Bike_Share_Analysis

January 24, 2018

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

> **Tip**: Quoted sections like this will provide helpful instructions on how to navigate and use a Jupyter notebook.

Over the past decade, bicycle-sharing systems have been growing in number and popularity in cities across the world. Bicycle-sharing systems allow users to rent bicycles for short trips, typically 30 minutes or less. Thanks to the rise in information technologies, it is easy for a user of the system to access a dock within the system to unlock or return bicycles. These technologies also provide a wealth of data that can be used to explore how these bike-sharing systems are used.

In this project, you will perform an exploratory analysis on data provided by Motivate, a bike-share system provider for many major cities in the United States. You will compare the system usage between three large cities: New York City, Chicago, and Washington, DC. You will also see if there are any differences within each system for those users that are registered, regular users and those users that are short-term, casual users.

Posing Questions

Before looking at the bike sharing data, you should start by asking questions you might want to understand about the bike share data. Consider, for example, if you were working for Motivate. What kinds of information would you want to know about in order to make smarter business decisions? If you were a user of the bike-share service, what factors might influence how you would want to use the service?

Question 1: Write at least two questions related to bike sharing that you think could be answered by data. XSCSF

Answer: From Business perspective, the following questions can be answered by the given data.

1) *no of bike users per route?* so as to provide more bikes at the stations which sees maximum users and similarly reduce at rarely used stations.

2) *what type of users use this service the most*?,*what is the ratio of casual to regulars or subcribers*?, so as to decide the pricing structure and attract more users to subcribe.

From the user perspective.
1)*what is the peak usage during any given day of the week*?, so i can plan my trip accordingly.
2)*what is the total no of users*? to know how good the service is

> **Tip**: If you double click on this cell, you will see the text change so that all of the formatting is removed. This allows you to edit this block of text. This block of text is written using Markdown, which is a way to format text using headers, links, italics, and many other options using a plain-text syntax. You will also use Markdown later in the Nanodegree program. Use **Shift** + **Enter** or **Shift** + **Return** to run the cell and show its rendered form.

Data Collection and Wrangling
Now it's time to collect and explore our data. In this project, we will focus on the record of individual trips taken in 2016 from our selected cities: New York City, Chicago, and Washington, DC. Each of these cities has a page where we can freely download the trip data.:

- New York City (Citi Bike): Link
- Chicago (Divvy): Link
- Washington, DC (Capital Bikeshare): Link

If you visit these pages, you will notice that each city has a different way of delivering its data. Chicago updates with new data twice a year, Washington DC is quarterly, and New York City is monthly. **However, you do not need to download the data yourself.** The data has already been collected for you in the `/data/` folder of the project files. While the original data for 2016 is spread among multiple files for each city, the files in the `/data/` folder collect all of the trip data for the year into one file per city. Some data wrangling of inconsistencies in timestamp format within each city has already been performed for you. In addition, a random 2% sample of the original data is taken to make the exploration more manageable.

Question 2: However, there is still a lot of data for us to investigate, so it's a good idea to start off by looking at one entry from each of the cities we're going to analyze. Run the first code cell below to load some packages and functions that you'll be using in your analysis. Then, complete the second code cell to print out the first trip recorded from each of the cities (the second line of each data file).

> **Tip**: You can run a code cell like you formatted Markdown cells above by clicking on the cell and using the keyboard shortcut **Shift** + **Enter** or **Shift** + **Return**. Alternatively, a code cell can be executed using the **Play** button in the toolbar after selecting it. While the cell is running, you will see an asterisk in the message to the left of the cell, i.e. `In`

[*]:. The asterisk will change into a number to show that execution has completed, e.g. In [1]. If there is output, it will show up as Out [1]:, with an appropriate number to match the "In" number.

In [3]:
```
## import all necessary packages and functions.
import csv # read and write csv files
from datetime import datetime # operations to parse dates
from pprint import pprint # use to print data structures like dictionaries in
                          # a nicer way than the base print function.
```

In [4]:
```
def print_first_point(filename):

    city = filename.split('-')[0].split('/')[-1]
    print('\nCity: {}'.format(city))

    with open(filename, 'r') as f_in:

        trip_reader = csv.DictReader(f_in)

        first_trip = next(trip_reader)

        pprint(first_trip)

    return (city, first_trip)


data_files = ['./data/NYC-CitiBike-2016.csv',
              './data/Chicago-Divvy-2016.csv',
              './data/Washington-CapitalBikeshare-2016.csv',]


example_trips = {}
for data_file in data_files:
    city, first_trip = print_first_point(data_file)
    example_trips[city] = first_trip
```

```
City: NYC
OrderedDict([('tripduration', '839'),
             ('starttime', '1/1/2016 00:09:55'),
             ('stoptime', '1/1/2016 00:23:54'),
             ('start station id', '532'),
             ('start station name', 'S 5 Pl & S 4 St'),
             ('start station latitude', '40.710451'),
             ('start station longitude', '-73.960876'),
             ('end station id', '401'),
             ('end station name', 'Allen St & Rivington St'),
             ('end station latitude', '40.72019576'),
```

```
            ('end station longitude', '-73.98997825'),
            ('bikeid', '17109'),
            ('usertype', 'Customer'),
            ('birth year', ''),
            ('gender', '0')])

City: Chicago
OrderedDict([('trip_id', '9080545'),
            ('starttime', '3/31/2016 23:30'),
            ('stoptime', '3/31/2016 23:46'),
            ('bikeid', '2295'),
            ('tripduration', '926'),
            ('from_station_id', '156'),
            ('from_station_name', 'Clark St & Wellington Ave'),
            ('to_station_id', '166'),
            ('to_station_name', 'Ashland Ave & Wrightwood Ave'),
            ('usertype', 'Subscriber'),
            ('gender', 'Male'),
            ('birthyear', '1990')])

City: Washington
OrderedDict([('Duration (ms)', '427387'),
            ('Start date', '3/31/2016 22:57'),
            ('End date', '3/31/2016 23:04'),
            ('Start station number', '31602'),
            ('Start station', 'Park Rd & Holmead Pl NW'),
            ('End station number', '31207'),
            ('End station', 'Georgia Ave and Fairmont St NW'),
            ('Bike number', 'W20842'),
            ('Member Type', 'Registered')])
```

