Bike_Share_Analysis

January 23, 2018

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

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

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

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

Posing Questions

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

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

Answer: 1.During which day of the week bike demand is more.So that we can make sure we will enough stock available on that day. 2.What is gender and age of people are taking more bikes and which types of bikes .So that we should have more stocks of that bike

> **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 [17]: ## import all necessary packages and functions.
         import csv # read and write csv files
         from datetime import datetime # operations to parse dates
         import pprint # use to print data structures like dictionaries in
                                    # a nicer way than the base print function.
```

```
In [18]: 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.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
             print(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')])
OrderedDict([('tripduration', '839'), ('starttime', '1/1/2016 00:09:55'), ('stoptime', '1/1/2016

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')])
OrderedDict([('trip_id', '9080545'), ('starttime', '3/31/2016 23:30'), ('stoptime', '3/31/2016 2

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')])
OrderedDict([('Duration (ms)', '427387'), ('Start date', '3/31/2016 22:57'), ('End date', '3/31/
```

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

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

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

             # YOUR CODE HERE
             if(city=='Washington'):
                 duration=(float((datum['Duration (ms)']))/(60*1000))
             else:
                 duration=(float((datum['tripduration']))/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 [20]:
```
import datetime
from dateutil import parser
day_of_week='Sunday'
def time_of_trip(datum, city):
    global day_of_week
    """
    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=='Washington'):
        datetime_obj=datetime.datetime.strptime(datum['Start date'],'%m/%d/%Y %H:%M')
    elif(city=='NYC'):
        datetime_obj=datetime.datetime.strptime(datum['starttime'],'%m/%d/%Y %H:%M:%S')
    else:
        datetime_obj=datetime.datetime.strptime(datum['starttime'],'%m/%d/%Y %H:%M')


    month=datetime_obj.month
    hour=datetime_obj.hour
    day=datetime_obj.weekday()
    if(day==0):
        day_of_week='Monday'
    elif(day==1):
        day_of_week='Tuesday'
    elif(day==2):
        day_of_week='Wednesday'
    elif(day==3):
        day_of_week='Thursday'
    elif(day==4):
        day_of_week='Friday'
```

```
    elif(day==5):
        day_of_week='Saturday'
    elif(day==6):
        day_of_week='Sunday'
    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 [21]:
```
user_type='sf'
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.
    """
    global user_type
    # YOUR CODE HERE
    if(city=='Washington'):
        user_type=datum['Member Type']
        if(user_type=='Registered'):
            user_type='Subscriber'
        else:
            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 [22]: new_point={}
         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
                     global new_point
                     new_point = {}
                     if(city=='Washington'):
                         datetime_obj=datetime.datetime.strptime(row['Start date'],'%m/%d/%Y %H:
                     elif(city=='Chicago'):
                         datetime_obj=datetime.datetime.strptime(row['starttime'],'%m/%d/%Y %H:%
                     else:
                         datetime_obj=datetime.datetime.strptime(row['starttime'],'%m/%d/%Y %H:%
                     month=datetime_obj.month
                     hour=datetime_obj.hour
                     day=datetime_obj.weekday()
                     if(day==0):
                         day_of_week='Monday'
                     elif(day==1):
                         day_of_week='Tuesday'
```

```
                elif(day==2):
                    day_of_week='Wednesday'
                elif(day==3):
                    day_of_week='Thursday'
                elif(day==4):
                    day_of_week='Friday'
                elif(day==5):
                    day_of_week='Saturday'
                elif(day==6):
                    day_of_week='Sunday'
                if(city=='Washington'):
                    dur=row['Duration (ms)']
                else:
                    dur=row['tripduration']
                # newpoint.append(month)
                # newpoint.append(hour)
                # newpoint.append(day_of_week)
                # print(month,hour,day_of_week)
                if(city=='Washington'):
                    type_mem=row['Member Type']
                else:
                    type_mem=row['usertype']
                new_point['duration']=dur
                new_point['month']=month
                new_point['hour']=hour
                new_point['day_of_week']=day_of_week
                new_point['user_type']=type_mem
                #print(new_point)
                trip_writer.writerow(new_point)
            #trip_writer.close()

                ## 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.         ##


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

In [23]:
```
# 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', '427387'),
             ('month', '3'),
             ('hour', '22'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Registered')])

