Bike_Share_Analysis

November 21, 2017

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.

Answer: 1. In which time the peoples are using the bike most? 2. When users returning the bike, are they bringing the bike in good shape or damaging it?

> **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 [64]: ## 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 [65]: def print_first_point(filename):
             """
             This function prints and returns the first data point (second row) from
             a csv file that includes a header row.
             """
             # print city name for reference
             city = filename.split('-')[0].split('/')[-1]
             print('\nCity: {}'.format(city))

             with open(filename, 'r') as f_in:
                 trip_reader = csv.DictReader(f_in)

                 ## This function read the first trip from the data file and store it in first_t
                 first_trip = trip_reader.__next__()

                 ## Used the pprint library's pprint function to print the first trip.

                 pprint(first_trip)

             # output city name and first trip for later testing
             return (city, first_trip)

         # list of files for each city
         data_files = ['./data/NYC-CitiBike-2016.csv',
                       './data/Chicago-Divvy-2016.csv',
                       './data/Washington-CapitalBikeshare-2016.csv',]

         # print the first trip from each file, store in dictionary
         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 (Wash-

ington). 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.

```python
In [66]: def duration_in_mins(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (datum) and
    its origin city (city) and returns the trip duration in units of minutes.

    Remember that Washington is in terms of milliseconds while Chicago and NYC
    are in terms of seconds.

    HINT: The csv module reads in all of the data as strings, including numeric
    values. You will need a function to convert the strings into an appropriate
    numeric type when making your transformations.
    see https://docs.python.org/3/library/functions.html
    """
    if(city == "Washington"):
        duration = int(datum["Duration (ms)"])/1000.0/60
    else:
        duration = int(datum["tripduration"])/60
    # YOUR CODE HERE

    return duration


    # Some tests to check that your code works. There should be no output if all of
    # the assertions pass. The `example_trips` dictionary was obtained from when
    # you printed the first trip from each of the original data files.
    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 [67]: def getMHDW(dt, dtformat):
             # Find month, hour and day_of_week from datetime string
             DAY_OF_WEEK = {0: 'Monday', 1:'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday'

             dt = datetime.strptime(dt, dtformat).timetuple()
             month = dt.tm_mon
             hour = dt.tm_hour
             day_of_week = DAY_OF_WEEK[dt.tm_wday]
             return (month, hour, day_of_week)

         def time_of_trip(datum, city):
             """
             Takes as input a dictionary containing info about a single trip (datum) and
             its origin city (city) and returns the month, hour, and day of the week in
             which the trip was made.

             Remember that NYC includes seconds, while Washington and Chicago do not.

             HINT: You should use the datetime module to parse the original date
             strings into a format that is useful for extracting the desired information.
             see https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

             c,w  = 3/31/2016 23:30
             """
             # YOUR CODE HERE
             if city == 'NYC':
                 return getMHDW(datum['starttime'], "%m/%d/%Y %H:%M:%S")
             elif city=="Washington":
                 return getMHDW(datum['Start date'], "%m/%d/%Y %H:%M")
             else:
                 return getMHDW(datum['starttime'], "%m/%d/%Y %H:%M")

         # Some tests to check that your code works. There should be no output if all of
         # the assertions pass. The `example_trips` dictionary was obtained from when
         # you printed the first trip from each of the original data files.
         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 [68]: # Get Start and End station Id and Name
         def getStartAndEndStationIdAndName(datum, city):
             if city == 'NYC':
                 return (datum['start station id'], datum['end station id'], datum['start statio
             elif city == 'Chicago':
                 return (datum['from_station_id'], datum['to_station_id'], datum['from_station_n
```

```
        elif city == 'Washington':
            return (datum['Start station number'], datum['End station number'], datum['Star
```

In [69]:
```python
def type_of_user(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (datum) and
    its origin city (city) and returns the type of system user that made the
    trip.

    Remember that Washington has different category names compared to Chicago
    and NYC.
    """

    # YOUR CODE HERE
    if city == 'Washington':
        user_type = "Subscriber" if datum['Member Type'] == 'Registered' else "Customer
    else:
        user_type = datum['usertype']
    return user_type


# Some tests to check that your code works. There should be no output if all of
# the assertions pass. The `example_trips` dictionary was obtained from when
# you printed the first trip from each of the original data files.
tests = {'NYC': 'Customer',
         'Chicago': 'Subscriber',
         'Washington': 'Subscriber'}

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 [70]:
```python
def condense_data(in_file, out_file, city):
    """
    This function takes full data from the specified input file
    and writes the condensed data to a specified output file. The city
    argument determines how the input file will be parsed.

