CS 205
Project 5

Due: Friday, April 5

Capital BikeShare is a bicycle-sharing system in Washington, D.C. At any of over 700 stations, a registered user can unlock and check out a bicycle. After use, the bike can be returned to the same station or any of the other stations.

Such sharing systems require careful management. There need to be enough bikes at each station to satisfy the demand, and enough empty docks at the destination station so that the bikes can be returned. At each station, bikes are checked out and are returned over the course of the day. An imbalance between bikes checked out and bikes returned calls for the system administration to truck bikes from one station to another. This is expensive!

In order to manage the system, and to support a smart-phone app so that users can find out when bikes are available, Capital BikeShare collects real-time data on when and where each bike is checked out or returned, and how many bikes and empty docks there are at each station. Capital BikeShare publishes the station-level information in real-time. The organization also publishes, at the end of each quarter of the year, the historical record of each bike rental in that time.

You can access the data from the Capital BikeShare web site. Doing this requires some translation and cleaning, skills that are introduced in Chapter 16. For this project, however, already translated and cleaned data are provided in two tables:

- Stations giving information about location of each of the stations in the system.
  ```
  Stations <- read_csv("https://mcsp.wartburg.edu/letsche/cs205/DC-Stations.csv")
  ```
- Trips giving the rental history over 2023:
  ```
  data_site <- "https://mcsp.wartburg.edu/letsche/cs205/Bike2023-Small.rds"
  Trips <- read_rds(data_site)
  ```

The `Trips` data table is a random subset of 20,000 trips from the full annual data. Start with this small data table to develop your analysis commands. When you have this working well, you can access the full data set of over 4 million events by removing `-Small` from the name of `data_site`. The .Rmd file you turn in should work with the full data set.

In this activity, you'll work with just a few variables:

- From `Stations`:
  - The latitude (`lat`) and longitude (`long`) of the station.
  - `name`: the station's name
- From `Trips`:
  - `sstation`: the name of the station where the bicycle was checked out (the start station)
  - `estation`: the name of the station to which the bicycle was returned (the end station)
  - `client`: indicates whether the customer is a "regular" customer who has paid a yearly membership fee, or a "casual" user who has paid a fee for a five-day membership
  - `sdate`: the time and data of check-out
  - `edate`: the time and date of return

Time/dates are stored as a special kind of number: a *POSIX date*. You can use sdate and edate in the same way that you would use a number. For instance, Figure 1 shows the distribution of times that bikes were checked out.

```
Trips %>%
   ggplot(aes ( x = sdate )) +
   geom_density ( fill = "gray", color = NA)
```

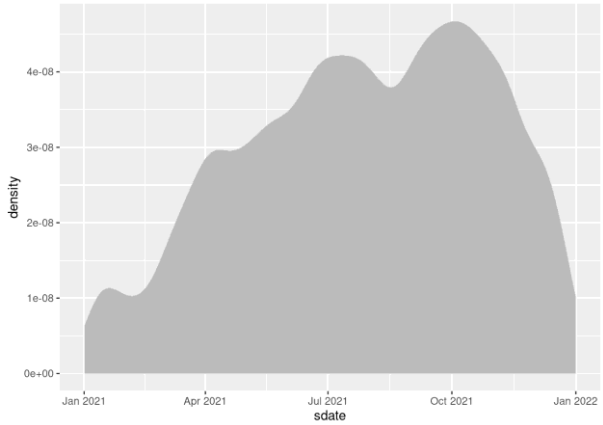

Figure 1. Use of shared bicycles over 2021.

How long?

Question 1: Make a box-and-whisker plot, like Figure 2, showing the distribution of the duration of rental events, broken down by client type. The duration of the rental can be calculated as `as.numeric( edate - sdate )`. The units will be either hours, minutes, or seconds. It should not be trouble for you to figure out which one. For this homework, express durations in minutes.

When you make your plot, you will likely find that the axis range is being set by a few outliers. These may be bikes that were lost or forgotten. Arrange your scale to ignore those outliers.

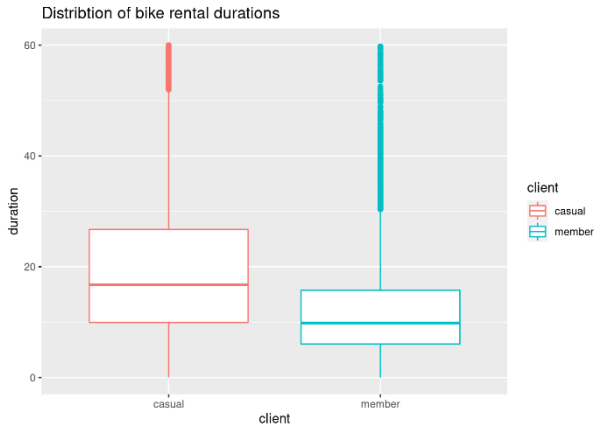

Figure 2. The distribution of bike-rental durations as a box-and-whisker plot.

When are bikes used?

The `sdate` variable in Trips indicates the date and time of day that the bicycle was checked out of the parking station.

