appropriate CRS for our data. In Washington State, we use **TWO** PCSs (including some contents from RCW 58.20.130):

- **North part of Washington:** NAD83(HARN)/Washington North (ftUS); EPSG:2926
  - Chelan, Clallam, Douglas, Ferry, Island, Jefferson, King, Kitsap, Lincoln, Okanogan, Pend Oreille, San Juan, Skagit, Snohomish, Spokane, Stevens, Whatcom, and that part of Grant lying north of parallel 47° 30′ north latitude.
- **South part of Washington:** NAD83(HARN)/Washington South (ftUS); EPSG:2927
  - Adams, Asotin, Benton, Clark, Columbia, Cowlitz, Franklin, Garfield, that part of Grant lying south of parallel 47° 30′ north latitude, Grays Harbor, Kittitas, Klickitat, Lewis, Mason, Pacific, Pierce, Skamania, Thurston, Wahkiakum, Walla Walla, Whitman and Yakima.

One more projected coordinate system is worth to know: EPSG:3857 - WGS 84 / Pseudo-Mercator, which is used for rendering maps in Google Maps, OpenStreetMap, etc.

Setting CRS in R

We can check the Coordinate Reference System setting using `st_crs()`. The light railway station data uses Projected Coordinate System with EPSG code 2926: **NAD83(HARN) [GCS] + Washington North (ftUS) [Projection]**.

```
st_crs(lrstations)
```

```
## Coordinate Reference System:
##   User input: NAD83(HARN) / Washington North (ftUS)
##   wkt:
## PROJCRS["NAD83(HARN) / Washington North (ftUS)",
##     BASEGEOGCRS["NAD83(HARN)",
##         DATUM["NAD83 (High Accuracy Reference Network)",
```

```
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4152]],
##     CONVERSION["SPCS83 Washington North zone (US survey foot)",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",47,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",-120.833333333333,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",48.7333333333333,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",47.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",1640416.667,
##             LENGTHUNIT["US survey foot",0.304800609601219],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",0,
##             LENGTHUNIT["US survey foot",0.304800609601219],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["US survey foot",0.304800609601219]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["US survey foot",0.304800609601219]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["United States (USA) - Washington - counties of Chelan; Clallam; Douglas; Ferry; Grant
##         BBOX[47.08,-124.79,49.05,-117.02]],
##     ID["EPSG",2926]]
```

We will transform all spatial data to EPSG 2926: NAD83(HARN) [GCS] + Washington North (ftUS).

```
airbnb <- st_transform(airbnb, crs = st_crs(lrstations))
kcm_routes <- st_transform(kcm_routes, crs = 2926)
king_affordability <- st_transform(king_affordability, crs = st_crs(lrstations))
urban_village <- st_transform(urban_village, crs = 2926)
```

Common `st_*()` Functions

Please note that all the data manipulation methods we learned earlier, such as `fliter`, `select`, and `groupby`, can still be applied to spatial data objects. In this section, we will focus specifically on spatial operations. Through a few small, hands-on tasks, you'll learn how to manipulate and analyze spatial features using the `sf` package.

Task 1: How Many Airbnb Listings Are within 2000 Feet of Each Light Rail Station?

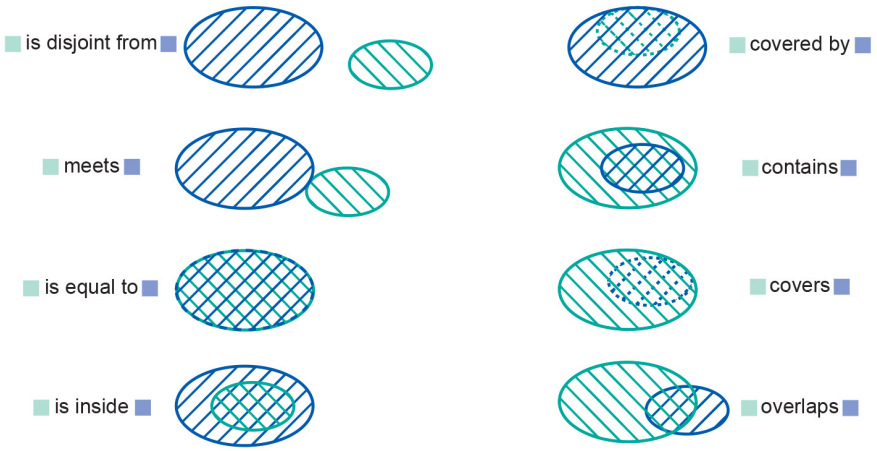
```
# the unit depends on selected crs
st_crs(lrstations)$units
```

```
## [1] "us-ft"
```

We first select all existing or under construction stations and create a buffer zone with radius of 2,000 feet using `st_buffer()`.

```
station_buffer <- st_buffer(lrstations %>% filter(STATUS == 'Existing / Under Construction'),
                            dist = 2000)
```

After that, we use `st_join()`. It is similar as `*_join()` functions we used before, but we use their spatial relations to join.



