# Bike_Share_Analysis

January 28, 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. Bike share distributions between customers and subscribers. 2. Frequency of usage of this system by the customers and subscribers.

> **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 [1]: ## 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 [2]: def print_first_point(filename):
            """
            Objective       : This function prints and returns the first data point (second row)
                              a csv file that includes a header row.
            Input Variables : filename - given file.
            Return Value    : Tuple containing information of city and first trip.
            """
            # Approach      : We will use csv library to open csv files and the methods to fetch

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

            with open(filename,newline='') 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 = trip_reader.__next__()

                ## 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 [3]: 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

            """
            Objective        : To calculate the duration of trips in minutes.
            Input Variables  : datum - Dictionary containing the information fo first ride.
                               city - given origin city (NYC/Chicago/Washington)
            Return Value     : duration - trip duration in minutes.
            """
            # Approach       : We will use basic conversions to convert the given time into minu
```

```
        # For Washington
        if city == 'Washington':
            temp = (10**3) * (60) # temp variable is used to convert ms into minutes.
            duration = int(datum['Duration (ms)'])/temp

        # For Chacago and NYC
        else:
            duration = int(datum['tripduration'])/60

        # Returning duration in minutes
        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 [4]:
```
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
    """

    # YOUR CODE HERE
    """
    Objective        : To calculate the time of trip whihc includes information of month
    Input Variables  : datum - Dictionary containing the information fo first ride.
                       city - given origin city (NYC/Chicago/Washington)
    Return Value     : a tuple containing information of maonth, hour and day_of_week.
    """
    # Approach       : We will use datetime library and its function to convert the give
    #                  format and function from datetime library to fetch required infor

    # For NYC, the format is mm/dd/yyyy HH:MM:SS
    if city == 'NYC':
```

```
            # dt is a variable used to store the datetime object
            dt = datetime.strptime(datum['starttime'],"%m/%d/%Y %H:%M:%S")

        # For Washington, the format is mm/dd/yyyy HH:MM
        elif city == 'Washington':
            dt = datetime.strptime(datum['Start date'],"%m/%d/%Y %H:%M")

        # For Chicago, the format is mm/dd/yyyy HH:MM
        elif city == 'Chicago':
            dt = datetime.strptime(datum['starttime'],"%m/%d/%Y %H:%M")

        # Fetching the required values using datetime library and strftime function.
        day_of_week = datetime.strftime(dt,"%A")
        month = int(datetime.strftime(dt,"%m"))
        hour = int(datetime.strftime(dt,"%H"))

        # Returning the values of month, hour and day_of_week as a tuple.
        return (month, 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 [5]:
```
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
    """
    Objective        : To find the type of user.
    Input Variables  : datum - Dictionary containing the information fo first ride.
                       city - given origin city (NYC/Chicago/Washington)
    Return Value     : user_type - containing information of type of user.
    """
    # Approach       : We will use simple dictionary value accessing techniques from the
```

```
    # For Washington, the attribute name is 'Member type'.
    if city == 'Washington':
        user_type = datum['Member Type']

    # For NYC and Chicago, the attribute name is 'usertype'.
    else:
        user_type = datum['usertype']

    # Returning the user_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': 'Registered'}

for city in tests:
    assert type_of_user(example_trips[city], city) == tests[city]
```

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

```
In [6]: 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!
    """

    """
    Objective        : To condense the given data into a new file.
    Input Variables  : in_file - Given input file.
                       out_file - Output file.
                       city - given origin city (NYC/Chicago/Washington)
    Return Value     : None.
    """
    # Approach       : We will use functionalities from csv library to condense the data
    #                  into new file.

    # Opening the files in required formats.
```

```
    with open(out_file, 'w',newline= '') 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)

        # writeheader() writes the ehader in the output file.
        trip_writer.writeheader()

        ## TODO: set up csv DictReader object ##
        trip_reader = csv.DictReader(f_in)

        # collect data from and process each row using a for loop.
        for row in trip_reader:
            # set up a dictionary to hold the values for the cleaned and trimmed
            # data point
            new_point = {}
            month, hour, day_of_week = time_of_trip(row,city) # Use of time_of_trip(...)

            ## TODO: use the helper functions to get the cleaned data from  ##
            ## the original data dictionaries.                              ##
            # Setting up the dictionary with appropriate values.
            new_point = {'duration': duration_in_mins(row,city),'month': month,'hour': h
                         'day_of_week':day_of_week,'user_type':type_of_user(row,city)}

            ## Note that the keys for the new_point dictionary should match ##
            ## the column names set in the DictWriter object above.         ##


            ## TODO: write the processed information to the output file.     ##

            # Using the csv libary, we write into the files.
            # Setting up DictWriter Object.
            writer = csv.DictWriter(f_out,fieldnames = out_colnames)

