Bike_Share_Analysis

March 8, 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:

From Business Perspective:

1. What type of customers are using the bike share service more? Regular or casual ones.
2. Cities in which the bike share is being used more? and if the factors motivating more usage can be added to areas where it is being used less.

From Customer Perspective:

1. Is the bike readily available in given area and in given time?
2. Do frequent usage result in any discount?

> **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 [27]: ## 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.
         import calendar # useful for converting numbers to months.
```

```
In [28]: 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 [29]: 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
             # Washington is in milliseconds so process it separately.
             if city == 'Washington':
```

```
            duration = (float(datum['Duration (ms)']))/1000
        else:
            duration = float(datum['tripduration'])

        #Convert to minutes and return.
        duration /= 60.0
        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 [30]:
```
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
    # If city is NYC, there is seconds field so process is separately
    if city == 'NYC':
        start_time = datum['starttime']
        #format of string to be used on datetime functions later
        time_format = '%m/%d/%Y %H:%M:%S'
    else:
        time_format = '%m/%d/%Y %H:%M'
        if 'starttime' in datum.keys():
            start_time = datum['starttime']
        else:
            start_time = datum['Start date']

    # Convert the date and time string into datetime object using striptime
    date_time = datetime.strptime(start_time, time_format)
```

```
        #Extract the month and strip the zero if present to get month as 1,2, 3
        month = int(date_time.strftime('%m').lstrip('0'))

        #Extract the hour information and convert to numeric form.
        hour = int(date_time.strftime('%H'))

        #Extract day of the week.
        day_of_week =date_time.strftime('%A')

        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 [31]:
```
def type_of_user(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (datum) and
    its origin city (city) and returns the type of system user that made the
    trip.

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

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

    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 [32]: 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()

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

                 # collect data from and process each row
                 for row in trip_reader:
                     # set up a dictionary to hold the values for the cleaned and trimmed
                     # data point
                     new_point = {}

                     ## TODO: use the helper functions to get the cleaned data from  ##
                     ## the original data dictionaries.                              ##
                     ## Note that the keys for the new_point dictionary should match ##
                     ## the column names set in the DictWriter object above.         ##
                     new_point['duration'] = duration_in_mins(row, city)
                     new_point['month'], new_point['hour'], new_point['day_of_week'] = time_of_t
                     new_point['user_type'] = type_of_user(row, city)


                     ## TODO: write the processed information to the output file.     ##
                     ## see https://docs.python.org/3/library/csv.html#writer-objects ##
                     trip_writer.writerow(new_point)
```

```
In [33]: # 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', 'Subscriber')])

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

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

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

Exploratory Data Analysis

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

Statistics

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

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

Answer:

```
NYC has the highest number of total trips.
NYC has the highest proportion of trips made by subscribers
Chicago has the highest proportion of trips made by short-term customers
```

```
In [34]: 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 [35]: def proportion(n_subset, n_total):
             """
             This function takes two inputs and computes the proportion of first input to the se
             The proportion is converted to float before returning.
             """
             return float(n_subset/n_total)
```

```
In [36]: ## 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.      ##
         def get_highest_trip_data(data_files):
             """
             This function  takes a list of file names and returns the name of the city with:
```

```
    1. Highest number of trips,
    2. Highest proportion of trips made by subscribers and
    3. Highest proportion of trips made by short-term customers
    """

    #setup empty dictionaries to hold the trip data for types of customers and proporti
    subscriber_data ={}
    customer_data ={}
    total_data ={}
    subscriber_proportion ={}
    customer_proportion={}

    #Iterate through data for each city and store the value of total number of trips,
    # number of trips by customers and number of trips by subscribers for each city in
    # with city as the key. Calculate proportion using the proportion function.
    for data_file in data_files:
        #get the city from file name name which will be the key of dictionaries.
        city = data_file.split('-')[0].split('/')[-1]

        #get the number of trips data
        subscriber_data[city], customer_data[city], total_data[city] = number_of_trips(

        #get the subscriber proportion data
        subscriber_proportion[city] = proportion(subscriber_data[city], total_data[city

        #get the customer proportion data
        customer_proportion[city] = proportion(customer_data[city], total_data[city])

    #find the city with max number of trips
    max_trips_city = max(total_data, key=total_data.get)

    #find the city with max proportion of trips made by subscribers
    max_subscribers_city = max(subscriber_proportion, key=subscriber_proportion.get)

    #find the city with max proportion of trips made by customers
    max_customers_city = max(customer_proportion, key=customer_proportion.get)

    return (max_trips_city, max_subscribers_city, max_customers_city )
```

