2016 US Bike Share Activity Snapshot

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 (https://www.motivateco.com/), 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:

What are some differences between the groups that use the bike share data

- Frequent users
- Occasional users
- Visitors

- Time that they use the bike
- How far they ride
- Popular pickup and dropoff locations for different groups

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 (https://www.citibikenyc.com/system-data)
- Chicago (Divvy): Link (https://www.divvybikes.com/system-data)
- Washington, DC (Capital Bikeshare): Link (https://www.capitalbikeshare.com/system-data)

If you visit these pages, you will notice that each city has a different way of delivering its data. Chicago updates with new data twice a year, Washington DC is quarterly, and New York City is monthly. **However, you do not need to download the data yourself.** The data has already been collected for you in the `/data/` folder of the project files. While the original data for 2016 is spread among multiple files for each city, the files in the `/data/` folder collect all of the trip data for the year into one file per city. Some data wrangling of inconsistencies in timestamp format within each city has already been performed for you. In addition, a random 2% sample of the original data is taken to make the exploration more manageable.

Question 2: However, there is still a lot of data for us to investigate, so it's a good idea to start off by looking at one entry from each of the cities we're going to analyze. Run the first code cell below to load some packages and functions that you'll be using in your analysis. Then, complete the second code cell to print out the first trip recorded from each of the cities (the second line of each data file).

> **Tip**: You can run a code cell like you formatted Markdown cells above by clicking on the cell and using the keyboard shortcut **Shift** + **Enter** or **Shift** + **Return**. Alternatively, a code cell can be executed using the **Play** button in the toolbar after selecting it. While the cell is running, you will see an asterisk in the message to the left of the cell, i.e. `In [*]:`. The asterisk will change into a number to show that execution has completed, e.g. `In [1]`. If there is output, it will show up as `Out [1]:`, with an appropriate number to match the "In" number.

In [1]:
```
## import all necessary packages and functions.
import csv # read and write csv files
from datetime import datetime # operations to parse dates
from pprint import pprint # use to print data structures like dictionari
es in
                              # a nicer way than the base print function.
```

In [2]:
```
def print_first_point(filename):
    """
    This function prints and returns the first data point (second row) f
rom
    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
{'bikeid': '17109',
 'birth year': '',
 'end station id': '401',
 'end station latitude': '40.72019576',
 'end station longitude': '-73.98997825',
 'end station name': 'Allen St & Rivington St',
 'gender': '0',
 'start station id': '532',
 'start station latitude': '40.710451',
 'start station longitude': '-73.960876',
 'start station name': 'S 5 Pl & S 4 St',
 'starttime': '1/1/2016 00:09:55',
 'stoptime': '1/1/2016 00:23:54',
 'tripduration': '839',
 'usertype': 'Customer'}

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

City: Washington
{'Bike number': 'W20842',
 'Duration (ms)': '427387',
 'End date': '3/31/2016 23:04',
 'End station': 'Georgia Ave and Fairmont St NW',
 'End station number': '31207',
 'Member Type': 'Registered',
 'Start date': '3/31/2016 22:57',
 'Start station': 'Park Rd & Holmead Pl NW',
 'Start station number': '31602'}
```

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` (https://docs.python.org/3/library/datetime.html) 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. It also turns out that there are some trips in the New York city dataset that do not have an attached user type. Since we don't have enough information to fill these values in, just leave them as-is for now.

Question 3a: Complete the helper functions in the code cells below to address each of the cleaning tasks described above.

