Bike_Share_Analysis

March 10, 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.

Answer: 1. Which stations are frequently visited ? 2. On what days and times do people use the bikeshares the most ? 3. How many bikes should each bikeshare have available ?

> **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 [7]: ## import all necessary packages and functions.
        import csv # read and write csv files
        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.
        import pandas as pd # converts CSV files into dataframes which are more
```

```
                        # practical to use than plain text
import numpy as np # performs calculations
#from ggplot import * # I know we're not supposed to do this but there is
                      # no other way.
import matplotlib as mpl # Still required to change the sizes of plots
```

In [8]:
```
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:
        ## TODO: Use the csv library to set up a DictReader object. ##
        ## see https://docs.python.org/3/library/csv.html           ##
        trip_reader = csv.DictReader(f_in)

        ## TODO: Use a function on the DictReader object to read the     ##
        ## first trip from the data file and store it in a variable.     ##
        ## see https://docs.python.org/3/library/csv.html#reader-objects ##
        first_trip = next(trip_reader)

        ## TODO: Use the pprint library to print the first trip. ##
        ## see https://docs.python.org/3/library/pprint.html     ##
        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 (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 [9]: 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
            """

            # YOUR CODE HERE
            if city == 'NYC' or city == 'Chicago':
                duration = int(datum['tripduration'])
            else:
                duration = int(datum['Duration (ms)'])/1000
```

```
    return duration/60


# 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 [10]: def date_string_to_weekday(date):
             """
             Takes date as a string in the form 'mm/dd/yyyy' and converts it
             to a week day
             """

             #dictionary to convert weekday number to day of week
             weekday_dictionary = {0: "Monday",
                                   1: "Tuesday",
                                   2: "Wednesday",
                                   3: "Thursday",
                                   4: "Friday",
                                   5: "Saturday",
                                   6: "Sunday"}

             #find weekday number
             month, day, year = date.split('/')
             week_day = datetime.datetime.weekday(datetime.date(int(year),
                                                                int(month),
                                                                int(day)))


             return weekday_dictionary[week_day]

         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.
             """

             if city == 'NYC'or city == 'Chicago':

                 # extract month
                 month =  datum['starttime'].split('/')[0]
```

```
            # extract hour
            hour = datum['starttime'].split()[1].split(':')[0]

            # get day of week
            day_of_week = date_string_to_weekday(datum['starttime'].split()[0])

        else:

            # extract month
            month =  datum['Start date'].split('/')[0]

            # extract hour
            hour = datum['Start date'].split()[1].split(':')[0]

            # get day of week
            day_of_week = date_string_to_weekday(datum['Start date'].split()[0])

        return (int(month), int(hour), day_of_week)
    # 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 [13]:
```
def correct_member_type(user_type):
    """
    Converts the user type for the Washington dataset so that it fits the other
    datasets.
    """
    # Dictionary for the conversion
    user_type_dictionary = {"Registered":"Subscriber",
                            "Casual":"Customer"}

    # Converting member type
    new_user_type = user_type_dictionary[user_type]

    return new_user_type


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.
        """

        if city == 'NYC': user_type = datum['usertype']
        elif city == 'Chicago': user_type = datum['usertype']
        else: user_type = correct_member_type(datum['Member Type'])



        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 [14]: 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', 'hour', 'day_of_week', 'user_type']
            trip_writer = csv.DictWriter(f_out, fieldnames = out_colnames)
            trip_writer.writeheader()
            trip_reader = csv.DictReader(f_in)

            #  Use a function on the DictReader object to read the
            #  first trip from the data file and store it in a variable.
            first_trip = next(trip_reader)
```

```
                """
                Variables I am working with because this function is a mess:
                    out_colnames
                """

                # 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 points
                    new_point = {}
                    month, hour, day_of_week = time_of_trip(row, city)
                    new_point[out_colnames[0]] = duration_in_mins(row, city)
                    new_point[out_colnames[1]] = month
                    new_point[out_colnames[2]] = hour
                    new_point[out_colnames[3]] = day_of_week
                    new_point[out_colnames[4]] = type_of_user(row, city)

                    # Write row to new csv file
                    trip_writer.writerow(new_point)
```

```
In [15]: # 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', '9.792516666666668'),
             ('month', '3'),
             ('hour', '22'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: Chicago
OrderedDict([('duration', '3.3'),
             ('month', '3'),
             ('hour', '22'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: NYC
```

```
OrderedDict([('duration', '11.433333333333334'),
             ('month', '1'),
             ('hour', '0'),
             ('day_of_week', 'Friday'),
             ('user_type', 'Subscriber')])
```

> **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 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. Which city has the highest number of trips? NYC 2. Which city has the highest proportion of trips made by subscribers? NYC 3. Which city has the highest proportion of trips made by short-term customers?
Chicago

```
In [16]: 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

                 # 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 [17]: ## 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.      ##

         filepaths = ['./data/NYC-2016-Summary.csv',
                      './data/Chicago-2016-Summary.csv',
                      './data/Washington-2016-Summary.csv']


         for path in filepaths:

             n_subscribers, n_customers, n_total = number_of_trips(path)
             print("n_subscribers: ", n_subscribers)
             print("n_customers: ", n_customers)
             print("n_total: ", n_total)
             print("proportion subscribers: ", n_subscribers/n_total)
             print("proportion customers: ", n_customers/n_total)
```

```
n_subscribers:  245896
n_customers:  30901
n_total:  276797
proportion subscribers:  0.8883622293594222
proportion customers:  0.11163777064057775
n_subscribers:  54981
n_customers:  17149
n_total:  72130
proportion subscribers:  0.7622487175932344
proportion customers:  0.23775128240676557
n_subscribers:  51752
n_customers:  14573
n_total:  66325
proportion subscribers:  0.780278929513758
proportion customers:  0.21972107048624198
```

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: NYC :
average travel time (min): 15.8125996067 proportion over 30 mins: 0.07302463538260892
Chicago :
average travel time (min): 16.563645039 proportion over 30 mins: 0.08332178011922917
Washington :
average travel time (min): 18.933051618 proportion over 30 mins: 0.10839050131926121

