Explore Citi Bike Data

Code

Citi Bike Data

Welcome to the off platform project focused on visualization. In this project, we will be exploring data associated with the New York City bike share program, Citi Bike! Remember, it may be easiest to read these instructions by clicking on the “Preview” button in RStudio.

There are over 850 Citi Bike stations in New York City — users check a bike out from a starting station and then dock that bike at a different station when they reach their destination. Citi Bike offers a variety of memberships, but most memberships allow for trips between 30 and 45 minutes — this will be relevant once we start digging into the dataset.

Citi Bike publically releases a variety of datasets. We’ve included a dataset containing information about individual trips from January of 2020. If you’d like to download a more recent dataset, or investigate other data that Citi Bike provides, take a look at their System Data (https://www.citibikenyc.com/system-data) page. There are so many interesting questions that you can investigate with this data — we’re about to walk you through a few, but we’d love to see what else you can discover!

Investigate the Data

We’ve included a file named `january_trips.csv` . Load this data into a dataframe using the `read.csv()` function. Note that this dataset is *big*. It may take a few minutes to load — if you’d like to use a subset of this data that will take less time to load, we’ve included a file named `january_trips_subset.csv` as well. We strongly recommend using this subset of the data. When loading these datasets, make sure that the .csv files are in the same directory as this .Rmd file.

Note that because this dataset is so large, it may take several seconds to load the data or preview your R Notebook.

Hide

```
# Load the data set into a data frame

all_data <- read.csv("january_trips_subset.csv")
```

Now that we’ve loaded the dataset, the easiest way to investigate the data is to click on the variable name in the “Global Environment” tab in RStudio. This will let you scroll through the data as if it were a spreadsheet. If you want to display some of the data in this document, call the `head()` function using your data frame as a parameter. Make sure to scroll through all of the columns!

Hide

```
# Investigate the data

head(all_data)
```

	X	tripduration	starttime	stoptime	start.station.
	<int>	<int>	<chr>	<chr>	<ir
1	1	1420	2020-01-12 11:58:32.4220	2020-01-12 12:22:12.5870	32
2	2	539	2020-01-05 09:37:51.3740	2020-01-05 09:46:50.7600	1
3	3	238	2020-01-25 10:47:55.5770	2020-01-25 10:51:53.9540	2
4	4	189	2020-01-17 08:03:12.6320	2020-01-17 08:06:21.8730	36
5	5	741	2020-01-23 22:42:59.9520	2020-01-23 22:55:21.5230	34
6	6	568	2020-01-15 20:37:44.8000	2020-01-15 20:47:13.2320	2

6 rows | 1-6 of 16 columns

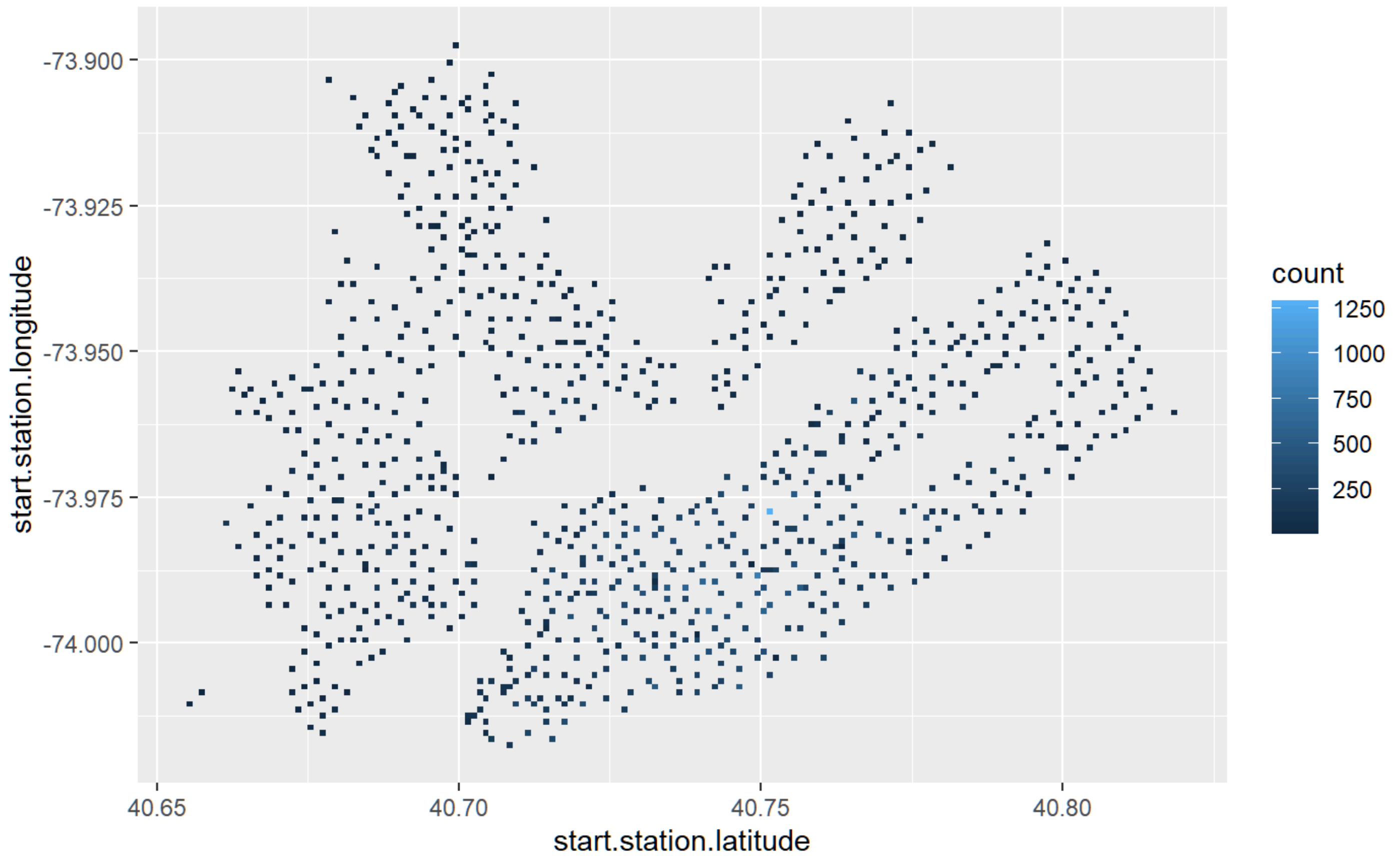
Since we have information about the starting and ending location for each trip, let's quickly make a heat map of the starting locations. Make a heat map using `ggplot()` and `geom_bin2d()` . If you make the bin width for each axis very small (we used `0.001`), you should see the shape of Manhattan, Brooklyn, and Queens! Check out the rectangle in Manhattan with no stations — that's Central Park!