    HINT: See the cell below to see how the arguments are structured!
    """

    with open(out_file, 'w') as f_out, open(in_file, 'r') as f_in:
        # set up csv DictWriter object - writer requires column names for the
```

```
                    # first row as the "fieldnames" argument
                    out_colnames = ['duration', 'month', 'month_str','hour','day_of_week', 'user_ty
                                    'start_station_id','end_station_id','start_station_name','end_s
                    trip_writer = csv.DictWriter(f_out, fieldnames = out_colnames)
                    trip_writer.writeheader()

                    ## TODO: set up csv DictReader object ##
                    trip_reader = csv.DictReader(f_in)
                    MONTH = {1:'January', 2:'February', 3:'March', 4:'April', 5:'May', 6:'June',
                        7:'July', 8:'August', 9:'September', 10:'October', 11:'November', 12:'Decem
                    # type_of_user
                    # time_of_trip
                    # duration_in_mins

                    # collect data from and process each row
                    for row in trip_reader:
                        # set up a dictionary to hold the values for the cleaned and trimmed
                        # data point
                        new_point = {}
                        new_point['duration'] = duration_in_mins(row, city)
                        month, hour, day_of_week = time_of_trip(row, city)
                        new_point['month'] = month
                        new_point['month_str'] = MONTH[month]
                        new_point['hour'] = hour
                        new_point['day_of_week'] = day_of_week
                        new_point['user_type'] = type_of_user(row, city)
                        (start_station_id, end_station_id, start_station_name, end_station_name ) =
                        new_point['start_station_id'] = start_station_id
                        new_point['end_station_id'] = end_station_id
                        new_point['start_station_name'] = start_station_name
                        new_point['end_station_name'] = end_station_name
                        trip_writer.writerow(new_point)


In [71]:  # 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'),
            ('month_str', 'March'),
            ('hour', '22'),
            ('day_of_week', 'Thursday'),
            ('user_type', 'Subscriber'),
            ('start_station_id', '31602'),
            ('end_station_id', '31207'),
            ('start_station_name', 'Park Rd & Holmead Pl NW'),
            ('end_station_name', 'Georgia Ave and Fairmont St NW')])


City: Chicago
OrderedDict([('duration', '15.433333333333334'),
            ('month', '3'),
            ('month_str', 'March'),
            ('hour', '23'),
            ('day_of_week', 'Thursday'),
            ('user_type', 'Subscriber'),
            ('start_station_id', '156'),
            ('end_station_id', '166'),
            ('start_station_name', 'Clark St & Wellington Ave'),
            ('end_station_name', 'Ashland Ave & Wrightwood Ave')])


City: NYC
OrderedDict([('duration', '13.983333333333333'),
            ('month', '1'),
            ('month_str', 'January'),
            ('hour', '0'),
            ('day_of_week', 'Friday'),
            ('user_type', 'Customer'),
            ('start_station_id', '532'),
            ('end_station_id', '401'),
            ('start_station_name', 'S 5 Pl & S 4 St'),
            ('end_station_name', 'Allen St & Rivington St')])
```

> **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. NYC has the highest number of trips. It has total of 276798 trips made 2. NYC has highest proportion of trips made by subscribers 3. Chicago has the highest proportion of trips made by short-term customers

```
In [72]: def number_of_trips(filename):
             """
             This function reads in a file with trip data and reports the number of
             trips made by subscribers, customers, and total overall.
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)

                 # initialize count variables
                 n_subscribers = 0
                 n_customers = 0
                 cities = {}
                 # tally up ride types
                 for row in reader:
                     if row['user_type'] == 'Subscriber':
                         n_subscribers += 1
                     else:
                         n_customers += 1

                 # compute total number of rides
                 n_total = n_subscribers + n_customers

                 # return tallies as a tuple
                 return(n_subscribers, n_customers, n_total)

In [73]: ## Modify this and the previous cell to answer Question 4a. Remember to run ##
         ## the function on the cleaned data files you created from Question 3.      ##

         data_file = {
                 'washington': {
                     'filename':'./data/Washington-2016-Summary.csv'
                 },
                  'chicago': {
                     'filename': './data/Chicago-2016-Summary.csv'
                 },
                 'nyc': {
                     'filename': './data/NYC-2016-Summary.csv'
                 }
```

```
                }

        for city in data_file:
            s, c, total = number_of_trips(data_file[city]['filename'])
            data_file[city]['subscriber'] = s/total
            data_file[city]['customer'] = c/total
            data_file[city]['total'] = total

        # 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?
        print(data_file)

{'washington': {'filename': './data/Washington-2016-Summary.csv', 'subscriber': 0.78028224225793
```

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:

```
In washington,
     Average trip length 18.93287355913721
     and proportion of rides made longer than 30 minutes is 0.10838886711093688

In chicago,
     Average trip length 16.563629368787335
     and proportion of rides made longer than 30 minutes is 0.08347312528593809

In nyc,
     Average trip length 15.81259299802294
     and proportion of rides made longer than 30 minutes is 0.07316888127804391
```

```
In [74]: def average_and_proportion_of_trip(filename):
             """
             This function reads in a file with trip data and reports the number of
             trips made by subscribers, customers, and total overall.
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
```

```
                reader = csv.DictReader(f_in)
                ap = {'totalTripDuration': 0 ,'total': 0, 'below30': 0, 'above30': 0 }
                # find TotalTripDuration and count for below 30 minutes and above 30 minutes
                for row in reader:
                    duration = float(row['duration'])
                    ap['total'] += 1
                    ap['totalTripDuration'] += duration
                    if duration < 30:
                        ap['below30'] += 1
                    else:
                        ap['above30'] += 1

                ap['avg'] = ap['totalTripDuration'] / ap['total']
                ap['proportion'] = ap['above30'] / ap['total']
                return ap