If everything has been filled out correctly, you should see below the printout of each city name (which has been parsed from the data file name) that the first trip has been parsed in the form of a dictionary. When you set up a `DictReader` object, the first row of the data file is normally interpreted as column names. Every other row in the data file will use those column names as keys, as a dictionary is generated for each row.

This will be useful since we can refer to quantities by an easily-understandable label instead of just a numeric index. For example, if we have a trip stored in the variable `row`, then we would rather get the trip duration from `row['duration']` instead of `row[0]`.

Condensing the Trip Data

It should also be observable from the above printout that each city provides different information. Even where the information is the same, the column names and formats are sometimes different. To make things as simple as possible when we get to the actual exploration, we should trim and clean the data. Cleaning the data makes sure that the data formats across the cities are consistent, while trimming focuses only on the parts of the data we are most interested in to make the exploration easier to work with.

You will generate new data files with five values of interest for each trip: trip duration, starting month, starting hour, day of the week, and user type. Each of these may require additional wrangling depending on the city:

- **Duration**: This has been given to us in seconds (New York, Chicago) or milliseconds (Washington). A more natural unit of analysis will be if all the trip durations are given in terms of minutes.
- **Month**, **Hour**, **Day of Week**: Ridership volume is likely to change based on the season, time of day, and whether it is a weekday or weekend. Use the start time of the trip to obtain these values. The New York City data includes the seconds in their timestamps, while Washington and Chicago do not. The `datetime` package will be very useful here to make the needed conversions.
- **User Type**: It is possible that users who are subscribed to a bike-share system will have different patterns of use compared to users who only have temporary passes. Washington divides its users into two types: 'Registered' for users with annual, monthly, and other longer-term subscriptions, and 'Casual', for users with 24-hour, 3-day, and other short-term passes. The New York and Chicago data uses 'Subscriber' and 'Customer' for these groups, respectively. For consistency, you will convert the Washington labels to match the other two.

Question 3a: Complete the helper functions in the code cells below to address each of the cleaning tasks described above.

```
In [5]: def duration_in_mins(datum, city):

            if city=="Washington":
                fd = float(datum["Duration (ms)"])
                duration = fd/60000
            else:
                fd = float(datum['tripduration'])
                duration = fd/60

            return duration


        tests = {'NYC': 13.9833,
        'Chicago': 15.4333,
        'Washington': 7.1231}
        for city in tests:
            assert abs(duration_in_mins(example_trips[city], city) - tests[city]) < .001
```

```
In [8]: def time_of_trip(datum, city):

            if city == "Washington":
                start_time = datum["Start date"]
            else:
                start_time = datum["starttime"]

            if city == "NYC":
                D = datetime.strptime(start_time, '%m/%d/%Y %H:%M:%S')
            else:
```

```
        D = datetime.strptime(start_time, '%m/%d/%Y %H:%M')

    month = int(datetime.strftime(D, '%m'))
    hour  = int(datetime.strftime(D, '%H'))
    day_of_week = str(datetime.strftime(D, '%A'))

    return (month, hour, day_of_week)

tests = {'NYC': (1, 0, 'Friday'),
'Chicago': (3, 23, 'Thursday'),
'Washington': (3, 22, 'Thursday')}
for city in tests:
    assert time_of_trip(example_trips[city], city) == tests[city]
```

```
In [9]: def type_of_user(datum, city):

            if city == 'Washington':
                user_type = datum['Member Type']
            else:
                user_type = datum['usertype']

            return user_type

        tests = {'NYC': 'Customer',
        'Chicago': 'Subscriber',
        'Washington': 'Registered'}
        for city in tests:
              assert type_of_user(example_trips[city], city) == tests[city]
```

Question 3b: Now, use the helper functions you wrote above to create a condensed data file for each city consisting only of the data fields indicated above. In the /examples/ folder, you will see an example datafile from the Bay Area Bike Share before and after conversion. Make sure that your output is formatted to be consistent with the example file.

```
In [10]: def condense_data(in_file, out_file, city):

             with open(out_file, 'w') as f_out, open(in_file, 'r') as f_in:

                 out_colnames = ['duration', 'month', 'hour', 'day_of_week', 'user_type']
                 trip_writer = csv.DictWriter(f_out, fieldnames = out_colnames)

                 trip_writer.writeheader()

                 trip_reader = csv.DictReader(f_in)
                 for row in trip_reader:
                     new_point = {}
                     new_point['duration'] = duration_in_mins(row, city)
                     new_point['month'] = time_of_trip(row, city)[0]
```

```
                new_point['hour'] = time_of_trip(row, city)[1]
                new_point['day_of_week'] = time_of_trip(row, city)[2]
                new_point['user_type'] = type_of_user(row, city)
                trip_writer.writerow(new_point)
        return None
```

```
In [11]: # Run this cell to check your work
         city_info = {'Washington': {'in_file': './data/Washington-CapitalBikeshare-2016.csv',
                                     'out_file': './data/Washington-2016-Summary.csv'},
                      'Chicago': {'in_file': './data/Chicago-Divvy-2016.csv',
                                  'out_file': './data/Chicago-2016-Summary.csv'},
                      'NYC': {'in_file': './data/NYC-CitiBike-2016.csv',
                              'out_file': './data/NYC-2016-Summary.csv'}}

         for city, filenames in city_info.items():
             condense_data(filenames['in_file'], filenames['out_file'], city)
             print_first_point(filenames['out_file'])
```

```
City: Washington
OrderedDict([('duration', '7.123116666666666'),
             ('month', '3'),
             ('hour', '22'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Registered')])

City: Chicago
OrderedDict([('duration', '15.433333333333334'),
             ('month', '3'),
             ('hour', '23'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: NYC
OrderedDict([('duration', '13.983333333333333'),
             ('month', '1'),
             ('hour', '0'),
             ('day_of_week', 'Friday'),
             ('user_type', 'Customer')])
```

