Bike_Share_Analysis

December 19, 2017

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

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

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

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

Posing Questions

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

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

Answer: Following two questions are most obvious and can be answered using data -

1 Approximate amount of bikes used in a particular location in some fixed amount of time in various cities.

2 Trend and comparision of type of users.

> **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 [2]: ## 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 [3]: 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 ##
                #next(trip_reader)
                first_trip = next(trip_reader)

                ## TODO: Use the pprint library to print the first trip. ##
                ## see https://docs.python.org/3/library/pprint.html     ##
                #print
                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 [4]: def duration_in_mins(datum, city):
            """
            Takes as input a dictionary containing info about a single trip (datum) and
            its origin city (city) and returns the trip duration in units of minutes.

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

            HINT: The csv module reads in all of the data as strings, including numeric
            values. You will need a function to convert the strings into an appropriate
            numeric type when making your transformations.
            see https://docs.python.org/3/library/functions.html
            """

            if city == 'Washington':
                #Converts millisecond to minutes for washington
                temp = datum['Duration (ms)']
                duration_int_millisecond = int(temp)
                duration = duration_int_millisecond / 60000
            else:
                #Converts seconds to minute for Chicago and NYC
                temp = datum['tripduration']
                duration_int_second = int(temp)
```

```
        duration = duration_int_second / 60


    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 [5]:
```
from datetime import datetime
import calendar

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
    """

    date_string = 'starttime'
    date_format = "%m/%d/%Y %H:%M"

    if city == 'NYC':
        #Modifying date_format according to entry in NYC
        date_format = "%m/%d/%Y %H:%M:%S"
    elif city == 'Washington':
        #Modifying date_string according to entry in Washington
        date_string = 'Start date'

    date_entry = datum[date_string]

    #strips date in tokens
    date_token = datetime.strptime(date_entry,date_format)
    month = date_token.month
    hour = date_token.hour
```

```
    day_of_week = calendar.day_name[date_token.weekday()]

    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 [6]:
```
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.
    """
    user_type = ''

    if city == 'Washington' and datum['Member Type'] == 'Registered':
        user_type = 'Subscriber'
    elif city == 'Washington' and datum['Member Type'] == 'Casual':
        user_type = 'Customer'
    else:
        user_type = datum['usertype']

    return user_type


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

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

Question 3b: Now, use the helper functions you wrote above to create a condensed data file for each city consisting only of the data fields indicated above. In the `/examples/` folder, you will

see an example datafile from the Bay Area Bike Share before and after conversion. Make sure that your output is formatted to be consistent with the example file.

```
In [7]: 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'] = time_of_trip(row,city)[0]
                    new_point['hour'] = time_of_trip(row,city)[1]
                    new_point['day_of_week'] = time_of_trip(row,city)[2]
                    new_point['user_type'] = type_of_user(row,city)

                    ## 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 [8]: # 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:
1 : Highest number of trips : **NYC [276798]**
2 : Highest proportion of Subscribers: **NYC [88.84 %]**
3 : Highest proportion of Customers(Short-Term) : **Chicago [23.78 %]**

In [9]:
```
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 [10]:
```
## 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 total_trip(filename):
    return number_of_trips(filename)[2]

def proportion_helper(cust_type,filename):
    """
    Helps to return trip proportion by accepting user type and city path

    returns proportion by calculating trip count and trip total
    """


    total = number_of_trips(filename)[2]

    #Based on user type fetch trip count
```

```
            if cust_type == 'Subscriber':
                trip_count = number_of_trips(filename)[0]
            else:
                trip_count = number_of_trips(filename)[1]

            return int(trip_count) / int (total) * 100

        def highest_calculation(city_info,cust_type='All',type_='Calculation'):
            """
            Function used to calulate either maximum of all types of user or proportion using t

            returns maximum proportion or maximum based on type
            """


            max_ = 0
            max_city = ''

            for city in city_info:
                if type_ == 'Proportion':
                    temp = proportion_helper(cust_type,city_info[city])
                else:
                    temp = total_trip(city_info[city])

                if temp > max_:
                    max_ = temp
                    max_city = city

            return max_city,max_
```

In [11]:
```
# test function to check corresponding answers for Q4a

city_info = {'Washington' : './data/Washington-2016-Summary.csv',
             'Chicago': './data/Chicago-2016-Summary.csv',
             'NYC': './data/NYC-2016-Summary.csv'}

print("Highest number of trips : {}".format(highest_calculation(city_info)))
print("Highest propotion of Subscribers : {} ".format(highest_calculation(city_info,"Su
print("Highest proportion of Customer : {} ".format(highest_calculation(city_info,"Cust
```

```
Highest number of trips : ('NYC', 276798)
Highest propotion of Subscribers : ('NYC', 88.83590199351151)
Highest proportion of Customer : ('Chicago', 23.774798630269924)
```

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:

Average trip length for NYC : 15.81 minutes
Average trip length for Chicago : 16.56 minutes
Average trip length for Washington : 15.81 minutes
Proportion of ride longer than 30 minutes for NYC : 10.84%
Proportion of ride longer than 30 minutes for Chicago : 8.33%