            # Write a row.
            writer.writerow(new_point)
            ## see https://docs.python.org/3/library/csv.html#writer-objects ##
```

In [7]:
```
# Run this cell to check your work
city_info = {'Washington': {'in_file': './data/Washington-CapitalBikeshare-2016.csv',
                            'out_file': './data/Washington-2016-Summary.csv'},
             'Chicago': {'in_file': './data/Chicago-Divvy-2016.csv',
                         'out_file': './data/Chicago-2016-Summary.csv'},
             'NYC': {'in_file': './data/NYC-CitiBike-2016.csv',
                     'out_file': './data/NYC-2016-Summary.csv'}}
```

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

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

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

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

> **Tip**: If you save a jupyter Notebook, the output from running code blocks will also be saved. However, the state of your workspace will be reset once a new session is started. Make sure that you run all of the necessary code blocks from your previous session to reestablish variables and functions before picking up where you 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**: NYC has highest number of trips (276798). NYC has the highest proportion of trips made by subscribers (88.83% of Total). Chicago has the highest proportion of trips made by short-term customers (23.77% of Total).

In [8]:
```
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.
    """
    """
    Objective        : To calculate the number of trips made by customers and subscriber
    Input Variables  : filename - given file.
    Return Value     : Tuple containing information of number of customers, number of su
    """
    # Approach       : We will use csv module to read the file and distionary value acce
    #                  check for customers and subscribers and counting them.

    # Opening the file in required mode.
    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': # For NYC and Chicago
                n_subscribers += 1
            elif row['user_type'] == 'Registered': # For Washington Data
                n_subscribers += 1
            elif row['user_type'] == 'Customer':  # For NYC and Chicago
                    n_customers += 1
            else: # For Washinton Data
                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 [9]:
```
## 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.      ##

# summary_files ad a list of all files to be iterated.
summary_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv',

# for loop to iterate through each file.
for file in summary_files:
```

```
            # Getting the required values.
            n_subscribers, n_customers, n_total = number_of_trips(file)

            ## Ratio of customers.
            ratio_customer = n_customers/n_total

            # Ratio of subscribers.
            ratio_subscriber = n_subscribers/n_total

            # Printing the information for analysis purpose.
            print(file)
            print('Total',n_total)
            print('Customer Proportion',ratio_customer)
            print('Subscriber Proportion',ratio_subscriber)
            print('\n')
```

```
./data/Washington-2016-Summary.csv
Total 66326
Customer Proportion 0.21971775774206193
Subscriber Proportion 0.7802822422579381


./data/Chicago-2016-Summary.csv
Total 72131
Customer Proportion 0.23774798630269925
Subscriber Proportion 0.7622520136973008


./data/NYC-2016-Summary.csv
Total 276798
Customer Proportion 0.11164098006488486
Subscriber Proportion 0.8883590199351151
```

> **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**:
Washington
Average time: 18.93
Proportion greater 30 minutes: 10.84 %
Chicago
Average time: 16.56
Proportion greater 30 minutes: 8.33 %
NYC
Average time: 15.81
Proportion greater 30 minutes: 7.3 %

In [10]:
```
## 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 average_trip_length(filename):
    """
    Objective        : To compute the average trip length by customers and subscribers
                       whose trips are longer than 30 minutes.
    Input Variables  : filename - given file.
    Return Value     : A list containing informtion of average trip length and proporti
                       is greater than 30 minutes.
    """
    # Approach       : We will use csv library to read the file and for loop to iterate
    #                  and computing the required counts.

    # opening file in required format.
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # initialize count variables
        total_time = 0
        count = 0
        count_greater_30 = 0

        # Iterating the file row by row using a for loop.
        for row in reader:

            # Adding the total time every row.
            total_time += float(row['duration'])
            count += 1

            # Counter for trips having duration greater than 30 minutes.
```

```
                    if float(row['duration']) > 30.0:
                        count_greater_30 += 1

                # Calculation of average time by variable average_length.
                average_length = total_time / count

                # Calculation of proportion greater than 30.
                proportion_greater_30 = count_greater_30 / count

                # Returning the values in form of list.
                return [average_length , proportion_greater_30]
```

```
In [11]: # Files as elements of list.
         summary_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv'

         # For loop for iterating trough all files.
         for file in summary_files:

             # average_proportion is list having values average_length and proportion_greater_30
             average_proportion = average_trip_length(file)
             print('----------------------------------------------------------------')
             print('Analysing',file,'file...')

             # Printing average length which is at 0th index of average_proportion list.
             print('Average time:',round(average_proportion[0],2))

             # Printing proportion gretare than 30 which is at 1th index of average_proportion l
             print('Proportion greater 30 minutes:',round(average_proportion[1]*100,2),'%')
             print('\n')
```

```
----------------------------------------------------------------
Analysing ./data/Washington-2016-Summary.csv file...
Average time: 18.93
Proportion greater 30 minutes: 10.84 %


----------------------------------------------------------------
Analysing ./data/Chicago-2016-Summary.csv file...
Average time: 16.56
Proportion greater 30 minutes: 8.33 %


----------------------------------------------------------------
Analysing ./data/NYC-2016-Summary.csv file...
Average time: 15.81
Proportion greater 30 minutes: 7.3 %
```

**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**:

If we choose Washington City, we found the following results:
Average time Customer: 41.68
Average time Subscriber: 12.53

This clearly shows that Customer takes longer rides on average.