> **Tip**: If you save a jupyter Notebook, the output from running code blocks will also be saved. However, the state of your workspace will be reset once a new session is started. Make sure that you run all of the necessary code blocks from your previous session to reestablish variables and functions before picking up where you last left off.

Exploratory Data Analysis

Now that you have the data collected and wrangled, you're ready to start exploring the data. In this section you will write some code to compute descriptive statistics from the data. You will also be introduced to the `matplotlib` library to create some basic histograms of the data.

Statistics

First, let's compute some basic counts. The first cell below contains a function that uses the csv module to iterate through a provided data file, returning the number of trips made by subscribers and customers. The second cell runs this function on the example Bay Area data in the `/examples/` folder. Modify the cells to answer the question below.

Question 4a: Which city has the highest number of trips? Which city has the highest proportion of trips made by subscribers? Which city has the highest proportion of trips made by short-term customers?

Answer: 1)*New York has the highest number of trips and the highest proportion of trips made by subscribers.* 2)*Chicago has the highest number of trips made by short-term customers.*

```
In [12]: ## ALl the Condensed Files ##
         import csv
         city_files = {'Washington':'./data/Washington-2016-Summary.csv',
                       'Chicago':'./data/Chicago-2016-Summary.csv',
                       'NYC': './data/NYC-2016-Summary.csv',
                       'Bay Area':'./examples/BayArea-Y3-Summary.csv'}
```

```
In [13]: def number_of_trips(filename):
             with open(filename, 'r') as f_in:
                 reader = csv.DictReader(f_in)
                 n_subscribers = 0
                 n_customers = 0
                 S_list = []
                 C_list = []
                 for row in reader:
                     if row['user_type'] == 'Subscriber' or row['user_type'] == 'Registered':
                         S_list.append(float(row['duration']))
                     else:
                         C_list.append(float(row['duration']))

             n_total_list = S_list + C_list

             n_subscribers = len(S_list)
             n_customers = len(C_list)
             n_total = len(n_total_list)
             C_p = (n_customers)/(n_total)*100
             S_p = (n_subscribers)/(n_total)*100

             return(n_customers, n_subscribers, C_p, S_p, n_total, S_list, C_list, n_total_list)
```

```
In [14]: for city, data_file in city_files.items():
             print("City: {} \n\tCasual: {:.2f}% \n\tSubscribers: {:.2f}% \n\tTotal Trips: {}".f
```

```
City: Washington
        Casual: 21.97%
        Subscribers: 78.03%
        Total Trips: 66326
City: Chicago
```

```
        Casual: 23.77%
        Subscribers: 76.23%
        Total Trips: 72131
City: NYC
        Casual: 11.16%
        Subscribers: 88.84%
        Total Trips: 276798
City: Bay Area
        Casual: 10.05%
        Subscribers: 89.95%
        Total Trips: 6299
```

> **Tip**: In order to add additional cells to a notebook, you can use the "Insert Cell Above" and "Insert Cell Below" options from the menu bar above. There is also an icon in the toolbar for adding new cells, with additional icons for moving the cells up and down the document. By default, new cells are of the code type; you can also specify the cell type (e.g. Code or Markdown) of selected cells from the Cell menu or the dropdown in the toolbar.

Now, you will write your own code to continue investigating properties of the data.

Question 4b: Bike-share systems are designed for riders to take short trips. Most of the time, users are allowed to take trips of 30 minutes or less with no additional charges, with overage charges made for trips of longer than that duration. What is the average trip length for each city? What proportion of rides made in each city are longer than 30 minutes?

Answer:1) Average length trip lengths for Chicago, NYC, Washington are 16.6, 16.3 and 16.9 minutes respectively. 2) The proportion of rides over 30 minutes in Chicago, NYC, Washington are 8.3%, 7.7% and 10.8% respectively.

```
In [15]: def trip_duration(filename):
             trip_list = number_of_trips(filename)[7]
             list_thirty = []
             i = 0
             for i in range(len(trip_list)):
                 if trip_list[i] > 30:
                     list_thirty.append(trip_list[i])
             avg = sum(trip_list)/len(trip_list)
             proportion = len(list_thirty)/len(trip_list)*100
             return(avg, proportion)
```

```
In [16]: for city, data_file in city_files.items():
             print("city: {} \n\t Average trip length is {:.1f} minutes \n\t and {:.1f}% of trip
```

```
city: Washington
         Average trip length is 18.9 minutes
         and 10.8% of trips are longer than 30 minutes
city: Chicago
         Average trip length is 16.6 minutes
         and 8.3% of trips are longer than 30 minutes
```

```
city: NYC
        Average trip length is 15.8 minutes
        and 7.3% of trips are longer than 30 minutes
city: Bay Area
        Average trip length is 14.0 minutes
        and 3.5% of trips are longer than 30 minutes
```

Question 4c: Dig deeper into the question of trip duration based on ridership. Choose one city. Within that city, which type of user takes longer rides on average: Subscribers or Customers?