Proportion of ride longer than 30 minutes for Washington : 7.30%

In [12]:
```
## 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_duration(filename, longer_duration_limit = 30):
    """
    Counts average and proportion of several variables
    default value for longer duration threshhold is 30

    returns Average Duration , Proportion of longer duration trips, Customers trip prop
    """
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # initialize count variables
        n_duration = []
        n_longer_duration = 0
        customer_duration = []

        # tally up ride duratiom
        for row in reader:
            temp = float(row['duration'])
            n_duration.append(temp)

            if row['user_type'] == 'Customer':
```

```
                customer_duration.append(temp)

            if temp > longer_duration_limit:
                n_longer_duration += 1

        sum_duration = sum(n_duration)
        sum_customer_duration = sum(customer_duration)
        sum_subscriber_duration = sum_duration - sum_customer_duration

        count_trip = len(n_duration)
        count_customer = len(customer_duration)
        count_subscriber = count_trip - count_customer

        return sum_duration/count_trip , n_longer_duration/count_trip * 100 , sum_custo
```

```
In [13]: # test function to get answers for 4b

         for city in city_info:
             average_duration, longer_duration , average_customer_duration , average_subscriber_
             print("Average Duration in city {} is {} and proportion of trip longer then 30 minu
```

```
Average Duration in city Washington is 18.93287355913721 and proportion of trip longer then 30 m
Average Duration in city Chicago is 16.563629368787335 and proportion of trip longer then 30 min
Average Duration in city NYC is 15.81259299802294 and proportion of trip longer then 30 minutes
```

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

Answer: In city Washington on average Customer(41.68 minutes) takes longer trip than Subscriber(12.53 minutes)

```
In [14]: ## 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?                                                ##

         for city in city_info:
             average_customer_duration = average_trip_duration(city_info[city])[2]
             average_subscriber_duration = average_trip_duration(city_info[city])[3]

             print("For city {} \n Customer Average trip duration is {} and Subscriber Average t
```

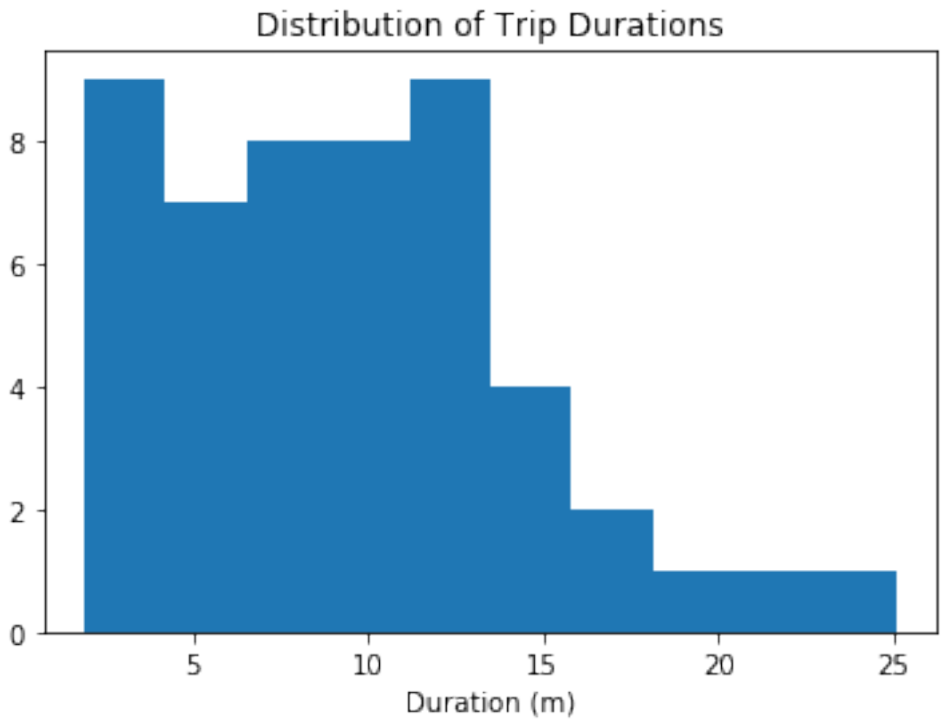
```
For city Washington
 Customer Average trip duration is 41.67803139252976 and Subscriber Average trip duration is 12.
```

```
For city Chicago
 Customer Average trip duration is 30.979781133982506 and Subscriber Average trip duration is 12
For city NYC
 Customer Average trip duration is 32.982004306775025 and Subscriber Average trip duration is 13
```

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 [15]: # 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()
```

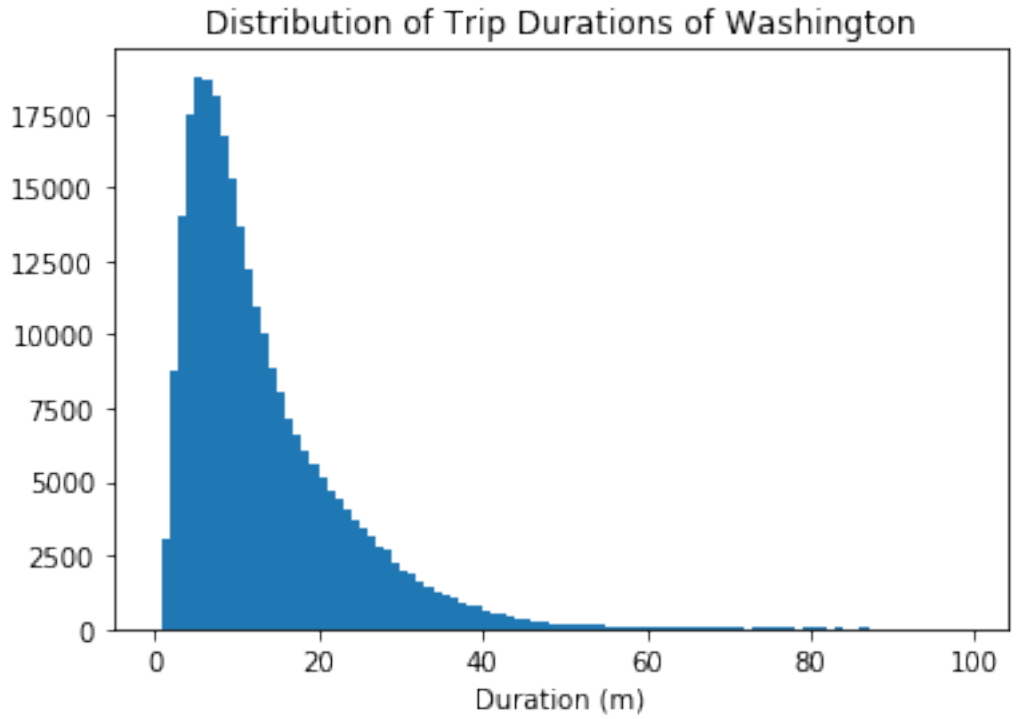


In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. 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 [16]: def trip_times(filename):

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

                 trip_duration = []

                 # tally up ride types
                 for row in reader:
                     temp = float(row['duration'])
                     trip_duration.append(temp)

                 #print(len(trip_duration))

                 return trip_duration
```

```
In [17]: ## 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.    ##

         %matplotlib inline

         data = trip_times('./data/NYC-2016-Summary.csv')

         plt.hist(data,range(0,100))
         plt.title('Distribution of Trip Durations of Washington')
         plt.xlabel('Duration (m)')
         plt.show()
```