Spatial Relationships between Two Regions. Source: University of Twente

```r
# whether left objects will be within right objects
# default as left join - keeping all row in the left object
airbnb_near_station <- st_join(airbnb, station_buffer, join = st_within)
```

```r
airbnb_near_station %>%
  st_drop_geometry() %>% # drop the geometry column (optional)
  filter(!is.na(STATION)) %>%
  group_by(STATION) %>%
  summarise(airbnb_count = n()) %>%
  arrange(desc(airbnb_count)) %>%
  head()
```

```
## # A tibble: 6 × 2
##   STATION           airbnb_count
##   <chr>                    <int>
## 1 Westlake                   342
## 2 University Street          223
## 3 Capitol Hill               213
## 4 U District                 160
## 5 Pioneer Square              83
## 6 Judkins Park                73
```

Task 2: How to Calculate the KCM Bus Density at Census Tract Level?

To simplify this task, we define bus density as **total route length / tract area**. So, we need to get the length of `kcm_routes` in each tract and the area of each tract.

`st_intersection()` can return geometric intersection of two spatial objects. It clips or cuts geometries based on where they overlap.

```r
bus_by_tract <- st_intersection(kcm_routes, king_affordability)
```

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

Check the route 67: previously, there is only a single line but now it was cut into 10 separate and smaller lines. Each row corresponding to the part of the route that lies within one tract.

```r
bus_by_tract %>% filter(ROUTE_NUM == 67) %>% select(NAME)
```

```
## Simple feature collection with 20 features and 1 field
## Geometry type: GEOMETRY
## Dimension:     XY
## Bounding box:  xmin: 1272282 ymin: 241318.1 xmax: 1282693 ymax: 262024.1
## Projected CRS: NAD83(HARN) / Washington North (ftUS)
## First 10 features:
##                                           NAME                       geometry
## 70   Census Tract 44.01; King County; Washington LINESTRING (1275143 248530,...
## 70.1 Census Tract 42.02; King County; Washington MULTILINESTRING ((1282187 2...
## 70.2    Census Tract 20; King County; Washington MULTILINESTRING ((1274936 2...
## 70.3 Census Tract 53.04; King County; Washington LINESTRING (1275999 242548....
## 70.4 Census Tract 53.03; King County; Washington MULTILINESTRING ((1280547 2...
## 70.5 Census Tract 44.02; King County; Washington LINESTRING (1275016 246063....
## 70.6 Census Tract 52.01; King County; Washington MULTILINESTRING ((1274753 2...
## 70.7 Census Tract 53.05; King County; Washington LINESTRING (1275435 243107....
## 70.8    Census Tract 45; King County; Washington LINESTRING (1274812 248653....
## 70.9 Census Tract 12.01; King County; Washington MULTILINESTRING ((1273607 2...
```

Then we calculate the total route length within each group. `st_length()` was used. Note: we have to define Projected Coordinate System in order to calculate length/area.

```
bus_by_tract <- bus_by_tract %>%
  mutate(route_length_ft = st_length(geometry))  # define a new column
```

Then, we need to aggregate the length by `GEOID`.

```
bus_length_summary <- bus_by_tract %>%
  st_drop_geometry() %>% # drop geometry (optional)
  group_by(GEOID) %>%
  summarise(total_length_ft = sum(as.numeric(route_length_ft))) # replace NA with 0
```

Similarly, we calculate the area of each census tract using `st_area()` and `*_join` the summary data:

```
tracts_density <- king_affordability %>%
  mutate(area_ft2 = as.numeric(st_area(geometry))) %>%
  left_join(bus_length_summary, by = "GEOID") %>%
  mutate(bus_density = (total_length_ft / area_ft2),
         bus_density = replace_na(bus_density, 0))
```