Make sure to install and load `ggplot2` and `dplyr` !

Hide

```
# Install and load ggplot2 and dplyr
library(ggplot2)
library(dplyr)

# Create a heatmap
starting_loc_hm = ggplot(data=all_data, aes(x=start.station.latitude, y =start.station.longitud
e)) +
  geom_bin2d(binwidth = c(0.001,0.001))
starting_loc_hm
```



We also have the duration of each trip. Using these features, we can calculate the average speed of each trip.

Finally, since we also have the date of birth for each rider, we can calculate their age.

Let's work towards building a line graph where age is on the x axis and average speed is on the y axis. This graph could help Citi Bike understand how their users are using their bikes. For example, if younger riders tend to bike much faster than older riders, Citi Bike may want to think about ways to encourage younger riders to bike more cautiously.

Before we begin to work on the visualization, we'll have to work a bit with our dataset to get all of the relevant columns.

Modifying the Data Frame: Subset and Age

Since this dataset is so big, we recommend using the `filter()` function to grab a subset of the data. For example, you could grab only the rows where the duration was under 900 seconds (15 minutes). We stored these rows in a new data frame named `short_trips`. Note that we did this only to speed up the runtime of upcoming operations. This is completely optional — if you're happy to wait a bit for each operation, feel free to use the complete data set.

Hide

```
# Create a subset of the data
short_trips = all_data %>%
  filter(tripduration < 900)
head(short_trips)
```

	X	tripduration	starttime	stoptime	start.station.
	<int>	<int>	<chr>	<chr>	<ir
1	2	539	2020-01-05 09:37:51.3740	2020-01-05 09:46:50.7600	1
2	3	238	2020-01-25 10:47:55.5770	2020-01-25 10:51:53.9540	2
3	4	189	2020-01-17 08:03:12.6320	2020-01-17 08:06:21.8730	36
4	5	741	2020-01-23 22:42:59.9520	2020-01-23 22:55:21.5230	34
5	6	568	2020-01-15 20:37:44.8000	2020-01-15 20:47:13.2320	2
6	7	710	2020-01-11 12:11:02.1250	2020-01-11 12:22:52.8890	2

6 rows | 1-6 of 16 columns

Hide

```
NA
```

Next, let's add a column called `age` to the data frame. `age` should be 2020 minus `birth.year` (this data was collected in 2020). Use the `mutate()` function to do this. After calling mutate, make sure to save the result in a variable. We should save the result back to `short_trips` .

Hide

```
# Add the age column
short_trips = short_trips %>%
  mutate(age = 2020 - birth.year)
head(short_trips)
```

	X	tripduration	starttime	stoptime	start.station.
	<int>	<int>	<chr>	<chr>	<ir
1	2	539	2020-01-05 09:37:51.3740	2020-01-05 09:46:50.7600	1
2	3	238	2020-01-25 10:47:55.5770	2020-01-25 10:51:53.9540	2
3	4	189	2020-01-17 08:03:12.6320	2020-01-17 08:06:21.8730	36
4	5	741	2020-01-23 22:42:59.9520	2020-01-23 22:55:21.5230	34
5	6	568	2020-01-15 20:37:44.8000	2020-01-15 20:47:13.2320	2
6	7	710	2020-01-11 12:11:02.1250	2020-01-11 12:22:52.8890	2

6 rows | 1-6 of 17 columns

Hide

```
NA
```

Modifying the Data Frame: Distance

In order to calculate the speed of each biker, we need to find the total distance they traveled. Luckily, we have information about the starting and ending latitudes and longitudes. Let’s use those four columns to create a new column named `distance`.

There are many different ways to calculate distance. We’ll walk you through the strategy we used. However, before following along with us, challenge yourself to solve this problem on your own — one of the goals of these off platform projects is to get comfortable problem solving on your own. Try to use Google to find the packages you might need to calculate the distance between latitude and longitude coordinates. Use the code block below to try solving this problem on your own. We’ll walk you through our solution in the following section.

As you write your code that edits the data frame, consider printing the head of the data frame to validate the work you are doing!

Hide

```
# Try creating a distance column in your data frame here:
```

There are many different strategies to calculate the distance between two points. The simplest way to do this would be to find the length of the straight line between the two points. This is a massive assumption to make — it would be remarkable if any of these bike trips traveled in a straight line between the two points without making any turns or curves.

That being said, finding the straight line distance is a good starting point. The `distHaversine()` function found in the `geosphere` library can calculate this distance.

First, install and load the `geosphere` library.

Next, use `dplyr` ’s `select()` function to create two new data frames that contain only the latitudes and longitudes of the starting and ending points. We called these data frames `starting_stations` and `ending_stations`.

Finally, use `dplyr` ’s `mutate()` function to add a column named `distance` to your data. `distance` should be calculated by calling `distHaversine()` using `starting_stations` and `ending_stations` as parameters.

If you get stuck, use `?distHaversine` to check the documentation to see more examples! You can also use the documentation to find the units of the result of `distHaversine()`!

Hide

```
# Use the geosphere library to create a distance column
library(geosphere)

starting_stations = short_trips %>%
  select(start.station.longitude, start.station.latitude)
ending_stations = short_trips %>%
  select(end.station.longitude, end.station.latitude)

# Using the mutate and distHaversine functions to create the distance column

short_trips = short_trips %>%
  mutate(distance = distHaversine(starting_stations, ending_stations))

head(short_trips)
```

	X	tripduration	starttime	stoptime	start.station.
	<int>	<int>	<chr>	<chr>	<ir
1	2	539	2020-01-05 09:37:51.3740	2020-01-05 09:46:50.7600	1
2	3	238	2020-01-25 10:47:55.5770	2020-01-25 10:51:53.9540	2
3	4	189	2020-01-17 08:03:12.6320	2020-01-17 08:06:21.8730	36
4	5	741	2020-01-23 22:42:59.9520	2020-01-23 22:55:21.5230	34
5	6	568	2020-01-15 20:37:44.8000	2020-01-15 20:47:13.2320	2
6	7	710	2020-01-11 12:11:02.1250	2020-01-11 12:22:52.8890	2

6 rows | 1-6 of 18 columns

Hide

```
NA
```

Modifying the Data Frame: Speed

Now that we've made a column containing the distance of each trip, let's make another column containing the average speed of each trip. This column should be easier to create than the previous — speed can be calculated by dividing the `distance` column by the `tripduration` column. This will give us the average speed in meters per second. Use the `mutate()` function to make the `speed` column!

Hide

```
# Create the speed column

short_trips = short_trips %>%
  mutate(speed = distance / tripduration)
head(short_trips$speed)
```

```
[1] 2.917876 2.393891 2.541098 1.858753 3.444751 1.600162
```

Modifying the Data Frame: Average Speed by Age

We're almost there! Now that we have the speed of every bike trip, we want to group those trips by `age` and find the average speed of each age.