In [75]: ## Use this and additional cells to answer Question 4b.            ##
         ##                                                                 ##
         ## HINT: The csv module reads in all of the data as strings, including  ##
         ## numeric values. You will need a function to convert the strings    ##
         ## into an appropriate numeric type before you aggregate data.       ##
         ## TIP: For the Bay Area example, the average trip length is 14 minutes ##
         ## and 3.5% of trips are longer than 30 minutes.                     ##

         data_file = {
                 'washington': {
                     'filename':'./data/Washington-2016-Summary.csv'
                 },
                  'chicago': {
                     'filename': './data/Chicago-2016-Summary.csv'
                 },
                 'nyc': {
                     'filename': './data/NYC-2016-Summary.csv'
                 }
                 }

        for city in data_file:
            ap = average_and_proportion_of_trip(data_file[city]['filename'])
            data_file[city]['avg'] = ap['avg']
            data_file[city]['proportion'] = ap['proportion']
            print('In {}, \n\t Average trip length {} \n\t and proportion of rides made longer

In washington,
        Average trip length 18.93287355913721
        and proportion of rides made longer than 30 minutes is 0.10838886711093688
In chicago,
        Average trip length 16.563629368787335
        and proportion of rides made longer than 30 minutes is 0.08347312528593809
```

In nyc,

       Average trip length 15.81259299802294

       and proportion of rides made longer than 30 minutes is 0.07316888127804391

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 washington Customers takes longer rides on average: 41.67803139252976
In chicago Customers takes longer rides on average: 30.979781133982506
In nyc Customers takes longer rides on average: 32.982004306775025

```
In [76]: def longerRideUserType(filename):
             with open(filename, 'r') as f_in:
                 reader = csv.DictReader(f_in);
                 lr = {'subscriberCount': 0, 'customerCount': 0, 'subscriberDuration':0, 'custom

                 for row in reader:
                     if row['user_type'] == 'Subscriber':
                         lr['subscriberCount'] += 1
                         lr['subscriberDuration']  += float(row['duration'])
                     elif row['user_type'] == 'Customer':
                         lr['customerCount'] += 1
                         lr['customerDuration']  += float(row['duration'])

                 # find the average duration for subscribers and customers
                 lr['avgSub'] = lr['subscriberDuration'] / lr['subscriberCount']
                 lr['avgCus'] = lr['customerDuration'] / lr['customerCount']
                 return lr

In [77]: ## Use this and additional cells to answer Question 4c. If you have    ##
         ## not done so yet, consider revising some of your previous code to     ##
         ## make use of functions for reusability.                              ##
         ##                                                                      ##
         ## TIP: For the Bay Area example data, you should find the average      ##
         ## Subscriber trip duration to be 9.5 minutes and the average Customer ##
         ## trip duration to be 54.6 minutes. Do the other cities have this      ##
         ## level of difference?                                                 ##

         data_file = {
                 'washington': {
                     'filename':'./data/Washington-2016-Summary.csv'
                 },
                  'chicago': {
                     'filename': './data/Chicago-2016-Summary.csv'
                 },
                 'nyc': {
                     'filename': './data/NYC-2016-Summary.csv'
```

```
                    }
                }

        for city in data_file:
            lr = longerRideUserType(data_file[city]['filename'])
            data_file[city]['avgSub'] = lr['avgSub']
            data_file[city]['avgCus'] = lr['avgCus']
            if(lr['avgSub'] > lr['avgCus']):
                print('In {} Subscribers takes longer rides on average: {}'.format(city,lr['avg
            else:
                print('In {} Customers takes longer rides on average: {}'.format(city,lr['avgCu
```

```
In washington Customers takes longer rides on average: 41.67803139252976
In chicago Customers takes longer rides on average: 30.979781133982506
In nyc Customers takes longer rides on average: 32.982004306775025
```

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 [78]: # 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()
```