In [18]:
```
## 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.                        ##

def travel_time_stats(filename):
    """
    This function calculates the average travel time of the CSV summaries.
    Input is the file path and the returned value is a float.
    """

    # Create dataframe
    df = pd.read_csv(filename)

    # Calculate average
    average = np.average(df['duration'])

    # Calculate proportion of rides above 30 mins
    proportion = len(df[df['duration']>30])/len(df)

    return average, proportion
```

In [19]:
```
filepaths = ['./data/NYC-2016-Summary.csv',
             './data/Chicago-2016-Summary.csv',
             './data/Washington-2016-Summary.csv']

# Display the data created by population_overtime function
for path in filepaths:
    average_travel, proportion_overtime = travel_time_stats(path)
    print(path,": \n")
    print("average travel time (min): ", average_travel)
    print("proportion over 30 mins: ", proportion_overtime)
    print("\n")
```

```
./data/NYC-2016-Summary.csv :

average travel time (min):  15.8125996067
proportion over 30 mins:  0.07302463538260892


./data/Chicago-2016-Summary.csv :

average travel time (min):  16.563645039
proportion over 30 mins:  0.08332178011922917


./data/Washington-2016-Summary.csv :

average travel time (min):  18.933051618
proportion over 30 mins:  0.10839050131926121
```

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: I selected city Washington. The subscriber trips and customer trips for Washington is 12.528224939776903, 41.67803139252976 respectively. Clearly, customers takes longer rides on average.

```
In [24]: ## 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?                                                ##
         def ridership_metrics(filename):
             """
             This function reads in a file with trip data and reports the ridership
             by subscribers, customers, and total overall.
             """
             subscribers = 0
             customers = 0
             subscriber_trips = 0
             customer_trips = 0
             with open(filename, 'r') as f_in:
                 reader = csv.reader(f_in)
                 next(reader, None)
                 for row in reader:
                     if row[4] in ['Subscriber','Registered']:
```

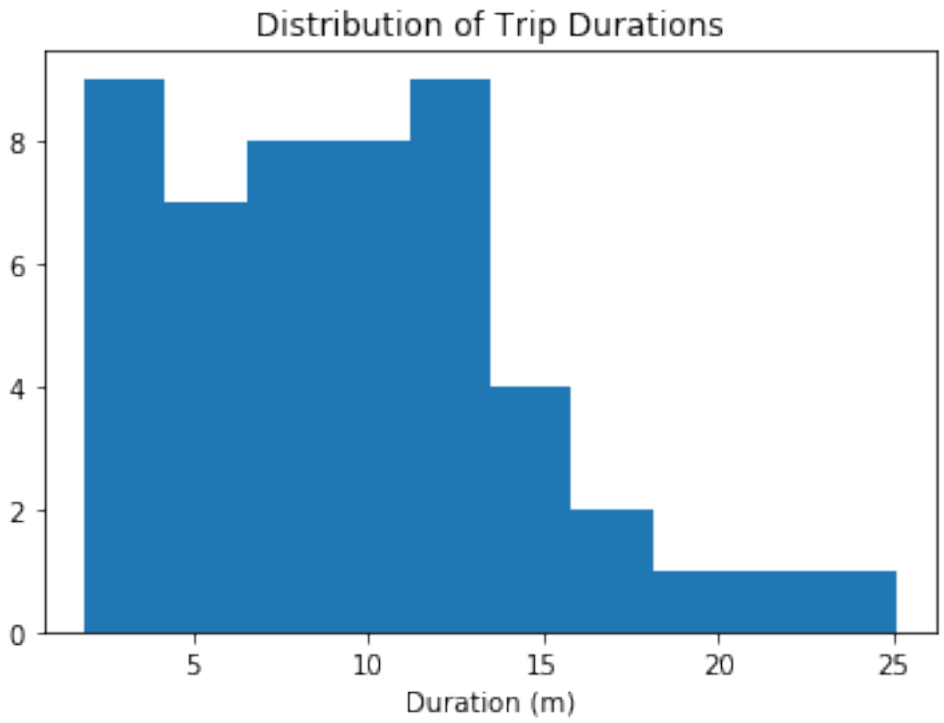
```
                    subscribers += 1
                    subscriber_trips += float(row[0])
                elif row[4] in ['Casual','Customer']:
                    customers += 1
                    customer_trips += float(row[0])
            return float(subscriber_trips/subscribers),float(customer_trips/customers)

    data_file_washington = './data/Washington-2016-Summary.csv'
    subscriber_trips,customer_trips = ridership_metrics(data_file_washington)
    print('The subscriber trips and customer trips for Washington is {}, {}'.format(subscri
```

```
The subscriber trips and customer trips for Washington is 12.528224939776903, 41.67803139252976
```