Do this by piping your data frame into the `group_by()` function using `age` as a parameter.

Then pipe the result of that function into the `summarize()` function. `summarize()` works similarly to `mutate()` — pass `mean_speed = mean(speed)` to the `summarize()` function to create a new column named `mean_speed` . Save this new data frame in a variable called `average_speed_by_age` .

Hide

```
# Use group_by() and summarize() to get the mean speed of each age
average_speed_by_age = short_trips %>%
  group_by(age) %>%
  summarize(mean_speed = mean(speed))
head(average_speed_by_age)
```

age <dbl>	mean_speed <dbl>
17	2.407569
18	2.636186
19	2.628663
20	2.550774
21	2.492042
22	2.628113

6 rows

Hide

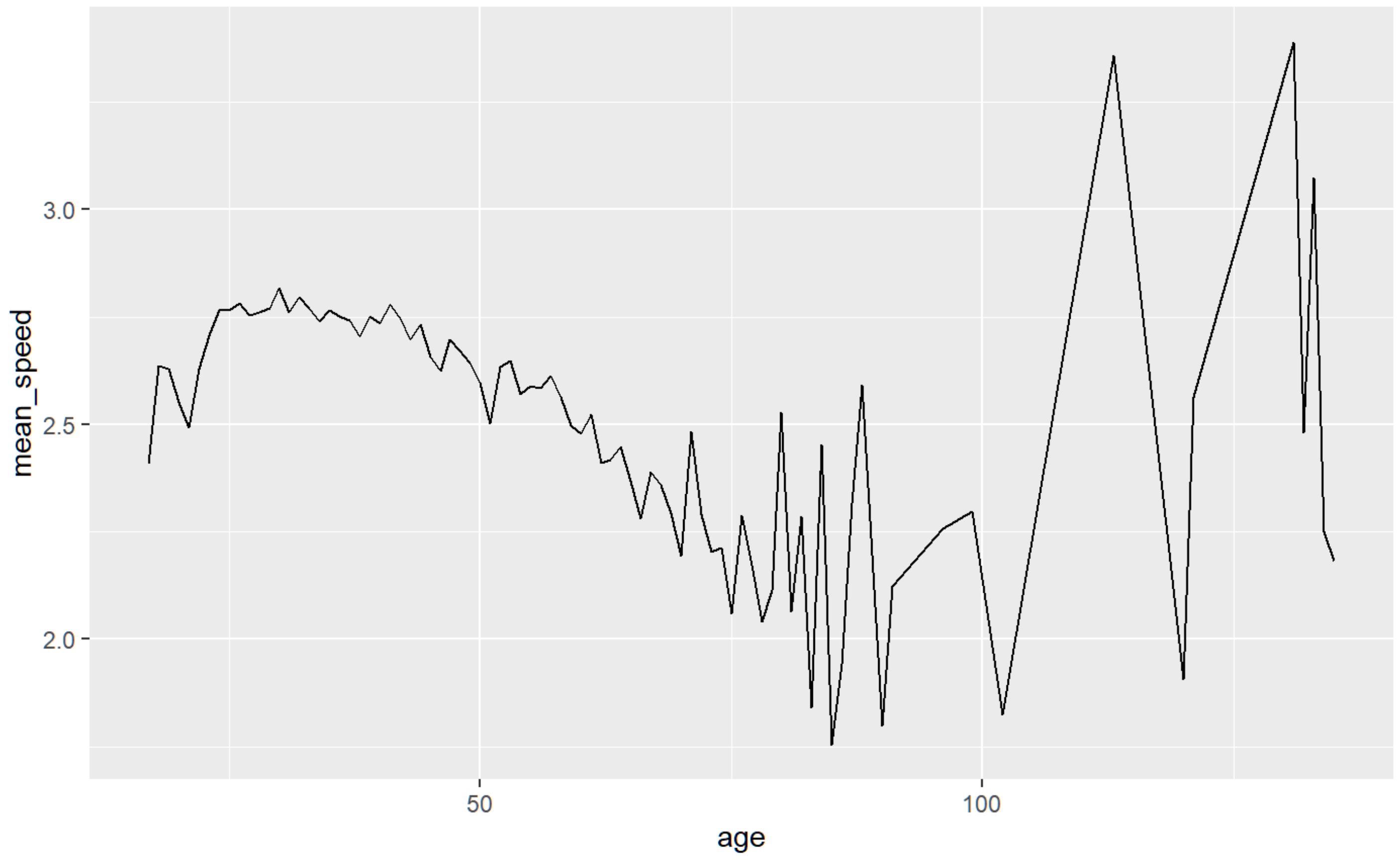
```
NA
```

Visualization!

We made it! We now have the average speed of every age in our dataset. Let's use `ggplot2` to make a line graph to see if younger people really do bike faster. Make sure to install and load `ggplot2` if you haven't done so already. Pass your data frame to `ggplot()` and add a `geom_line()` . `geom_line()` should contain an aesthetic where `x = age` and `y = mean_speed` .