City: Chicago
OrderedDict([('duration', '926'),
             ('month', '3'),
             ('hour', '23'),
             ('day_of_week', 'Thursday'),
             ('user_type', 'Subscriber')])

City: NYC
OrderedDict([('duration', '839'),
             ('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: Newyork has the highest number of trips . Washington has the highest proportion of trips made by subscribers. Newyork has the highest proportion of trips made by short-term customers .

In [24]:
```
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 [25]:
```
## Modify this and the previous cell to answer Question 4a. Remember to run ##
## the function on the cleaned data files you created from Question 3.      ##
data_file_w='./data/Washington-2016-Summary.csv'
data_file_n='./data/NYC-2016-Summary.csv'
data_file_c='./data/Chicago-2016-Summary.csv'
#data_file = './examples/BayArea-Y3-Summary.csv'
n_total_w,n_sub_w,n_cus_w=number_of_trips(data_file_w)
n_total_n,n_sub_n,n_cus_n=number_of_trips(data_file_n)
n_total_c,n_sub_c,n_cus_c=number_of_trips(data_file_c)
max_total=n_total_w
max_total_city='Washington'
max_sub=n_sub_w
max_sub_city='Washington'
max_cus=n_cus_w
max_cus_city='Washington'
if(n_total_n>max_total):
    max_total_city='Newyork'
    max_total=n_total_n
if(n_total_c>max_total):
    max_total_city='Chicago'
    max_total=n_total_c
```

```
        if(n_sub_n>max_sub):
            max_sub_city='Newyork'
            max_sub=n_sub_n
        if(n_sub_c>max_sub):
            max_sub_city='Chicago'
            max_sub=n_sub_c

        if(n_cus_n>max_cus):
            max_cus_city='Newyork'
            max_cus=n_total_n
        if(n_cus_c>max_cus):
            max_cus_city='Chicago'
            max_cus=n_cus_c
        print(max_total_city,"has the highest number of trips ")
        print(max_sub_city,"has the highest proportion of trips made by subscribers")
        print(max_cus_city,"has the highest proportion of trips made by short-term customers ")
```

```
Newyork has the highest number of trips
Washington has the highest proportion of trips made by subscribers
Newyork 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: Newyork:the average trip length is 4376894.116666754 minutes and 32.653152256020626 % of trips are longer than 30 minutes.

Chicago:the average trip length is 1194751.1499999992 minutes and 32.265632610327856 % of trips are longer than 30 minutes.

Washington:the average trip length is 1255741.7716833346 minutes and 44.2792998572706 % of trips are longer than 30 minutes.