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 [25]: # 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()
```

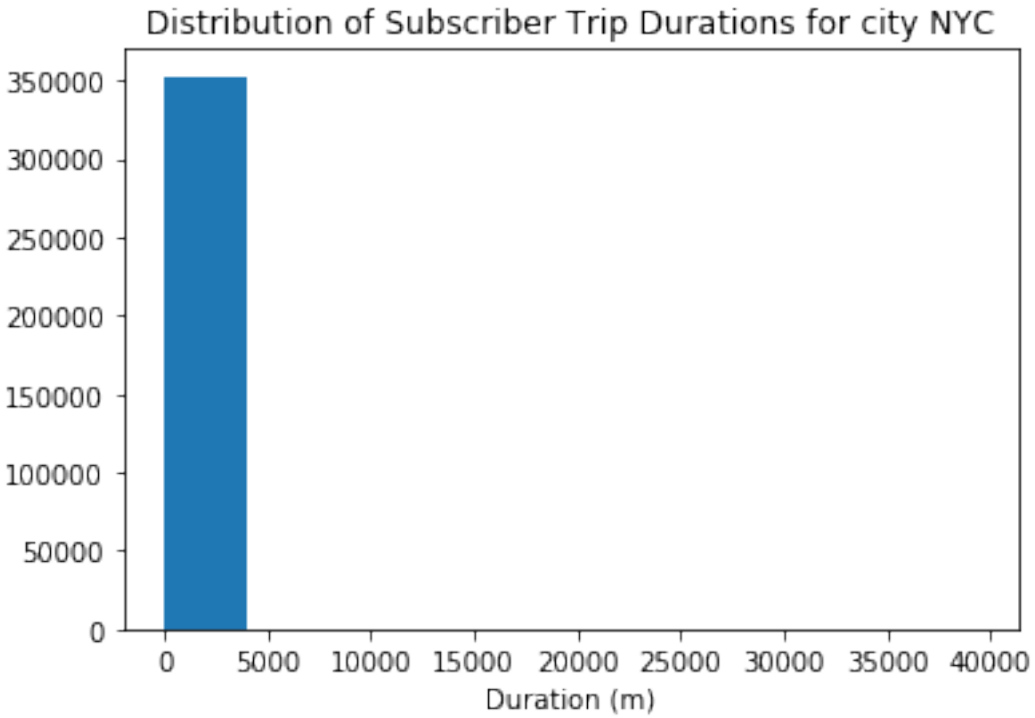


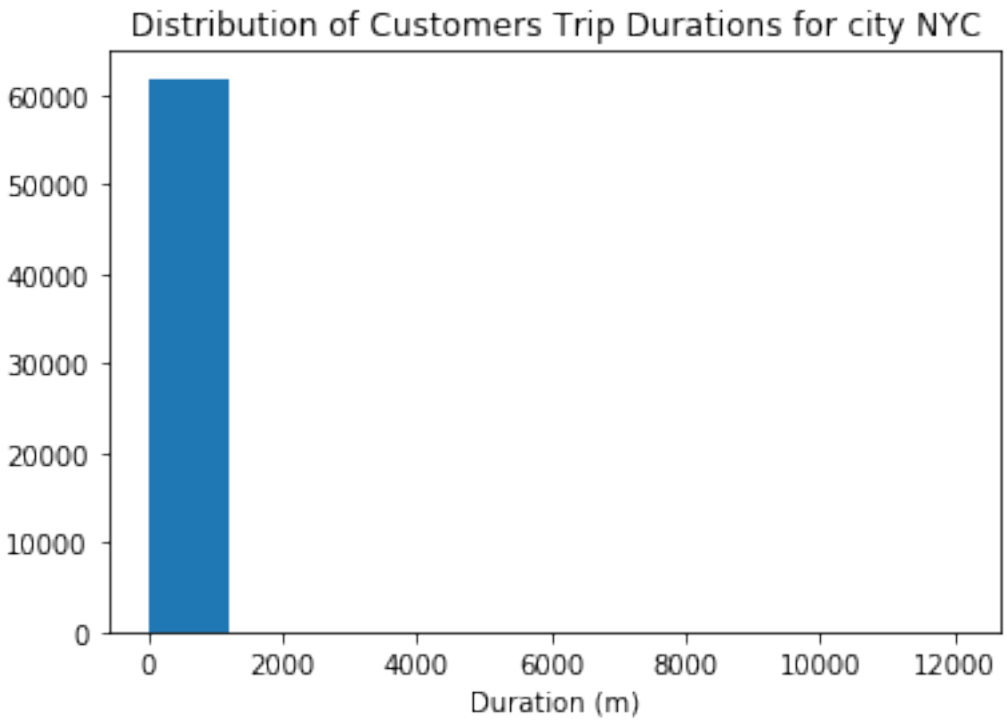
In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. 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 [32]: ## 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.     ##
         customers = []
         subscribers = []
         def get_durations_as_list(filename):
             """
             This function reads in a file with trip data and returns durations
             as list for subscribers, customers, and total overall.
             """
             with open(filename, 'r') as f_in:
                 reader = csv.reader(f_in)
                 next(reader, None)
                 for row in reader:
                     if row[4] in ['Subscriber','Registered']:
                         subscribers.append(float(row[0]))
                     elif row[4] in ['Casual','Customer']:
                         customers.append(float(row[0]))
```

```
        return subscribers,customers

data_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv','.
for file in data_files:
    city = file.split('-')[0].split('/')[-1]
    subscribers,customers = get_durations_as_list(file)

plt.hist(subscribers)
plt.title('Distribution of Subscriber Trip Durations for city {}'.format(city))
plt.xlabel('Duration (m)')
plt.show()

plt.hist(customers)
plt.title('Distribution of Customers Trip Durations for city {}'.format(city))
plt.xlabel('Duration (m)')
plt.show()
```





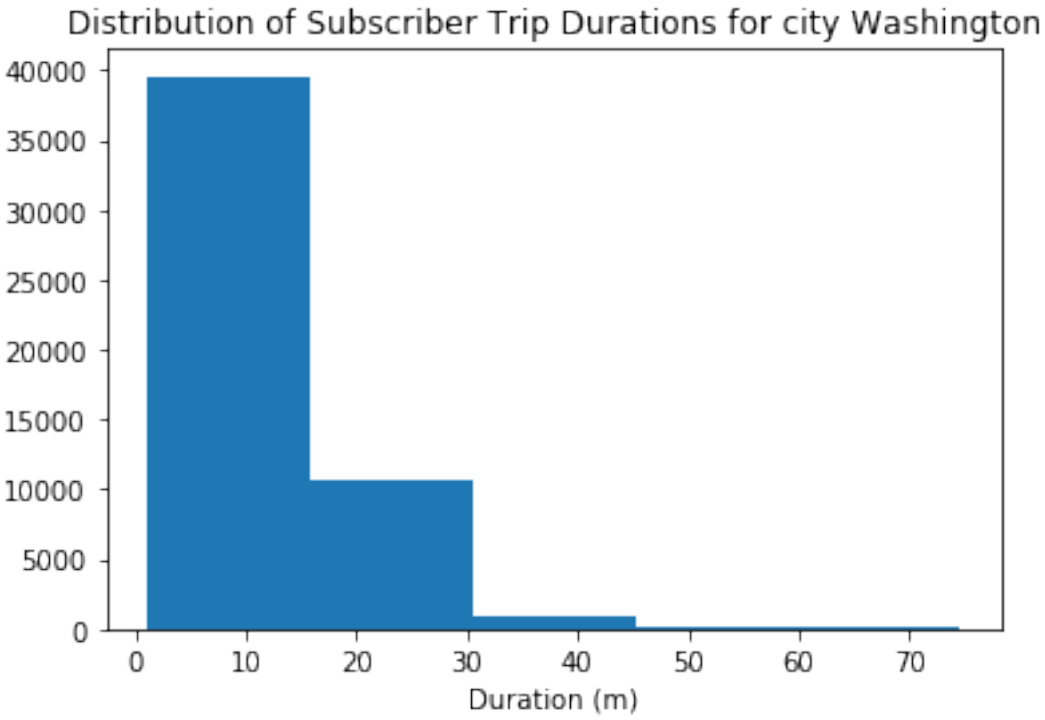
If you followed the use of the `.hist()` and `.show()` functions exactly like in the example, you're probably looking at a plot that's completely unexpected. The plot consists of one extremely tall bar on the left, maybe a very short second bar, and a whole lot of empty space in the center and right. 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].