Hide

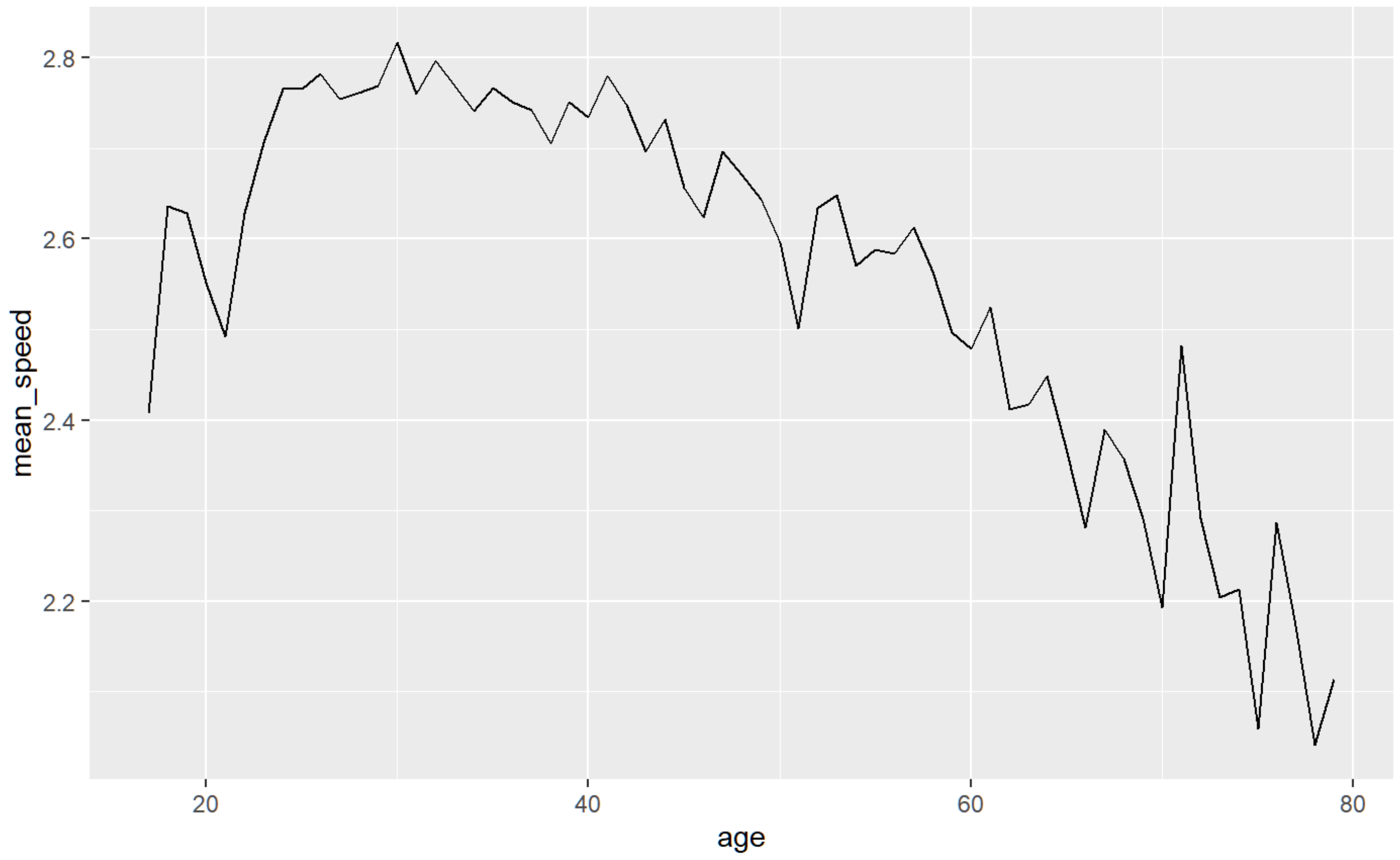
```
# Install and load ggplot2 to create a line graph of age and mean speed
line_viz = ggplot(data = average_speed_by_age,aes(x=age, y=mean_speed)) +
  geom_line()
line_viz
```



Nice work! Our intuition seems to be right — there's a steady drop in speed until we hit some outliers. It would be pretty surprising to see someone over the age of 100 using a bike share program! Let's filter the data to only show ages less than 80 and redraw our visualization.

Hide

```
# Filter the data frame to only contain rows where the age is less than 80
average_speed_by_age <- average_speed_by_age %>%
  filter(age < 80)
average_speed_by_age %>%
  ggplot() + geom_line(aes(x = age, y = mean_speed))
```

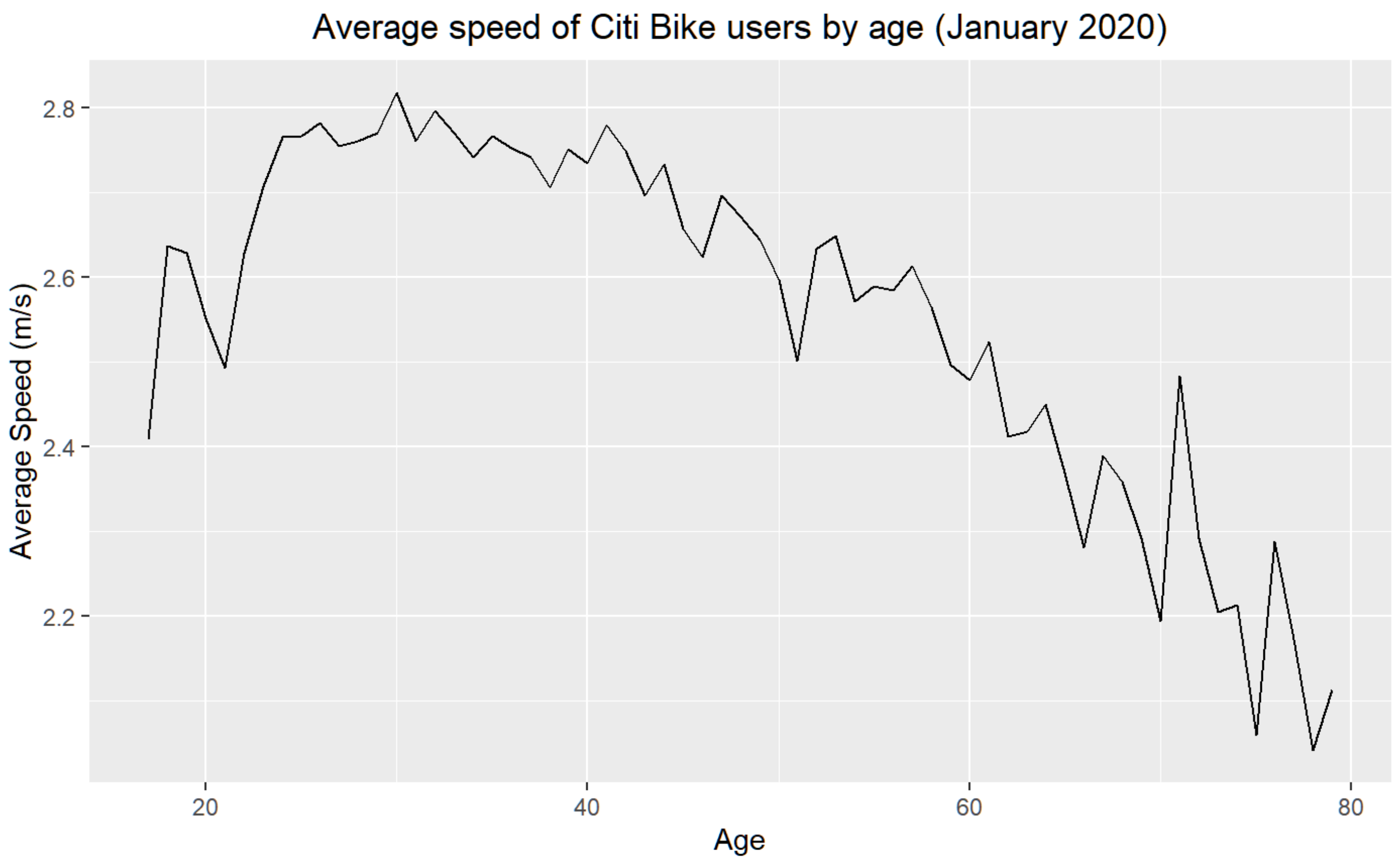


That looks a bit better! Let's do some work to make our graph look a bit more professional. Add a title and axis labels. We also centered our title!