In [26]:
```
## Use this and additional cells to answer Question 4b.                ##
##                                                                     ##
## HINT: The csv module reads in all of the data as strings, including ##
## numeric values. You will need a function to convert the strings     ##
## into an appropriate numeric type before you aggregate data.         ##
## TIP: For the Bay Area example, the average trip length is 14 minutes ##
## and 3.5% of trips are longer than 30 minutes.
```

```
data_file_w='./data/Washington-2016-Summary.csv'
data_file_n='./data/NYC-2016-Summary.csv'
data_file_c='./data/Chicago-2016-Summary.csv'
def total_duration(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        total_dur_min=0
        #count the total duration in min(newyrork,chicago in sec,washington ms)
        for row in reader:
           # print(row['duration'])
            if row['duration'].isdigit():
                total_dur_min+=float(row['duration'])/60
    return total_dur_min
def total_duration_washington(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        total_dur_min=0
        #count the total duration in min(newyrork,chicago in sec,washington ms)
        for row in reader:
            if row['duration'].isdigit():
                total_dur_min+=float(row['duration'])/(60*1000)
    return total_dur_min

    #
def total_duration_grt_30min(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        total_dur_min_30=0
        #count the total duration in min(newyrork,chicago in sec,washington ms)
        for row in reader:
            if row['duration'].isdigit():
                dur=float(row['duration'])/(60)
                if(dur>30):
                    total_dur_min_30+=dur

    return total_dur_min_30
def total_duration_grt_30min_washington(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        total_dur_min_30=0
        #count the total duration in min(newyrork,chicago in sec,washington ms)
        for row in reader:
            if row['duration'].isdigit():
                dur=float(row['duration'])/(60*1000)
                if(dur>30):
                    total_dur_min_30+=dur

    return total_dur_min_30
```

```
total_dur=total_duration(data_file_n)
total_dur_30=total_duration_grt_30min(data_file_n)
#print(total_dur,total_dur_30)
pro=(total_dur_30/total_dur)*100
print("Newyork:the average trip length is ",total_dur," minutes and ",pro,"% of trips a
total_dur=total_duration(data_file_c)
total_dur_30=total_duration_grt_30min(data_file_c)
#print(total_dur,total_dur_30)
pro=(total_dur_30/total_dur)*100
print("Chicago:the average trip length is ",total_dur," minutes and ", pro,"% of trips
total_dur= total_duration_washington(data_file_w)
total_dur_30= total_duration_grt_30min_washington(data_file_w)
#print(total_dur,total_dur_30)
pro=(total_dur_30/total_dur)*100
print("Washington:the average trip length is ",total_dur," minutes and ", pro,"% of tri
```

```
Newyork:the average trip length is  4376894.116666754  minutes and  32.653152256020626 % of trip
Chicago:the average trip length is  1194751.1499999992  minutes and  32.265632610327856 % of tri
Washington:the average trip length is  1255741.7716833346  minutes and  44.2792998572706 % of tr
```

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: Newyork Subscriber takes longer rides on average: 995561.8000000041

In [27]:
```
## 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?
## Use this and additional cells to answer Question 4b.                 ##
##                                                                      ##
## HINT: The csv module reads in all of the data as strings, including  ##
## numeric values. You will need a function to convert the strings      ##
## into an appropriate numeric type before you aggregate data.          ##
## TIP: For the Bay Area example, the average trip length is 14 minutes ##
## and 3.5% of trips are longer than 30 minutes.


data_file_newyork='./data/NYC-2016-Summary.csv'
```

```
        def total_duration(filename):
            with open(filename,'r') as f_in:
                reader=csv.DictReader(f_in)
                dur_min=0
                total_dur_sub=0
                total_dur_cus=0
                #count the total duration in min(newyrork,chicago in sec,washington ms)
                for row in reader:
                    dur_min=float(row['duration'])/(60)
                    if(row['user_type']=='Customer'):
                        total_dur_cus+=dur_min
                    else:
                        total_dur_sub+=dur_min

            #print(total_dur_cus,total_dur_sub)
            return total_dur_cus,total_dur_sub

        total_dur_customer,total_dur_subscriber=total_duration(data_file_newyork)
        #print(total_dur_customer,total_dur_subscriber)
        if(total_dur_customer>total_dur_subscriber):
            print("Newyork Customer takes longer rides on average:",total_dur_customer)
        else:
            print("Newyork Subscriber takes longer rides on average:",total_dur_customer)
```