In [3]:
```
def duration_in_mins(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (dat
um) and
    its origin city (city) and returns the trip duration in units of min
utes.

    Remember that Washington is in terms of milliseconds while Chicago a
nd 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 appr
opriate
    numeric type when making your transformations.
    see https://docs.python.org/3/library/functions.html
    """

    # Getting duration which is in seconds and changing it to minutes
    if city == "NYC" or city == "Chicago":

        duration = float(datum['tripduration'])
        duration = duration/60

    # Getting duration which is in milliseconds and changing to minutes.
    elif city == "Washington":

        duration = float(datum['Duration (ms)'])
        duration = duration/60000


    else:
        print("Error - {} - is not a recognized city!".format(city))

    return duration


# Some tests to check that your code works. There should be no output if
 all of
# the assertions pass. The `example_trips` dictionary was obtained from
 when
# you printed the first trip from each of the original data files.
tests = {'NYC': 13.9833,
         'Chicago': 15.4333,
         'Washington': 7.1231}

for city in tests:
    assert abs(duration_in_mins(example_trips[city], city) - tests[city
]) < .001
```

In [4]:
```
def time_of_trip(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (dat
um) and
    its origin city (city) and returns the month, hour, and day of the w
eek 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 info
rmation.
    see https://docs.python.org/3/library/datetime.html#strftime-and-str
ptime-behavior
    """

    # YOUR CODE HERE

    # Converting time that includes seconds - Example format of date: 1/
1/2016 00:09:55
    if city == "NYC":
        starttime_string = datum['starttime']

        starttime = datetime.strptime(starttime_string, '%m/%d/%Y %H:%
M:%S')

    # Converting time that does not include seconds - Example format of
 date: 3/31/2016 23:30
    elif city == "Washington" or city == "Chicago":
        if city == "Washington":
            starttime_string = datum['Start date']
        else:
            starttime_string = datum['starttime']

        starttime = datetime.strptime(starttime_string, '%m/%d/%Y %H:%M'
)

    # Just in case there is bad data
    else:
        print("Error - {} - is not a recognized city!".format(city))

    # Setting up return values

    month = starttime.month

    hour = starttime.hour

    day = starttime.weekday()
    #  Converting day to day of the week
    days = ["Monday","Tuesday","Wednesday","Thursday","Friday","Saturda
y","Sunday"]
    day_of_week = days[day]
```

```
    return (month, hour, day_of_week)


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

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

In [5]:
```
def type_of_user(datum, city):
    """
    Takes as input a dictionary containing info about a single trip (dat
um) 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 Ch
icago
    and NYC. NYC has some data points with a missing user type; you can
 leave
    these as they are (empty string).
    """

    # YOUR CODE HERE

    # Washington uses "Member Type" for usertype

    if city == "Washington":

        user_type = datum['Member Type']

        # Changing "Registered" to "Subscriber" to match other data

        if user_type == "Registered":
            user_type = "Subscriber"

    # Chicago and NYC use "usertype" for usertype.
    # There may be some empty strings returned for NYC

    elif city == "Chicago" or city == "NYC":

        user_type = datum['usertype']


    else:
        print("Error - {} - is not a recognized city!".format(city))


    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 (http://www.bayareabikeshare.com/open-data) before and after conversion. Make sure that your output is formatted to be consistent with the example file.

In [6]:
```
import csv

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 fo
r the
        # first row as the "fieldnames" argument
        out_colnames = ['duration', 'month', 'hour', 'day_of_week', 'use
r_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 fr
om  ##
            ## the original data dictionaries.
    ##
            ## Note that the keys for the new_point dictionary should ma
tch ##
            ## the column names set in the DictWriter object above.
    ##
            datum = row

            new_point['duration'] = duration_in_mins(datum, city)
            new_point['month'], new_point['hour'], new_point['day_of_wee
k'] = time_of_trip(datum, city)
            new_point['user_type'] = type_of_user(datum, city)

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

            trip_writer.writerow(new_point)
```

In [7]:
```
# Run this cell to check your work
city_info = {'Washington': {'in_file': './data/Washington-CapitalBikesha
re-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
{'day_of_week': 'Thursday',
 'duration': '7.123116666666666',
 'hour': '22',
 'month': '3',
 'user_type': 'Subscriber'}

City: NYC
{'day_of_week': 'Friday',
 'duration': '13.983333333333333',
 'hour': '0',
 'month': '1',
 'user_type': 'Customer'}

City: Chicago
{'day_of_week': 'Thursday',
 'duration': '15.433333333333334',
 'hour': '23',
 'month': '3',
 'user_type': 'Subscriber'}
```

> **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:

- The city with the largest number of total trips is NYC.
- The city with the largest proportion of Subscriber trips is NYC.
- The city with the largest proportion of Customer trips is Chicago.