Hide

```
# Add a title and label the axes

average_speed_by_age %>%
  ggplot() +
  geom_line(aes(x = age, y = mean_speed)) +
  labs(title = "Average speed of Citi Bike users by age (January 2020)", x = "Age", y = "Average
Speed (m/s)") +
  theme(plot.title = element_text(hjust = 0.5))
```



Filtering By Gender

Great work! This visualization gives us some great insights on how Citi Bike users are using their bikes. Let's dive even deeper! We can group our data by more than one feature.

Find your line of code that grouped our data by `age` . Copy it, but add `gender` as a parameter to the `group_by()` function. Save the result in a data frame named `average_speed_by_age_and_gender` . Inspect this data frame to see what it contains.

Hide

```
# Use group_by() again to group by both age and gender
average_speed_by_age_and_gender = short_trips %>%
  group_by(age,gender) %>%
  summarize(mean_speed = mean(speed))
```

```
`summarise()` has grouped output by 'age'. You can override using the `.groups` argument.
```

Hide

```
average_speed_by_age_and_gender
```

age <dbl>	gender <int>	mean_speed <dbl>
17	1	2.397661

age <dbl>	gender <int>	mean_speed <dbl>
17	2	2.526463
18	0	2.177695
18	1	2.690855
18	2	2.195785
19	0	1.603640
19	1	2.658394
19	2	2.514846
20	0	2.481425
20	1	2.599900

1-10 of 203 rows Previous **1** 2 3 4 5 6 ... 21 Next

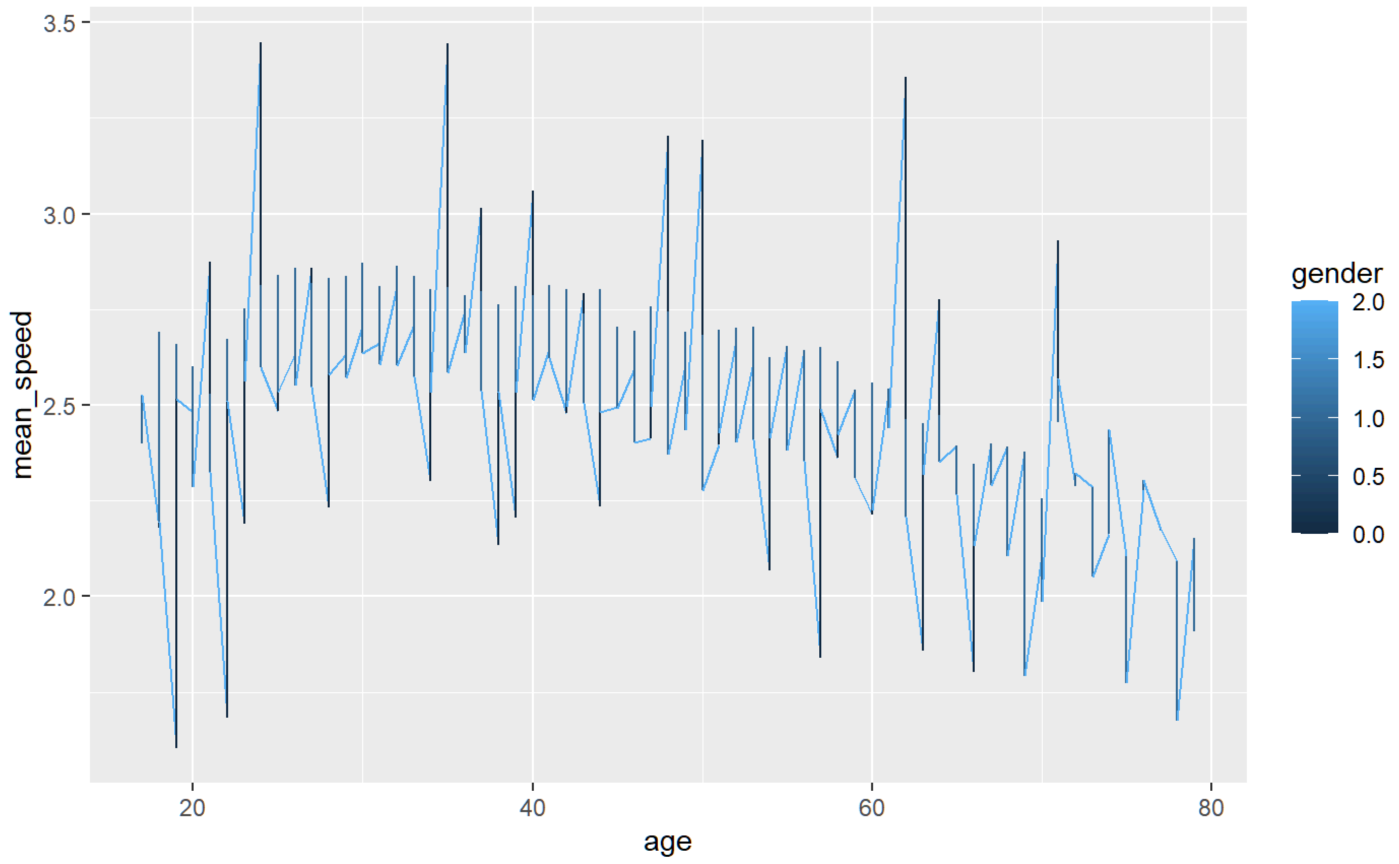
Let's now visualize the difference in average speed by age *and* gender. Note that if you look in the documentation for the data, a `0` represents a user that didn't specify their gender as male or female, a `1` represents a user identifying as male, and a `2` represents a user identifying as female.

The previous call to `ggplot()` and `geom_line()` should be close to what we want. Add the parameter `color = gender` to the aesthetic in `geom_line()` . Make sure you use the new data containing the gender information! You once again may want to filter out the ages greater than 80.

Note that this graph won't quite be what we want yet, but we're getting close!

Hide

```
# Make a line graph of your new filtered data frame
average_speed_by_age_and_gender %>%
  filter(age < 80) %>%
  ggplot() +
  geom_line(aes(x=age, y=mean_speed, color=gender))
```



It’s a bit hard to tell what is happening in that graph — but one oddity that sticks out is the scale used for gender. We know that our gender data is represented as three distinct values — `0` , `1` , and `2` . However, `ggplot()` is using gender as a continuous variable.

We can turn this column into a factor by using the `as.factor()` and `mutate()` functions. Pipe your data frame into the `mutate()` function and use `gender = as.factor(gender)` as the parameter.