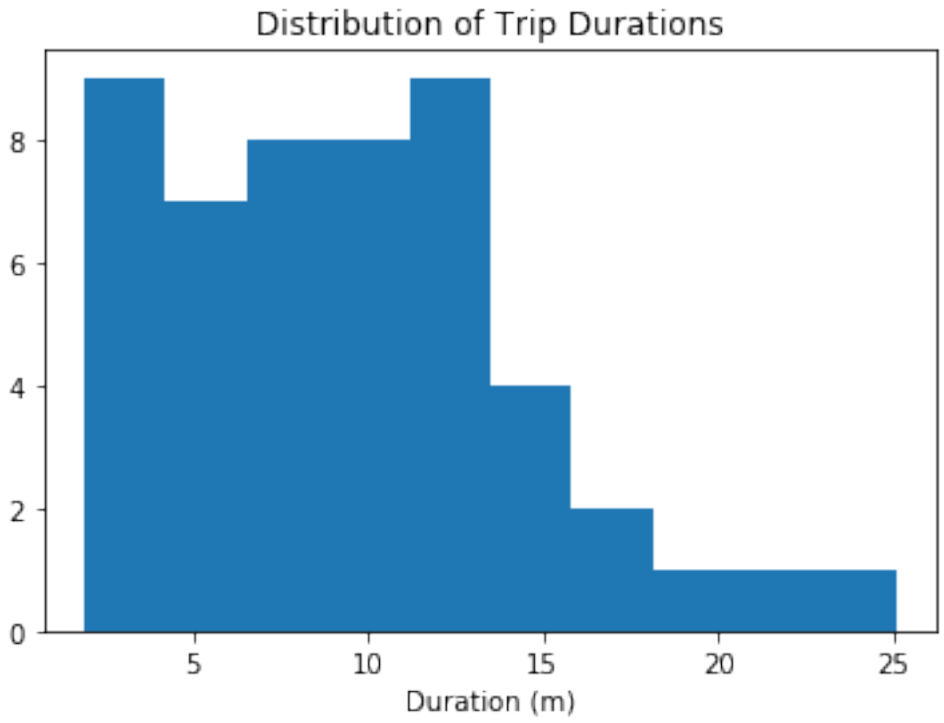
```
Newyork Subscriber takes longer rides on average: 995561.8000000041
```

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 [28]:
```
        # 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()
```