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

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

Answer:
For **Subscribers** residing in NYC : **Peak Timings was in Interval : 5-10 minutes**
For **Customers** residing in NYC : **Peak Timings was in Interval : 20-25 minutes**

- **Shape of distribution** -

  Both subscriber and customers of NYC gives distribution similar to *Bell Curve* with increasing and decreasing values in trend
  Also **Subscribers** are more comparitive to Customers hence **have more weightage** in defining overall trend

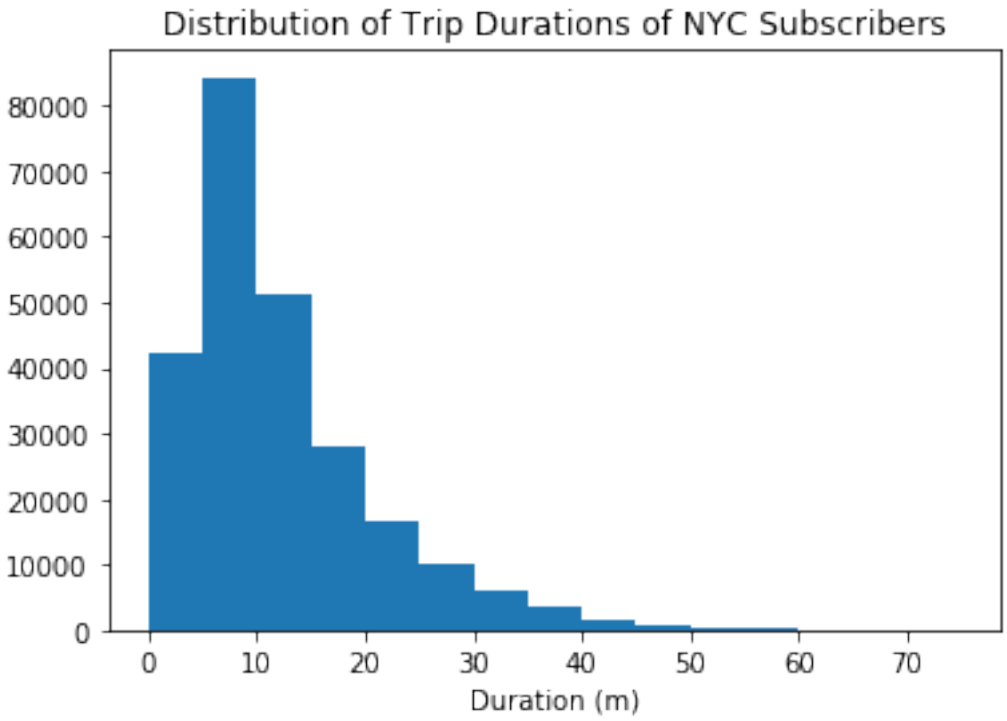
  Both subscriber and customer has *right tail longer* and *mass of distribution in concentrated towards left* which makes it **right skewed** and hence **positive skew**

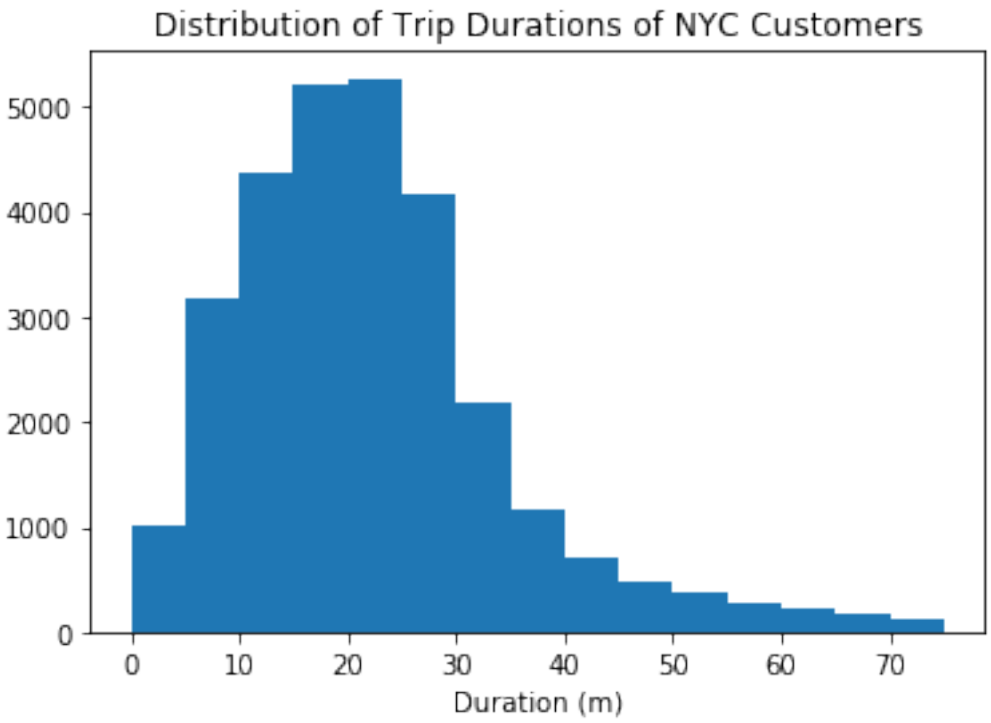
  Mode of distribution for **NYC customer is unimodal around 5-10 minutes** while for **NYC subscriber is also unimodal around 20 minutes**