Answer: In *Chicago* on an average the customers take longer rides.

```
In [17]: def rider_duration(filename):
             s = number_of_trips(filename)[5]
             c = number_of_trips(filename)[6]
             avg_s = sum(s)/len(s)
             avg_c = sum(c)/len(c)
             return(avg_s, avg_c)
```

```
In [18]: for city, data_file in city_files.items():
             print("\nIn {}, average subscriber trip duration is {:.1f} minutes and the average
```

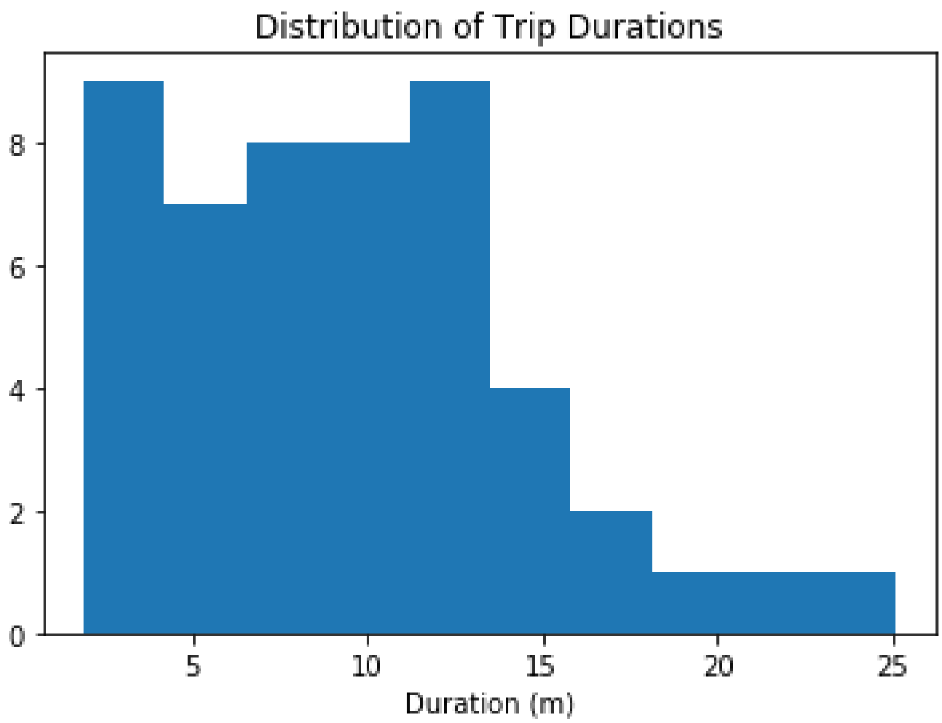
```
In Washington, average subscriber trip duration is 12.5 minutes and the average Customer trip du

In Chicago, average subscriber trip duration is 12.1 minutes and the average Customer trip durat

In NYC, average subscriber trip duration is 13.7 minutes and the average Customer trip duration

In Bay Area, average subscriber trip duration is 9.5 minutes and the average Customer trip durat
```

Visualizations

The last set of values that you computed should have pulled up an interesting result. While the mean trip time for Subscribers is well under 30 minutes, the mean trip time for Customers is actually *above* 30 minutes! It will be interesting for us to look at how the trip times are distributed. In order to do this, a new library will be introduced here, `matplotlib`. Run the cell below to load the library and to generate an example plot.

```
In [19]: # load library
         import matplotlib.pyplot as plt

         # this is a 'magic word' that allows for plots to be displayed
         # inline with the notebook. If you want to know more, see:
         # http://ipython.readthedocs.io/en/stable/interactive/magics.html
         %matplotlib inline

         # example histogram, data taken from bay area sample
         data = [ 7.65,  8.92,  7.42,  5.50, 16.17,  4.20,  8.98,  9.62, 11.48, 14.33,
```

```
             19.02, 21.53,  3.90,  7.97,  2.62,  2.67,  3.08, 14.40, 12.90,  7.83,
             25.12,  8.30,  4.93, 12.43, 10.60,  6.17, 10.88,  4.78, 15.15,  3.53,
              9.43, 13.32, 11.72,  9.85,  5.22, 15.10,  3.95,  3.17,  8.78,  1.88,
              4.55, 12.68, 12.38,  9.78,  7.63,  6.45, 17.38, 11.90, 11.52,  8.63,]
        plt.hist(data)
        plt.title('Distribution of Trip Durations')
        plt.xlabel('Duration (m)')
        plt.show()
```