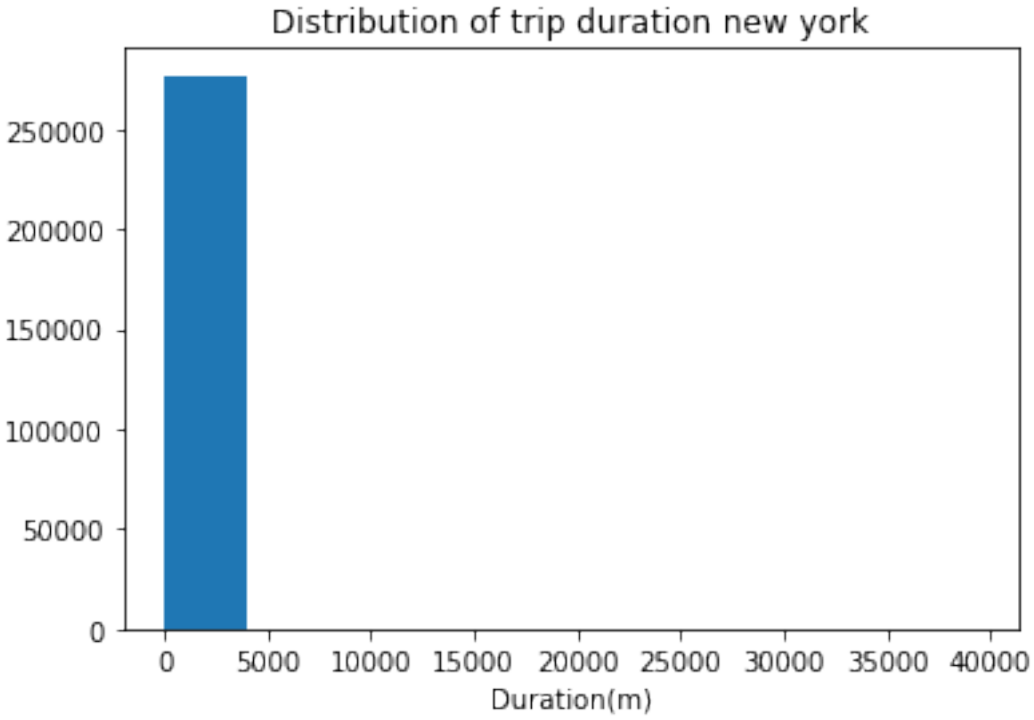


In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. 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 [29]:
```
## Use this and additional cells to collect all of the trip times as a list ##
## and then use pyplot functions to generate a histogram of trip times.     ##
import matplotlib.pyplot as plt
data_file_newyork='./data/NYC-2016-Summary.csv'
def total_time(filename):
    with open(filename,'r')as f_in:
            reader=csv.DictReader(f_in)
            time_dur=[]
            for row in reader:
                time_dur.append(float(row['duration'])/60)
    return time_dur
```

```
#print(total_time(data_file_newyork))
plt.hist(total_time(data_file_newyork))
plt.title("Distribution of trip duration new york")
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: Distribution of customer trip duration Newyork:-