Assuming user_type with no value is "Customer"

In [8]:
```
def number_of_trips(filename):
    """
    This function reads in a file with trip data and reports the number
 of
    trips made by subscribers, customers, and total overall.
    """
    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
        total_check = 0

        # tally up ride types
        for row in reader:
            if row['user_type'] == 'Subscriber':
                n_subscribers += 1
            elif row['user_type'] == 'Customer' or row['user_type'] ==
'Casual' or row['user_type'] == '':
                n_customers += 1
            total_check += 1

        # compute total number of rides
        n_total = n_subscribers + n_customers
        if n_total != total_check:
            print ("Something is wrong.  Subscribers + Customers - {} -
 doesn\'t add up to the total:  {}".format(n_total,total_check))

        # return tallies as a tuple
        return(n_subscribers, n_customers, n_total)
```

In [9]:
```
'''
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 custom
ers?
'''

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

# Finding out the key stats for each city
city_data = {}

for city in city_info:
    print(city + ':')
    n_subscribers,n_customers,n_total = number_of_trips(city_info[city])

    print("There are {} subscribers".format(n_subscribers))
    print("There are {} customers".format(n_customers))
    print("There are {} total".format(n_total))

    subscriber_proportion = n_subscribers/n_total
    customer_proportion = n_customers/n_total

    city_data[city] = {'total': n_total, 'subscriber_proportion' : subsc
riber_proportion, 'customer_proportion' : customer_proportion}



# Calculating the deired stats to answer the question

max_total=0
max_subscriber_proportion=0
max_customer_proportion=0

for city in city_info:
    if max_total < city_data[city]['total']:
        max_total = city_data[city]['total']
        largest_total_city = city
    if max_subscriber_proportion < city_data[city]['subscriber_proportio
n']:
        max_subscriber_proportion = city_data[city]['subscriber_proporti
on']
        largest_subscriber_proportion_city = city
    if max_customer_proportion < city_data[city]['customer_proportion']:
        max_customer_proportion = city_data[city]['customer_proportion']
        largest_customer_proportion_city = city

# Printing out the answer

print("The city with the largest number of total trips is {}.".format(la
rgest_total_city))
print("The city with the largest proportion of Subscriber trips is {}.".
format(largest_subscriber_proportion_city))
print("The city with the largest proportion of Customer trips is {}.".fo
```

```
rmat(largest_customer_proportion_city))
```

```
Washington:
There are 51753 subscribers
There are 14573 customers
There are 66326 total
NYC:
There are 245896 subscribers
There are 30902 customers
There are 276798 total
Chicago:
There are 54982 subscribers
There are 17149 customers
There are 72131 total
The city with the largest number of total trips is NYC.
The city with the largest proportion of Subscriber trips is NYC.
The city with the largest proportion of Customer trips is Chicago.
```

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.
    ##

# SEE NEW CELL ABOVE FOR ANSWER

data_file = './examples/BayArea-Y3-Summary.csv'
print(number_of_trips(data_file))
```

```
(5666, 633, 6299)
```

> **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:

Chicago:

- The average trip length for Chicago is : 16.6
- The proportion of trips over thirty minutes in Chicago is : 8.3%

NYC:

- The average trip length for NYC is : 15.8
- The proportion of trips over thirty minutes in NYC is : 7.3%

Washington:

- The average trip length for Washington is : 18.9
- The proportion of trips over thirty minutes in Washington is : 10.8%

In [127]:
```
## 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 duration_numbers(filename,users='all'):
    if users != 'all' and users!= 'Subscriber' and users != 'Customer':
        print('users must be set to all,Subscriber or Customer')


