Bike_Share_Analysis

February 11, 2018

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

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, I will perform an exploratory analysis on data provided by Motivate, a bike-share system provider for many major cities in the United States. I will compare the system usage between three large cities: New York City, Chicago, and Washington, DC. I 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, I should start by asking questions I might want to understand about the bike share data.

Questions i wll answer in this project : what is the freqency of 3 kinds of users' bike use? what is the most popular use time/ month for three city ? How much is per users' cost for three different city?

Data Collection and Wrangling

Now it's time to collect and explore our data. In this project, I 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. **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).

```
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 ##

                first_trip = next(trip_reader)

                ## TODO: Use the pprint library to print the first trip. ##
                ## see https://docs.python.org/3/library/pprint.html     ##
                pprint(first_trip)

            # output city name and first trip for later testing
            return (city, first_trip)

        # list of files for each city
        data_files = ['./data/NYC-CitiBike-2016.csv',
```

```
                    './data/Chicago-Divvy-2016.csv',
                    './data/Washington-CapitalBikeshare-2016.csv',]

        # print the first trip from each file, store in dictionary
        example_trips = {}
        for data_file in data_files:
            city, first_trip = print_first_point(data_file)
            example_trips[city] = first_trip
```

```
City: NYC
OrderedDict([('tripduration', '839'),
             ('starttime', '1/1/2016 00:09:55'),
             ('stoptime', '1/1/2016 00:23:54'),
             ('start station id', '532'),
             ('start station name', 'S 5 Pl & S 4 St'),
             ('start station latitude', '40.710451'),
             ('start station longitude', '-73.960876'),
             ('end station id', '401'),
             ('end station name', 'Allen St & Rivington St'),
             ('end station latitude', '40.72019576'),
             ('end station longitude', '-73.98997825'),
             ('bikeid', '17109'),
             ('usertype', 'Customer'),
             ('birth year', ''),
             ('gender', '0')])

City: Chicago
OrderedDict([('trip_id', '9080545'),
             ('starttime', '3/31/2016 23:30'),
             ('stoptime', '3/31/2016 23:46'),
             ('bikeid', '2295'),
             ('tripduration', '926'),
             ('from_station_id', '156'),
             ('from_station_name', 'Clark St & Wellington Ave'),
             ('to_station_id', '166'),
             ('to_station_name', 'Ashland Ave & Wrightwood Ave'),
             ('usertype', 'Subscriber'),
             ('gender', 'Male'),
             ('birthyear', '1990')])

City: Washington
OrderedDict([('Duration (ms)', '427387'),
             ('Start date', '3/31/2016 22:57'),
             ('End date', '3/31/2016 23:04'),
             ('Start station number', '31602'),
             ('Start station', 'Park Rd & Holmead Pl NW'),
```

```
('End station number', '31207'),
('End station', 'Georgia Ave and Fairmont St NW'),
('Bike number', 'W20842'),
('Member Type', 'Registered')])
```

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 [70]:
```
datum ={}
```

```
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 =='NYC':
        duration = int(datum['tripduration'])/60
    elif city =='Chicago':
        duration = int(datum['tripduration'])/60

    else:
        duration = int(datum['Duration (ms)'])/60000

    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 [71]:
```
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
    """
    if city =='NYC':
        dt = datetime.strptime((datum['starttime']), "%m/%d/%Y %H:%M:%S")
        month = int(dt.strftime("%m"))
        hour = int(dt.strftime("%H"))
        day_of_week = dt.strftime("%A")

    elif city =='Chicago':
        dt2 = datetime.strptime((datum['starttime']), "%m/%d/%Y %H:%M")
        month = int(dt2.strftime("%m"))
        hour = int(dt2.strftime("%H"))
```

```
        day_of_week = dt2.strftime("%A")

    else:
        dt3 = datetime.strptime((datum['Start date']), "%m/%d/%Y %H:%M")
        month = int(dt3.strftime("%m"))
        hour = int(dt3.strftime("%H"))
        day_of_week = dt3.strftime("%A")


    return (month, hour, day_of_week)



# Some tests to check that your code works. There should be no output if all of
# the assertions pass. The `example_trips` dictionary was obtained from when
# you printed the first trip from each of the original data files.
tests = {'NYC': (1, 0, 'Friday'),
         'Chicago': (3, 23, 'Thursday'),
         'Washington': (3, 22, 'Thursday')}