In [18]:
```
def trip_times_per_user(filename,type_):

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

        trip_duration = []

        # tally up ride types
        for row in reader:
            if row['user_type'] == type_:
                temp = float(row['duration'])
                trip_duration.append(temp)

        #print(len(trip_duration))
        return trip_duration
```

In [19]:
```
import numpy as np
## 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.     ##

%matplotlib inline

data = trip_times_per_user('./data/NYC-2016-Summary.csv','Subscriber')

#plt.hist(data,range(0,75))
binwidth = 5
bins=np.arange(0, 75 + binwidth, binwidth)
plt.hist(data,bins)
plt.title('Distribution of Trip Durations of NYC Subscribers')
plt.xlabel('Duration (m)')
plt.show()
```

```
data = trip_times_per_user('./data/NYC-2016-Summary.csv','Customer')

binwidth = 5
bins=np.arange(0, 75 + binwidth, binwidth)
plt.hist(data,bins)
plt.title('Distribution of Trip Durations of NYC Customers')
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:

Proposed Question : Requirement of optimal amount of bike required in a city based on peak time plotted for year, month and day of the week. Reason being selecting these parameters is because of requirement of proper maintainance and instant service without over investing. Amount of trips trend can give usage pattern based on seleted parameter and hence for providing better customer service and availability.

Subscriber and Customers are not discriminated for observations since the idea is to optimaly placing bikes and providing better service rather than increasing subscriber base.

1.1.1 Analysing based on Hour of Day

Comparing trips made by user based on hour of day gives us following observation -

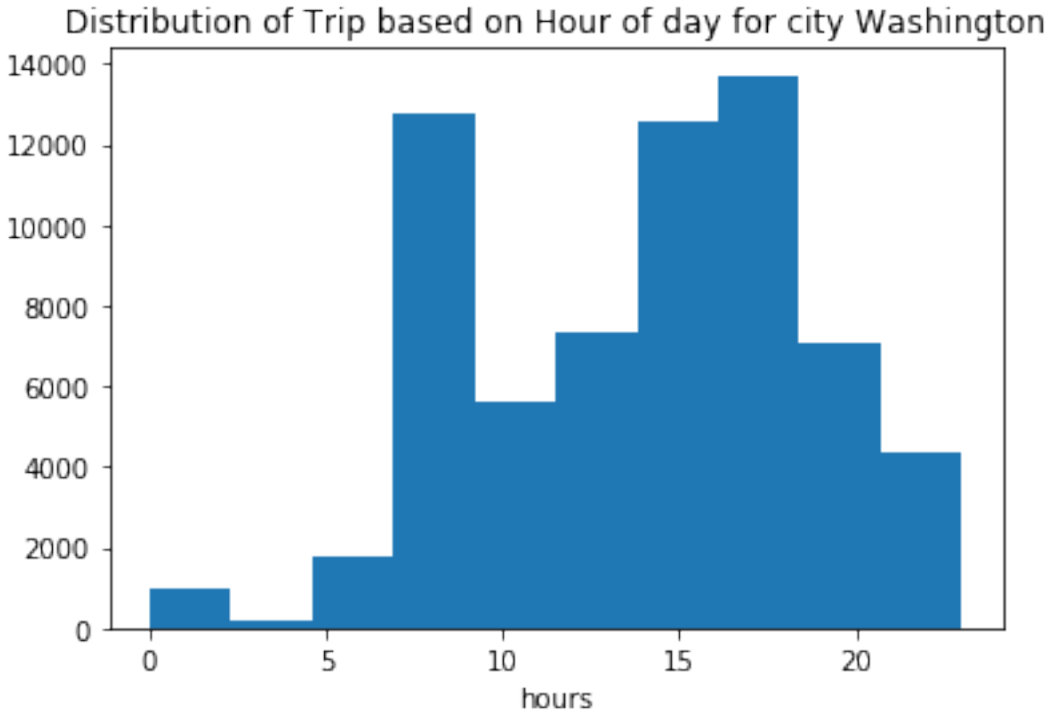
- **All three cities has similar usage pattern** : Trends in all three cities are almost similar
- **Most usage in early morning or early evening** : All three cities sees peak usage during early morning and early evening
- **Almost no usage from midnight to early morning** : Same pattern of almost no usage after midnight for three cities. Rest of hours has constant trend

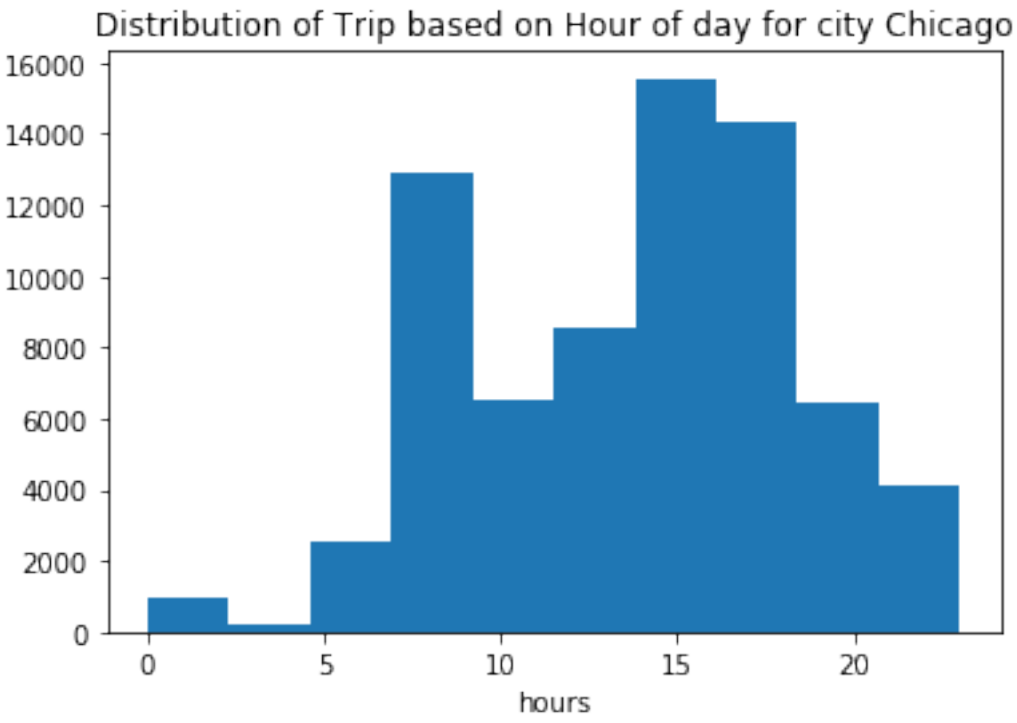
In [20]:
```
## Use this and additional cells to continue to explore the dataset. ##
## Once you have performed your exploration, document your findings  ##
## in the Markdown cell above.                                       ##