Often, you will want discrete components of a date. For instance:

Date Component	Function (`lubridate` package)
Day of the year (1-365)	`lubridate::yday(sdate)`
Day of the Week (Sunday to Saturday)	`lubridate::wday(sdate)`
Hour of the Day	`lubridate::hour(sdate)`
Minute in the Hour	`lubridate::minute(sdate)`

Question 2: Make histograms or density plots of each of these discrete components, comparing and contrasting what you see in the data for the two client types. For example, Figure 3 shows *combined* data for both members and causal users that indicates that few bikes are checked out before 5 am, and that there are busy times around the rush hour 5 pm**. Remember to look at both client types unless told otherwise.**

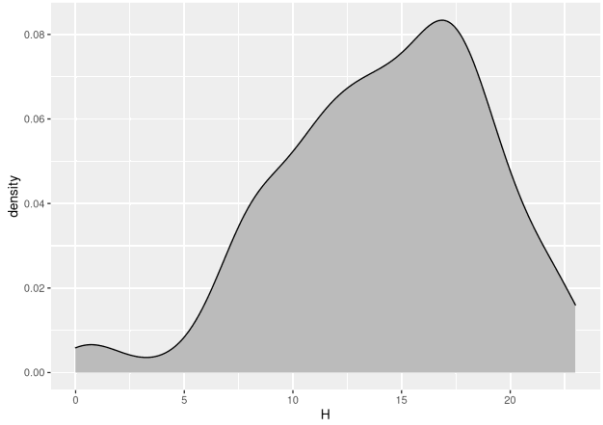

Figure 3. Distribution of bike trips by hour of the day.

A similar sort of display of events per hour can be accomplished by calculating and displaying each hour's count, as in Figure 4.

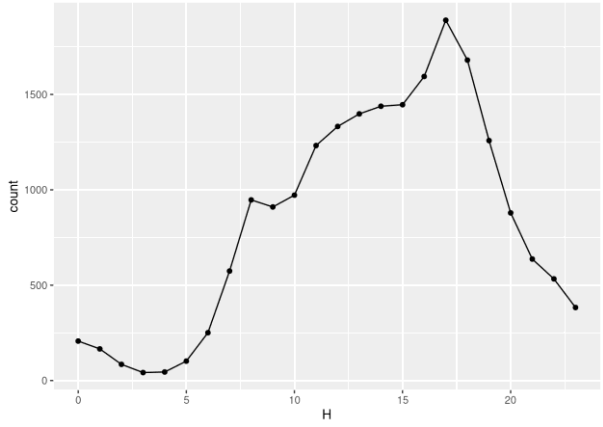

Figure 4. The number of events in each hour of the day. Compare to the scale in Figure 3. Why are the scales different?

The graphic shows a lot of variation in bike use over the course of the day. Now consider an additional variable.

Question 3: Group the bike rentals by three variables: hour of the day, day of the week, and client type. Find the total number of events in each grouping and plot this count versus hour. Use the `group` aesthetic to represent one of the other variables and faceting to represent the other.

Question 4: Make the same sort of display of how bike rentals vary by hour, day of the week, and client type, but use `geom_density()` rather than grouping and counting. Compare the two displays – one of discrete counts and one of density – and describe any major differences.

How far?

Find the distance between each pair of stations. You know the position from the `lat` and `long` variables in `Stations`. This is enough information to find the distance. The calculation has been implemented in the `haversine()` function:

```
source("https://mcsp.wartburg.edu/letsche/cs205/haversine.R")
```

`haversine()` is a transformation function. To use it, create a data table where a case is a *pair* of stations and there are variables for the latitude and the longitude of the starting and ending station. To do this, join the `Station` data to itself. The following statement shows how to create appropriately named variables for joining.

```
Simple <-
   Stations %>%
   select(name, lat, long) %>%
   rename( sstation = name )
Simple2 <-
   Simple %>%
   rename( estation = sstation, lat2 = lat, long2 = long)
```

Look at the `head()` of `Simple` and `Simple2` and make sure you understand how they are related to `Stations`.

The joining of `Simple` and `Simple2` should match every station to every other station. This sort of matching doesn't make use of any matching variables; everything is matched to everything else. (Since a ride can start and end at the same station, it also makes sense to match each station to itself.) This is called a *full outer join*. (A full outer join matches every case in the left table to each and every case in the right table.)

First, try the full outer join of just a few cases from each table, for example, four from the left and four from the right.

```
merge( head ( Simple, 4), head( Simple2, 3), by = NULL)
```

Make sure you understand what the full outer join does before proceeding. For instance, you should be able to predict how many cases the output will have when the left input has *n* cases and the right has *m* cases.

Question 5: There are 766 cases in the `Stations` data table. How many cases will there be in a full outer join of `Simple` to `Simple2`?

It's often impractical to carry out a full outer join. For example, joining `BabyNames` to itself with a full outer join will generate a result with more than *four trillion* cases!