Distribution of Trip Durations
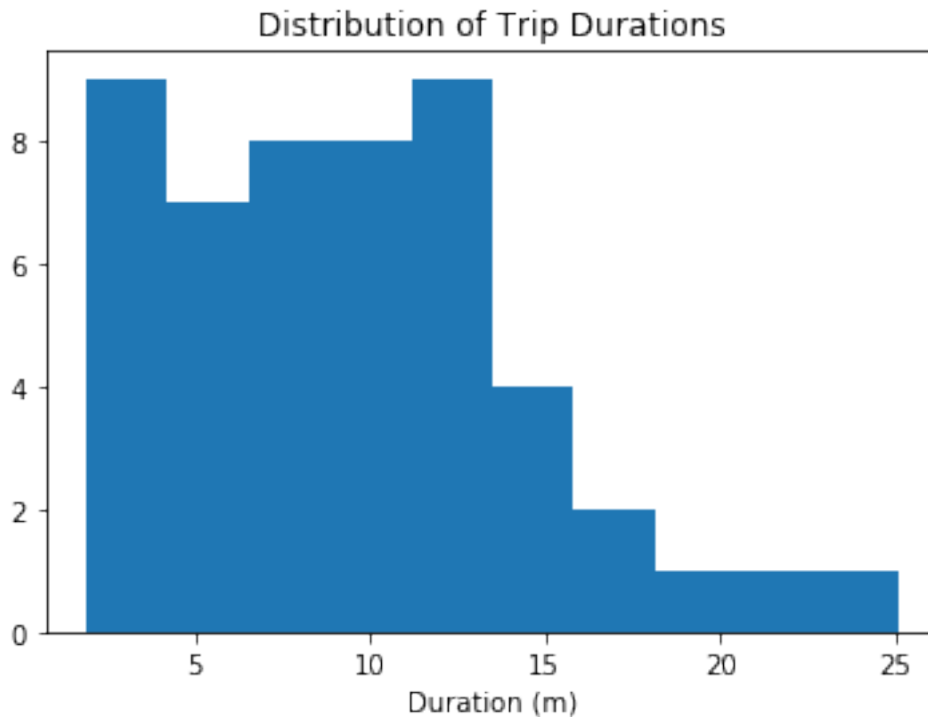
   In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. The `.title()` and `.xlabel()` functions provide some labeling for plot context.
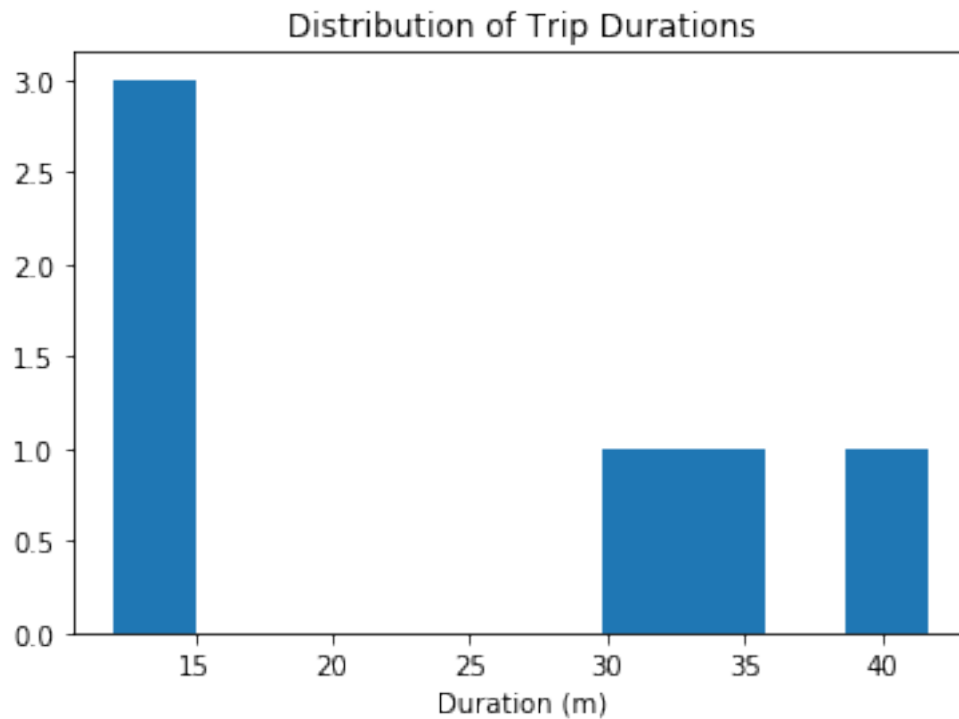
   You will now use these functions to create a histogram of the trip times for the city you selected in question 4c. Don't separate the Subscribers and Customers for now: just collect all of the trip times and plot them.

```
In [79]:  ## Use this and additional cells to collect all of the trip times as a list ##
          ## and then use pyplot functions to generate a histogram of trip times.    ##
          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
          '''
          {'washington':
           {'filename': './data/Washington-2016-Summary.csv', 'avgSub': 12.528120499294745, 'avgC
           'chicago': {'filename': './data/Chicago-2016-Summary.csv', 'avgSub': 12.06720169025007
           'nyc': {'filename': './data/NYC-2016-Summary.csv', 'avgSub': 13.680790523907177, 'avgC
          '''
          # example histogram, data taken from bay area sample
          data = [ 12.528120499294745, 41.67803139252976, 12.067201690250076, 30.979781133982506,
          plt.hist(data)
```

15

```
plt.title('Distribution of Trip Durations')
plt.xlabel('Duration (m)')
plt.show()
```


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:

```
For subscribers peak distribution is 10-15 minutes
For customers peak distribution is 15-25 minutes



Shapes:

For Subscriper, the shape of the distribution is "SKEWED" type
```

For Customer, the shape of the distribution is "SYMMETRICAL" type

```
In [80]: def trip_times(filename):
             """
             This function reads in a file with trip data and reports the number of
             trips made by subscribers, customers, and total overall.
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)
                 tripTimes = {'subTripTime': [], 'custTripTime': [] }
                 # initialize count variables
                 # tally up ride types
                 for row in reader:
                     duration = float(row['duration'])
                     if row['user_type'] == 'Subscriber':
                         tripTimes['subTripTime'].append(duration)
                     elif row['user_type'] == 'Customer':
                         tripTimes['custTripTime'].append(duration)

                 return tripTimes

         data_file = {
                 'washington': {
                     'filename':'./data/Washington-2016-Summary.csv',
                     'subTripTime': []
                 },
                  'chicago': {
                     'filename': './data/Chicago-2016-Summary.csv',
                     'subTripTime': []
                 },
                 'nyc': {
                     'filename': './data/NYC-2016-Summary.csv' ,
                     'subTripTime': []
                 }
                 }
         # where is the peak of each distribution?
         # How would you describe the shape of each distribution?
         for city in data_file:
             tripTimes = trip_times(data_file[city]['filename'])
             data_file[city]['subTripTime'] = tripTimes['subTripTime']
             data_file[city]['custTripTime'] = tripTimes['custTripTime']

In [81]: ## Use this and additional cells to answer Question 5. ##
         ## Use this and additional cells to collect all of the trip times as a list ##
         ## and then use pyplot functions to generate a histogram of trip times.      ##
         import matplotlib.pyplot as plt
```

```python
# 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
plt.hist(data_file['washington']['subTripTime'],None, range=(0,75), density=None,
         weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orie
         rwidth=None, log=False, color=None, label=None, stacked=False)
plt.title('Distribution of Subscribers Trip Durations')
plt.xlabel('Duration (m)')
plt.xlim(xmin=0, xmax = 75)
plt.show()

plt.hist(data_file['washington']['custTripTime'],None, range=(0,75), density=None,
         weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orie
         rwidth=None, log=False, color=None, label=None, stacked=False)
plt.title('Distribution of Customer Trip Durations')
plt.xlabel('Duration (m)')
plt.xlim(xmin=0, xmax = 75)
plt.show()
```