In [37]:
```
#Run the below code to get answer to question 4a
data_files = ['./data/Washington-2016-Summary.csv',
              './data/Chicago-2016-Summary.csv',
              './data/NYC-2016-Summary.csv']
highest_trip_data = {}

highest_trip_data['max_trips'], \
highest_trip_data['max_subscriber_prop'], \
highest_trip_data['max_customer_prop'] = get_highest_trip_data(data_files)
```

```
        print('{} has the highest number of total trips.'.format(highest_trip_data['max_trips']

        print('{} has the highest proportion of trips made '
              'by subscribers'.format(highest_trip_data['max_subscriber_prop']))

        print('{} has the highest proportion of trips '
              'made by short-term customers'.format(highest_trip_data['max_customer_prop']))
```

```
NYC has the highest number of total trips.
NYC has the highest proportion of trips made by subscribers
Chicago has the highest proportion of trips made by short-term customers
```

> **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:

```
The average trip length for Washington is 18.93287355913721 minutes
The proportion of rides made in Washington that are longer than 30 minutes is:10.83888671109369

The average trip length for Chicago is 16.563629368787335 minutes
The proportion of rides made in Chicago that are longer than 30 minutes is:8.332062497400562 per

The average trip length for NYC is 15.81259299802294 minutes
The proportion of rides made in NYC that are longer than 30 minutes is:7.3024371563378345 percen

The average trip length for BayArea is 14.038656929671422 minutes
The proportion of rides made in BayArea that are longer than 30 minutes is:3.5243689474519764 pe
```

```
In [38]: ## 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 trip_time_details(filename, duration=30.0):
             """
```

```
    This function reads in a file(filename) with trip data and
    time duration(duration) and computes:
    1.The average trip length and
    2.The proportion of rides made in the city which are longer than
      the passed in duration
    """

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

        # initialize counters
        total_trip_time = 0.0
        trip_count = 0
        longer_trips = 0
        longer_trips_proportion = 0.0
        average_trip_time = 0.0

        #iterate over each row in csv and add the trip time and also
        #keep track of number of trips.
        for row in reader:
            trip_time = float(row['duration'])
            total_trip_time += trip_time
            trip_count += 1

            # Longer trips greater than 'duration'
            if trip_time > duration:
                longer_trips += 1

        #Average trip time
        average_trip_time = total_trip_time/trip_count

        #proportion of longer trips expressed as percent.
        longer_trips_proportion =((longer_trips/trip_count) *100)

        return(average_trip_time, longer_trips_proportion)
```

In [39]:
```
#Run the below code to get answer to question 4b
data_files = ['./data/Washington-2016-Summary.csv',
              './data/Chicago-2016-Summary.csv',
              './data/NYC-2016-Summary.csv',
             './examples/BayArea-Y3-Summary.csv']

# To print the duration and can also be passed as optional
# input to trip_time_details function
duration = 30
```

```
        #for all the csv files available
        for data_file in data_files:

            #Extract city name to print the city.
            city = data_file.split('-')[0].split('/')[-1]

            #Compute the values
            avg_trip_len, long_trip_prop = trip_time_details(data_file)

            #Print
            print('The average trip length for {} is {} minutes'.format(city, avg_trip_len))
            print('The proportion of rides made in {} that are longer '
                  'than {} minutes is:{} percent\n'.format(city, duration, long_trip_prop))
```

```
The average trip length for Washington is 18.93287355913721 minutes
The proportion of rides made in Washington that are longer than 30 minutes is:10.83888671109369

The average trip length for Chicago is 16.563629368787335 minutes
The proportion of rides made in Chicago that are longer than 30 minutes is:8.332062497400562 per

The average trip length for NYC is 15.81259299802294 minutes
The proportion of rides made in NYC that are longer than 30 minutes is:7.3024371563378345 percen

The average trip length for BayArea is 14.038656929671422 minutes
The proportion of rides made in BayArea that are longer than 30 minutes is:3.5243689474519764 pe
```