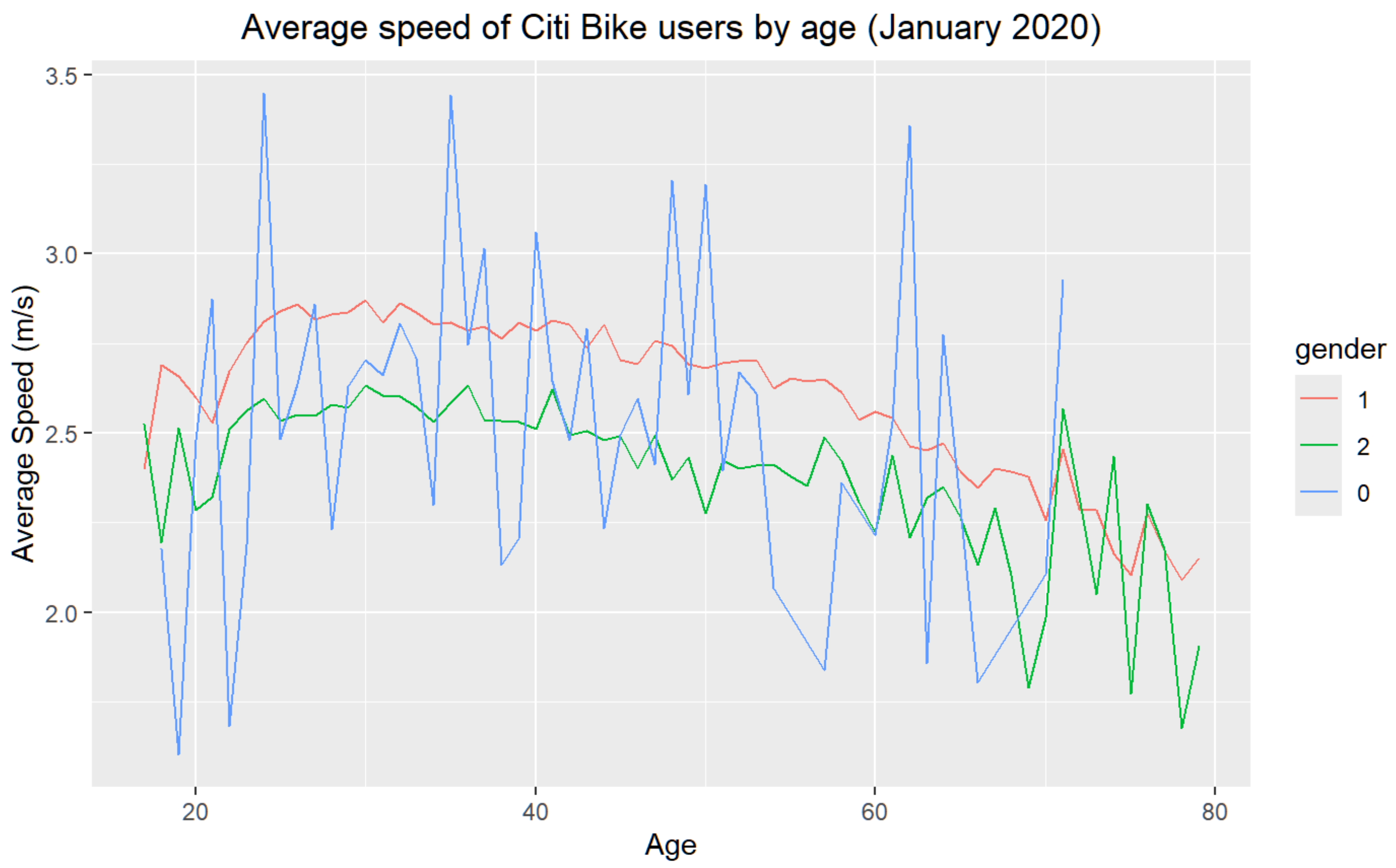
Note that when you make this column a factor, you will see a number of warnings. This warning is telling you that the type of the values in the gender column have been changed from integers to characters.

Then redraw your graph.

Hide

```
# Use mutate() and as.factor() to change the gender column into a factor.
average_speed_by_age_and_gender = average_speed_by_age_and_gender %>%
  mutate(gender = as.factor(gender))

average_speed_by_age_and_gender %>%
  filter(age < 80) %>%
  ggplot() +
  geom_line(aes(x = age, y = mean_speed, color = gender)) +
  labs(title = "Average speed of Citi Bike users by age (January 2020)", x = "Age", y = "Average
Speed (m/s)") +
  theme(plot.title = element_text(hjust = 0.5))
```



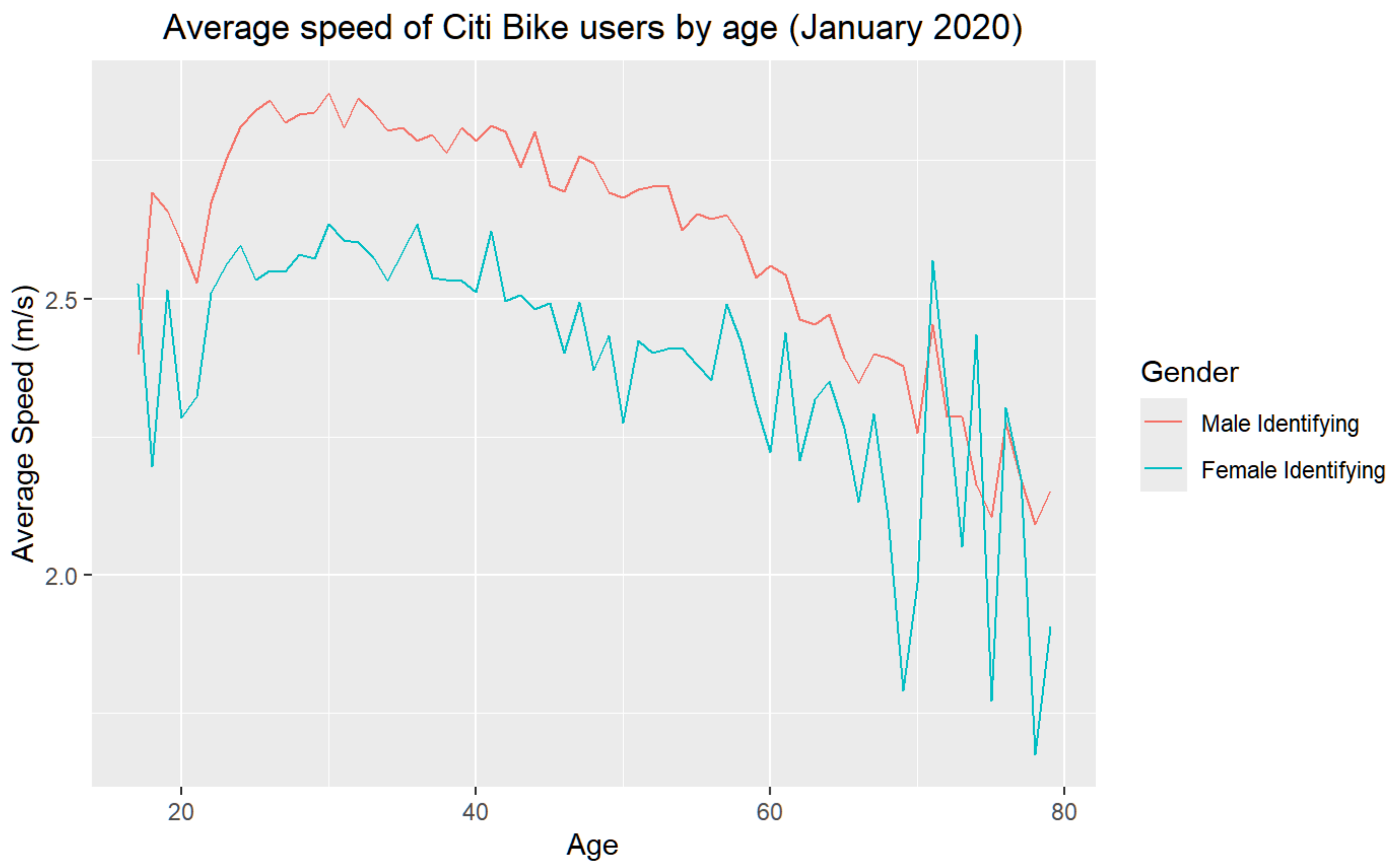
Nice work! We can now see the average speeds by age broken into the 3 genders Citi Bike accounts for. You can see that male-identifying users typically bike faster than female-identifying users. Users with an unknown gender don't follow a specific pattern — it is likely that there isn't enough data to properly visualize those users.