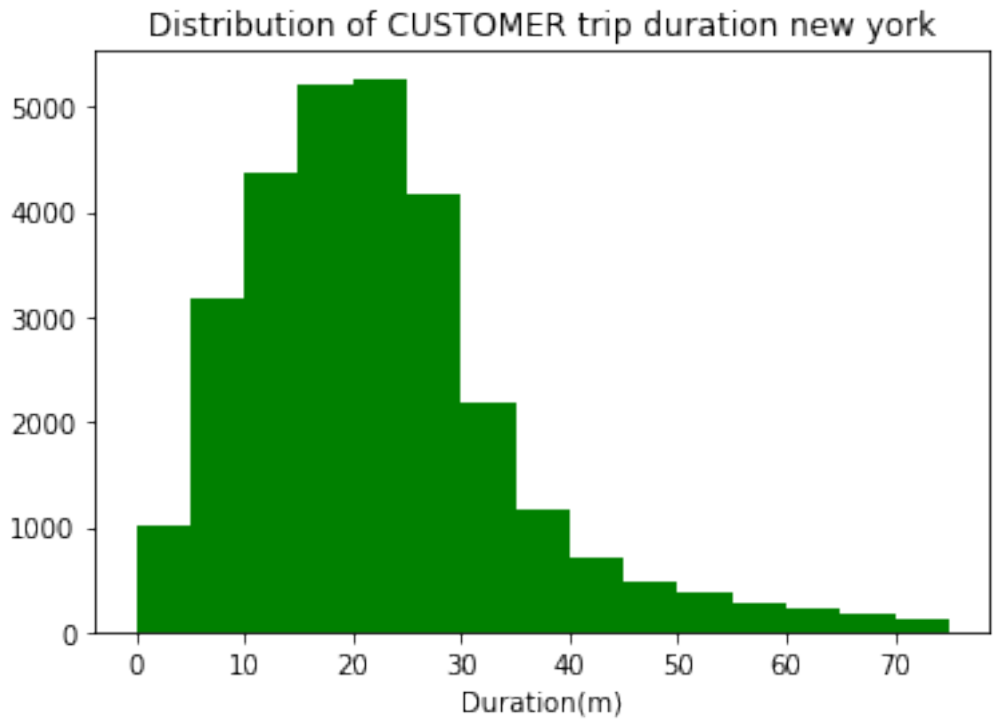
Peak of distribution is between to 15 to 25 min .It is not symmetric graph.Most of the customer are using bicycle for 5 min to 35 min.Very few customer are using bicycle for longer duration that is greater than 40 min.It means most of the customer are using bicycle for shorter duration.

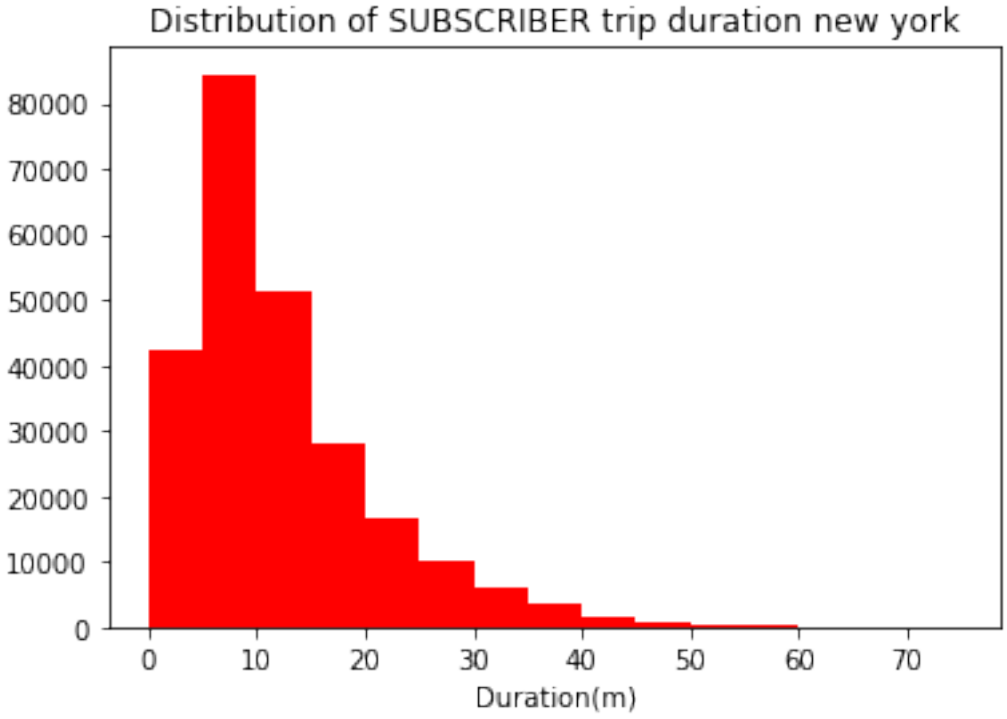
Distribution of Subscriber trip duration Newyork:-

Peak of distribution is between 0 to 20 min .It is not symmetric graph.Most of the customer are

using bicycle for 1 min to 15 min.Very few customer are using bicycle for longer duration that is greater than 30 min.It means most of the Subscriber are using bicycle for shorter duration.

In [30]:
```
## Use this and additional cells to answer Question 5. ##
#%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
data_file_newyork='./data/NYC-2016-Summary.csv'