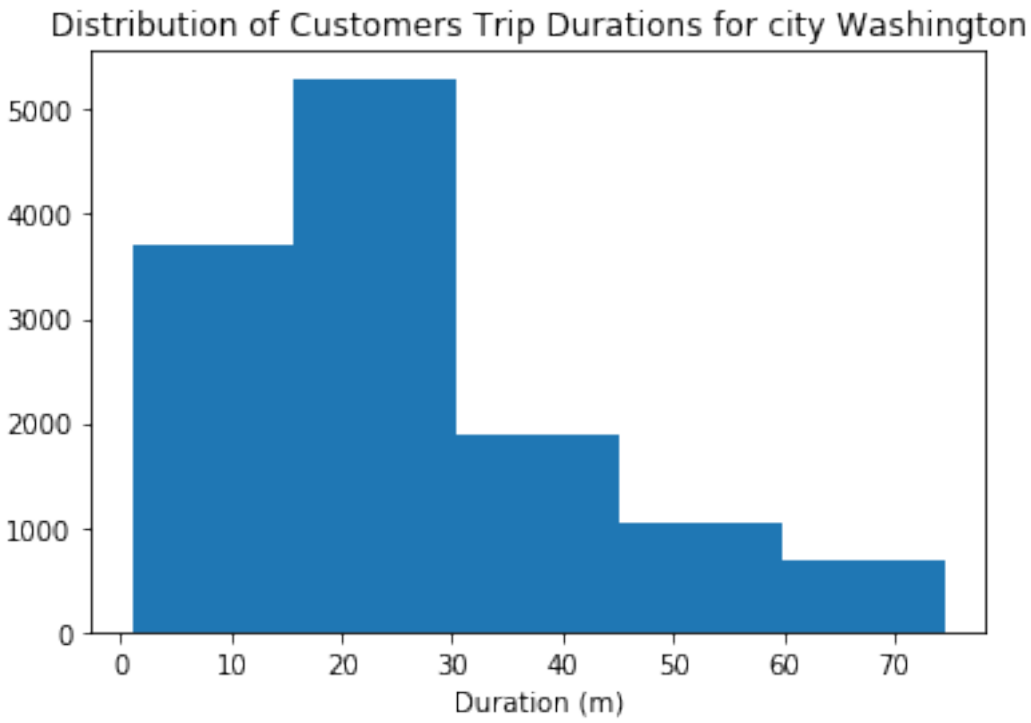
Question 5: Use the parameters of the `.hist()` function to plot the distribution of trip times for the Subscribers in your selected city. Do the same thing for only the Customers. Add limits to the plots so that only trips of duration less than 75 minutes are plotted. As a bonus, set the plots up so 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?

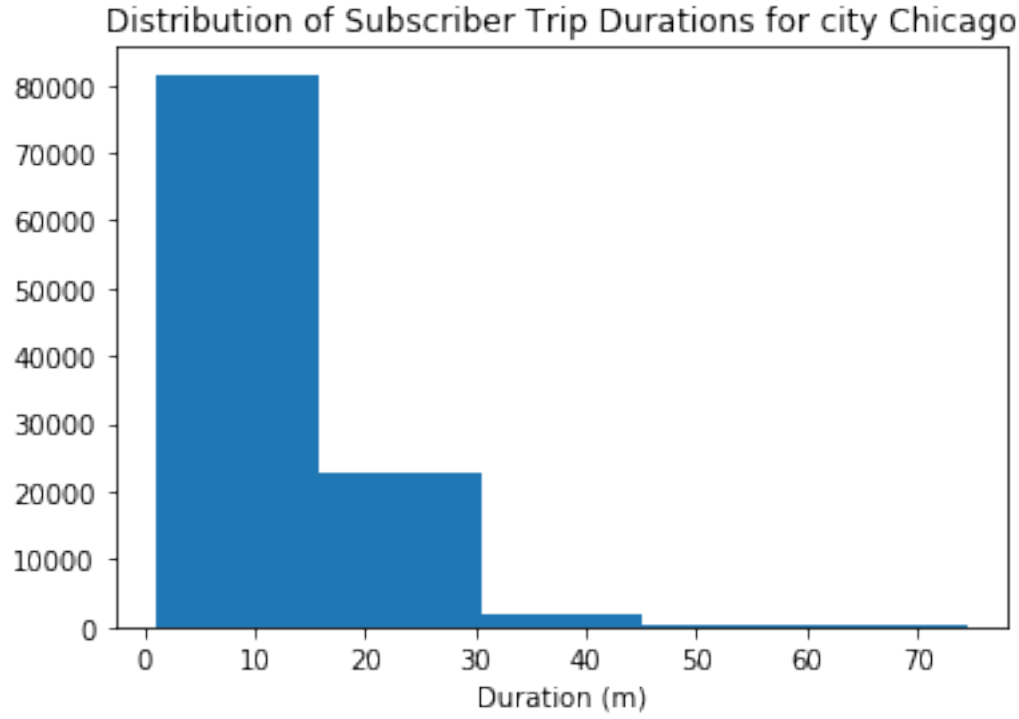
Answer: Selected city NYC. The peak for each distribution, for subscribers it is at 250000 and 25500 approximately. The shapes of the graphs are skewed towards the right.