For our final version of this graph, we filtered out the users with a gender of `0`, and we labeled the lines as "Male Identifying" and "Female Identifying" using the `scale_color_discrete()` function. Take a look at the documentation for this function using `?scale_color_discrete` to change the label of each line.

Hide

```
# Filter the data frame to only include genders 1 and 2. Set appropriate labels for the legend

average_speed_by_age_and_gender %>%
  filter(age < 80, gender == 1 | gender == 2) %>%
  ggplot() +
  geom_line(aes(x = age, y = mean_speed, color = gender)) +
  labs(title = "Average speed of Citi Bike users by age (January 2020)", x = "Age", y = "Average
Speed (m/s)") +
  theme(plot.title = element_text(hjust = 0.5)) + scale_color_discrete(name = "Gender", labels =
c("Male Identifying", "Female Identifying"))
```



Making a Stacked Bar Plot Of Ages

Let’s make one final graph. For this graph, we’re interested in seeing the distribution of Citi Bike users’ age and gender. Let’s use a stacked bar plot to do this. We’ll want to create a bar plot where age is on the x axis, count is on the y axis, and each bar is split into genders.

Let’s start by using our `short_trips` dataset. We’ll want to call `group_by()` using this dataset and pipe the result to `tally()`. This will let us get a count of the number of bikers for each age and gender. We saved this new data frame in a variable called `age_counts`.

Hide

```
# Create the age_counts data frame
age_counts <- short_trips %>%
  group_by(age, gender) %>%
  tally()
head(age_counts)
```

age <dbl>	gender <int>	n <int>
17	1	48
17	2	4
18	0	1
18	1	97

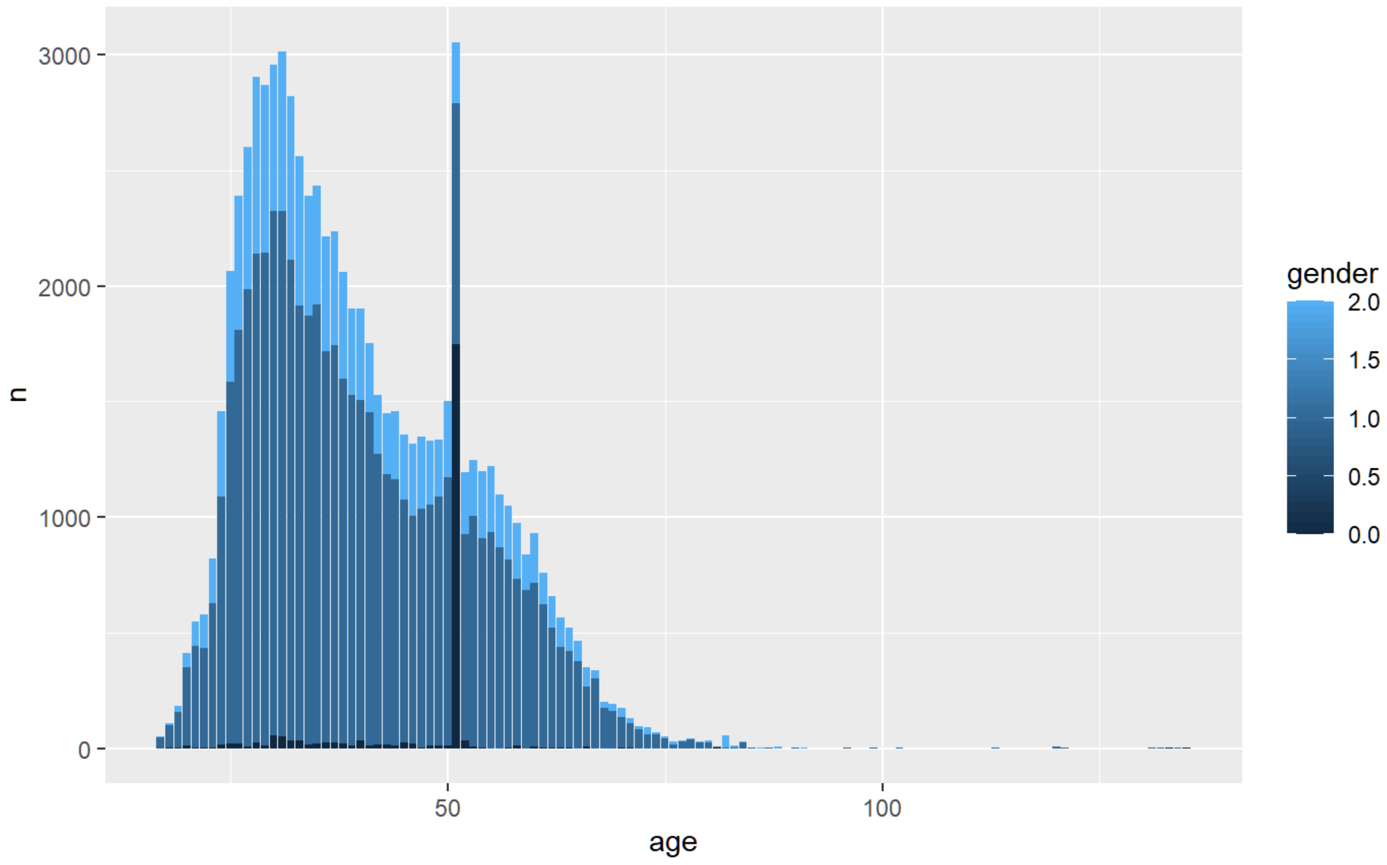
age <dbl>	gender <int>	n <int>
18	2	11
19	0	2

6 rows

If you look at the head of this new data frame, you'll see the counts are stored in a column named `n`. Let's now use `ggplot()` and `geom_col()` to create a stacked bar plot. `ggplot()` should have an `aes()` where `x = age`, `y = n` and `fill = gender`.