def list_hour(filename):
    """
    Function to return count based on hour of day

    returns list with count of each hour of day
    """
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        trip_hour = []

        # tally up ride types
        for row in reader:
                temp = int(row['hour'])
                trip_hour.append(temp)

        return trip_hour
```

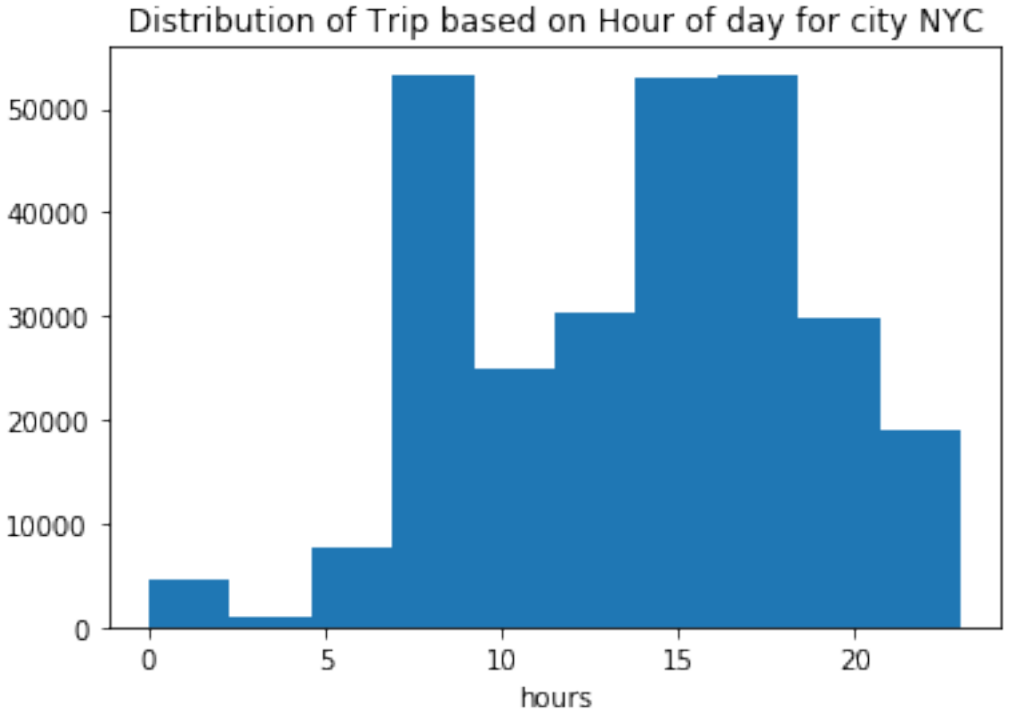
In [21]:
```
def plot_comparision(data , city):
    """
    Helper function to plot data
```