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

Answer:

```
In NYC the Customers take longer rides on an average.
The Subscriber average is 13.680790523907177 minutes
The Customer average is 32.77595139473187 minutes
```

```
In [40]: ## 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 trip_length_by_usertype(filename):
             """
             This function reads in a file with trip data for a city and returns
             the average duration of trips made by subscribers and customers and
             the type of user whose average duration is higher.
```

```
    """
    # Extract city name to print the city.
    city = data_file.split('-')[0].split('/')[-1]

    with open(filename, 'r') as f_in:
        # Set up csv reader object
        reader = csv.DictReader(f_in)
        # Set up empty lists to collect the duration based on the user type.
        subscriber_total_trip_time = []
        customer_total_trip_time = []

        # Iterate over the file row by row
        for row in reader:
            if row['user_type'] == 'Subscriber':
                subscriber_total_trip_time.append(float(row['duration']))
            else:
                customer_total_trip_time.append(float(row['duration']))

        #calculate the average times based on the user type.
        subscriber_avg_trip_time = sum(subscriber_total_trip_time)/len(subscriber_total
        customer_avg_trip_time = sum(customer_total_trip_time)/len(customer_total_trip_

        #Compute which user type average is more
        if subscriber_avg_trip_time > customer_avg_trip_time:
            long_ride_user_type = 'Subscribers'
        else:
            long_ride_user_type = 'Customers'

        return(long_ride_user_type, subscriber_avg_trip_time, customer_avg_trip_time )
```

In [41]:
```
#Run the below code to get answer to question 4c
data_files = ['./data/NYC-2016-Summary.csv']

#Iterate over the files in data_files
for data_file in data_files:
    #Extract city name to print the city.
    city = data_file.split('-')[0].split('/')[-1]

    #Call the computations
    long_user, sub_avg_time, cust_avg_time = trip_length_by_usertype(data_file)

    print('In {} the {} take longer rides on an average.\n'
          'The Subscriber average is {} minutes\n'
          'The Customer average is {} minutes '.format(city,
                                                       long_user,
                                                       sub_avg_time,
                                                       cust_avg_time))
```

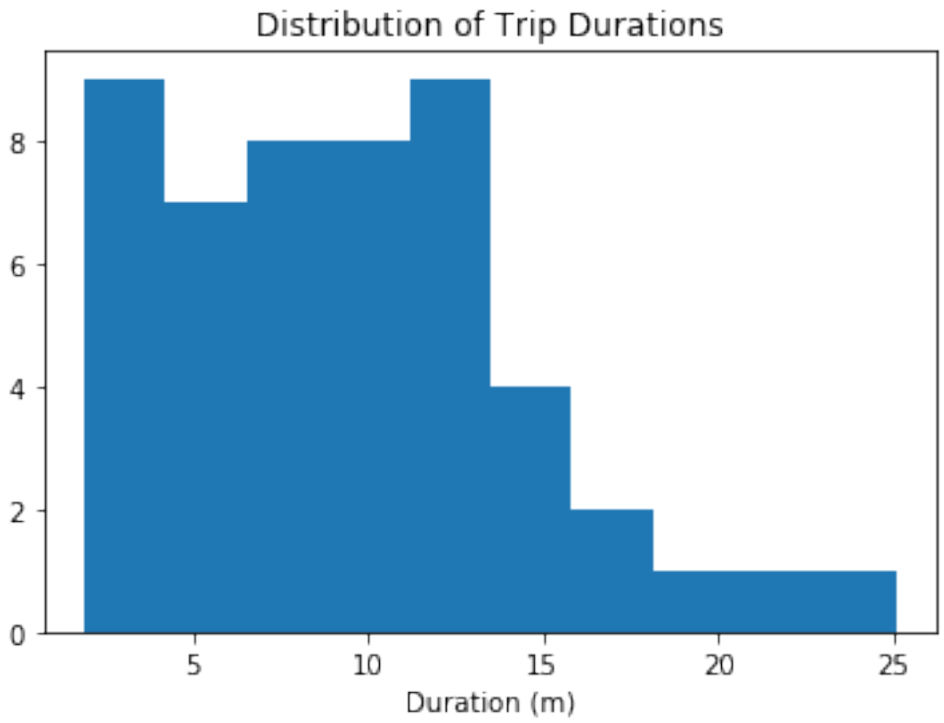
```
In NYC the Customers take longer rides on an average.
```

```
The Subscriber average is 13.680790523907177 minutes
The Customer average is 32.77595139473187 minutes
```

