5.4.1

Summary Statistics for Number of Rides by City Type

After Omar looks at your bubble chart, he suggests that you should add some statistical analysis because it will help you demonstrate the relevance of the data, especially the number of rides for each city. This will help V. Isualize and other stakeholders make decisions about which types of cities need more driver support.

The old adage "There are many ways to skin a cat" comes to mind when getting the summary statistics. We'll use and compare the following three ways to calculate the summary statistics:

- The Pandas `describe()` function on the DataFrame or Series.
- The Pandas `mean()`, `median()`, and `mode()` methods on a Series.
- The NumPy `mean()` and `median()` functions, and the SciPy stats `mode()` function on a Series.

REWIND

The **measures of central tendency** refer to the tendency of data to be toward the middle of the dataset. The three key measures of central tendency are the mean, median, and mode.

Pandas describe() Function

The `describe()` function is a convenient tool to get a high-level summary statistics on a DataFrame or Series. After running the function, the output will show the count, mean, standard deviation, minimum value, 25%, 50%, and 75% percentiles, and maximum value from a DataFrame column that has numeric values.

REWIND

Remember—quartiles are percentiles. The lower quartile is the 25th percentile. The upper quartile is the 75th percentile.

Let's use the `describe()` function on the urban, suburban, and rural DataFrames. Add the following code to a new cell:

```
# Get summary statistics.
urban_cities_df.describe()
```

The output from running this cell is:

```
# Get summary statistics.
urban_cities_df.describe()
```

	fare	ride_id	driver_count
count	1625.000000	1.625000e+03	1625.000000
mean	24.525772	4.873485e+12	36.678154
std	11.738649	2.907440e+12	20.075545
min	4.050000	1.458810e+10	3.000000
25%	14.550000	2.400244e+12	22.000000
50%	24.640000	4.711188e+12	37.000000
75%	34.580000	7.451579e+12	52.000000
max	44.970000	9.991538e+12	73.000000

SKILL DRILL

Use the `describe()` function on the `suburban_cities_df` and `rural_cities_df` DataFrames and compare the outputs of all three DataFrames.

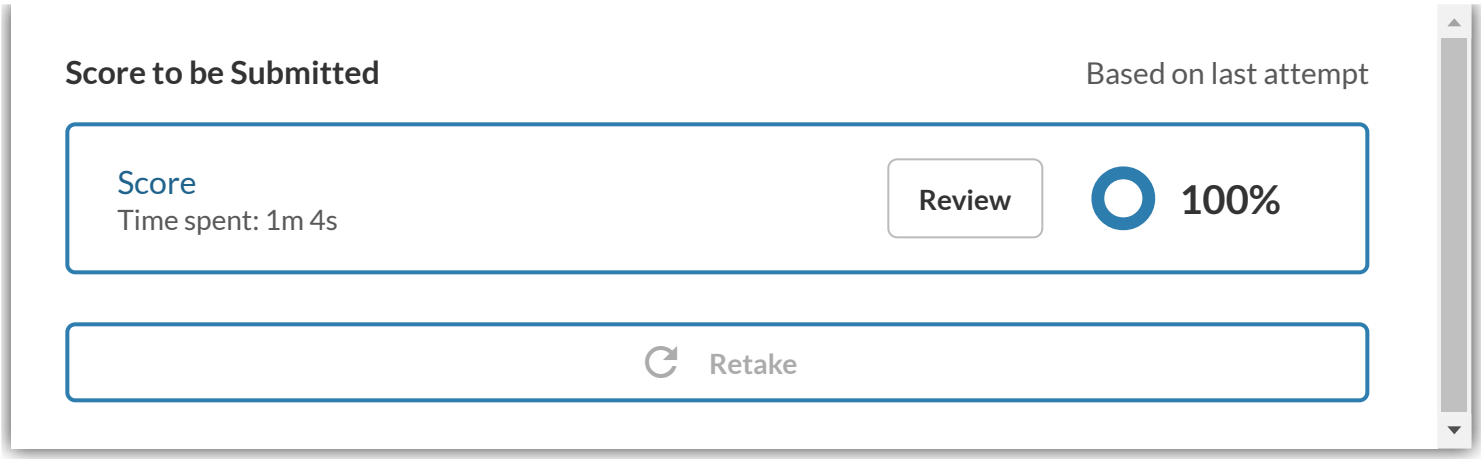
Now let's calculate the summary statistics of the ride count for each city type. Add the following code to a new cell and run the cell.

```
# Get summary statistics.
urban_ride_count.describe()
```

The output from running this cell will show the total number, the average, the standard deviation, the maximum, minimum, and the 25%, 50%, and 75% quartiles.

```
# Get summary statistics.
urban_ride_count.describe()
```

```
count    66.000000
mean     24.621212
std       5.408726
min      12.000000
25%      21.000000
50%      24.000000
75%      28.000000
max      39.000000
Name: ride_id, dtype: float64
```



Pandas mean(), median(), and mode() Methods

If we want to get only the mean without getting the complete summary statistics, we can use the `mean()` method.