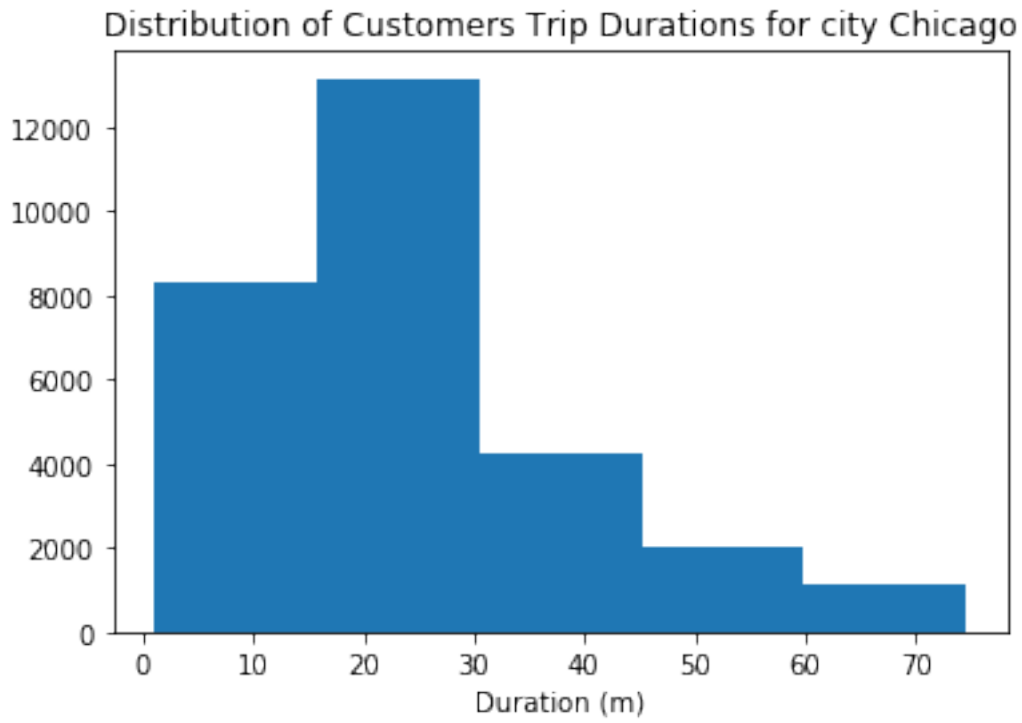
```
In [26]: ## Use this and additional cells to answer Question 5. ##
         customers = []
         subscribers = []
         def get_durations_as_list(filename):
             with open(filename, 'r') as f_in:
                 reader = csv.reader(f_in)
                 next(reader, None)
                 for row in reader:
                     if row[4] in ['Subscriber','Registered'] and float(row[0]) < 75:
                         subscribers.append(float(row[0]))
```

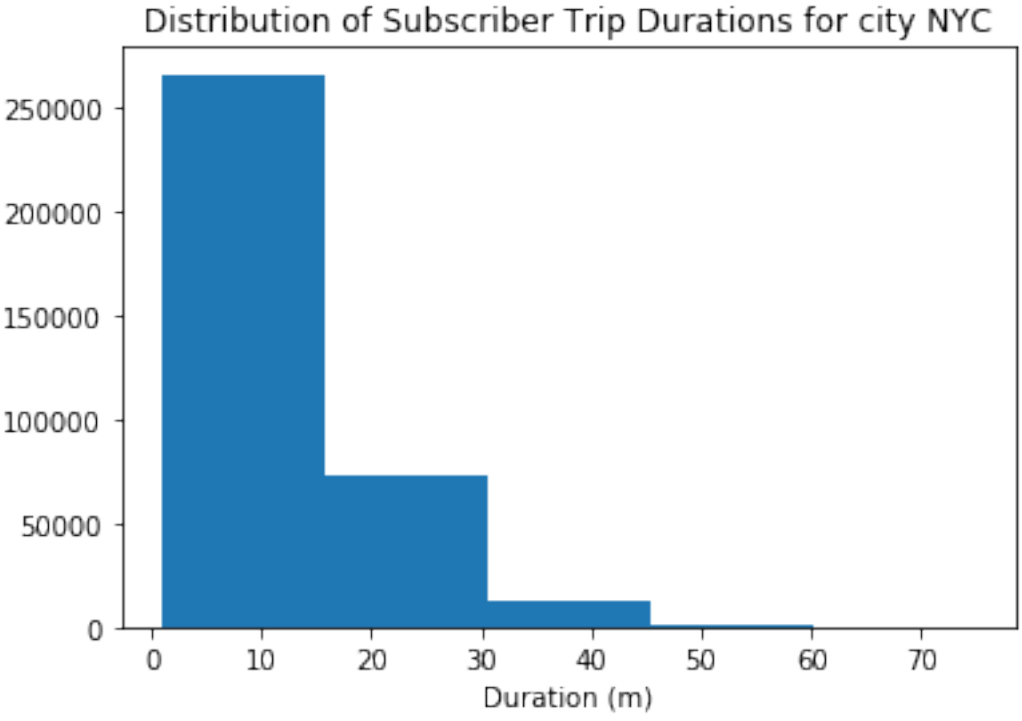
```
            elif row[4] in ['Casual','Customer'] and float(row[0]) < 75:
                customers.append(float(row[0]))
        return subscribers,customers

data_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv','.
for file in data_files:
    city = file.split('-')[0].split('/')[-1]
    subscribers,customers = get_durations_as_list(file)

    plt.hist(subscribers,range=[min(subscribers),max(subscribers)],bins=5)
    plt.title('Distribution of Subscriber Trip Durations for city {}'.format(city))
    plt.xlabel('Duration (m)')
    plt.show()

    plt.hist(customers,range=[min(subscribers),max(subscribers)],bins=5)
    plt.title('Distribution of Customers Trip Durations for city {}'.format(city))
    plt.xlabel('Duration (m)')
    plt.show()
```