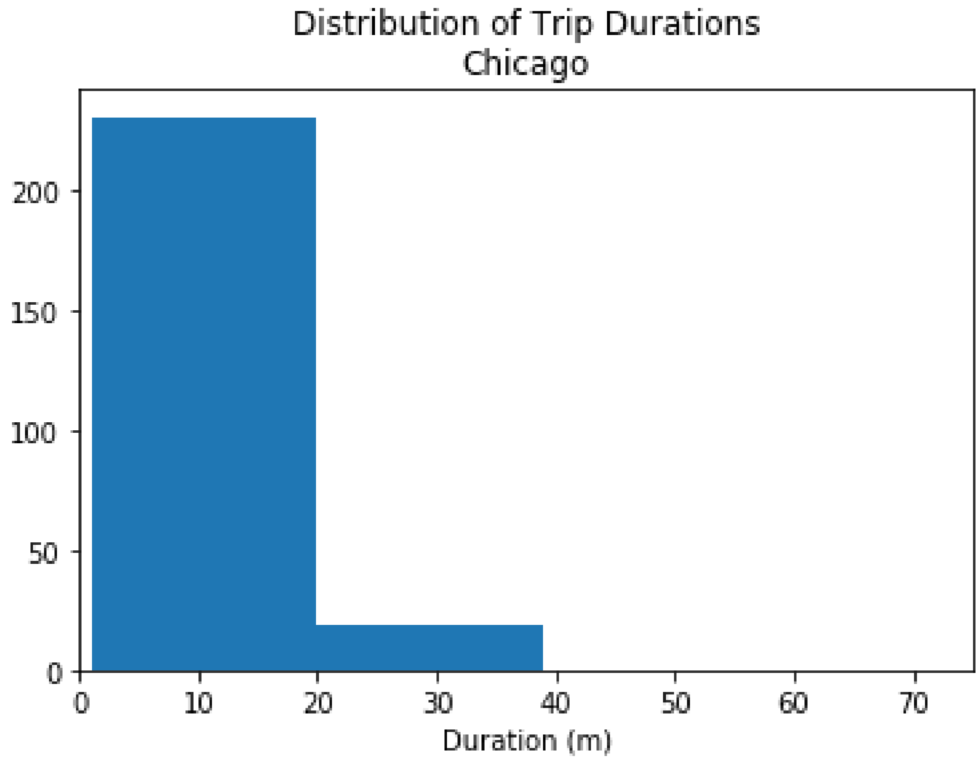


In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. The `.title()` and `.xlabel()` functions provide some labeling for plot context.

You will now use these functions to create a histogram of the trip times for the city you selected in question 4c. Don't separate the Subscribers and Customers for now: just collect all of the trip times and plot them.

```
In [20]: myList = number_of_trips('./data/Chicago-2016-Summary.csv')[7]
         Data1 = [ round(elem, 2) for elem in myList ]
         data = Data1[0:250]
         #print(data)
         plt.hist(data)
         plt.title('Distribution of Trip Durations\nChicago')
         plt.xlabel('Duration (m)')
```

```
plt.xlim(0,75)
plt.show()
```




```
In [21]: ##To Increase Size of Graphs
         # Get current size
         fig_size = plt.rcParams["figure.figsize"]

         # Prints: [8.0, 6.0]
         print("Current size:", fig_size)

         # Set figure width to 12 and height to 9
         fig_size[0] = 10
         fig_size[1] = 8
         plt.rcParams["figure.figsize"] = fig_size
```