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

In [73]:
```
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 =='NYC':
        user_type = str(datum['usertype'])

    elif city =='Chicago':
        user_type = str(datum['usertype'])

    else:
        if datum['Member Type'] == 'Registered':
            user_type = 'Subscriber'
        else:
            user_type = 'Customer'

    return user_type
```

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

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

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

In [74]:
```
import csv
def condense_data(in_file, out_file, city):
    """
    This function takes full data from the specified input file
    and writes the conde nsed 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 [68]: # 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')])
```

Exploratory Data Analysis

Now that you have the data collected and wrangled, you're ready to start exploring the data.

In this section you will write some code to compute descriptive statistics from the data. You will also be introduced to the `matplotlib` library to create some basic histograms of the data.

Statistics

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

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

Answer: NYC has the max number of trips, the number is 276798 NYC has the highest proportion of trips made by subscribers, the pecentage is 88.84% Chicago has the highest proportion of trips made by customer, the pecentage is 23.77%

```
In [75]: 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
                 len_subs_ride =0
                 len_cust_ride = 0

                 # tally up ride types
                 for row in reader:
                     if row['user_type'] == 'Subscriber':
                         n_subscribers += 1
                         len_subs_ride += float(row['duration'])
                     else:
                         n_customers += 1
                         len_cust_ride += float(row['duration'])

                 # compute total number of rides
                 n_total = n_subscribers + n_customers
                 pct_subscribers = n_subscribers/n_total
                 pct_customers = n_customers/n_total

                 avg_subs_ride = len_subs_ride/n_subscribers
                 avg_cust_ride = len_cust_ride/n_customers


                 # return tallies as a tuple
```

```
    return(n_subscribers, n_customers, n_total,pct_subscribers,pct_customers,avg_su
```

In [76]:
```
## 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_file1 = './data/Washington-2016-Summary.csv'
data_file2 = './data/NYC-2016-Summary.csv'
data_file3 = './data/Chicago-2016-Summary.csv'

list_n_trips= {"Washington":number_of_trips(data_file1)[2],
               "NYC": number_of_trips(data_file2)[2],"Chicago": number_of_trips(data_fi
max_n_trips = max(list_n_trips, key=list_n_trips.get)
print('{} has the max number of trips, the number is {}'.format(max_n_trips, list_n_tri


list_pcts= {"Washington":number_of_trips(data_file1)[3],
               "NYC": number_of_trips(data_file2)[3],"Chicago": number_of_trips(data_fi
max_pcts =max(list_pcts, key=list_pcts.get)
print('{} has the highest proportion of trips made by subscribers, the pecentage is {:.


list_pctc= {"Washington":number_of_trips(data_file1)[4],
               "NYC": number_of_trips(data_file2)[4],"Chicago": number_of_trips(data_fi
max_pctc =max(list_pctc, key=list_pctc.get)
print('{} has the highest proportion of trips made by customer, the pecentage is {:.2%}
```

```
NYC has the max number of trips, the number is 276798
NYC has the highest proportion of trips made by subscribers, the pecentage is 88.84%
Chicago has the highest proportion of trips made by customer, the pecentage is 23.77%
```

Question 4b: Bike-share systems are designed for riders to take short trips. Most of the time, users are allowed to take trips of 30 minutes or less with no additional charges, with overage charges made for trips of longer than that duration. What is the average trip length for each city? What proportion of rides made in each city are longer than 30 minutes?

Answer: The average trip length for Washington is 18.93(min),the proportion of rides made are longer than 30 minutes is 10.84% The average trip length for NYC is 15.81(min),the proportion of rides made are longer than 30 minutes is 7.30% The average trip length for Chicago is 16.56(min),the proportion of rides made are longer than 30 minutes is 8.33%