Distribution of Subscribers Trip Durations

Distribution of Customer Trip Durations
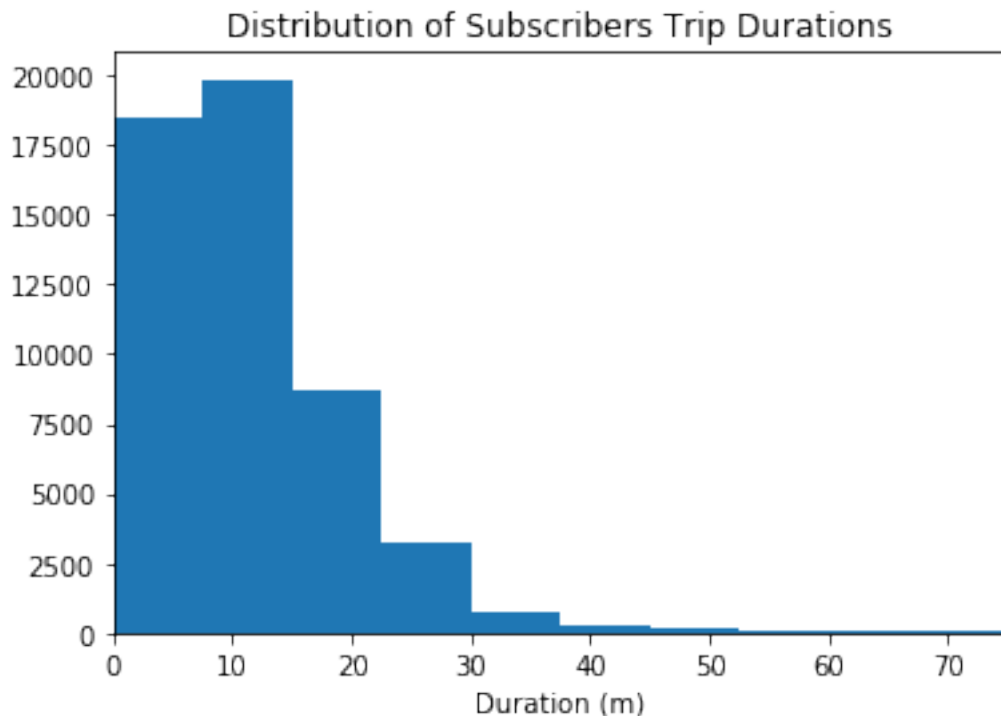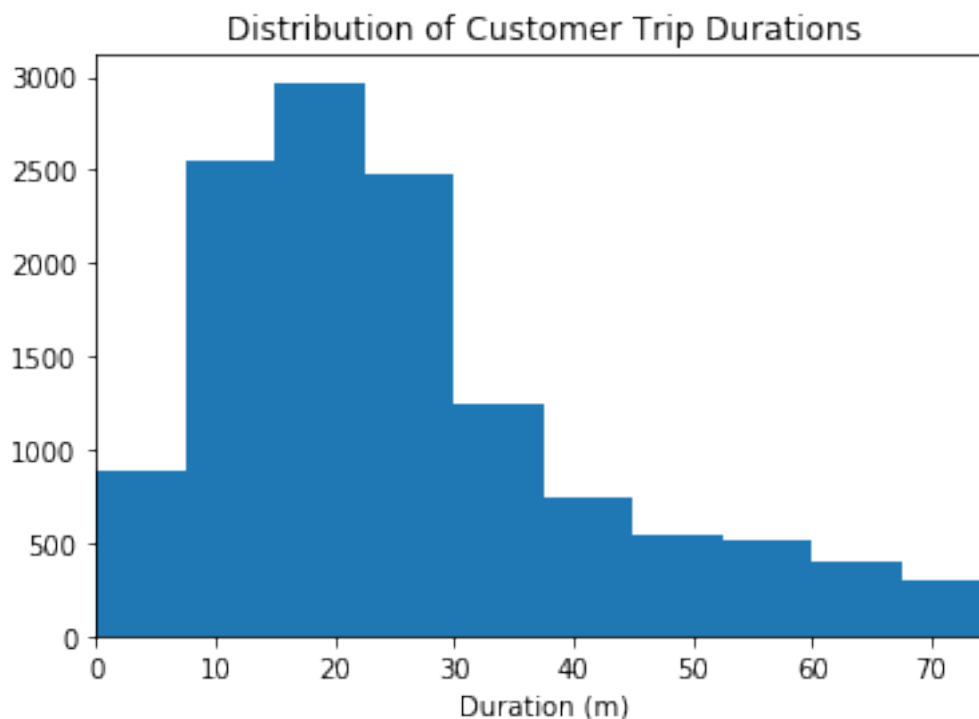
Performing Your Own Analysis