In [12]:
```
## 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?                                                ##

# find_longer is funcion telling the average subscriber trip
# and average customer trip duration.
def find_longer(filename):
    """
    Objective        : To compute the average trip duration of customers and subscriber
    Input Variables  : filename - given file.
    Return Value     : List containing information of average trip duration of customer
    """
    # Approach       : We will use csv module to read the file and dictionary value acc
    #                  check for customers and subscribers and counting the duration.

    # Opening the file in required mode.
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # initialize count variables
        total_time_customer = 0
        total_time_subscriber = 0

        # For loop for iterating the file row by row.
        for row in reader:
            if filename != './data/Washington-2016-Summary.csv':
                if row['user_type'] == 'Subscriber':
                    total_time_subscriber += float(row['duration'])
                else:
                    total_time_customer += float(row['duration'])
```

```
                    else:
                        if row['user_type'] == 'Registered':
                            total_time_subscriber += float(row['duration'])

                        else:
                            total_time_customer += float(row['duration'])

                # Getting use of helper function (number_of_trips(...)) to get the required val
                # and total count.
                count_subscriber, count_customer, count_total = number_of_trips(filename)

                # Average length of Customer.
                average_length_customer = total_time_customer / count_customer

                # Average length of Subscriber.
                average_length_subscriber = total_time_subscriber / count_subscriber

                # Returning average length of customer and subcriber as a list.
                return [average_length_customer, average_length_subscriber]
```

```
In [13]: # Summary files as element of list.
         summary_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv'

         # For loop for iterating all files.
         for file in summary_files:

             # average_time is a list returned from find_longer function.
             average_time = find_longer(file)
             print('-----------------------------------------------------------------')
             print('Analysing',file,'file...')

             # Average customer time, at 0th index of average_time list.
             print('Average time Customer:',round(average_time[0],2))

             # Average subscriber time, at 1th index of average_time list.
             print('Average time Subscriber:',round(average_time[1],2))
             print('\n')
```

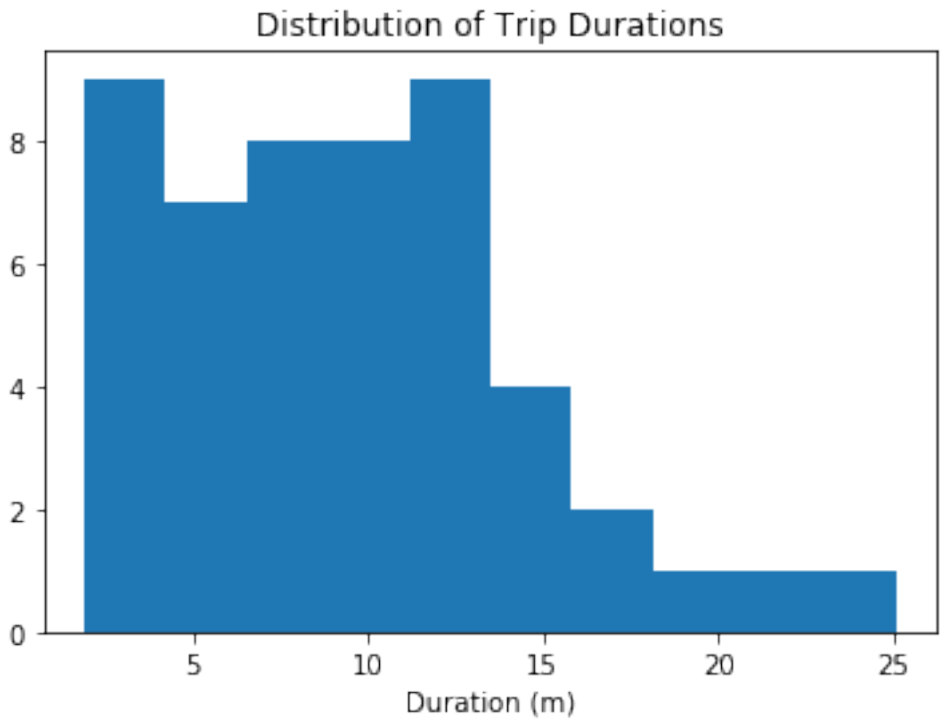
```
-----------------------------------------------------------------
Analysing ./data/Washington-2016-Summary.csv file...
Average time Customer: 41.68
Average time Subscriber: 12.53
```

```
----------------------------------------------------------------------
Analysing ./data/Chicago-2016-Summary.csv file...
Average time Customer: 30.98
Average time Subscriber: 12.07


----------------------------------------------------------------------
Analysing ./data/NYC-2016-Summary.csv file...
Average time Customer: 32.78
Average time Subscriber: 13.68
```

### 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 [14]:
```
# Given example for visualization.

# 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()
```