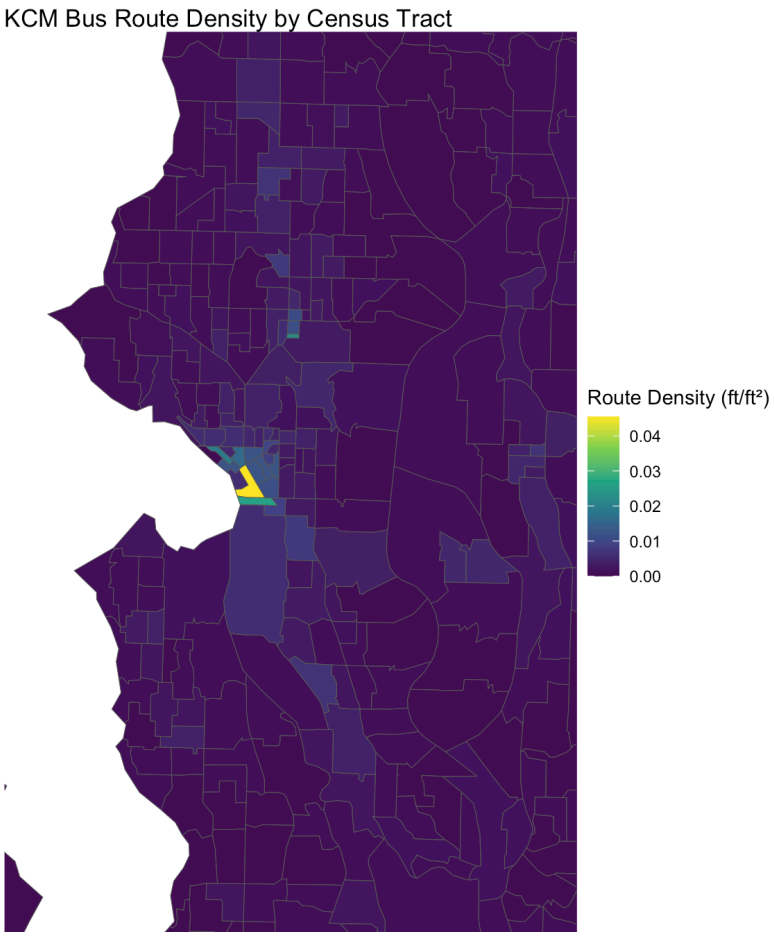
Remember, we are using the data for the whole King County. But we may only want to focus on Seattle, we can adjust the boundary of map by `coord_sf`. Because we are using a

projected coordinate system, we do not know the real boundary (X and Y). `st_bbox()` can be used to check it.

```
st_bbox(tracts_density)
```

```
##       xmin       ymin       xmax       ymax
## 1220277.22   31385.15 1583151.89  287656.30
```