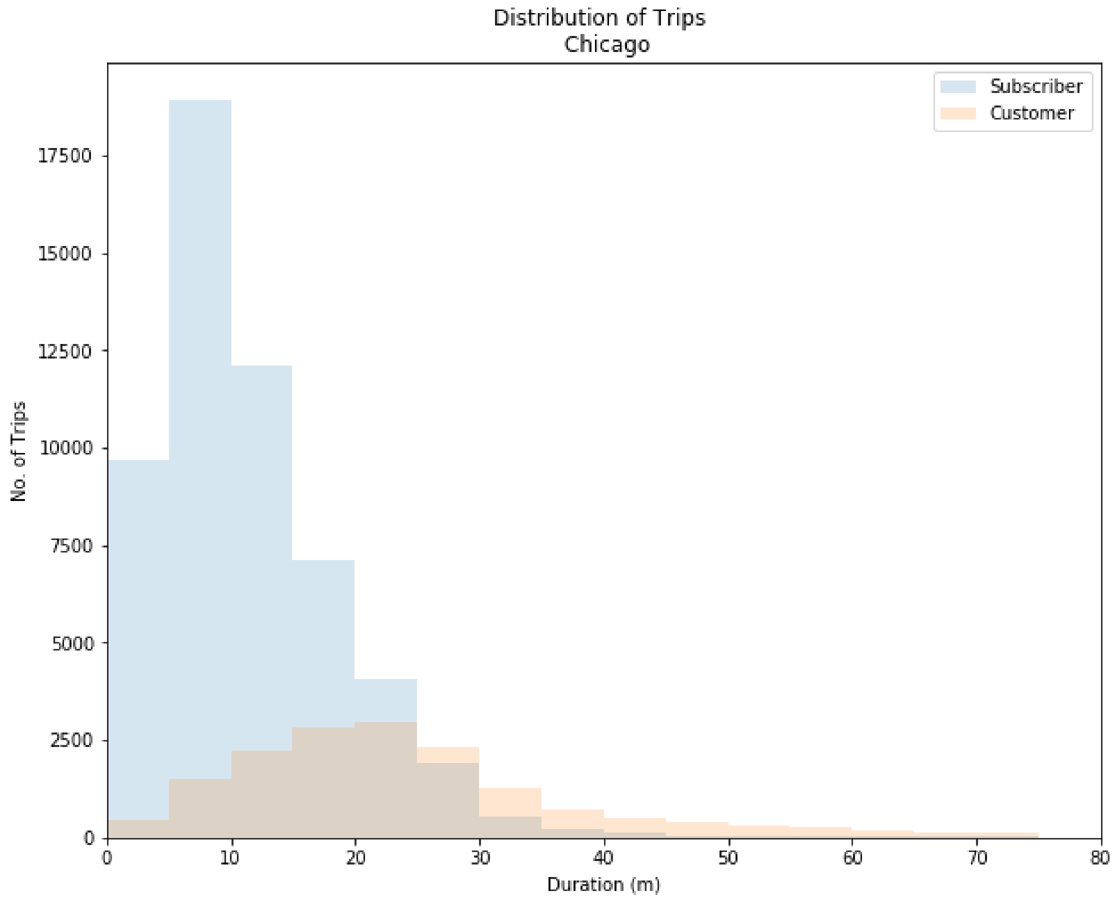
```
Current size: [6.0, 4.0]
```

If you followed the use of the `.hist()` and `.show()` functions exactly like in the example, you're probably looking at a plot that's completely unexpected. The plot consists of one extremely tall bar on the left, maybe a very short second bar, and a whole lot of empty space in the center and right. Take a look at the duration values on the x-axis. This suggests that there are some highly infrequent outliers in the data. Instead of reprocessing the data, you will use additional parameters with the `.hist()` function to limit the range of data that is plotted. Documentation for the function can be found [here].

Question 5: Use the parameters of the `.hist()` function to plot the distribution of trip times for the Subscribers in your selected city. Do the same thing for only the Customers. Add limits to the plots so that only trips of duration less than 75 minutes are plotted. As a bonus, set the plots up so that bars are in five-minute wide intervals. For each group, where is the peak of each distribution? How would you describe the shape of each distribution?

Answer: Peak of each distribution is towards right, hence i'd describe them as a *'Skewed right'* or positively skewed distribution. where as the subscriber trip distribution has a clear peak, therefore it is also an *'Unimodal'* distribution.

```
In [22]: import numpy
         import matplotlib.pyplot as plt
         filename = './data/Chicago-2016-Summary.csv'
         s = number_of_trips(filename)[5]
         c = number_of_trips(filename)[6]
         x1 = [ round(elem, 2) for elem in s ]
         y1 = [ round(elem, 2) for elem in c ]

         bins = range(0,80,5)
         plt.hist(x1, bins, alpha = 0.18, label='Subscriber')
         plt.hist(y1, bins, alpha = 0.19, label='Customer')
         plt.xlabel('Duration (m)')
         plt.ylabel('No. of Trips')
         plt.title('Distribution of Trips \n Chicago')
         plt.xlim(0,80)
         plt.legend(loc='upper right')

         plt.show()
```



Performing Your Own Analysis

So far, you've performed an initial exploration into the data available. You have compared the relative volume of trips made between three U.S. cities and the ratio of trips made by Subscribers and Customers. For one of these cities, you have investigated differences between Subscribers and Customers in terms of how long a typical trip lasts. Now it is your turn to continue the exploration in a direction that you choose. Here are a few suggestions for questions to explore:

- How does ridership differ by month or season? Which month / season has the highest ridership? Does the ratio of Subscriber trips to Customer trips change depending on the month or season?
- Is the pattern of ridership different on the weekends versus weekdays? On what days are Subscribers most likely to use the system? What about Customers? Does the average duration of rides change depending on the day of the week?
- During what time of day is the system used the most? Is there a difference in usage patterns for Subscribers and Customers?

If any of the questions you posed in your answer to question 1 align with the bullet points above, this is a good opportunity to investigate one of them. As part of your investigation, you will need to create a visualization. If you want to create something other than a histogram, then you might want to consult the Pyplot documentation. In particular, if you are plotting values across a categorical variable (e.g. city, user type), a bar chart will be useful. The documentation page for `.bar()` includes links at the bottom of the page with examples for you to build off of for your own use.

Question 6: Continue the investigation by exploring another question that could be answered by the data available. Document the question you want to explore below. Your investigation should involve at least two variables and should compare at least two groups. You should also use at least one visualization as part of your explorations.

Answer: MyQuestion: *What is the peak usage on any given day during the week?* Answer: According to my analysis for Chicago city, monday seems to be the most busy day of the week. As many users(Mostly subscribers) use the service on monday. Also many users take long Bike trips on weekend as the average trip duration is maximum for sunday.

```
In [23]: def number_of_trips_per_day(filename, day, usertype):
             with open(filename, 'r') as f_in:
                 reader = csv.DictReader(f_in)
                 d_list = []

                 for row in reader:
                     if row['user_type'] == usertype and row['day_of_week'] == day:
                         d_list.append(float(row['duration']))

             return (d_list)                                        ##

In [24]: def avg_weekly(filename, day):
             with open(filename, 'r') as f_in:
                 reader = csv.DictReader(f_in)
                 avg_list = []
                 for row in reader:
                     if row['day_of_week'] == day:
                         avg_list.append(float(row['duration']))
             avg = sum(avg_list)/len(avg_list)

             return(avg)

In [25]: filename = './data/Chicago-2016-Summary.csv'
         Days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
         U_type = ['Subscriber', 'Customer']

         A_list = [len(number_of_trips_per_day(filename, Days[0], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[1], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[2], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[3], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[4], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[5], U_type[0])),
                   len(number_of_trips_per_day(filename, Days[6], U_type[0]))]

         B_list = [len(number_of_trips_per_day(filename, Days[0], U_type[1])),
                   len(number_of_trips_per_day(filename, Days[1], U_type[1])),
```

```
                    len(number_of_trips_per_day(filename, Days[2], U_type[1])),
                    len(number_of_trips_per_day(filename, Days[3], U_type[1])),
                    len(number_of_trips_per_day(filename, Days[4], U_type[1])),
                    len(number_of_trips_per_day(filename, Days[5], U_type[1])),
                    len(number_of_trips_per_day(filename, Days[6], U_type[1]))]


        print("Subcriber_weekly_trips {},\nCustomer_weekly_trips {}".format(A_list, B_list))
```

```
Subcriber_weekly_trips [8840, 9356, 8447, 8643, 8648, 5676, 5372],
Customer_weekly_trips [2446, 1555, 1157, 1365, 2093, 4251, 4282]
```

```
In [26]: avg_list = []
         avg_list.extend([avg_weekly(filename, Days[0]),
                          avg_weekly(filename, Days[1]),
                          avg_weekly(filename, Days[2]),
                          avg_weekly(filename, Days[3]),
                          avg_weekly(filename, Days[4]),
                          avg_weekly(filename, Days[5]),
                          avg_weekly(filename, Days[6])])

         per_day_average = [round(elem, 2) for elem in avg_list]
         print(per_day_average)
```