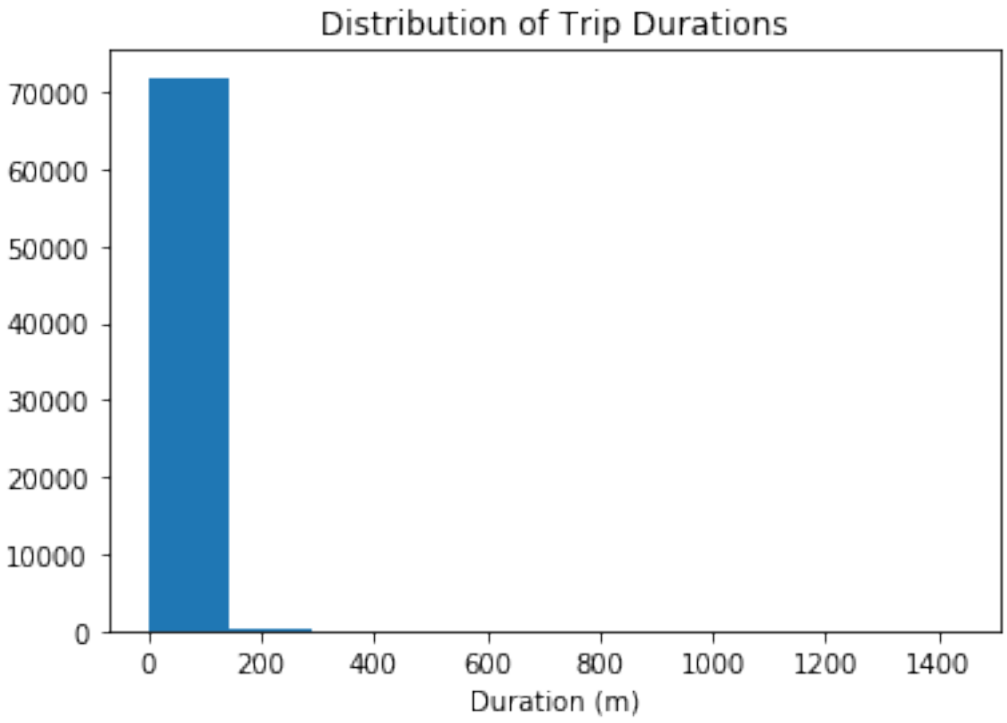



In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. 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 [15]: ## 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

         # Initialising a list - empty.
         trip_time = []

         # We are analysing Chicago Data
         with open('./data/Chicago-2016-Summary.csv', 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)
```

```
        # Iterating the file row by row.
        for row in reader:
            trip_time.append(round(float(row['duration']),2))

# Plotting the histogram.
plt.hist(trip_time)
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**:

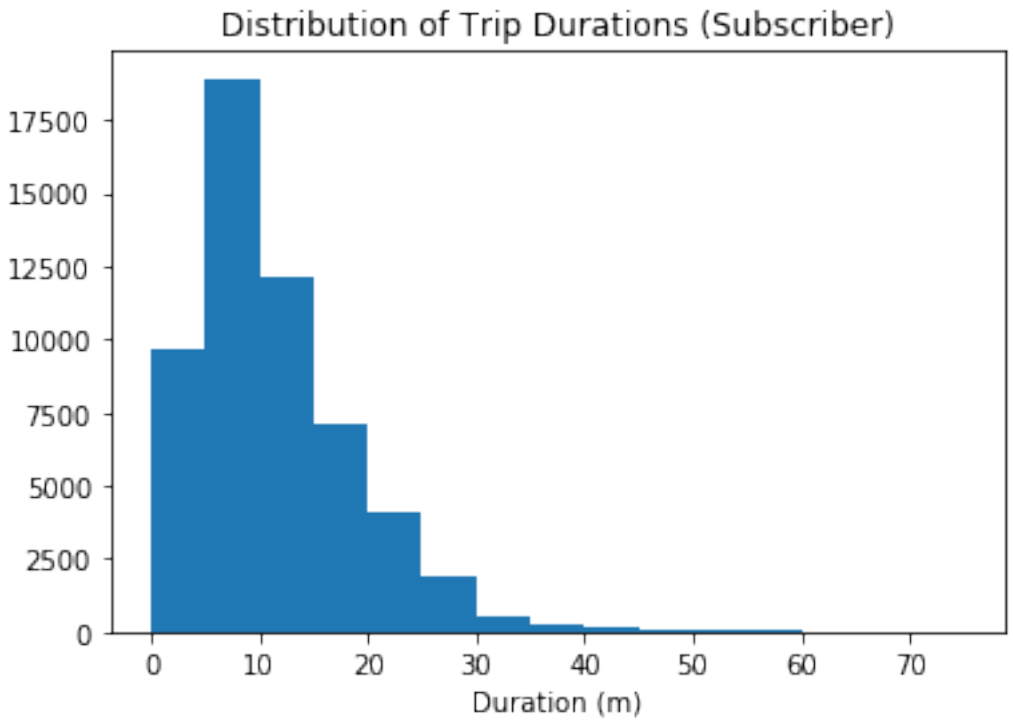
We have chosen Chicago city for our analysis. We found the following conslusions: In the subscriber histogram, we found that the **peak** is near 10 minutes. On the other hand,in customer

histogram, the **peak** is near 25 minutes (Approx Mode Value). This visualization helps her to conclude that mean/average of Subsriber is near 12 minutes which has been confirmed from above problems and in case of Customer, it is near 30 minutes, which is again confirmed with refernece to previous problem.