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 [42]: # 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.

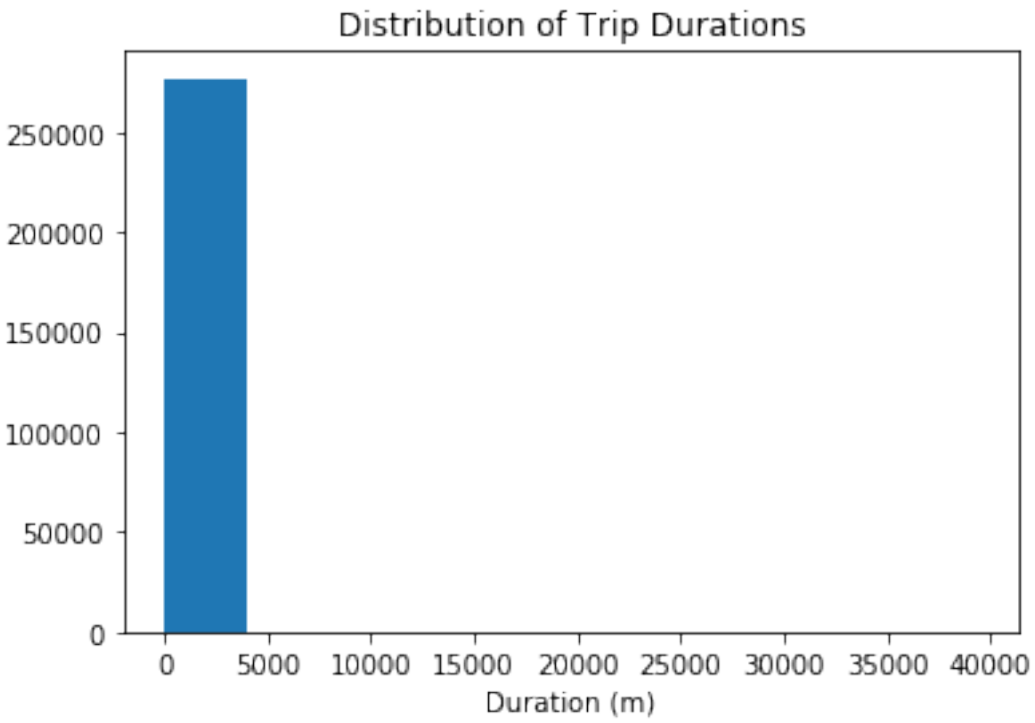
You will now use these functions to create a histogram of the trip times for the city you selected in question 4c. Don't separate the Subscribers and Customers for now: just collect all of the trip times and plot them.

```
In [43]: ## 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.     ##
         def get_trip_durations(filename, user_type=None):
             """
             This function reads in a file with trip data(filename) and an optional
             user type and returns a list of trip durations based on the user type.
             If user type is not specified, all the trip durations in the file will
             be returned.
             """
             # Empty list to hold the trip durations
             trip_duration_list =[]
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)
                 #build list of trip durations in float
                 for row in reader:
                     # If user specific trip details only are needed
```

```
                    if user_type:
                        if row['user_type'] == user_type:
                            trip_duration_list.append(float(row['duration']))
                    else:
                        trip_duration_list.append(float(row['duration']))

            return trip_duration_list
```