Hide

```
# Create the stacked bar plot
age_counts %>%
  ggplot() +
  geom_col(aes(x=age,y=n, fill=gender), position = "stack")
```



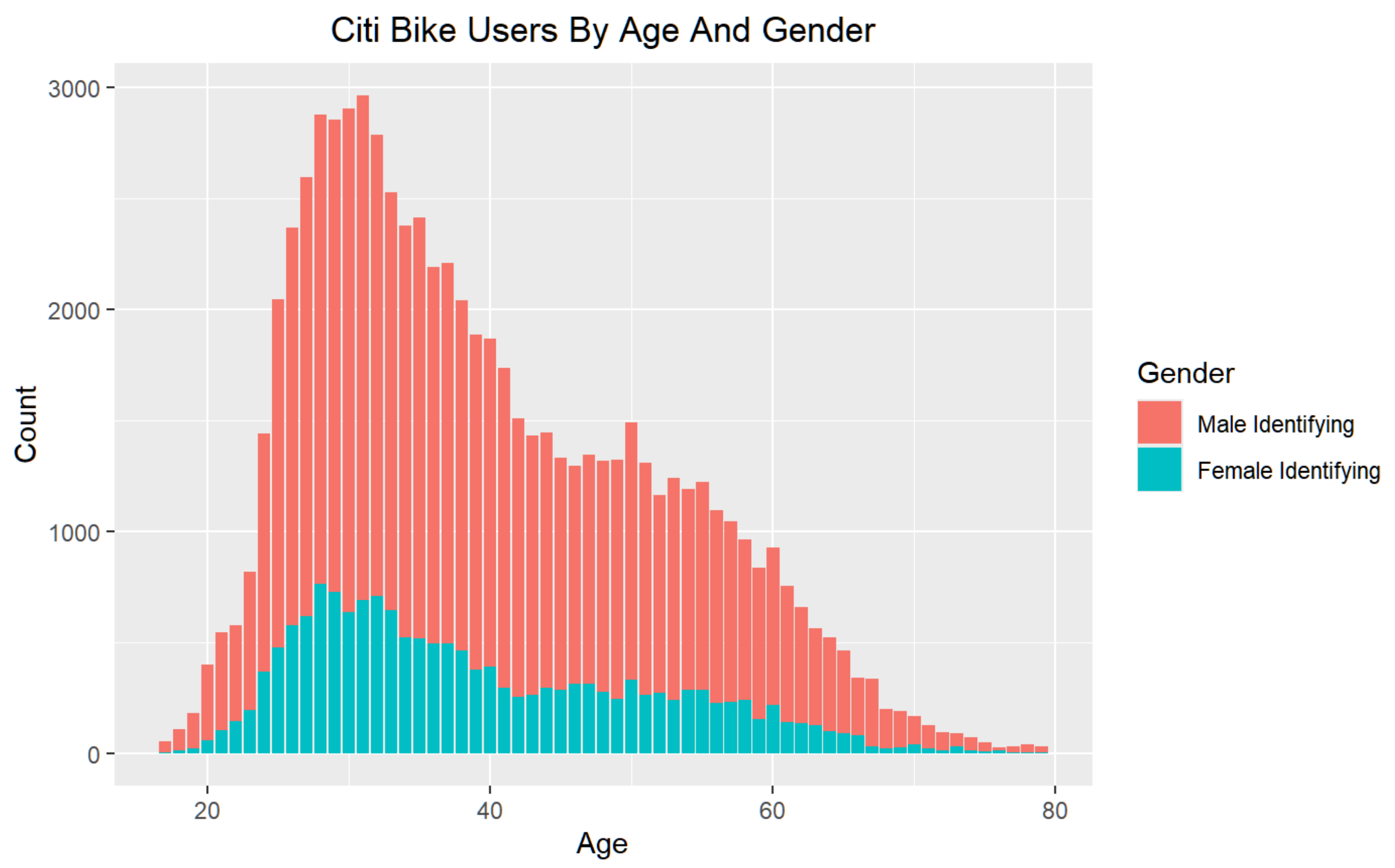
Great! There are some tweaks that we might want to make to this graph. First, gender right now is represented as an integer. It will make more sense if that column is represented as a factor. To do this, we can pass `as.factor(gender)` as the value for the x axis.

Next, it looks like we have some unusual data around the age of 50. It looks like there are a ton of bikers with an unknown gender at that age. This might be something we want to dig into a bit more, but for now, let's filter out the bikers with a gender of `0`. We also filtered out bikers with an age over `100` — that seems like an error in data collection as well.

Finally, we labeled and titled our graph using `labs()` and `scale_fill_discrete()`

Hide

```
# Filter and label your graph
age_counts %>%
  filter(age < 80, gender == 1 | gender == 2) %>%
  ggplot() +
  geom_col(aes(x = age, y = n, fill = as.factor(gender))) +
  labs(title = "Citi Bike Users By Age And Gender", x = "Age", y = "Count") +
  theme(plot.title = element_text(hjust = 0.5)) + scale_fill_discrete(name="Gender", labels = c
("Male Identifying","Female Identifying"))
```



Further Work

Great work! You've made several graphs that show a real difference in the way different groups of Citi Bike users bike. This could be a valuable asset in helping Citi Bike understand how to make bike riding safer in New York. However, there is so much more you can do with this data!

To begin, there are some major flaws in the way we calculated the speed. Specifically, we made some *huge* assumptions when calculating the distance of each bike ride. Instead of calculating the straight line distance using the geosphere library, we could take advantage of a service like Google Map's API to get a more accurate measurement of distance. If you're interested in look more into this problem, investigate getting a Google Maps API key (https://developers.google.com/maps/documentation/geocoding/get-api-key) and using a library like gmapsdistance (https://cran.r-project.org/web/packages/gmapsdistance/gmapsdistance.pdf).

Another great way to extend this project is to investigate other data that Citi Bike makes available. We used data found in the *Citi Bike Trip Histories* section on the System Data (https://www.citibikenyc.com/system-data) page. On the System Data page, you can find different dataset, including information about membership data and real time station data. You could use this real time data to track how the flow of bikes changes over the course of the day. You could investigate how the weather impacts membership. We would love to see any graphs or insights you produce!