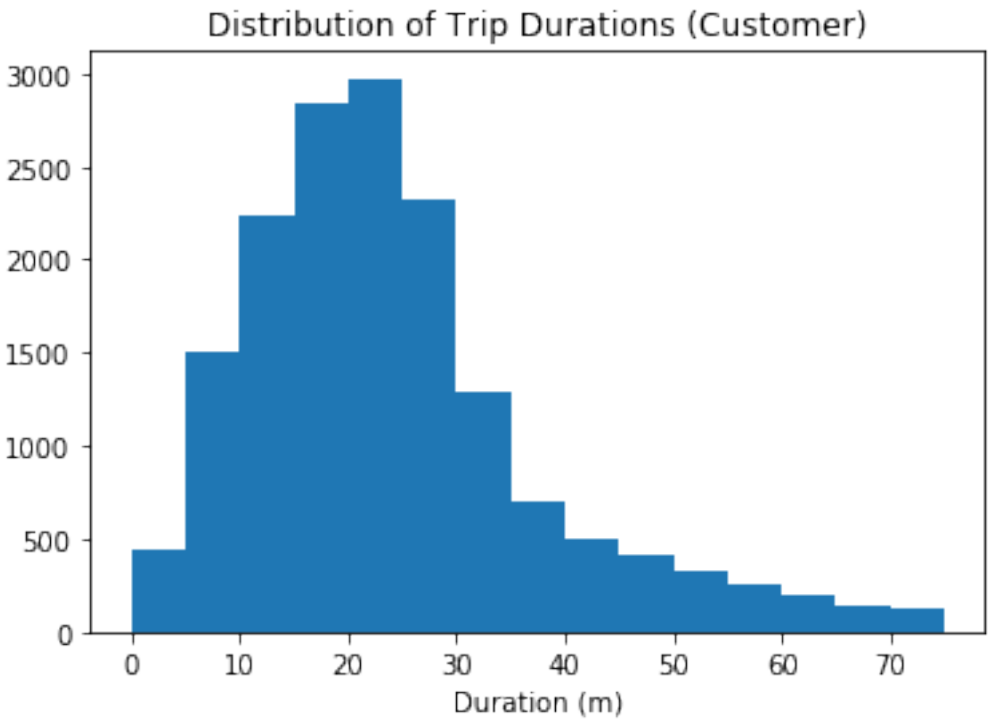
The shape of each distribution is as follows: * Subscriber's plot is right skewed i.e, positively skewed data is there in case of subscribers. * Customer's plot is not as much skewed as of subscriber, it is following nearly normal distribution.

In [16]:
```
## 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

# 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

# Initialising lists - empty.
trip_time_sub = []
trip_time_cus = []

# Analysing Chicago File
with open('./data/Chicago-2016-Summary.csv', 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # Iterating the files row by row.
        for row in reader:
            if row['user_type'] == 'Subscriber' and float(row['duration']) < 75:
                trip_time_sub.append(round(float(row['duration']),2))
            elif row['user_type'] == 'Customer' and float(row['duration']) < 75:
                trip_time_cus.append(round(float(row['duration']),2))
```

In [17]:
```
# Plotting histogram for subscribers.
plt.hist(trip_time_sub,bins = [x for x in range(0,80,5)])
plt.title('Distribution of Trip Durations (Subscriber)')
plt.xlabel('Duration (m)')
plt.show()
```




In [18]:
```
# Plotting histogram for customers.
plt.hist(trip_time_cus,bins = [x for x in range(0,80,5)])
plt.title('Distribution of Trip Durations (Customer)')
plt.xlabel('Duration (m)')
plt.show()
```




## Performing Your Own Analysis

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

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

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

**Question 6**: Continue the investigation by exploring another question that could be answered by the data available. Document the question you want to explore below. Your investigation

should involve at least two variables and should compare at least two groups. You should also use at least one visualization as part of your explorations.

**Answer**:

From the observation of three cities, we found the following results: Customers of Chicago has used the system most often (Around 49.8% of total trips by customers are made on Saturday and Sunday). Moreover, customers of other cities have used the system more on weekends as compared to subscribers. Subscribers are not signifantly active on specific days but customers are active on weekends.

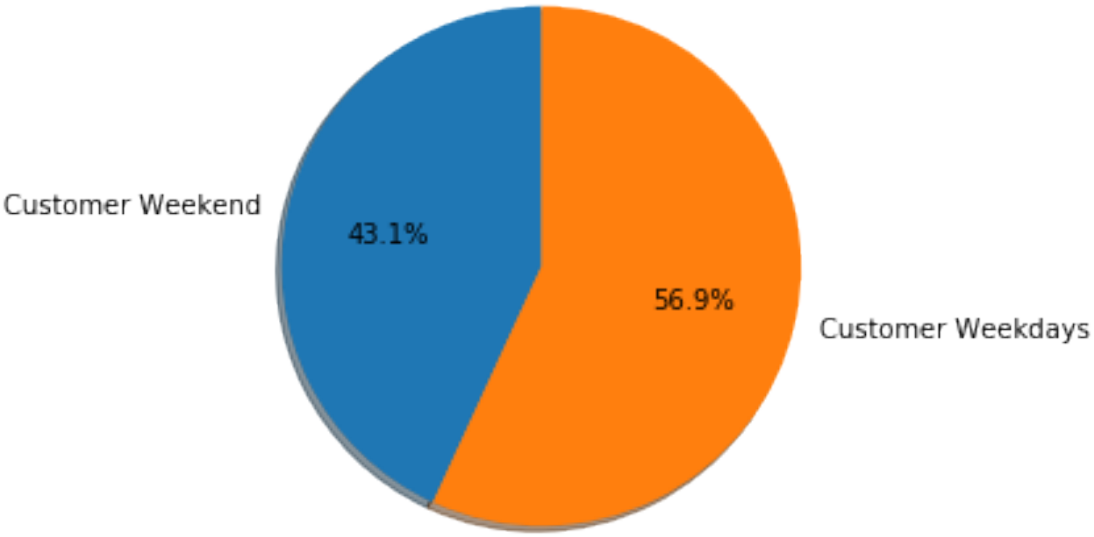
We have used Pie Chart to show the difference more clearly.