def total_duration(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        dur_min=0
        total_dur_sub=[]
        total_dur_cus=[]
        #count the total duration in min(newyrork,chicago in sec,washington ms)
        for row in reader:
            dur_min=row['duration']
            if(row['user_type']=='Customer'):
                total_dur_cus.append(float(dur_min)/60)
            else:
                total_dur_sub.append(float(dur_min)/60)

    #print(total_dur_cus,total_dur_sub)
    return total_dur_cus,total_dur_sub
total_dur_cus,total_dur_sub=total_duration(data_file_newyork)
#bin_subscribers=int(float(max(total_dur_sub))/5)
#bin_customers=int(float(max(total_dur_cus))/5)
#print(total_dur_cus)
#plt.clf()

plt.hist(total_dur_cus,bins=15,range=[0,75],color='green')
plt.title("Distribution of CUSTOMER trip duration new york")
plt.xlabel("Duration(m)")
plt.show()

#plt.clf()

plt.hist(total_dur_sub,color='red',bins=15,range=[0,75])
plt.title("Distribution of SUBSCRIBER trip duration new york")
plt.xlabel("Duration(m)")
plt.show()
```


Distribution of CUSTOMER trip duration new york
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
60
70
Duration(m)


Distribution of SUBSCRIBER trip duration new york
80000
70000
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
70
Duration(m)

Performing Your Own Analysis

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

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

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

Question 6: Continue the investigation by exploring another question that could be answered by the data available. Document the question you want to explore below. Your investigation should involve at least two variables and should compare at least two groups. You should also use at least one visualization as part of your explorations.

Answer: Distribution of Customer time NewYork :-

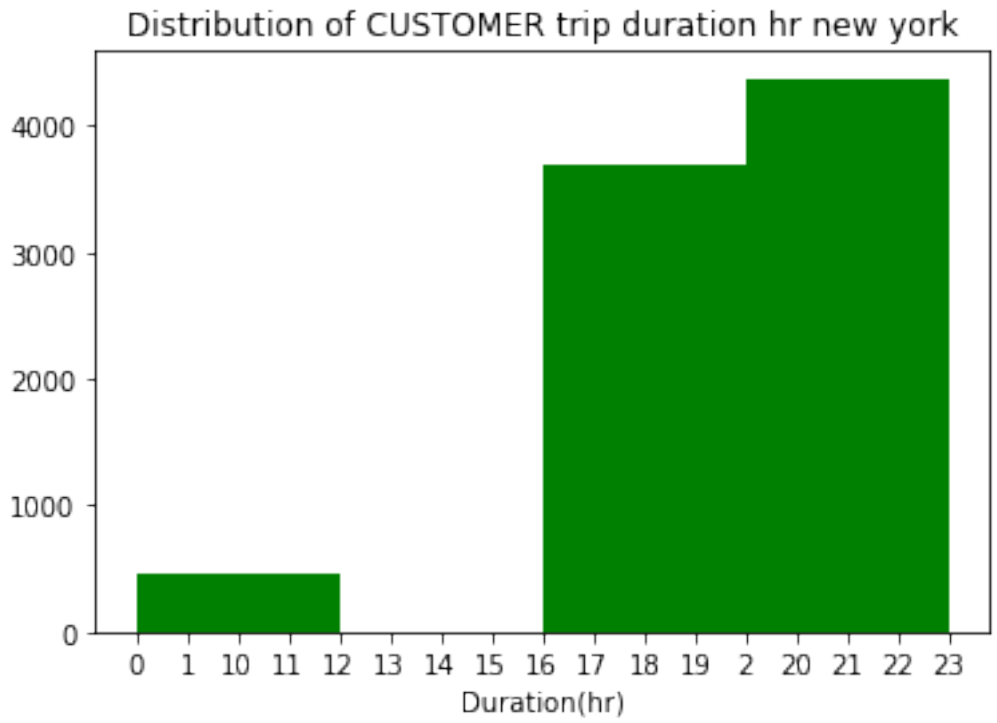
We can see clearly from the graph most of the short term member are using from 16:00 hr to 23:00 hr .It means in evening and in night more number of cycle required.Speacilly between 20:00 hr to 23:00 more number of cycle required for customer in Newyork. Less number of cycle required between 00:00 hr to 12:00 compared to evening.Between 12:00 hr to 16:00 hr no cycle required .We should have some stock of cycle also present during this time in case of emergency somebody may need.

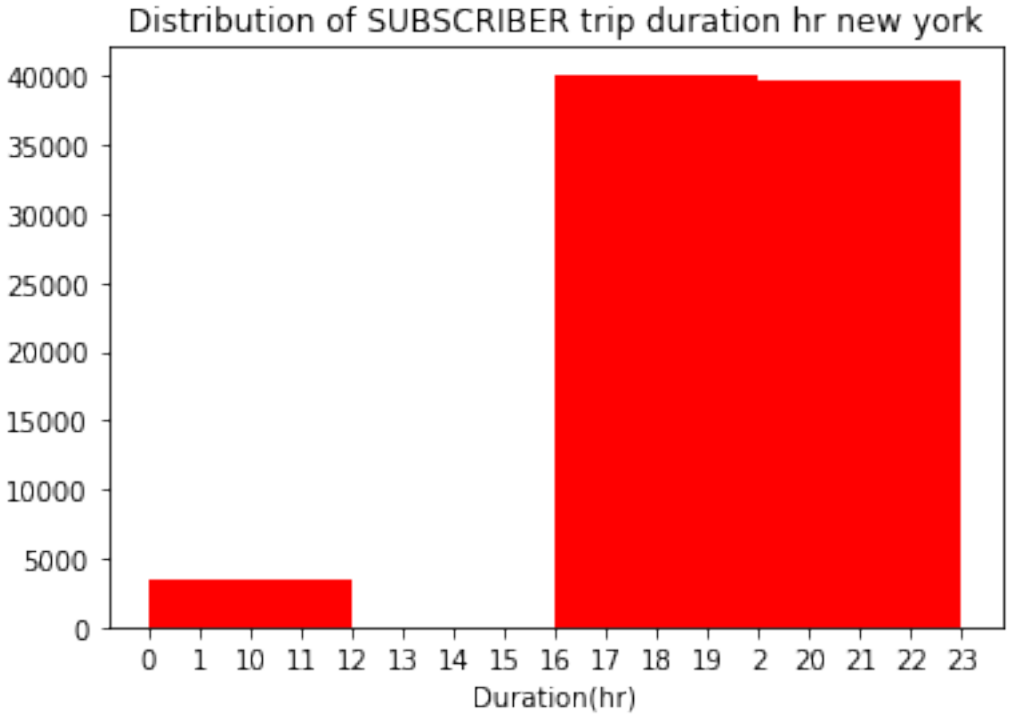
Distribution of Subscriber time NewYork:-

We can see clearly from the graph most of the short term member are using from 16:00 hr to 23:00 hr .It means in evening and in night more number of cycle required.In evening and night similar demand of cycle so we should have equal stock during evening and night.Less number of cycle required between 00:00 hr to 12:00 compared to evening.Between 12:00 hr to 16:00 hr no cycle required .We should have some stock of cycle also present during this time in case of emergency somebody may need.