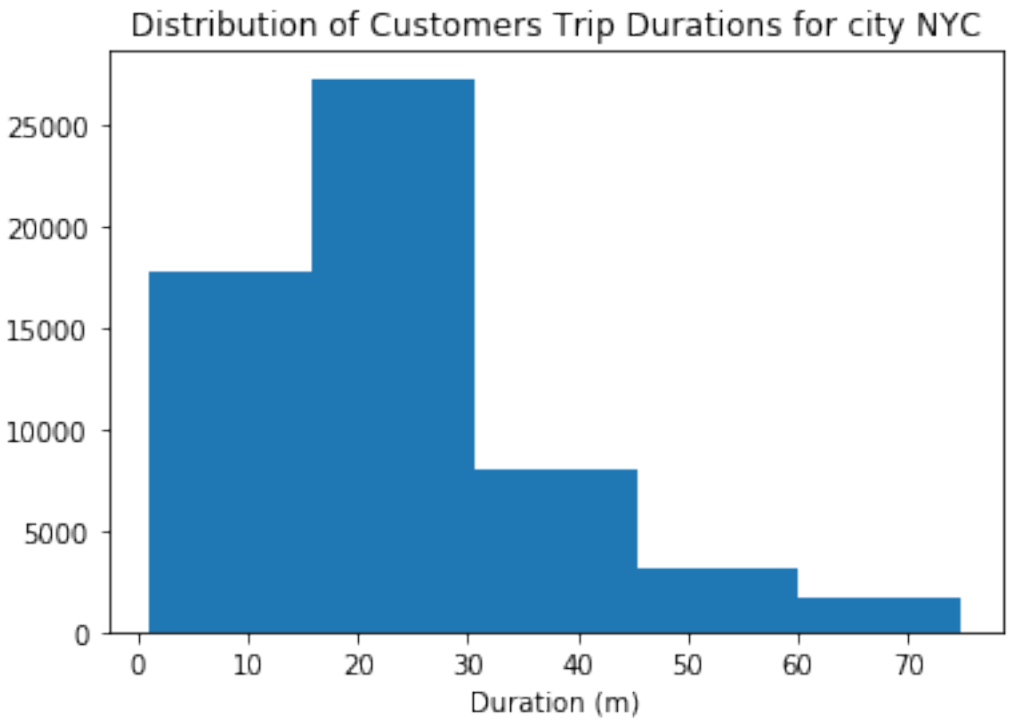



Distribution of Customers Trip Durations for city Washington
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
60
70
Duration (m)


Distribution of Subscriber Trip Durations for city Chicago
80000
70000
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
70
Duration (m)


Distribution of Customers Trip Durations for city Chicago
12000
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
60
70
Duration (m)


Distribution of Subscriber Trip Durations for city NYC
250000
200000
150000
100000
50000
0
0
10
20
30
40
50
60
70
Duration (m)



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: Which month has highest ridership ? Which month has highest subscriber ridership and which month has highest customer ridership? How ridership affected due to weekends? Which hour of day has highest ridership ? Answers of these below mentioned. - The month of july has highest ridership. - The month of june has highest subscriber ridership. - The month of july has highest customer ridership. - The ridership is high during weekdays. - The ridership for both subscribers and customers is high during weekdays. - At the hour of 17 the ridership for both subscribers and customers is high.

In [30]:
```
## Use this and additional cells to continue to explore the dataset. ##
## Once you have performed your exploration, document your findings  ##
## in the Markdown cell above.
##
def month_mapper(x):
    """
    This function reads the input month number as a string
    and maps it to the appropriate month in the calender.
    """
    return {
        '1':'january',
        '2':'february',
        '3':'march',
        '4':'april',
        '5':'may',
        '6':'june',
        '7':'july',
        '8':'august',
        '9':'september',
        '10':'october',
        '11':'november',
        '12':'december',
    }[x]

def week_mapper(x):
    """
    This function reads the input weekday/weekend as a string
    and maps it to the appropriate week in the calender.
    """
    return {
        'monday':'weekday',
        'tuesday':'weekday',
        'wednesday':'weekday',
        'thursday':'weekday',
        'friday':'weekday',
        'saturday':'weekend',
        'sunday':'weekend',
    }[x]
```

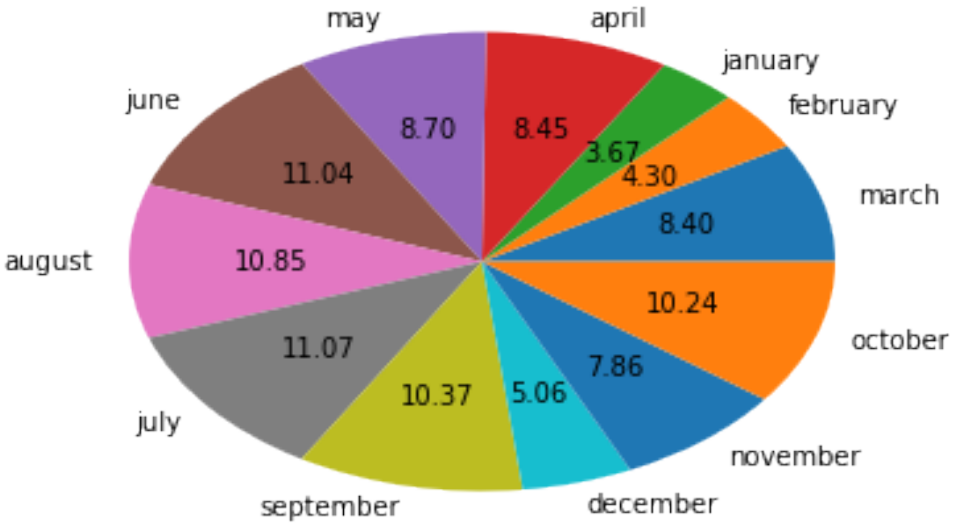
```
def monthly_metrics(filename):
    """
    This function reads the input file with all the trips
    and returns monthly metrics as a dict.
    """
    metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            month_name = month_mapper(row[1])
            metrics[month_name] = metrics.get(month_name, 0) + 1
        return metrics

def monthly_metrics_subscribers(filename):
    """
    This function reads the input file with all the trips
    and returns monthly metrics for subscriber as a dict.
    """
    subscriber_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Subscriber','Registered']:
                month_name = month_mapper(row[1])
                subscriber_metrics[month_name] = subscriber_metrics.get(month_name, 0)
        return subscriber_metrics

def monthly_metrics_customers(filename):
    """
    This function reads the input file with all the trips
    and returns monthly metrics for customer as a dict.
    """
    customers_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Casual','Customer']:
                month_name = month_mapper(row[1])
                customers_metrics[month_name] = customers_metrics.get(month_name, 0) +
        return customers_metrics

def weekly_metrics(filename):
    """
    This function reads the input file with all the trips
```

```
    and returns weekly metrics as a dict.
    """
    metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            week_name = week_mapper(row[3].lower())
            metrics[week_name] = metrics.get(week_name, 0) + 1
        return metrics

def weekly_metrics_subscribers(filename):
    """
    This function reads the input file with all the trips
    and returns weekly metrics for subscriber as a dict.
    """
    subscriber_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Subscriber','Registered']:
                week_name = week_mapper(row[3].lower())
                subscriber_metrics[week_name] = subscriber_metrics.get(week_name, 0) +
        return subscriber_metrics

def weekly_metrics_customers(filename):
    """
    This function reads the input file with all the trips
    and returns weekly metrics for customer as a dict.
    """
    customers_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Casual','Customer']:
                week_name = week_mapper(row[3].lower())
                customers_metrics[week_name] = customers_metrics.get(week_name, 0) + 1
        return customers_metrics

def hourly_metrics(filename):
    """
    This function reads the input file with all the trips
    and returns hourly metrics as a dict.
    """
    metrics = {}
    with open(filename, 'r') as f_in:
```