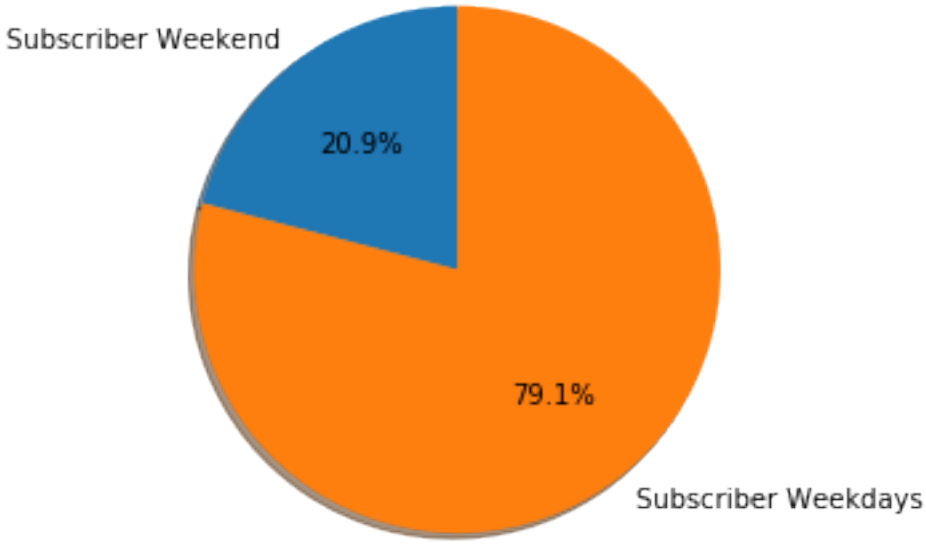
In [19]:
```
## Use this and additional cells to continue to explore the dataset. ##
## Once you have performed your exploration, document your findings  ##
## in the Markdown cell above.     ##

## 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.                                       ##

# busy_days function will check if weekends or weekdays are more busy.
def busy_days(filename):
    """
    Objective        : To find wheher customers or subscribers have used the system mor
                       and weekdays.
    Input Variables  : filename - given file.
    Return Value     : Tuple containing information of number of customers, number of s
                       weekdays.
    """
    # Approach       : We will use csv module to read the file and dictionary value acc
    #                  check for customers and subscribers and counting them.

    # Opening the file in required mode.
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # initialize count variables
        n_customer_weekend = 0
        n_customer_weekdays = 0

        n_subscriber_weekend = 0
        n_subscriber_weekdays = 0

        # Iterating the file row by row.
        for row in reader:

            if filename != './data/Washington-2016-Summary.csv':
                if (row['user_type'] == 'Subscriber') and (row['day_of_week'] == 'Satur
                    n_subscriber_weekend += 1
```

```
                    elif (row['user_type'] == 'Customer') and (row['day_of_week'] == 'Satur
                        n_customer_weekend += 1
                    elif (row['user_type'] == 'Subscriber') and not(row['day_of_week'] == '
                        n_subscriber_weekdays += 1
                    elif (row['user_type'] == 'Customer') and not(row['day_of_week'] == 'Sa
                        n_customer_weekdays += 1

                # For Washington.
                else:
                    if (row['user_type'] == 'Registered') and (row['day_of_week'] == 'Satur
                        n_subscriber_weekend += 1
                    elif (row['user_type'] == 'Casual') and (row['day_of_week'] == 'Saturda
                        n_customer_weekend += 1
                    elif (row['user_type'] == 'Registered') and not(row['day_of_week'] == '
                        n_subscriber_weekdays += 1
                    elif (row['user_type'] == 'Casual') and not(row['day_of_week'] == 'Satu
                        n_customer_weekdays += 1

            # Returning the values as a tuple.
            return (n_customer_weekend, n_customer_weekdays,
                    n_subscriber_weekend,n_subscriber_weekdays)
```