In [44]:
```
def plot_histogram(data, title, xlabel, data_range=None, no_of_bars=None ):
    """
    This function reads in a list with data and plots the histogram.
    The parameters and their description is as follows:
    data- data to be plotted
    title- title of the histogram
    xlabel - label for the x-axis in the histogram
    data_range - range of values in the data to be considered. Values
                 out of this range will be ignored. The type is tuple
                 and the values are min and max of range.
    no_of_bars - Number of bars in histogram. The value is derived
                from ((max_range - min_range)/desired_width)
    """
    if data_range and no_of_bars:
        plt.hist(data,
                 bins=no_of_bars,
                 range=data_range)
    elif data_range:
        plt.hist(data,
                 range=data_range)
    elif no_of_bars:
        plt.hist(data,
                 bins=no_of_bars)
    else:
        plt.hist(data)

    plt.title(title)
    plt.xlabel(xlabel)
    plt.show()
```

In [45]:
```
data_files = ['./data/NYC-2016-Summary.csv']

for data_file in data_files:
    data = get_trip_durations(data_file)
    plot_histogram(data, 'Distribution of Trip Durations','Duration (m)' )
```



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

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

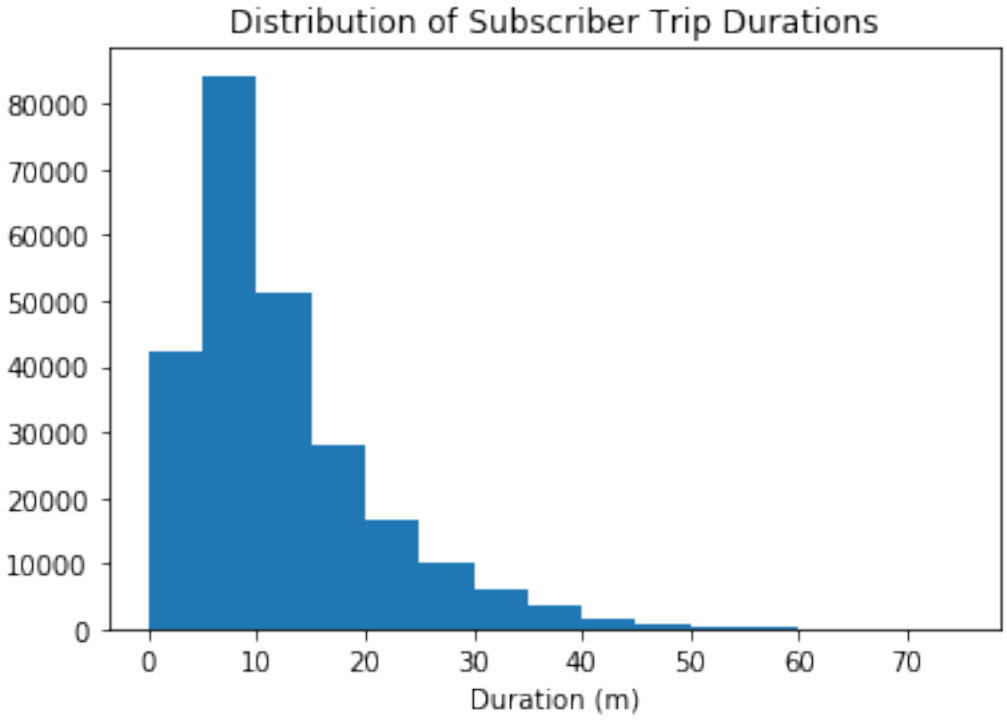
Answer:

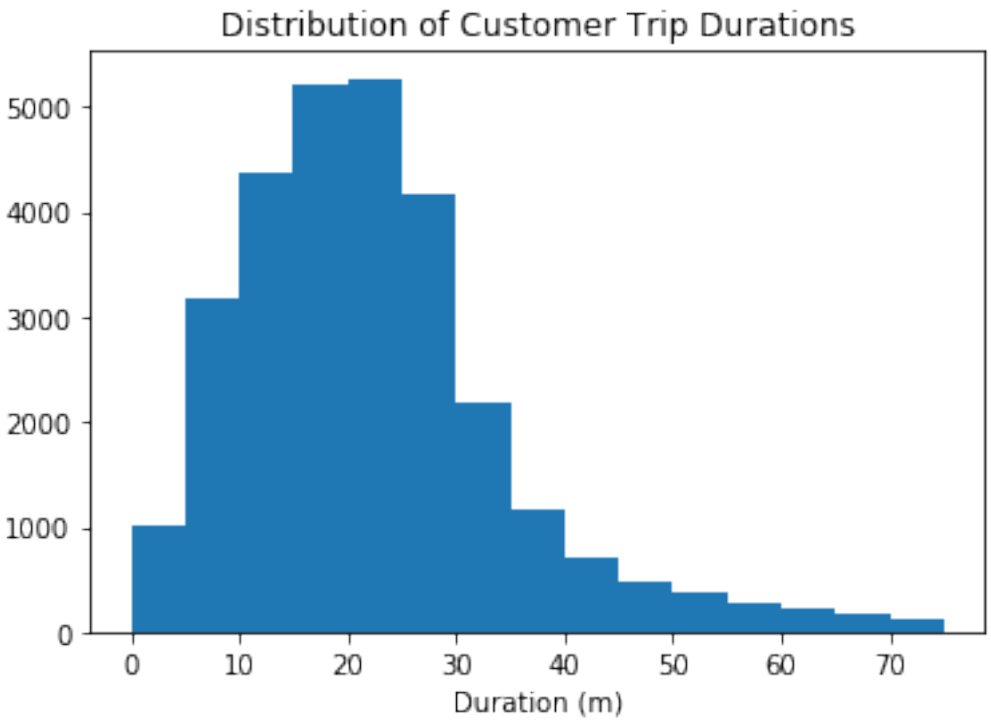
For Subscribers Group in NYC: The peak is 5-10 minutes. The distribution is right skewed.

For Customers Group in NYC: The peak is 20-25 minutes. The distribution is right skewed.

```
In [46]: ## Use this and additional cells to answer Question 5. ##
         data_files = ['./data/NYC-2016-Summary.csv']
         user_types = ['Subscriber', 'Customer']

         for data_file in data_files:
             for user_type in user_types:
                 data = get_trip_durations(data_file, user_type)
                 plot_histogram(data,
```

```
'Distribution of {} Trip Durations'.format(user_type),
'Duration (m)',
(0,75),
15)
```





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.

6a. Which month of the year has the highest number of trips in all the cities? In which month of the year the subscribers use the most? In which month of the year Customers use the most?

6b. Using visualization, Analyze how the ratio of Subscriber trips to Customer trips change depending on the month or season for NYC? Plot it on a bar graph.

Answer:

Answer for 6a:

```
In Washington:
July had the highest number of total trips in the year
June had the highest number of trips by Subscribers
July had the highest number of trips by Customers

In Chicago:
July had the highest number of total trips in the year
June had the highest number of trips by Subscribers
July had the highest number of trips by Customers

In NYC:
September had the highest number of total trips in the year
September had the highest number of trips by Subscribers
August had the highest number of trips by Customers

In BayArea:
October had the highest number of total trips in the year
October had the highest number of trips by Subscribers
August had the highest number of trips by Customers
```

Answer for 6b:

We can see from the visualization that for NYC, the ratio of subscribers to customers does vary a lot with months. The ratio value jumps high in the beginning and ending of the year (December- February). For the rest of the months the ratio does vary, but not drastically. The ratio is higher in December- February, which is winter in NYC, highest being in January. The ratio is lower in July- August which is summer, lowest being in July. The ratio is never less that 1, implying that in any given month, the number of trips by subscribers is always higher than the number of trips by customers.

```
In [47]: ## 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 ridership_data_by_month(filename):
             """
             This function reads in a file with trip data(filename) and returns
             the month wise trip details for each user type.
             """
```

```
    #Set up empty dictionaries to hold month wise trip data for each user type.
    total_month_dict = {}
    sub_month_dict = {}
    cust_month_dict ={}

    # Read the contents of the file.
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)
        # Count the number of trips month wise. Increment the count if month is
        # available. Else add a new month as key and set value as 1. Do this for
        # subscriber, customer and all users.
        for row in reader:
            if row['user_type'] == 'Subscriber':
                if not row['month'] in sub_month_dict:
                    sub_month_dict[row['month']] =1
                else:
                    sub_month_dict[row['month']] += 1
            else:
                if not row['month'] in cust_month_dict:
                    cust_month_dict[row['month']] =1
                else:
                    cust_month_dict[row['month']] += 1

            if not row['month'] in total_month_dict:
                total_month_dict[row['month']] =1
            else:
                total_month_dict[row['month']] += 1

        return(total_month_dict, sub_month_dict, cust_month_dict)
```