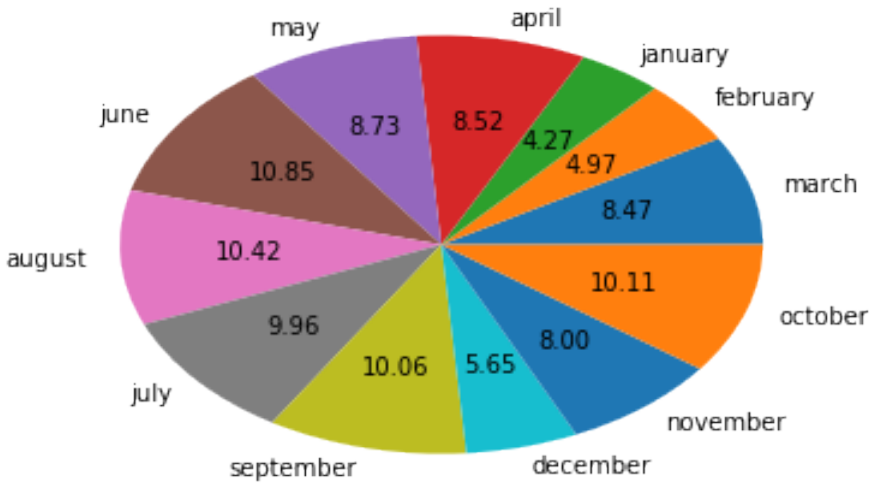
```
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            metrics[row[2]] = metrics.get(row[2], 0) + 1
        return metrics

def hourly_metrics_subscribers(filename):
    """
    This function reads the input file with all the trips
    and returns hourly metrics for subscribers as a dict.
    """
    subscriber_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Subscriber','Registered']:
                subscriber_metrics[row[2]] = subscriber_metrics.get(row[2], 0) + 1
        return subscriber_metrics

def hourly_metrics_customers(filename):
    """
    This function reads the input file with all the trips
    and returns hourly metrics  for customers as a dict.
    """
    customers_metrics = {}
    with open(filename, 'r') as f_in:
        reader = csv.reader(f_in)
        next(reader, None)
        for row in reader:
            if row[4] in ['Casual','Customer']:
                customers_metrics[row[2]] = customers_metrics.get(row[2], 0) + 1
        return customers_metrics

data_files = ['./data/Washington-2016-Summary.csv',]
for file in data_files:
    city = file.split('-')[0].split('/')[-1]
    metrics = monthly_metrics(file)
    plt.title('Monthly ridership stats for all types of users for the city {}'.format(c
    plt.pie([float(v) for v in metrics.values()],labels=[str(k) for k in metrics],
           autopct='%.2f')
    plt.show()

    subscriber_metrics = monthly_metrics_subscribers(file)
    plt.title('Monthly ridership stats for subscriber type of users for the city {}'.fo
    plt.pie([float(v) for v in subscriber_metrics.values()],labels=[str(k) for k in sub
           autopct='%.2f')
    plt.show()
```

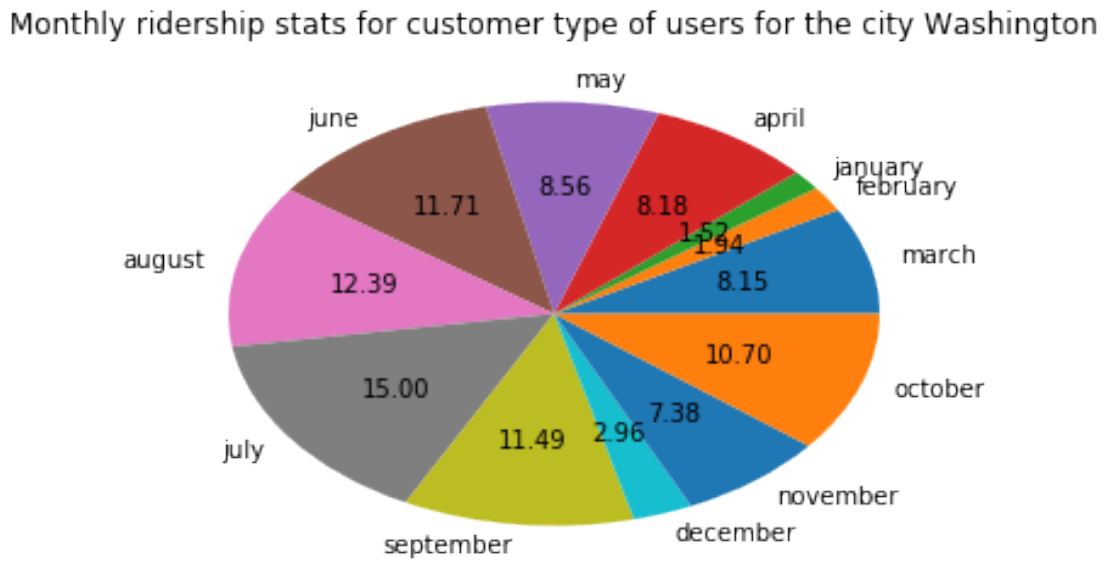
```
customer_metrics = monthly_metrics_customers(file)
plt.title('Monthly ridership stats for customer type of users for the city {}'.form
plt.pie([float(v) for v in customer_metrics.values()],labels=[str(k) for k in custo
        autopct='%.2f')
plt.show()
```