```
    """
    %matplotlib inline

    plt.hist(data)
    plt.title('Distribution of Trip based on Hour of day for city {}'.format(city))
    plt.xlabel('hours')
    plt.show()
```

In [22]:
```
for city in city_info:
    hour = list_hour(city_info[city])
    plot_comparision(hour , city)
```




Distribution of Trip based on Hour of day for city Chicago
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5
10
15
20
hours

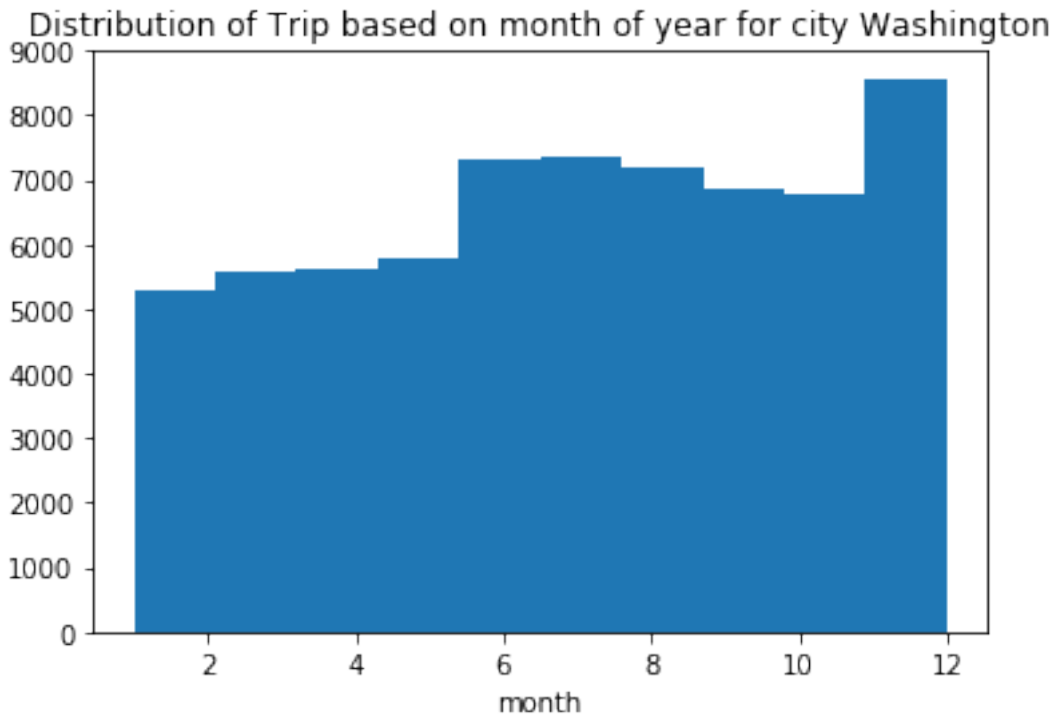

Distribution of Trip based on Hour of day for city NYC
50000
40000
30000
20000
10000
0
0
5
10
15
20
hours

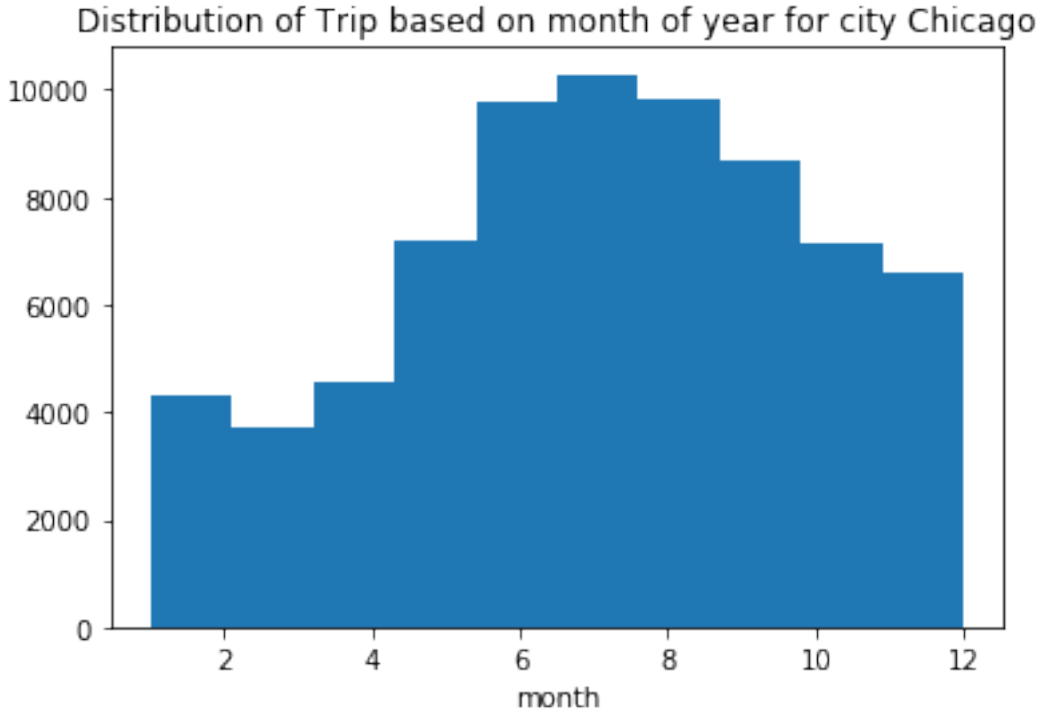
1.1.2 Analysing based on Month of Year

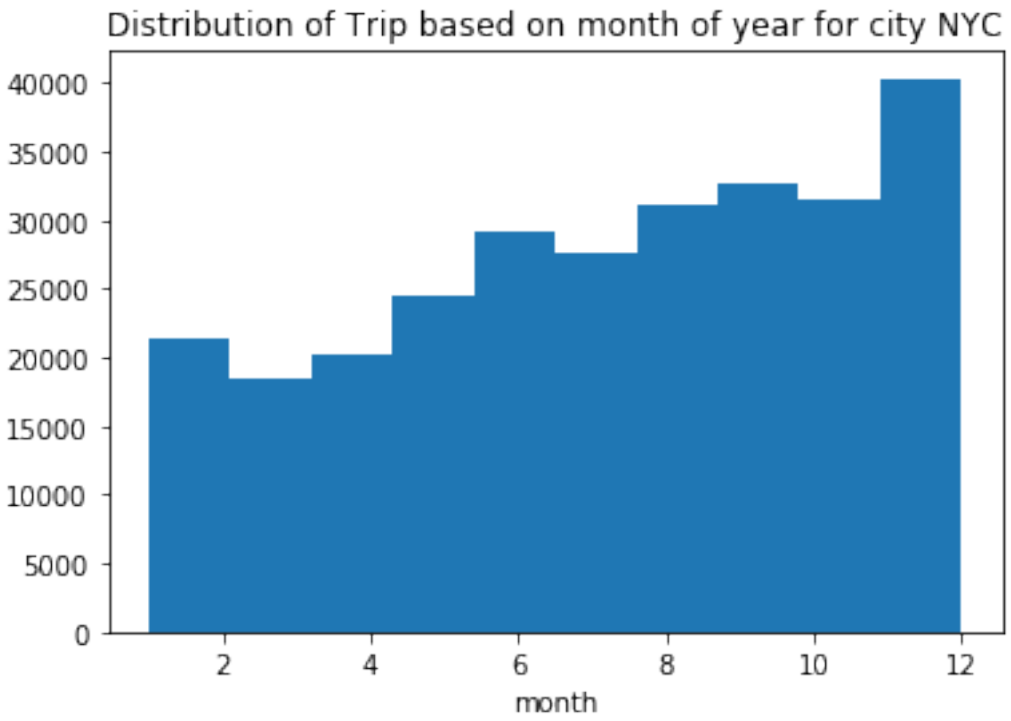
Comparing trips made by user based on month of year gives following observation -

- **Washington and NYC sees peak in December** : Washington has highest number of trips made in December than any other month of year. That helps in placing higher bikes in Washington during December. Both Washington and NYC has increasing trend based on month.
- **Chicago sees peak during June-September** : Chicago doesnt have particular month as peak but has a duration where usage is much higher than other months i.e. June-September.