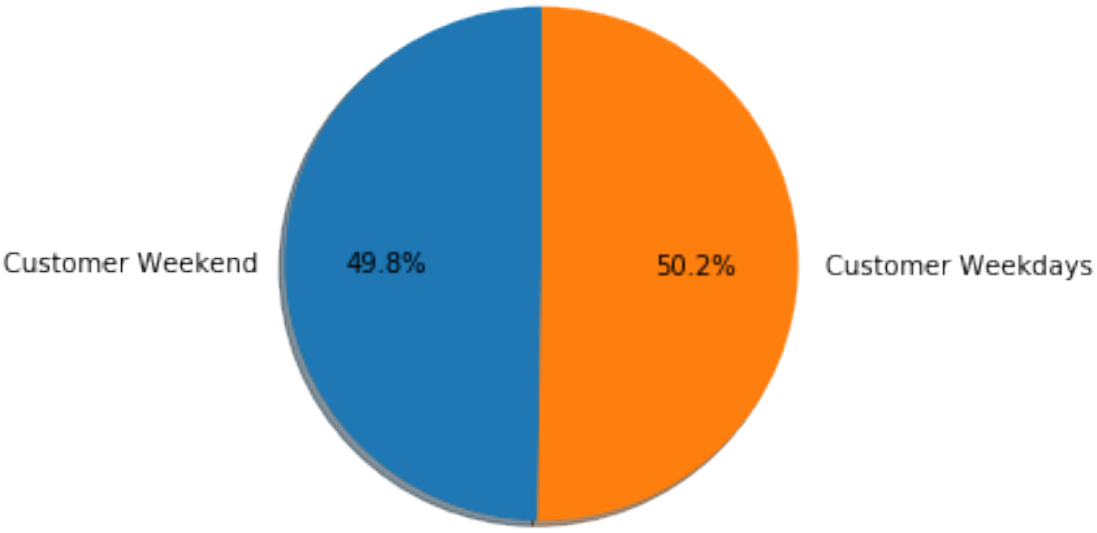
In [20]:
```
# load library
import matplotlib.pyplot as plt

# 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

# Pie Chart Visualization
# As part of our analysis, we will be using pie charts to demonstrate the results in be

# Labels of pie charts.
labels_cus = ['Customer Weekend','Customer Weekdays']
labels_sub = ['Subscriber Weekend','Subscriber Weekdays']

# Summary files as element of list.
summary_files = ['./data/Washington-2016-Summary.csv','./data/Chicago-2016-Summary.csv'

# Iterating the files using for loop.
for file in summary_files:

    print('-----------------------------------------------------------------------------
    print('Analysing ',file,'file...')

    # Initializing count variables.
```

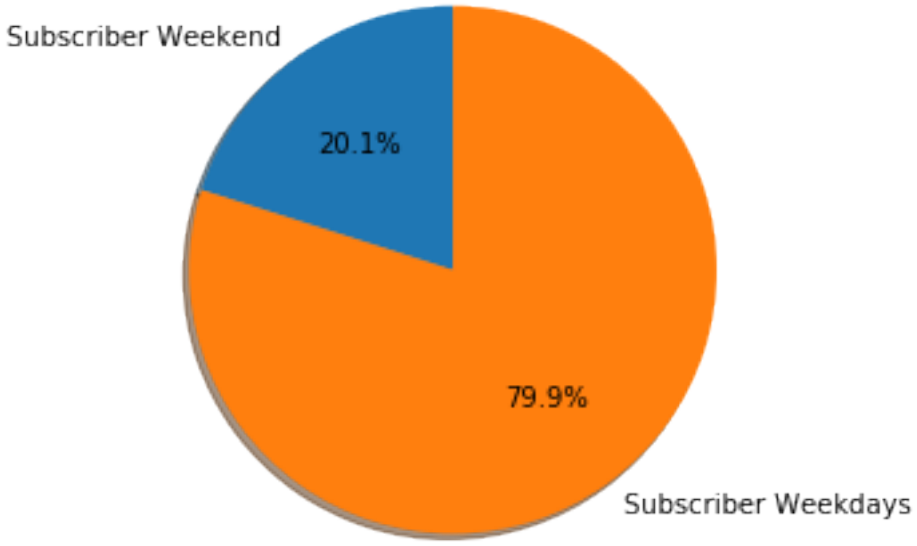
```
            n_customer_weekend = 0
            n_customer_weekdays = 0
            n_subscriber_weekend = 0
            n_subscriber_weekdays = 0

            # Getting values from busy_days(...) function.
            (n_customer_weekend, n_customer_weekdays,
            n_subscriber_weekend,n_subscriber_weekdays) = busy_days(file)

            # Calculating Ratio of Customer who used the system on Weekend.
            cus_weekend_ratio = n_customer_weekend / (n_customer_weekend + n_customer_weekdays)

            # Calculating Ratio of Customer who used the system on Weekdays.
            cus_weekdays_ratio = n_customer_weekdays / (n_customer_weekend + n_customer_weekday

            # Calculating Ratio of Subscriber who used the system on Weekend.
            sub_weekend_ratio = n_subscriber_weekend / (n_subscriber_weekend + n_subscriber_wee

            # Calculating Ratio of Subscriber who used the system on Weekdays.
            sub_weekdays_ratio = n_subscriber_weekdays / (n_subscriber_weekend + n_subscriber_w

            # Visualization though pie chart.
            sizes_cus = [cus_weekend_ratio*100,cus_weekdays_ratio*100]
            sizes_sub = [sub_weekend_ratio*100,sub_weekdays_ratio*100]
            explode = (0, 0)
            fig1, ax1 = plt.subplots()
            fig2, ax2 = plt.subplots()

            ax1.pie(sizes_cus, explode=explode, labels = labels_cus,autopct='%1.1f%%' ,shadow =
            ax2.pie(sizes_sub, explode=explode, labels = labels_sub,autopct='%1.1f%%' ,shadow =

            ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
            ax2.axis('equal')

            # Showing the final visualization.
            plt.show()


--------------------------------------------------------------------------------
Analysing  ./data/Washington-2016-Summary.csv file...
```