Monthly ridership stats for all types of users for the city Washington



Monthly ridership stats for subscriber type of users for the city Washington






```
In [28]: for file in data_files:
             city = file.split('-')[0].split('/')[-1]
             metrics = monthly_metrics(file)
             pprint(metrics)
             print('----------------------------')
             print(max((value,key) for (key,value) in metrics.items()))
             print('----------------------------')
             subscriber_metrics = monthly_metrics_subscribers(file)
             pprint(subscriber_metrics)
             print('----------------------------')
             print(max((value,key) for (key,value) in subscriber_metrics.items()))
             print('----------------------------')
             customer_metrics = monthly_metrics_customers(file)
             pprint(customer_metrics)
             print('----------------------------')
             print(max((value,key) for (key,value) in customer_metrics.items()))
             print('----------------------------')
             weekly_metrics = weekly_metrics(file)
             pprint(weekly_metrics)
             print('----------------------------')
             print(max((value,key) for (key,value) in weekly_metrics.items()))
             print('----------------------------')
             subscriber_metrics = weekly_metrics_subscribers(file)
             pprint(subscriber_metrics)
             print('----------------------------')
             print(max((value,key) for (key,value) in subscriber_metrics.items()))
             print('----------------------------')
```

```
            customer_metrics = weekly_metrics_customers(file)
            pprint(customer_metrics)
            print('----------------------------')
            print(max((value,key) for (key,value) in customer_metrics.items()))
            print('----------------------------')
            hourly_metrics = hourly_metrics(file)
            pprint(hourly_metrics)
            print('----------------------------')
            print(max((value,key) for (key,value) in hourly_metrics.items()))
            print('----------------------------')
            subscriber_metrics = hourly_metrics_subscribers(file)
            pprint(subscriber_metrics)
            print('----------------------------')
            print(max((value,key) for (key,value) in subscriber_metrics.items()))
            print('----------------------------')
            customer_metrics = hourly_metrics_customers(file)
            pprint(customer_metrics)
            print('----------------------------')
            print(max((value,key) for (key,value) in customer_metrics.items()))
```

```
{'april': 5602,
 'august': 7198,
 'december': 3354,
 'february': 2854,
 'january': 2434,
 'july': 7341,
 'june': 7320,
 'march': 5570,
 'may': 5768,
 'november': 5214,
 'october': 6792,
 'september': 6878}
----------------------------
(7341, 'july')
----------------------------
{'april': 4410,
 'august': 5392,
 'december': 2922,
 'february': 2571,
 'january': 2212,
 'july': 5155,
 'june': 5613,
 'march': 4382,
 'may': 4520,
 'november': 4139,
 'october': 5232,
 'september': 5204}
----------------------------
```

```
(5613, 'june')
----------------------------
{'april': 1192,
 'august': 1806,
 'december': 432,
 'february': 283,
 'january': 222,
 'july': 2186,
 'june': 1707,
 'march': 1188,
 'may': 1248,
 'november': 1075,
 'october': 1560,
 'september': 1674}
----------------------------
(2186, 'july')
----------------------------
{'weekday': 49198, 'weekend': 17127}
----------------------------
(49198, 'weekday')
----------------------------
{'weekday': 40911, 'weekend': 10841}
----------------------------
(40911, 'weekday')
----------------------------
{'weekday': 8287, 'weekend': 6286}
----------------------------
(8287, 'weekday')
----------------------------
{'0': 515,
 '1': 299,
 '10': 2608,
 '11': 2984,
 '12': 3667,
 '13': 3638,
 '14': 3620,
 '15': 3829,
 '16': 5104,
 '17': 7600,
 '18': 6111,
 '19': 4188,
 '2': 184,
 '20': 2911,
 '21': 2072,
 '22': 1458,
 '23': 858,
 '3': 89,
 '4': 67,
```

```
 '5': 407,
 '6': 1347,
 '7': 3779,
 '8': 5615,
 '9': 3375}
----------------------------
(7600, '17')
----------------------------
{'0': 385,
 '1': 224,
 '10': 1713,
 '11': 1989,
 '12': 2477,
 '13': 2395,
 '14': 2219,
 '15': 2547,
 '16': 3730,
 '17': 6225,
 '18': 5131,
 '19': 3380,
 '2': 146,
 '20': 2295,
 '21': 1640,
 '22': 1137,
 '23': 665,
 '3': 64,
 '4': 55,
 '5': 391,
 '6': 1276,
 '7': 3584,
 '8': 5256,
 '9': 2828}
----------------------------
(6225, '17')
----------------------------
{'0': 130,
 '1': 75,
 '10': 895,
 '11': 995,
 '12': 1190,
 '13': 1243,
 '14': 1401,
 '15': 1282,
 '16': 1374,
 '17': 1375,
 '18': 980,
 '19': 808,
 '2': 38,
```

```
 '20': 616,
 '21': 432,
 '22': 321,
 '23': 193,
 '3': 25,
 '4': 12,
 '5': 16,
 '6': 71,
 '7': 195,
 '8': 359,
 '9': 547}
----------------------------
(1401, '14')
```

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 would like to analyze the data regarding various garments, sales, forcasting. I would like to understand generate new fabric, designs for fashion industry

> **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 [31]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Bike_Share_Analysis.ipynb'])
```

```
Out[31]: 0
```