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

        # initialize count variables
        number_of_trips = 0
        total_trips = 0
        total_number_of_minutes = 0
        number_of_trips_over_thirty_minutes = 0

        # tally up ride durations

        for row in reader:
            total_trips += 1
            if (row['user_type'] == 'Subscriber' and users ==  'Subscrib
er') or users == 'all':
                number_of_trips += 1
                total_number_of_minutes += float(row['duration'])
                if float(row['duration']) > 30:
                    number_of_trips_over_thirty_minutes += 1
            elif (row['user_type'] != 'Subscriber' and users == 'Custome
r'):
                number_of_trips += 1
                total_number_of_minutes += float(row['duration'])
                if float(row['duration']) > 30:
                    number_of_trips_over_thirty_minutes += 1



        average_trip_duration = round(total_number_of_minutes/number_of_
trips,1)
        proportion_of_trips_over_thirty = round((number_of_trips_over_th
irty_minutes/number_of_trips) * 100, 1)


    return (average_trip_duration, proportion_of_trips_over_thirty)
```

In [128]:
```
'''
Question 4b: Bike-share systems are designed for riders to take short tr
ips.
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 tha
n that duration.
What is the average trip length for each city? What proportion of rides
 made
in each city are longer than 30 minutes?
'''
#city = "bay area"
#data_file = './examples/BayArea-Y3-Summary.csv'

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

# Reporting the key stats for each city

for city in city_info:

    average_trip_duration, proportion_of_trips_over_thirty = duration_nu
mbers(city_info[city])

    print(city + ":")
    print("The average trip length for {} is : {}".format(city,average_t
rip_duration))
    print("The proportion of trips over thirty minutes in {} is : {}%".f
ormat(city,proportion_of_trips_over_thirty))
```

```
Washington:
The average trip length for Washington is : 18.9
The proportion of trips over thirty minutes in Washington is : 10.8%
NYC:
The average trip length for NYC is : 15.8
The proportion of trips over thirty minutes in NYC is : 7.3%
Chicago:
The average trip length for Chicago is : 16.6
The proportion of trips over thirty minutes in Chicago is : 8.3%
```

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:

Washington: Customers take longer rides on average than Subscribers in Washington. The average trip length for Subscribers in Washington is : 12.5 The average trip length for Customers in Washington is : 41.7

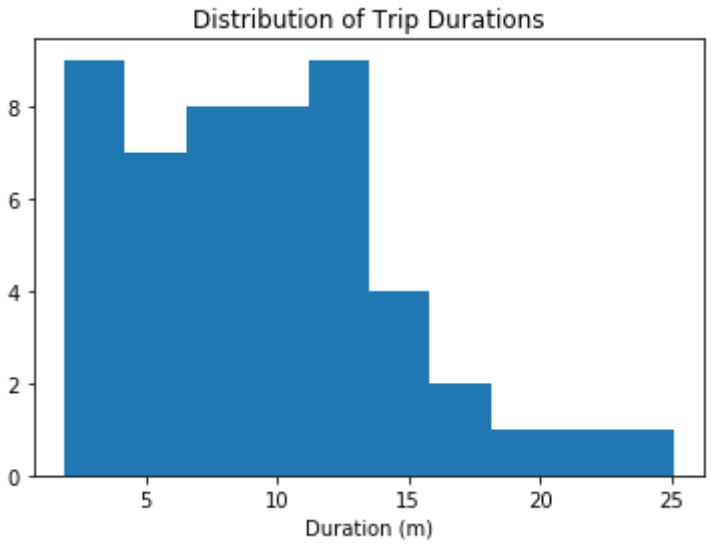
In [129]:
```
## Use this and additional cells to answer Question 4c. If you have    #
#
## not done so yet, consider revising some of your previous code to    #
#
## make use of functions for reusability.                              #
#
##                                                                     #
#
## 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? ##