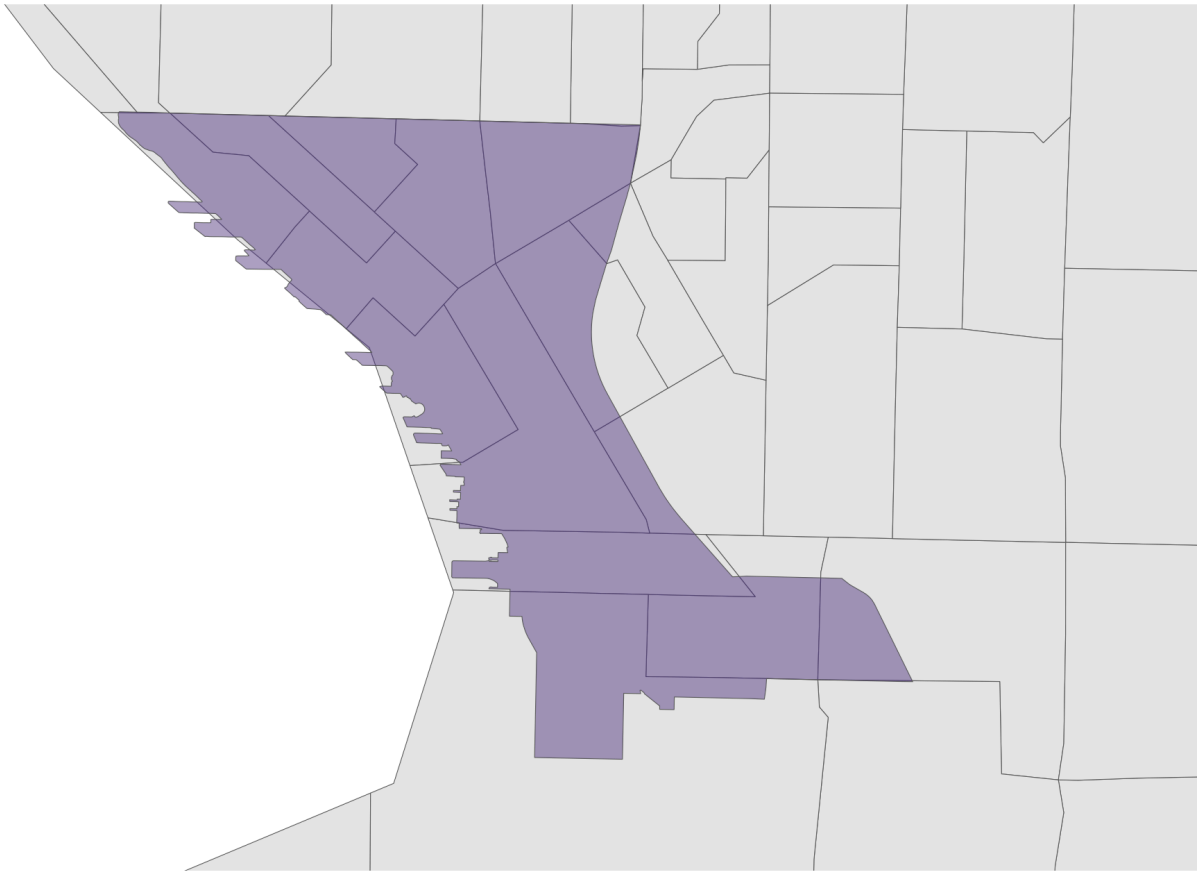
```
ggplot(tracts_density) +
  geom_sf(aes(fill = bus_density)) +
  scale_fill_viridis_c(option = "viridis") +
  labs(
    title = "KCM Bus Route Density by Census Tract",
    fill = "Route Density (ft/ft²)"
  ) +
  theme_void() +
  coord_sf(
    xlim = c(1240000, 1310000), # it is kind of annoying that we need to adjust them
    ylim = c(170000, 280000),
    expand = FALSE)
```



Task 3: How Many People are Living in Seattle Downtown?

Downtown is defined in `urban_village` but population data are in `king_affordability` at census tract level. They have total different boundary:

```
ggplot() +
  geom_sf(data = king_affordability) +
  geom_sf(data = urban_village %>% filter(UV_NAME == 'Downtown'), fill = "#4B2E83", alpha=0.5) +
  theme_void() +
  coord_sf(
    xlim = c(1260000, 1280000), # it is kind of annoying that we need to adjust them again
    ylim = c(218000, 231000),
    expand = FALSE)
```



Based on the data we have, we assume the population are evenly distributed within the tract and calculate the population by proportion. Please note, in reality, City of Seattle may have population data for downtown, or we can use smaller unit, such as census block, to avoid such boundary issue.

The estimated downtown population can be represented as (i is the index for census tracts

with overlaps):

$$\text{Estimated Population}_{DT} = \sum_i \left(\frac{\text{Area of Overlap}_i}{\text{Total Tract Area}_i} \times \text{Population}_i\right)$$

```
# calculate the area of each tract
king_affordability <- king_affordability %>%
  mutate(area_ft2 = as.numeric(st_area(geometry)))


downtown_pop <- st_intersection(king_affordability,
                                urban_village %>% filter(UV_NAME == "Downtown")) %>%
  mutate(
    overlap_area = as.numeric(st_area(geometry)),
    # calculate the area of overlap
    pop_est = (overlap_area / area_ft2) * total_pop
  ) %>% # calculate the population for each overlap
  summarise(total_pop_est = round(sum(pop_est, na.rm = TRUE))) # sum to the total population


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

In the end, you may find how painful or tedious mapping can feel in R (or in any programming language). For spatial data exploration and mapping, I strongly recommend using ArcGIS or QGIS. They're built for that purpose. However, R remains extremely powerful for spatial data analysis—especially when it comes to working with attributes, performing statistical summaries, and combining spatial and non-spatial data in a fully reproducible workflow. So, **be strategic and combine the best of both!**

📚 TODO: Review Spatial Data and Mapping in R

2 points

Carefully review and run all code chunks. You are encouraged to modify the arguments to explore how they affect. Reply "yes" in Canvas once you have reviewed and tested them.

Acknowledgement

The materials are developed by Haoyu Yue based materials from Dr. Feiyang Sun at UC San Diego, Siman Ning and Christian Phillips at University of Washington, Dr. Charles Lanfear at University of Cambridge.