In [48]:
```
def month_of_max_ridership(total_month_dict, sub_month_dict, cust_month_dict):
    """
    This function takes dictionary for each user as inputs. The dictionaries
    have  month as key and number of trips in the month as value. The function
    returns the month with maximum number of trips for each of the user type.
    """
    # Find the month with max trips for total users.
    n_max_month = max(total_month_dict, key=total_month_dict.get)
    # Convert the month in number to name. Ex: From 3 to March
    n_max_month_name = calendar.month_name[int(n_max_month)]

    # Find the month with max trips for subscribers.
    n_max_subs_month = max(sub_month_dict, key=sub_month_dict.get)
    n_max_subs_month_name = calendar.month_name[int(n_max_subs_month)]

    # Find the month with max trips for customers.
    n_max_cust_month = max(cust_month_dict, key=cust_month_dict.get)
```

```
            n_max_cust_month_name = calendar.month_name[int(n_max_cust_month)]

            # Return the month with max number of trips for each user type.
            return(n_max_month_name, n_max_subs_month_name, n_max_cust_month_name)
```

```
In [49]: #Run the below code to get answer to question 6a
         data_files = ['./data/Washington-2016-Summary.csv',
                       './data/Chicago-2016-Summary.csv',
                       './data/NYC-2016-Summary.csv',
                       './examples/BayArea-Y3-Summary.csv']
         #for all the csv files available
         for data_file in data_files:

             #Extract city name to print the city.
             city = data_file.split('-')[0].split('/')[-1]
             # Data dict to with key as type of users and value as the month in which max
             # trips were done by the user.
             max_data={}

             # Returns dictionaries with key as month and value as no. of trips in month.
             # Separate dictionary for each user type.
             total_ride_data, sub_ride_data, cust_ride_data = ridership_data_by_month(data_file)

             # get the month of max trips for each user type.
             max_data['total'], max_data['subscriber'],max_data['customer'] = month_of_max_rider


             # Print the results.
             print('In {}:\n'
                   '{} had the highest number of total trips in the year \n'
                   '{} had the highest number of trips by Subscribers \n'
                   '{} had the highest number of trips by Customers\n'.format(city,
                                                                              max_data['total'],
                                                                              max_data['subscrib
                                                                              max_data['customer
```

```
In Washington:
July had the highest number of total trips in the year
June had the highest number of trips by Subscribers
July had the highest number of trips by Customers

In Chicago:
July had the highest number of total trips in the year
June had the highest number of trips by Subscribers
July had the highest number of trips by Customers

In NYC:
September had the highest number of total trips in the year
```