Perform the full outer join and then use `haversine()` to compute the distance between each pair of stations.

```
StationPairs <- merge( Simple, Simple2, by = NULL)
```

Check your result for sensibility. Make a histogram of the station-to-station distances and explain where it looks like what you would expect. (*Hint:* you could use the Internet to look up the distance from one end of Washington, D.C. to the other.)

```
PairDistances <-
   StationPairs %>%
   mutate( distance = haversine( lat, long, lat2, long2 )) %>%
   select( sstation, estation, distance)
```

Once you have `PairDistances`, you can join it with `Trips` to calculate the start-to-end distance of each trip.

Question 6: Look at the variables in `Stations` and `Trips` and explain why `Simple` and `Simple2` were given different variable names for the station.

An `inner_join()` is appropriate for finding the distance of each ride. (Watch out! The `Trips` data and the PairDistances data are large enough that the join is expensive! It will take some time.)

Question 7: Display the distribution of the ride distances of the rides. Compare it to the distances between pairs of stations. Are they similar? Why or why not?

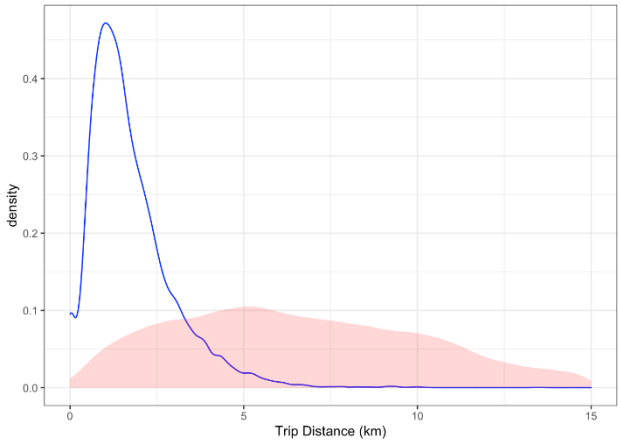

Figure 5. The distribution of trip lengths compared to the distribution of distances between pairs of stations (shaded).

Mapping the Stations

You can draw detailed maps with the `leaflet` package. You will probably need to install it. Don't include the installation in your .Rmd file, but assume it's already been installed.

```
install.packages("leaflet")
```

`leaflet` works much like `ggplot`, but provides special facilities for maps. Here's how to make the simple map:

```
library(leaflet)
stationMap <-
   leaflet( Stations ) %>%                          # like ggplot
   addTiles() %>%                                   # add the map
   addCircleMarkers( radius = 2, color = "red " ) %>%
   setView( -77.04, 38.9, zoom = 12 )
```

To display the map, use the object name as a command:

```
stationMap
```

Notice that in your knit html, the map is interactive. Exploring the map will explain (to some extent) the distribution you found in question 7.

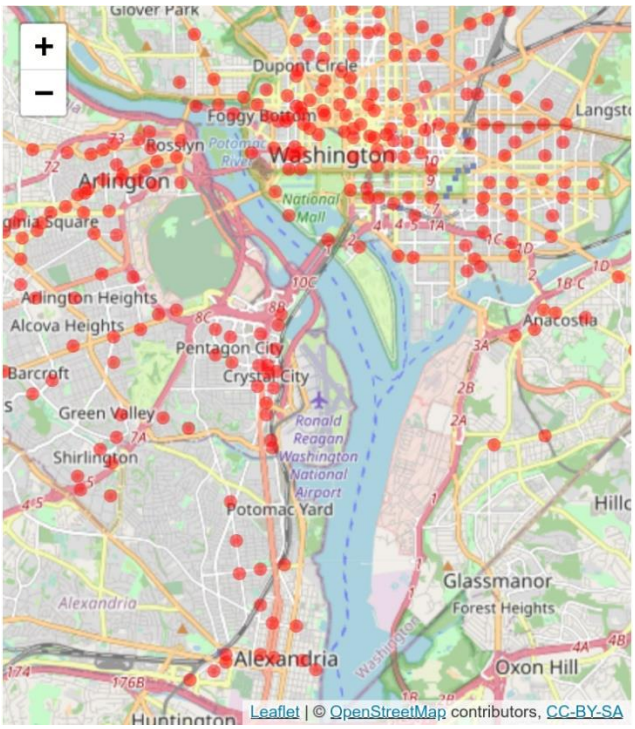

Figure 6. A map of station locations drawn with the `leaflet()` function in the `leaflet` package.

Long-Distance Stations

Question 8 (more challenging): Around each station on the map, draw a circle whose radius reflects the median distance covered by rentals starting at that station. To draw the circles, use the same leaflet commands as before, but add in a line like this:

```
addCircles( radius = ~ med, color = "blue", opacity = .0001 )
```

For `addCircles()` to draw circles at the right scale, the units of median distance should be presented in meters rather than kilometers. This will create too much overlap, unfortunately. So, set the radius to be half or one-third the median distance in meters. From your map, explain the pattern you see in the relationship between station location and median distance. **Given the amount of data in the full table, this may take your computer a <u>long</u> time. If that is a problem, perform this operation on the smaller data set, but be sure to indicate you did that. Your code should be able to work on the full data set.**