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: 1. Which month has highest riders rate? - (It will helps to increase the bikes on that month) 1. In Washington July has highest rider rate = 7341. 2. In Chicago July has highest rider rate = 10286. 3. In NYC September has highest rider rate = 32699. 2. Which Station's bikes most used (It will also helps to increase bike on that stations) 1. In Chicago Streeter Dr & Grand Ave station has been used by overall 1836 users. 2. In NYC Pershing Square North has been used by overall 2964 users. 3. In Washington Columbus Circle / Union Station has been used by overall 1387 users. 3. Ratio of subscriber trips to customer trips changes depending season 1. In Winter season Dec, Jan, Feb has highest ratio for the subscribers when compared to customers.

```
In [82]: # To find which station has highest bikeshare system users

         def findMostUsedStations(filename):
             stationMap = {}

             with open(filename, 'r') as f_in:
                 reader = csv.DictReader(f_in)

                 # initialize subscribers, customers count variables
                 result = {}
                 """
                 new_point['start_station_id'] = start_station_id
                 new_point['end_station_id'] = end_station_id
                 new_point['start_station_name'] = start_station_name
                 new_point['end_station_name'] = end_station_name
                 """
                 # this loop will calculate total riders rate seperated by subscriber/customer
                 for row in reader:
                     # generate a hashmap with stationId and name
                     if(result.get(row['start_station_id']) == None):
                         result[row['start_station_id']] = {
                             'name': row['start_station_name'],
                             'subscriber': 0,
                             'customer': 0,
                             'total': 0
                         }
                     else:
                         if row['user_type'] == 'Subscriber':
                             result[row['start_station_id']]['subscriber'] += 1
                             result[row['start_station_id']]['total'] += 1
                         elif row['user_type'] == 'Customer':
                             result[row['start_station_id']]['customer'] += 1
                             result[row['start_station_id']]['total'] += 1

                 # Find total count of station Id's
                 mostUsedStationName = "",
                 mostUsedStation = 0
```

20

```python
            for stationId in result:
                if (result[stationId]['total'] > mostUsedStation):
                    mostUsedStation = result[stationId]['total']
                    mostUsedStationName = result[stationId]['name']

            return (mostUsedStationName, mostUsedStation)


    data_file = {
        'washington': {
            'filename':'./data/Washington-2016-Summary.csv'
        },
         'chicago': {
             'filename': './data/Chicago-2016-Summary.csv'
        },
        'nyc': {
            'filename': './data/NYC-2016-Summary.csv'
        }
    }

    for city in data_file:
        ( mostUsedStationName, totalCount) = findMostUsedStations(data_file[city]['filename
        data_file[city]['most_used_station_name']= mostUsedStationName
        data_file[city]['most_used_station_total_count']= totalCount
        print('In {}, {} has been used by overall {} users'.format(city, mostUsedStationNam
```

```
In washington, Columbus Circle / Union Station has been used by overall 1387 users
In chicago, Streeter Dr & Grand Ave has been used by overall 1836 users
In nyc, Pershing Square North has been used by overall 2964 users
```

```python
In [83]: # This function will count riding rate for every months.
         # Each month having total number of riders rate seperated by subscriber/customer
         # And also having ratio of riders by every month

         def calculate_ratio_by_months(filename):
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)

                 # initialize subscribers, customers count variables
                 result = {}

                 # this loop will calculate total riders rate seperated by subscriber/customer
                 for row in reader:
                     if row['user_type'] == 'Subscriber':
                         if(result.get(row['month_str']) == None):
                             # set initial month key value pair
```

```python
                            result[row['month_str']] = {
                                'total': 1, 'subscriber': 1, 'customer': 0, 'ratio': ()
                            }
                        else:
                            result[row['month_str']]['subscriber'] += 1
                            result[row['month_str']]['total'] += 1
                else:
                    if(result.get(row['month_str']) == None):
                        # set initial month key value pair
                        result[row['month_str']] = {
                            'total': 1, 'subscriber': 0, 'customer': 1, 'ratio': ()
                        }
                    else:
                        result[row['month_str']]['customer'] += 1
                        result[row['month_str']]['total'] += 1

            # This loop will calculate ratio of subscriber/customer rider count
            for month in result:
                ratio = result[month]['subscriber'] / result[month]['customer']
                result[month]['ratio']= ratio

            return result


In [84]: data_file = {
                'washington': {
                    'filename':'./data/Washington-2016-Summary.csv'
                },
                'chicago': {
                    'filename': './data/Chicago-2016-Summary.csv'
                },
                'nyc': {
                    'filename': './data/NYC-2016-Summary.csv'
                }
        }


        for city in data_file:
            ratioByMonths = calculate_ratio_by_months(data_file[city]['filename'])
            max_rides = ratioByMonths['January']['total']
            max_ride_month = 'Jan'

            # find which month having maximum rides by city
            for month in ratioByMonths:
                total = ratioByMonths[month]['total']
                if max_rides < total:
                    max_rides = total
                    max_ride_month = month
```