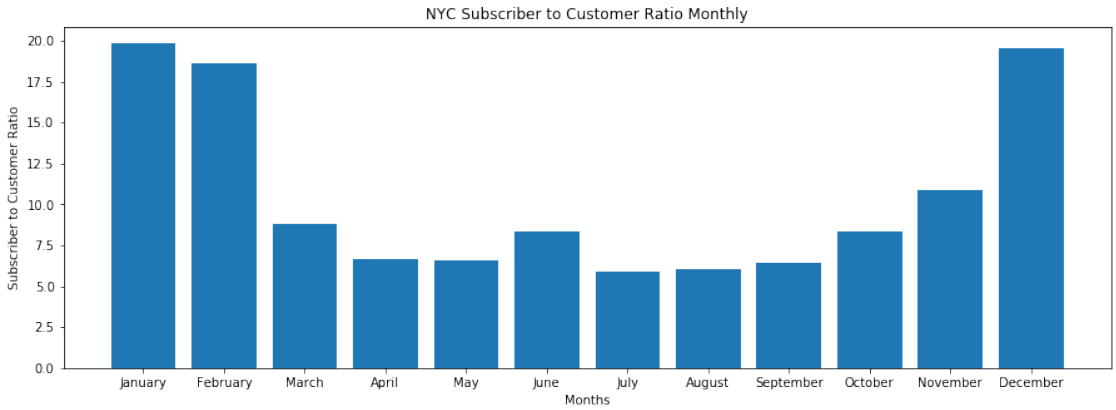
```
September had the highest number of trips by Subscribers
August had the highest number of trips by Customers

In BayArea:
October had the highest number of total trips in the year
October had the highest number of trips by Subscribers
August had the highest number of trips by Customers
```

```
In [50]: def get_ratio_monthwise(sub_ride_data, cust_ride_data):
             """
             This function take input monthly trip data for subscribers and customers
             and returns the ratio of subscriber data to customer data month wise.
             """
             #Empty dictionary to hold the ratio data month wise. Key= month
             # Value = subscriber/customer trip proportion.
             ratio_data ={}

             #keylist as months to iterate over the user type dictionaries.
             keylist =[1,2,3,4,5,6,7,8,9,10,11,12]

             #the key of dictionaries are months represented as numbers(1-12) formatted
             # as strings. To compare and compute proportion, converting keys from
             # string to numeric form(int).
             sub_ride_data = {int(i):int(j) for i,j in sub_ride_data.items()}
             cust_ride_data = {int(i):int(j) for i,j in cust_ride_data.items()}

             # for each month type compute the ratio of subscribers to customers.
             for key in keylist:
                 key_month_name = calendar.month_name[int(key)]
                 ratio_data[key_month_name] = sub_ride_data[key]/cust_ride_data[key]

             return ratio_data
```

```
In [51]: def plot_bar_graph(data, title, xlabel, ylabel):
             """
             This function plots the bar graph for the passed in data
             and the title of the graph. Below are the inputs
             data- data to be plotted in dictionary format.
             title- title of the histogram
             xlabel - label for the x-axis in the graph
             ylabel - label for the y-axis in the graph
             """
             #Names of month on x-axis
             names = list(data.keys())
             values = list(data.values())
```

```
    # Set appropriate figure size so that the months are visible and
    # bars are not cramped.
    plt.figure(figsize=(15, 5))

    # Setup the graph.
    plt.bar(range(len(data)),values,tick_label=names)
    # Graph title, xlabel and ylabel
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    # Plot the graph.
    plt.show()
```

In [52]:
```
#Run the below code to get answer to question 6b
data_files = ['./data/NYC-2016-Summary.csv']

#for all the csv files available
for data_file in data_files:
    #Extract city name to print the city.
    city = data_file.split('-')[0].split('/')[-1]
    # Returns dictionaries with key as month and value as no. of trips in month.
    # Separate dictionary for each user type.
    total_ride_data, sub_ride_data, cust_ride_data = ridership_data_by_month(data_file)
    ratio_data = get_ratio_monthwise(sub_ride_data, cust_ride_data)
    # Plot the data
    plot_bar_graph(ratio_data,
                   '{} Subscriber to Customer Ratio Monthly'.format(city),
                   'Months',
                   'Subscriber to Customer Ratio')
```



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: Recently there was a news report that the major cities around the world would run out of water. Cities like Sao Paulo are already on the verge of zero water. I believe data science can help solve the water shortage problems.

Data analytics can be carried out on the water usage pattern. This can be used to analyze the supply demand probem. Usage data based on the household requirements, per person usage, usage in affluent areas of the city, usage in industries and other sources can be collected analyzed. Usage at household can be further narrowed down whether houses using more water have big gardens, animals, etc which can cause more water usage. Water consumption becasue of wastage and leaking pipelines, etc can be analyzed.

Supply related data, like the sources of water and quantity, historic data and usages can be analyzed to see if we can come up wih an accurate model of demand-supply and assess the risk. Ground water levels, rainfall levels, river water availability data can be considered. Based on the data analysis, we come up with effective measures to solve the water problems in the areas of water supply, usage and storage.

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