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

# Modified definition above to return trip duration based on subscriber
 or customer

# city_info = {'Bay Area': './examples/BayArea-Y3-Summary.csv'}

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


for city in city_info:

    subscriber_average_trip_duration, subscriber_proportion_of_trips_ove
r_thirty = duration_numbers(city_info[city],'Subscriber')
    customer_average_trip_duration, customer_proportion_of_trips_over_th
irty = duration_numbers(city_info[city],'Customer')
    print(city + ":")
    print("The average trip length for Subscribers in {} is : {}".format
(city,subscriber_average_trip_duration))
    print("The average trip length for Customers in {} is : {}".format(c
ity,customer_average_trip_duration))
```

```
Washington:
The average trip length for Subscribers in Washington is : 12.5
The average trip length for Customers in Washington is : 41.7
NYC:
The average trip length for Subscribers in NYC is : 13.7
The average trip length for Customers in NYC is : 32.8
Chicago:
The average trip length for Subscribers in Chicago is : 12.1
The average trip length for Customers in Chicago is : 31.0
```

Visualizations

The last set of values that you computed should have pulled up an interesting result. While the mean trip time for Subscribers is well under 30 minutes, the mean trip time for Customers is actually above 30 minutes! It will be interesting for us to look at how the trip times are distributed. In order to do this, a new library will be introduced here, `matplotlib`. Run the cell below to load the library and to generate an example plot.

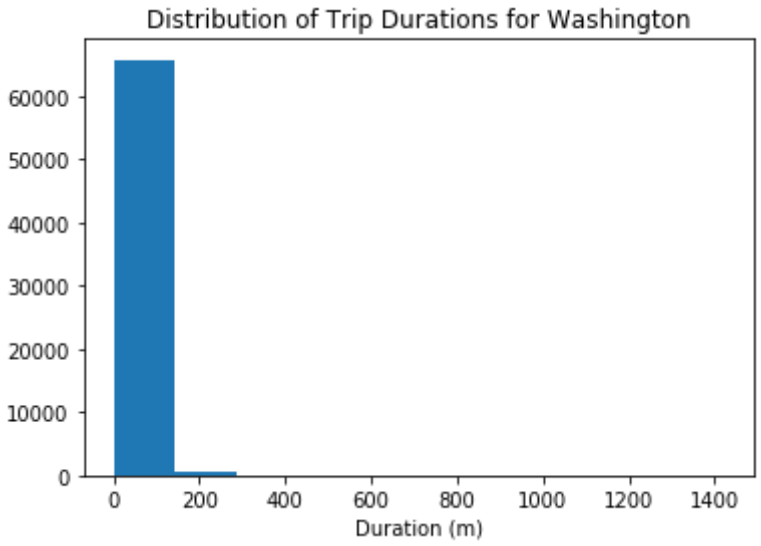
In [14]:
```
# load library
import matplotlib.pyplot as plt

# 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, 1
4.33,
        19.02, 21.53,  3.90,  7.97,  2.62,  2.67,  3.08, 14.40, 12.90,
7.83,
        25.12,  8.30,  4.93, 12.43, 10.60,  6.17, 10.88,  4.78, 15.15,
3.53,
         9.43, 13.32, 11.72,  9.85,  5.22, 15.10,  3.95,  3.17,  8.78,
1.88,
         4.55, 12.68, 12.38,  9.78,  7.63,  6.45, 17.38, 11.90, 11.52,
8.63,]
plt.hist(data)
plt.title('Distribution of Trip Durations')
plt.xlabel('Duration (m)')
plt.show()
```



In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. The `.title()` and `.xlabel()` functions provide some labeling for plot context.

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

In [15]:
```
## Use this and additional cells to collect all of the trip times as a l
ist ##
## and then use pyplot functions to generate a histogram of trip times.
    ##