In [77]:
```
## 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 length_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_shortride = 0
        n_longride= 0
        len_shortride =0
        len_longride=0

        # tally up duration
        for row in reader:
            if float(row['duration']) <= 30:
                n_shortride += 1
                len_shortride += float(row['duration'])
            else:
                n_longride += 1
                len_longride +=float(row['duration'])

        # compute total number of rides
        n_total = n_shortride + n_longride
        len_total = len_shortride + len_longride

        avg_len = len_total/n_total
        pct_longride = n_longride/n_total
        pct_shortride = n_shortride/n_total

        # return tallies as a tuple
        return(len_total, n_total,avg_len,pct_longride,pct_shortride)


data_file1 = './data/Washington-2016-Summary.csv'
data_file2 = './data/NYC-2016-Summary.csv'
data_file3 = './data/Chicago-2016-Summary.csv'

list_len= {"Washington":length_of_trips(data_file1)[2],
               "NYC": length_of_trips(data_file2)[2],"Chicago": length_of_trips(data_fi

print('The average trip length for {} is {:.2f}(min),the proportion of rides made are l
      format(list(list_len.keys())[list(list_len.values()).index(length_of_trips(data_f
             length_of_trips(data_file1)[2],
             length_of_trips(data_file1)[3]))
```

```
        print('The average trip length for {} is {:.2f}(min),the proportion of rides made are l
              format(list(list_len.keys())[list(list_len.values()).index(length_of_trips(data_f
                     length_of_trips(data_file2)[2],
                     length_of_trips(data_file2)[3]))

        print('The average trip length for {} is {:.2f}(min),the proportion of rides made are l
              format(list(list_len.keys())[list(list_len.values()).index(length_of_trips(data_f
                     length_of_trips(data_file3)[2],
                     length_of_trips(data_file3)[3]))
```

```
The average trip length for Washington is 18.93(min),the proportion of rides made are longer tha
The average trip length for NYC is 15.81(min),the proportion of rides made are longer than 30 mi
The average trip length for Chicago is 16.56(min),the proportion of rides made are longer than 3
```

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

Answer: The customer in Washington takes longer rides on average.The average Subscriber trip duration is 12.53 (min), The average customer trip duration is 41.68 (min)

In [79]:
```
## 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?                                                ##

data_file = './data/Washington-2016-Summary.csv'

if number_of_trips(data_file)[5] > number_of_trips(data_file)[6]:
    print('The subscriber in Washington takes longer rides on average. The average Subs
else:
    print('The customer in Washington takes longer rides on average.The average Subscri
```

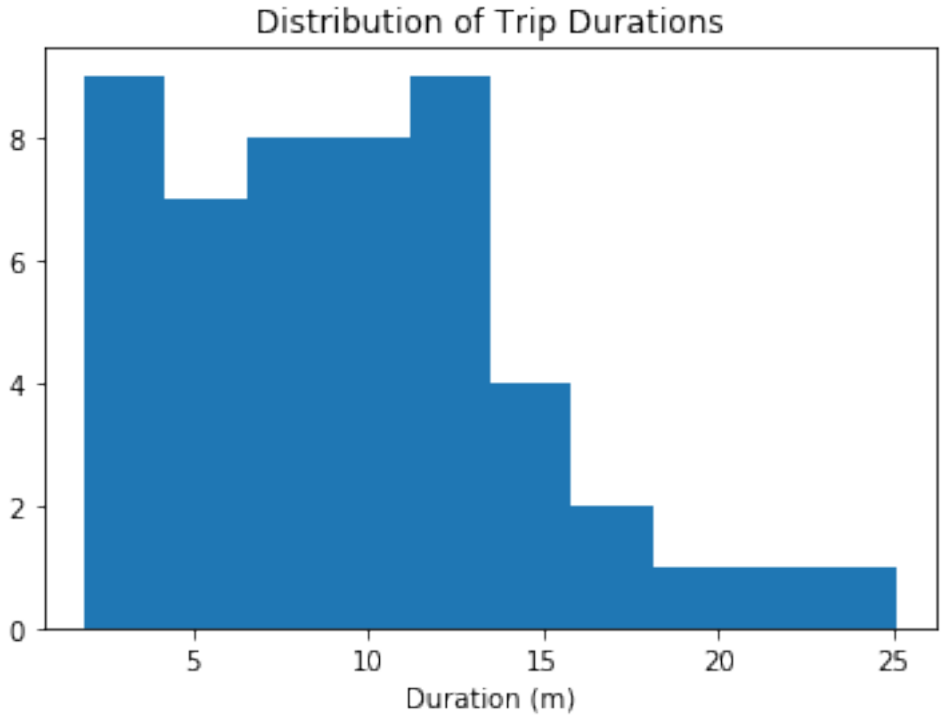
```
The customer in Washington takes longer rides on average.The average Subscriber trip duration is
```