```
In [23]: def list_month(filename):
             """
             Function to return count based on month of year

             returns list with count of each month
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)

                 trip_month = []

                 for row in reader:
                         temp = int(row['month'])
                         trip_month.append(temp)


                 #print(len(trip_duration))
                 return trip_month

In [24]: def plot_comparision(data , city):
             """
             Helper function to plot data
             """
             %matplotlib inline
             plt.hist(data)
             plt.title('Distribution of Trip based on month of year for city {}'.format(city))
             plt.xlabel('month')
             plt.show()

In [25]: for city in city_info:
             month = list_month(city_info[city])
             plot_comparision(month , city)
```


Distribution of Trip based on month of year for city Washington
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
2
4
6
8
10
12
month


Distribution of Trip based on month of year for city Chicago
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
month



1.1.3 Analysing trips based on day of the week

Comparing trips made by users by day of week gives us pattern where we can see which day system has been used most.

Following observation has been made -

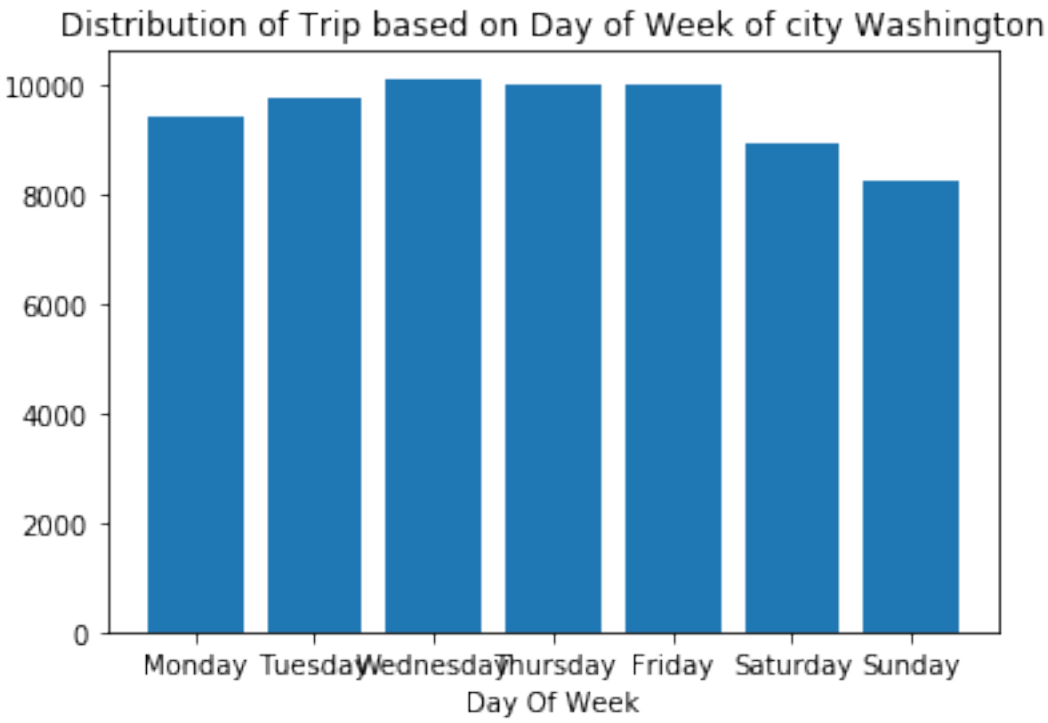
- **Less Usage during weekends** : During Saturday and Sunday usage of the system dips but not significantly. Washington and NYC sees less usage only in weekends but Chicago has less usage in Wednesday and Thursday too.
- **Overall less change in trend** : Idea of placing optimal number of bikes can be used because there is not much change in patterns of bikeshare usage over days in a week.

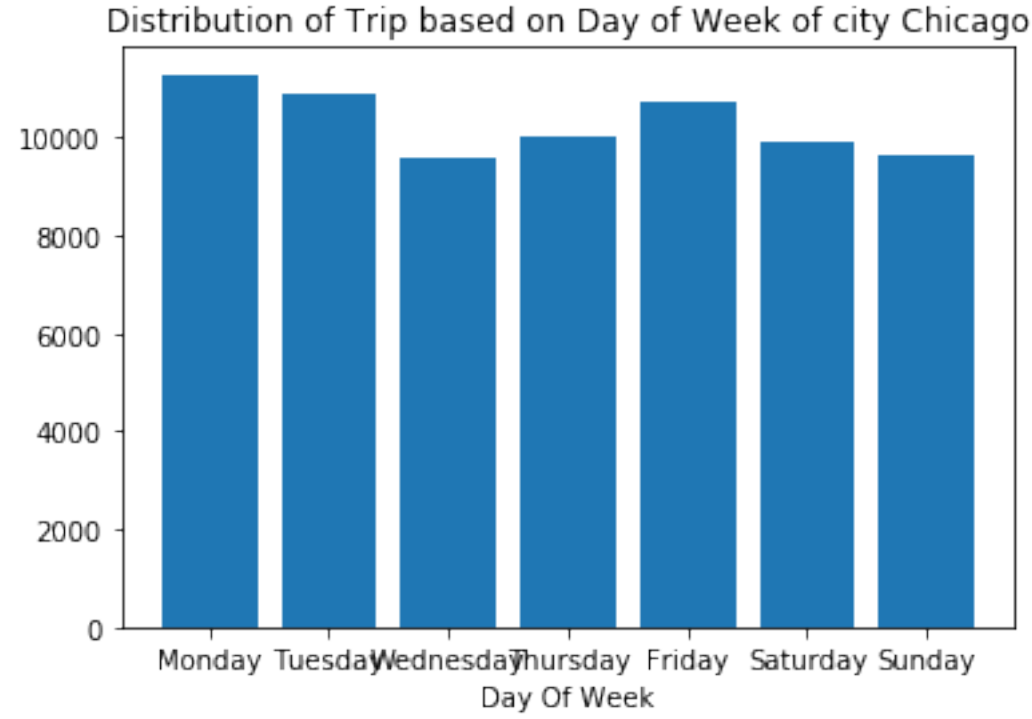
```
In [26]: def list_day(filename):
             """
             Helper function to fetch day and count of trip per day