22

```
            data_file[city]['monthly_result']= total
            data_file[city]['max_rides'] = max_rides
            data_file[city]['max_ride_month'] = max_ride_month
            #print('In {}, {} has highest rider rate => {}'.format(city, max_ride_month, max_ri

        print(data_file)

{'washington': {'filename': './data/Washington-2016-Summary.csv', 'monthly_result': 6792, 'max_r
```

In [85]:
```
data_file = {
            'washington': {
                'filename':'./data/Washington-2016-Summary.csv'
            },
             'chicago': {
                    'filename': './data/Chicago-2016-Summary.csv'
            },
            'nyc': {
                    'filename': './data/NYC-2016-Summary.csv'
            }
        }

        import numpy as np
        import matplotlib.pyplot as plt

        N = 12

        for city in data_file:
            ratioByMonths = calculate_ratio_by_months(data_file[city]['filename'])
            data_file[city]['all'] = ratioByMonths
            subscriberMeans = (data_file[city]['all']['January']['subscriber'],
                        data_file[city]['all']['February']['subscriber'],
                        data_file[city]['all']['March']['subscriber'],
                        data_file[city]['all']['April']['subscriber'],
                        data_file[city]['all']['April']['subscriber'],
                        data_file[city]['all']['June']['subscriber'],
                        data_file[city]['all']['July']['subscriber'],
                        data_file[city]['all']['August']['subscriber'],
                        data_file[city]['all']['September']['subscriber'],
                        data_file[city]['all']['October']['subscriber'],
                        data_file[city]['all']['November']['subscriber'],
                        data_file[city]['all']['December']['subscriber'])

            ind = np.arange(N)   # the x locations for the groups
            width = 0.40         # the width of the bars

            fig, ax = plt.subplots()
```

```python
rects1 = ax.bar(ind, subscriberMeans, width, color='r')