# Create a list with durations from file

def get_list(filename):
    count = 0
    subscriber_durations = []
    customer_durations = []
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)
        for row in reader:
            count += 1
            item = round(float(row['duration']),2)
            if row['user_type'] == 'Subscriber':
                subscriber_durations.append(item)
            else:
                customer_durations.append(item)

            city_durations = customer_durations + subscriber_durations

    return subscriber_durations, customer_durations, city_durations

# Define city and file info

city = 'Washington'
city_datafile = './data/Washington-2016-Summary.csv'
subscriber_durations, customer_durations, city_durations = get_list(city
_datafile)

%matplotlib inline

# Return Graph

plt.hist(city_durations)
plt.title('Distribution of Trip Durations for {}'.format(city))
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] (https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html#matplotlib.pyplot.hist).

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?

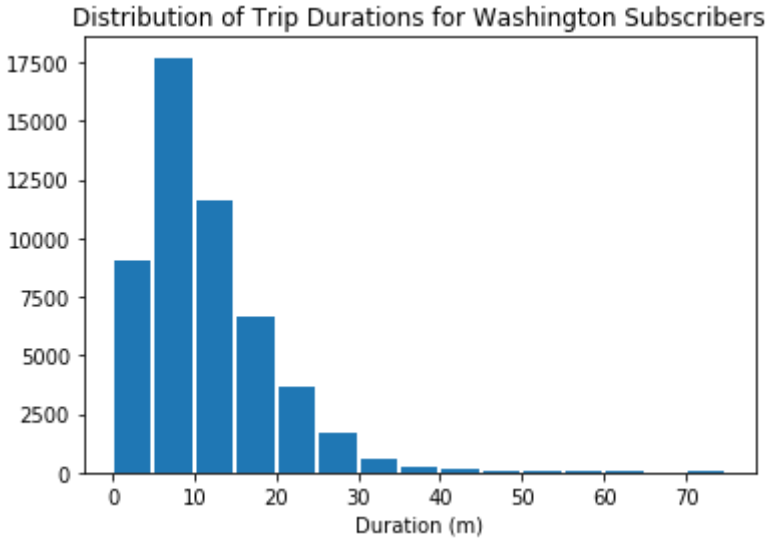
Answer:

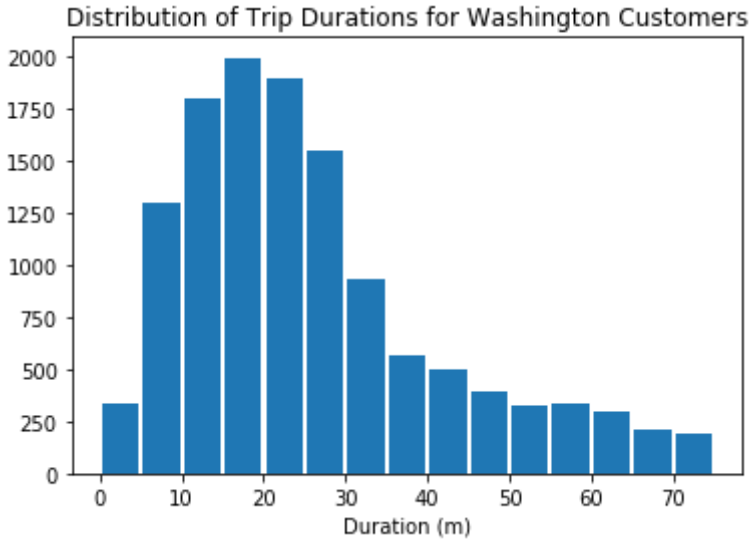
City: Washington

- For Subscribers max is between 5-10 minutes
- For Customers max is between 15-20 minutes

Description: A scewed right distribution (both)