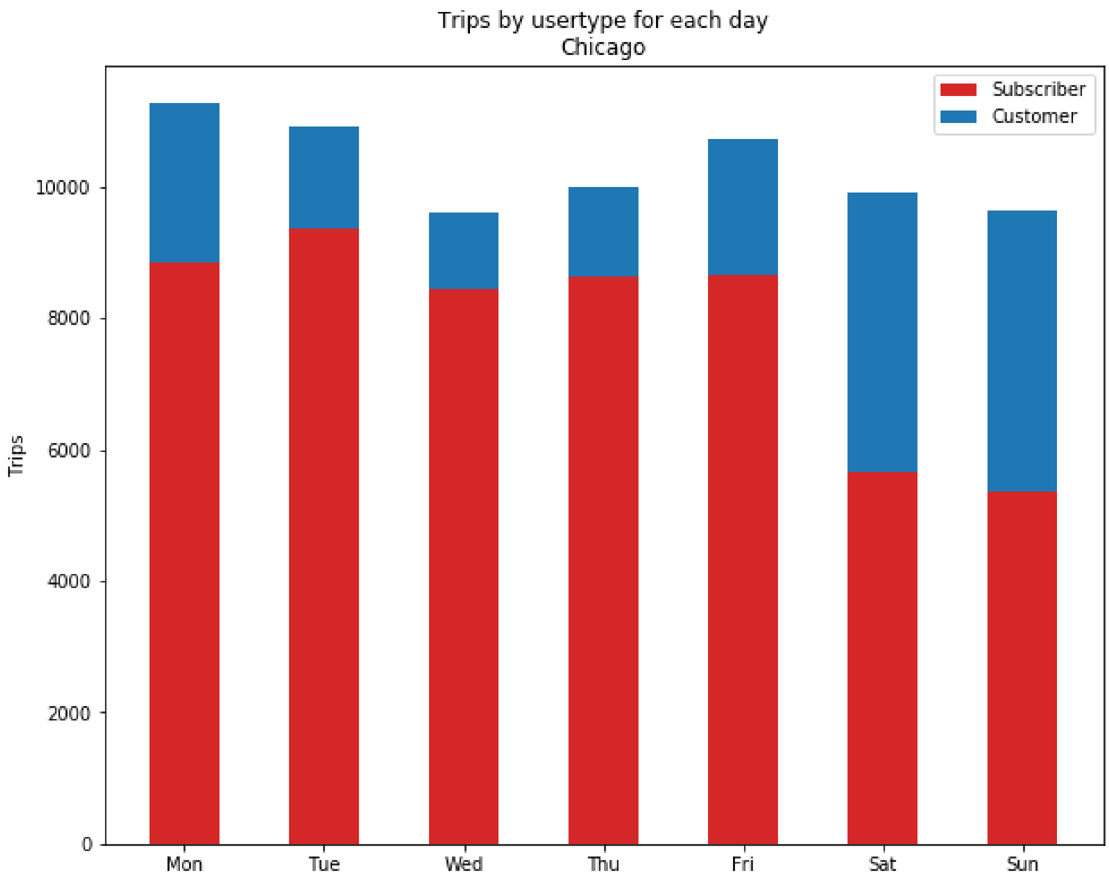
```
[16.12, 14.3, 14.46, 13.93, 15.58, 20.63, 21.38]
```

```
In [27]: import numpy as np
         import matplotlib.pyplot as plt

         A = [8840, 9356, 8447, 8643, 8648, 5676, 5372]
         B = [2446, 1555, 1157, 1365, 2093, 4251, 4282]
         Days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
         print(max(A+B))
         print(min(A+B))
```

```
9356
1157
```