customerMeans = (data_file[city]['all']['January']['customer'],
                 data_file[city]['all']['February']['customer'],
                 data_file[city]['all']['March']['customer'],
                 data_file[city]['all']['April']['customer'],
                 data_file[city]['all']['April']['customer'],
                 data_file[city]['all']['June']['customer'],
                 data_file[city]['all']['July']['customer'],
                 data_file[city]['all']['August']['customer'],
                 data_file[city]['all']['September']['customer'],
                 data_file[city]['all']['October']['customer'],
                 data_file[city]['all']['November']['customer'],
                 data_file[city]['all']['December']['customer'])
rects2 = ax.bar(ind + width, customerMeans, width, color='g')

# add text for labels, title and axes ticks
title = city + " - Rides by months"
ax.set_ylabel('Total Rides')
ax.set_title(title)
ax.set_xticks(ind + width / 2)
ax.set_xticklabels(('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',

ax.legend((rects1[0], rects2[0]), ('Sub', 'Cus'))


def drawBar(rects):
    """
    Add total ride count label above each bar
    """
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/1.5, 1.0*height,
                '%d' % int(height),
                ha='center', va='bottom')

drawBar(rects1)
drawBar(rects2)
plt.show()
```
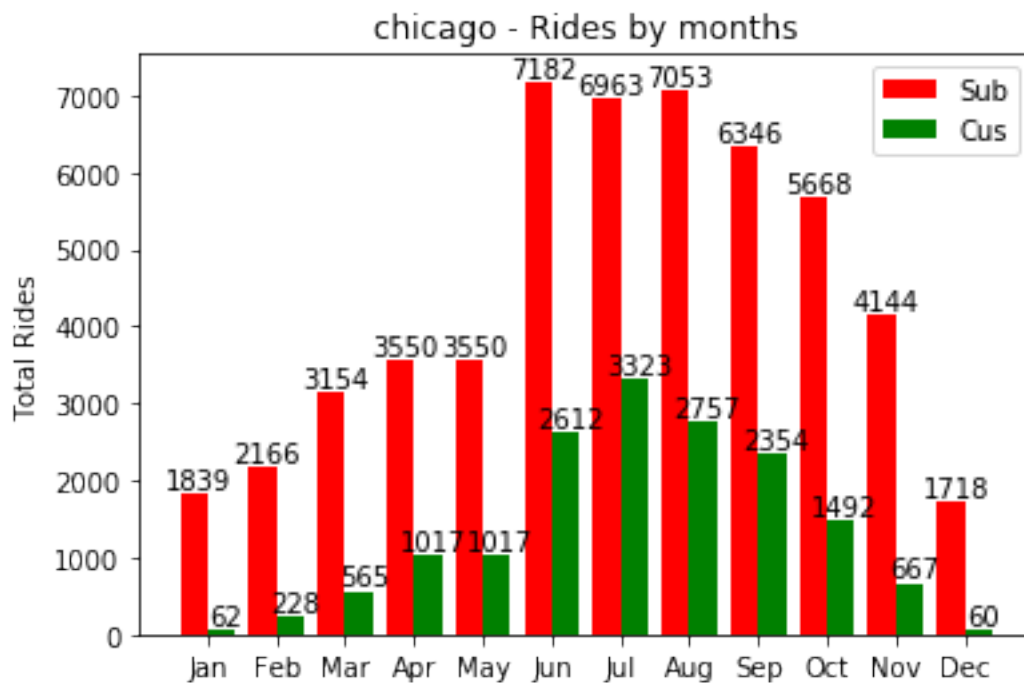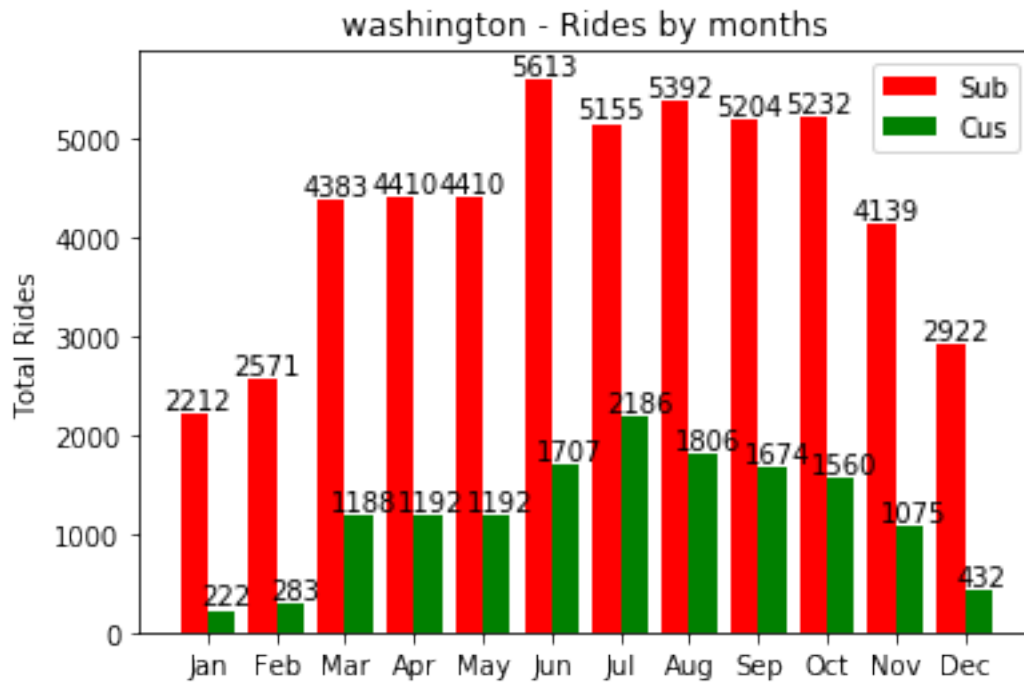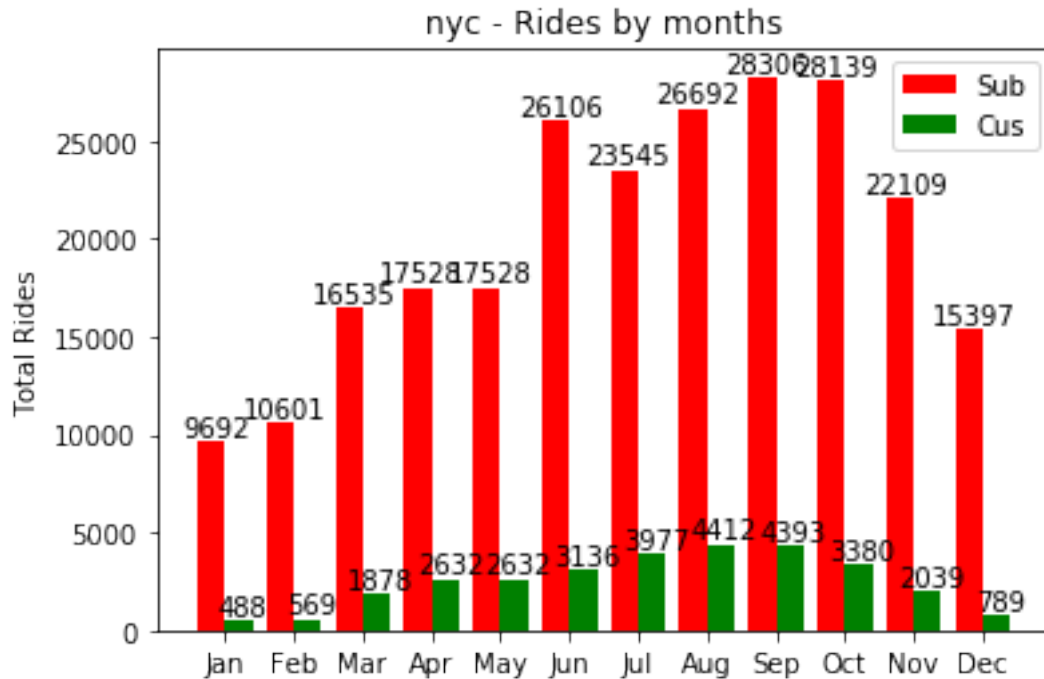
washington - Rides by months

chicago - Rides by months

Figure: nyc - Rides by months

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:

```
I want to analyse real estate property value increasing/decreasing based on seasons, flood, natu

And I am very much interested in physical activity. So i want analyse how body and brain changes
```

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.

2 Resources

For graphs and visualization part, i used https://matplotlib.org/devdocs/contents.html webpage f

For date and time https://docs.python.org/3/library/datetime.html?highlight=time#module-datetime