Add the following code to a new cell and run the cell.

```
# Calculate the mean of the ride count for each city type.
round(urban_ride_count.mean(),2), round(suburban_ride_count.mean(),2), rc
```

The output will be the average ride count for each city type rounded to two decimal places:

```
# Calculate the mean of the ride count for each city type.
round(urban_ride_count.mean(), 2), round(suburban_ride_count.mean(),2), round(rural_ride_count.mean(),2)
```

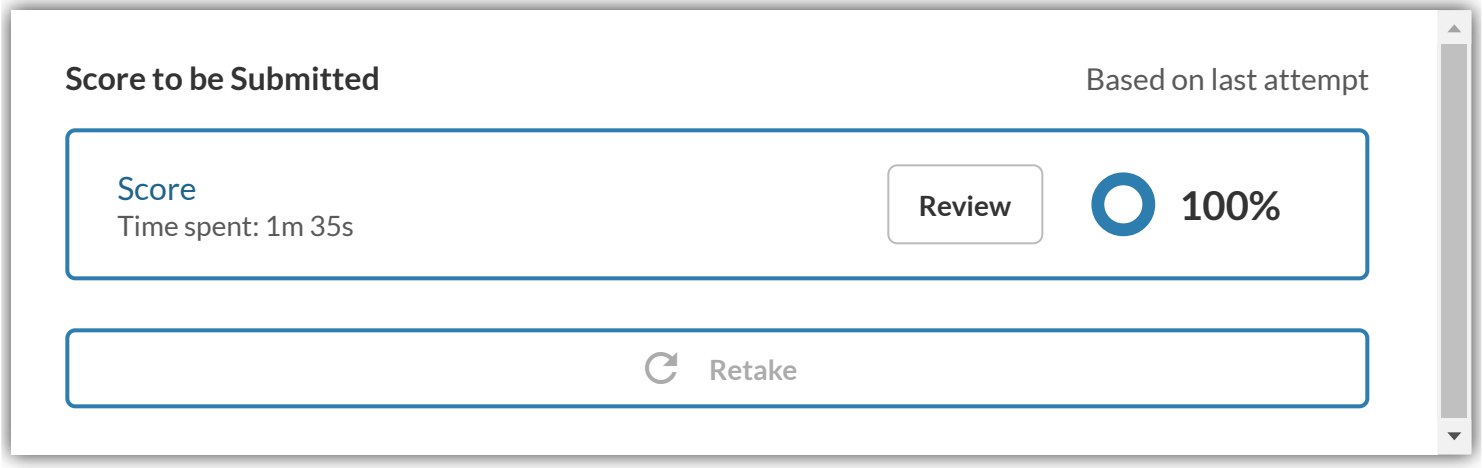
```
(24.62, 17.36, 6.94)
```

Notice that the mean of the ride count for each city type using the `mean()` method is the same value that was returned using the `describe()` function.

FINDING

If we compare the average number of rides between each city type, we'll notice that the average number of rides in the rural cities is about 3.5 and 2.5 times lower than urban and suburban cities, respectively.

To get the median of DataFrame or Series, we can also use the Pandas `median()` method in the same way as we used the `mean()` method.



Similarly, we can use the `mode()` method to get the mode of the ride counts for each city. Add the following code to a new cell:

```
# Calculate the mode of the ride count for the urban cities.
urban_ride_count.mode()
```

The output will be the mode or modes of the Series. In this case, we have two modes—one at 22 and one at 25:

```
# Calculate the mode of the ride count for the urban cities.
urban_ride_count.mode()
```

```
0    22
1    25
dtype: int64
```

REWIND

A dataset like this Series can have any number of modes—or even no mode!

Score to be Submitted Based on last attempt

Score
Time spent: 1m 4s

Review 100%

Retake

NumPy mean() and median() Functions and SciPy mode() Function

An optional approach to calculating the mean, median, and mode of a DataFrame or Series is to use the NumPy and SciPy statistics modules. We introduce these methods because there might come a time when you're working in the Python interpreter or VS Code environment instead of the Jupyter Notebook environment.

Whether you are using the use the Python interpreter, VS Code, or Jupyter Notebook environment, we will need to import the NumPy and SciPy statistics modules. Add the following import statements to a new cell in your `PyBer.ipynb` file and run the cell.

```
# Import NumPy and the stats module from SciPy.
import numpy as np
import scipy.stats as sts
```

Let's calculate the mean, median, and mode—otherwise known as the **measures of central tendency** for the ride counts—and print out those measures.

To get the measures of central tendency of the ride counts for the urban cities, add the following code block.

```
# Calculate the measures of central tendency for the ride count for the u
mean_urban_ride_count = np.mean(urban_ride_count)
print(f"The mean for the ride counts for urban trips is {mean_urban_ride_

median_urban_ride_count = np.median(urban_ride_count)
print(f"The median for the ride counts for urban trips is {median_urban_r

mode_urban_ride_count = sts.mode(urban_ride_count)
print(f"The mode for the ride counts for urban trips is {mode_urban_ride_
```

When we run the cell, we get the following output:

```
The mean for the ride counts for urban trips is 24.62.
The median for the ride counts for urban trips is 24.0.
The mode for the ride counts for urban trips is ModeResult(mode=array([22]), count=array([7])).
```

Let's go over what the output gives us:

- The mean and median values that were returned are the same values that were returned using the `describe()` function and the `mean()` and `median()` methods, respectively.
- With SciPy statistics, the mode result that's returned is the mode that appears the most frequently.
- `ModeResult` returned two attributes:
  - The first attribute, `mode`, is 22.
  - The second attribute, `count`, is the number of times it occurs in the dataset, in this case, 7.

Unlike the Pandas `mode()` method, the `sts.mode()` method will return the number of times the mode appears in the dataset.

There are a few choices of methods to use to get the mean, median, and mode of a dataset. The method you choose is a matter of preference and depends on whether you're working with the Pandas, NumPy, or statistics modules.