```
In [31]: ## Use this and additional cells to continue to explore the dataset. ##
         ## Once you have performed your exploration, document your findings  ##
         ## in the Markdown cell above.
         ##
         ## Use this and additional cells to answer Question 5. ##
         #%matplotlib notebook
```

```
import matplotlib.pyplot as plt
import numpy as np
data_file_newyork='./data/NYC-2016-Summary.csv'

def total_hr_duration(filename):
    with open(filename,'r') as f_in:
        reader=csv.DictReader(f_in)
        hr=0
        hr_distribution_sub=[]
        hr_distribution_cus=[]
        #count the total month distributation(newyrork)
        for row in reader:
            hr=row['hour']
            if(row['user_type']=='Customer'):
                hr_distribution_cus.append(hr)
            else:
                hr_distribution_sub.append(hr)

    #print(total_dur_cus,total_dur_sub)
    return hr_distribution_sub,hr_distribution_cus
hr_distribution_sub,hr_distribution_cus=total_hr_duration(data_file_newyork)

plt.hist(hr_distribution_cus,color='green',bins=4,range=[0,23])
plt.title("Distribution of CUSTOMER trip duration hr new york")
plt.xlabel("Duration(hr)")
plt.show()

#plt.clf()

plt.hist(hr_distribution_sub,color='red',bins=4,range=[0,23])
plt.title("Distribution of SUBSCRIBER trip duration hr new york")
plt.xlabel("Duration(hr)")
plt.show()
```


Distribution of CUSTOMER trip duration hr new york
4000
3000
2000
1000
0
0 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23
Duration(hr)


Distribution of SUBSCRIBER trip duration hr new york
40000
35000
30000
25000
20000
15000
10000
5000
0
0 1 10 11 12 13 14 15 16 17 18 19 2 20 21 22 23
Duration(hr)

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: Twitter Sentiment analysis

I would like to find positive and negative tweets based on the words present in tweets.I will first collect a list of positive words and list of negative words .I will train my data through SVM (support vector machine)algorithm.

Eg:-

tweet1:-Today is awesome weather.

The word awesome determine is positive tweet.

tweet2:-The food is bad here

The word bad determine ,it is negative tweet.

Some sarcasm is really difficult to predict.

I will take the tweet as input and pass it through my algorithm.It will determine how much percentage ,it is positive tweet

> **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 [32]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Bike_Share_Analysis.ipynb'])
```

```
Out[32]: 0
```