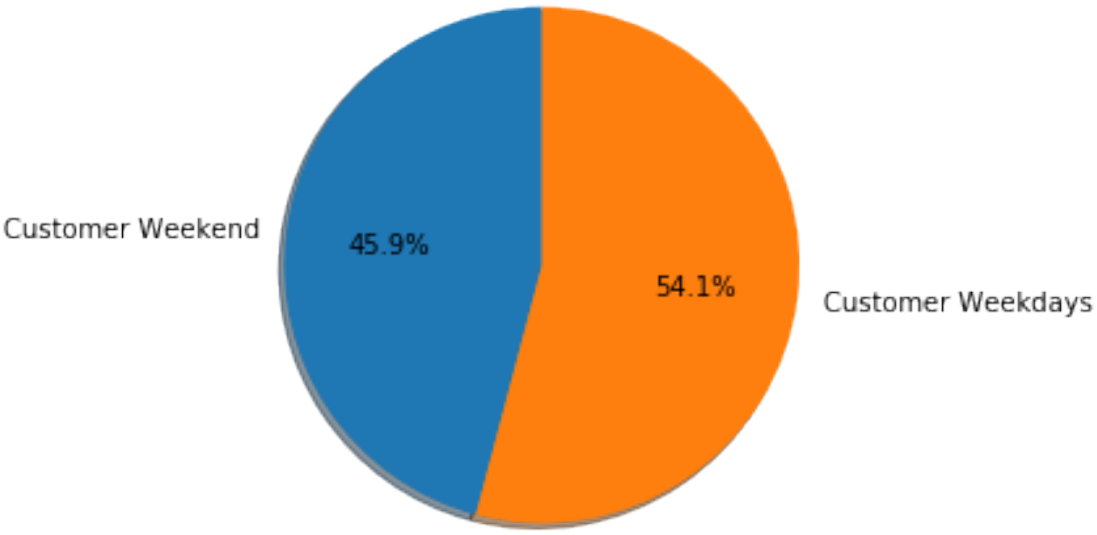



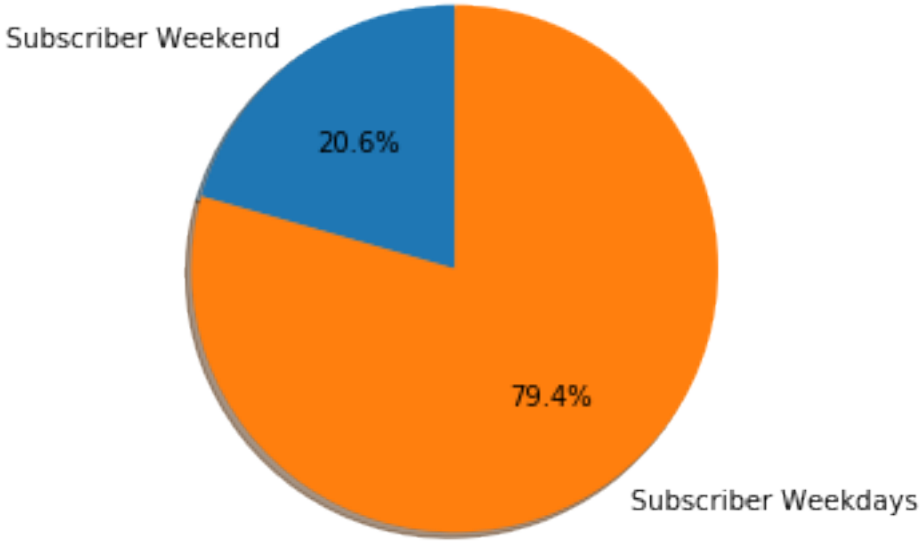

```
----------------------------------------------------------------------------------
Analysing  ./data/Chicago-2016-Summary.csv file...
```




```
--------------------------------------------------------------------------------
Analysing  ./data/NYC-2016-Summary.csv file...
```




## 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**: Bike Share Data provides almost information to deduce the observations and to find the answers for the questions asked. However, we can apply the techniques of data analysis for more broad analysis and conclusions, for example, we can create a model which will predict the class from a given set of data and the attributes values. Statistical tests includes hypothesis testing requires more data and hypothesis.
With a rich data in hand, we can use the techniques of data analsysis to predict and conclude more informations which is however not possible by just seeing the data. As a field of interest, I would like to apply the techniques in field of data mining, machine learning and AI as these areas requires a lot of data analsysis. From this field, I would like to learn more data mining techniques and data analysis more deeper, able to dig the data more deeper, able to conclude more information, improve the knowledge discovery from data and make the best use of data. > **Tip**: If we want to share the results of our analysis with others, we aren't limited to giving them a copy of the jupyter Notebook (.ipynb) file. We can also export the Notebook output in a form that can be opened even for those without Python installed. From the **File** menu in the upper left, go to the **Download as** submenu. You can then choose a different format that can be viewed more generally, such as HTML (.html) or PDF (.pdf). You may need additional packages or software to perform these exports.

> If you are working on this project via the Project Notebook page in the classroom, you can also submit this project directly from the workspace. **Before you do that**, you should save an HTML copy of the completed project to the workspace by running the code cell below. If it worked correctly, the output code should be a 0, and if you click on the jupyter icon in the upper left, you should see your .html document in the workspace directory. Alternatively, you can download the .html copy of your report following the steps in the previous paragraph, then *upload* the report to the directory (by clicking the jupyter icon).
>
> Either way, once you've gotten the .html report in your workspace, you can complete your submission by clicking on the "Submit Project" button to the lower-right hand side of the workspace.

## 1.2 Resources

- Python Documentation
- Concept revision from tutorials point

```
In [21]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Bike_Share_Analysis.ipynb'])
```

```
Out[21]: 0
```