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 [3]:
```
# 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()
```

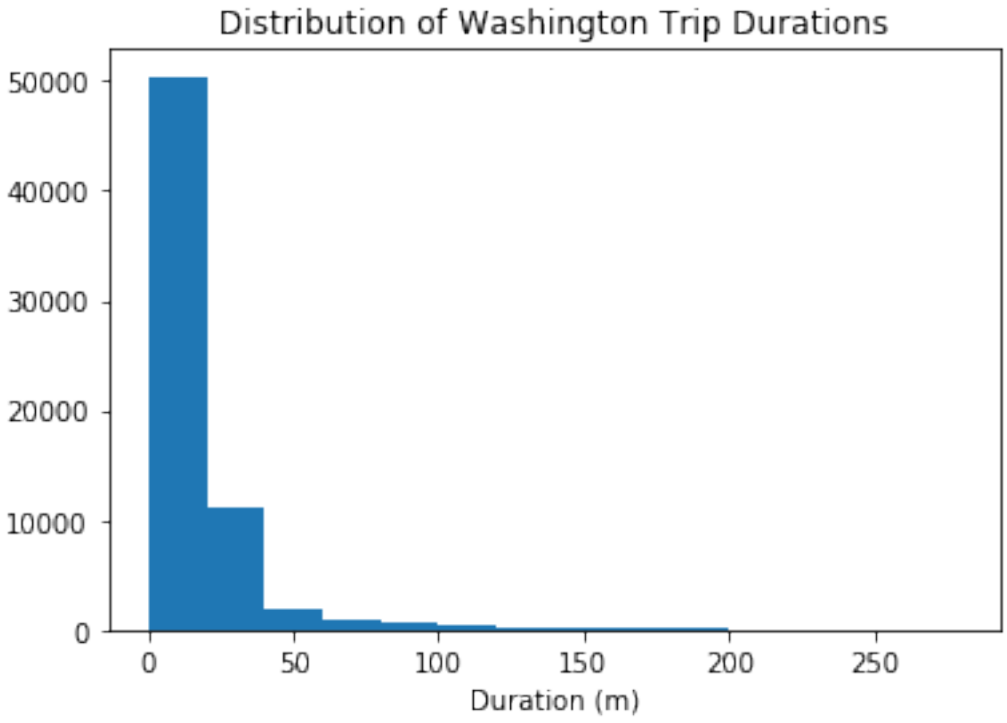


In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. 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 [80]: ## 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.    ##

tripdata=[]

def list_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)

        # tally up duration
        for row in reader:
            tripdata.append(float(row['duration']))
        return tripdata

data_file = './data/Washington-2016-Summary.csv'
bins =[0,20,40,60,80,100,120,140,160,180,200,220,240,260,280]

plt.hist(list_trips(data_file),bins)
plt.title('Distribution of Washington Trip Durations')
plt.xlabel('Duration (m)')
plt.legend()
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 Washington Customer Trip Durations, the peak is in 5-10 mins bin. for Washington Customer Trip Durations, the peak is in 15-20 mins bin. Both histogram are right skewed.

In [67]:
```
## Use this and additional cells to answer Question 5. ##

substriber_data=[]
customer_data=[]

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

        # tally up ride types
        for row in reader:
            if row['user_type'] == 'Subscriber':
                substriber_data.append(float(row['duration']))
            else:
                customer_data.append(float(row['duration']))
        return (substriber_data,customer_data)

data_file = './data/Washington-2016-Summary.csv'
bins =[0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75]

plt.hist(list_trips(data_file)[0],bins,histtype='bar',rwidth=0.8)
plt.title('Distribution of Washington Subscriber Trip Durations')
plt.xlabel('Duration (m)')
plt.legend()
plt.show()

plt.hist(list_trips(data_file)[1],bins,histtype='bar',rwidth=0.8)
plt.title('Distribution of Washington Customer Trip Durations')
plt.xlabel('Duration (m)')
```

```
plt.legend()
plt.show()
```