             returns dictionary value with key as day and vakue as count
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)

                 #Dictionary to store values
```

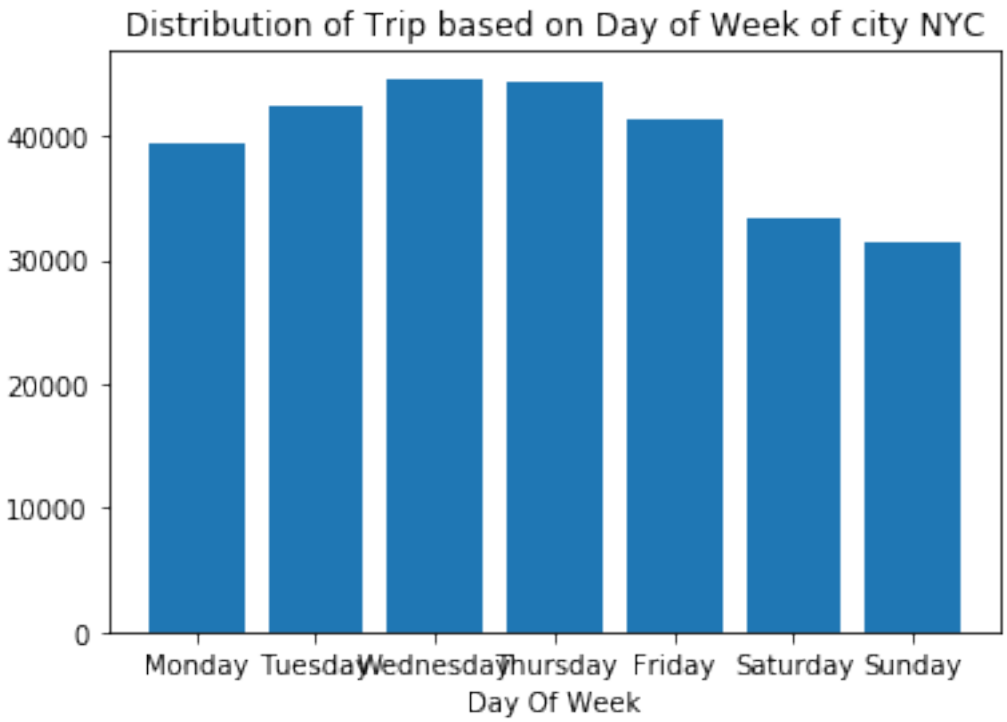
```
    trip_day_of_week = {'Monday': 0 , 'Tuesday' : 0 , 'Wednesday': 0 , 'Thursday' :

    for row in reader:
            day = row['day_of_week']
            trip_day_of_week[day] += 1

    return trip_day_of_week
```

In [27]:
```
def plot_comparision(data , city):
    """
    Helper function to plot data
    """

    %matplotlib inline

    #plots bar by fetching dictionary values from data
    plt.bar(range(len(data)), data.values(), align='center')

    #plots x axis as key value from dictionry data
    plt.xticks(range(len(data)), list(data.keys()))

    plt.title('Distribution of Trip based on Day of Week of city {}'.format(city))
    plt.xlabel('Day Of Week')
    plt.show()
```

In [28]:
```
for city in city_info:
    day = list_day(city_info[city])

    plot_comparision(day , city)
```


Distribution of Trip based on Day of Week of city Washington
10000
8000
6000
4000
2000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day Of Week


Distribution of Trip based on Day of Week of city Chicago
10000
8000
6000
4000
2000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day Of Week



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: Education in India is a major concern as India has large population and very less portion of people are able to complete basic education. People who have completed basic education have difficulties in pursuing higher education like (Masters/Phd) or getting a job. Hence a great topic to analyse.

I want to see the trend of people starting education and how they are performing in each stage and when they finish/stop education. The trends of there grade will help in analysing reasons they quit studying. Also demographic data will help to analyse conditions that are effecting them for getting a better education.

Similarly people who have completed atleast there under graduate course, there data can help in analysing factors that make them either get a job or remaining unemployed. Jobs can also be further categorised based on salary to analyse education v/s salary trend.

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

```
Out[29]: 0
```