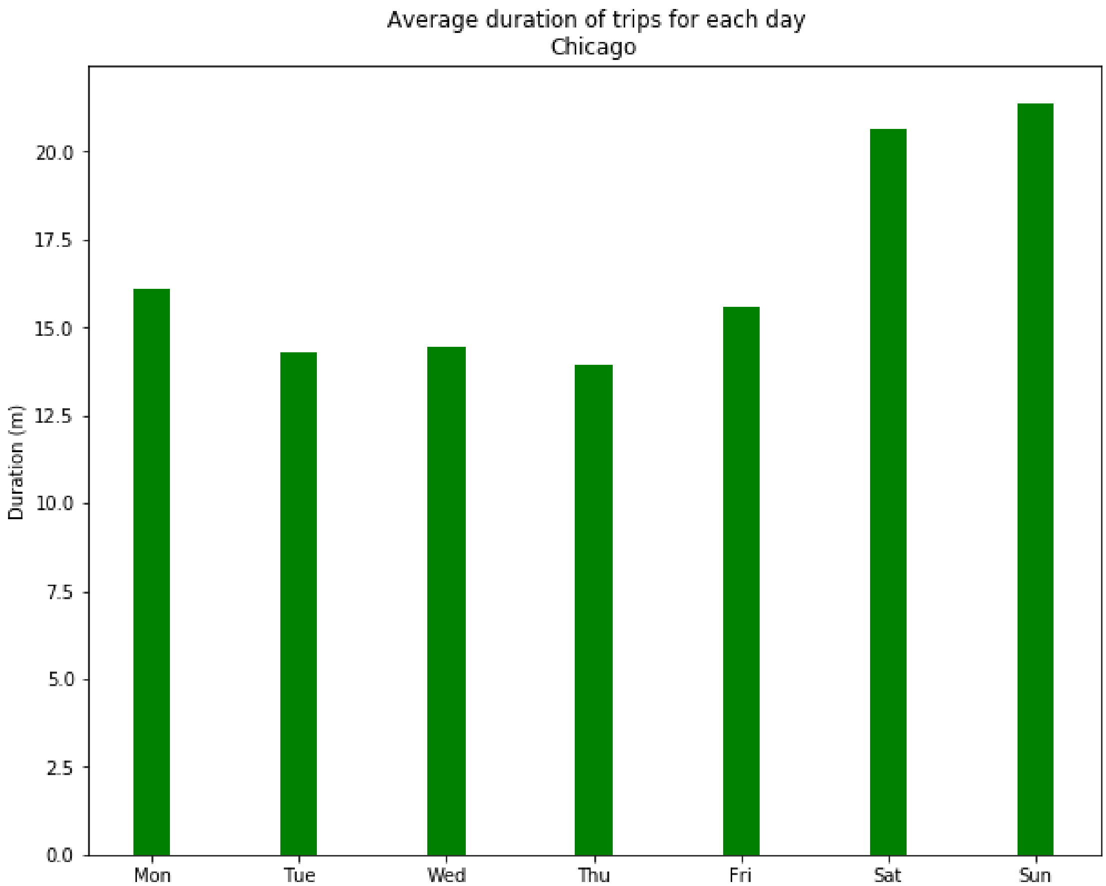
```
In [28]: import numpy as np
         import matplotlib.pyplot as plt


         A = [8840, 9356, 8447, 8643, 8648, 5676, 5372]
         B = [2446, 1555, 1157, 1365, 2093, 4251, 4282]
         Days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
         N = 7
```

```
ind = np.arange(N)
width = 0.5

p1 = plt.bar(ind, A, width, color ='#d62728')
p2 = plt.bar(ind, B, width, bottom = A)
plt.legend(loc='upper right')
plt.ylabel('Trips')
plt.title('Trips by usertype for each day\nChicago')
plt.xticks(ind, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
#plt.yticks(np.arange(1000, 10500, 500))
plt.legend((p1[0], p2[0]), ('Subscriber', 'Customer'))

plt.show()
```




```
In [29]: import numpy as np
         import matplotlib.pyplot as plt
         N = 7
         ind = np.arange(N)
         width = 0.25
         per_day_average = [16.12, 14.3, 14.46, 13.93, 15.58, 20.63, 21.38]
```

```
p1 = plt.bar(ind, per_day_average, width, color ='g')
plt.ylabel('Duration (m)')
plt.title(' Average duration of trips for each day\nChicago')
plt.xticks(ind, ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'))
plt.show()
```



Conclusions

Congratulations on completing the project! This is only a sampling of the data analysis process: from generating questions, wrangling the data, and to exploring the data. Normally, at this point in the data analysis process, you might want to draw conclusions about the data by performing a statistical test or fitting the data to a model for making predictions. There are also a lot of potential analyses that could be performed on the data which are not possible with only the data provided. For example, detailed location data has not been investigated. Where are the most commonly used docks? What are the most common routes? As another example, weather has potential to have a large impact on daily ridership. How much is ridership impacted when there is rain or snow? Are subscribers or customers affected more by changes in weather?

Question 7: Putting the bike share data aside, think of a topic or field of interest where you would like to be able to apply the techniques of data science. What would you like to be able to learn from your chosen subject?

Answer: In India, certain religious places see huge rush of devotees throughout the year. Some of them even donate large sums of money. I think we can apply the techniques of data science to analyse the crowd and at the same time maintain an accurate record of donations. Tracking the flow of devotees against particular month or season will help the temple authorities to be prepared in advanced to ensure smooth transition and tackle any kind of adverse situation. I could even use data science to analyse the performance of a team/player in a particular sport over a course of time to be able to predict the outcome of events better. I'd like to learn the predictive analysis of data. where we make future decisions based on the past and current data.

> **Tip**: If we want to share the results of our analysis with others, we aren't limited to giving them a copy of the jupyter Notebook (.ipynb) file. We can also export the Notebook output in a form that can be opened even for those without Python installed. From the **File** menu in the upper left, go to the **Download as** submenu. You can then choose a different format that can be viewed more generally, such as HTML (.html) or PDF (.pdf). You may need additional packages or software to perform these exports.
>
> If you are working on this project via the Project Notebook page in the classroom, you can also submit this project directly from the workspace. **Before you do that**, you should save an HTML copy of the completed project to the workspace by running the code cell below. If it worked correctly, the output code should be a 0, and if you click on the jupyter icon in the upper left, you should see your .html document in the workspace directory. Alternatively, you can download the .html copy of your report following the steps in the previous paragraph, then *upload* the report to the directory (by clicking the jupyter icon).
>
> Either way, once you've gotten the .html report in your workspace, you can complete your submission by clicking on the "Submit Project" button to the lower-right hand side of the workspace.

```
In [30]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Bike_Share_Analysis.ipynb'])

Out[30]: 0
```