In [16]:
```
## Use this and additional cells to answer Question 5. ##
import numpy as np

def plot_data_graph(data,header):
    bins1=np.arange(0, 80, 5)
    plt.hist(data,bins=bins1,range=(0, 75),rwidth=0.9)

    plt.title('Distribution of Trip Durations for {}'.format(header))
    plt.xlabel('Duration (m)')
    plt.show()

city = 'Washington'
city_datafile = './data/Washington-2016-Summary.csv'
subscriber_durations, customer_durations, city_durations = get_list(city
_datafile)

%matplotlib inline

# Return Graph for subscriber durations

plot_data_graph(subscriber_durations, "Washington Subscribers")
plot_data_graph(customer_durations, "Washington Customers")
```





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 (https://matplotlib.org/devdocs/api/pyplot_summary.html). 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()` (https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.bar.html#matplotlib.pyplot.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:

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?

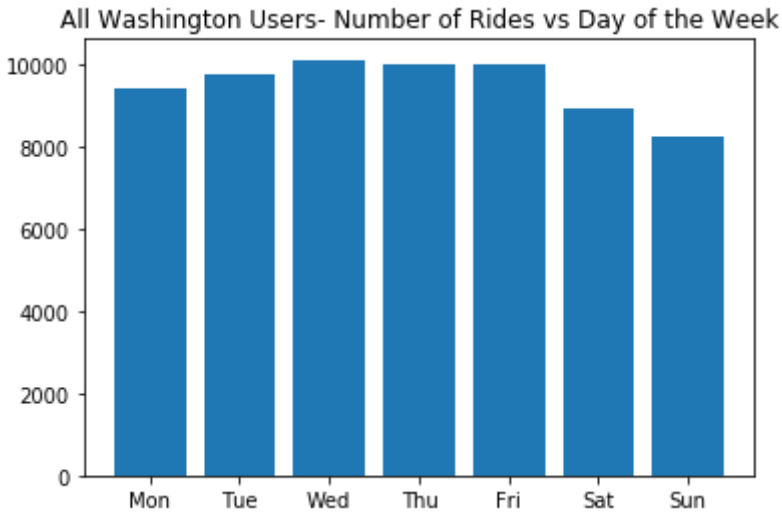
- Subscribers average number of rides is fairly consistent during weekdays.
- Customer use peaks on Saturday and Sunday with smaller peaks on Monday and Friday.
- It appears that Subscribers are more likely to use the service for their commute than the customers who are more likley to use the service for fun.
- It appears that a user who uses the service to bike to work has < 15 minute commute by bike.
- Customers are more likely to take longer rides and it would be useful to examine where they are riding

In [130]:
```
## 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.                                       ##

'''
Is the pattern of ridership different on the weekends versus weekdays?
On what days are Subscribers most likely to use the system? What about C
ustomers?
Does the average duration of rides change depending on the day of the we
ek?
'''

# Plots a graph and chart based on the numbers by day of the week data p
assed.
def plot_day_graph(data,header,week):
    labels = []
    for day in week:
        shortened_name = day[:3]
        labels.append(shortened_name)


    y_pos = np.arange(len(week))
    counts_per_day= [data.count(week[0]),data.count(week[1]),data.count(
week[2]),data.count(week[3]),data.count(week[4]),data.count(week[5]),dat
a.count(week[6])]

    plt.bar(y_pos, counts_per_day, align='center')
    plt.xticks(y_pos, labels)
    plt.title(header + '- Number of Rides vs Day of the Week')
    plt.show()

    print(header)
    for day in week:
        print("Day of the week:  {}   Counts:  {}".format(day,str(data.c
ount(day))))

def plot_duration_by_day_graph(average_durations,header,week):
    labels = []
    for day in week:
        shortened_name = day[:3]
        labels.append(shortened_name)


    y_pos = np.arange(len(week))
    counts_per_day= average_durations

    plt.bar(y_pos, counts_per_day, align='center')
    plt.xticks(y_pos, labels)
    plt.title(header + '- Duration vs Day of Week')

    plt.show()

    print(header)
    count = 0
    for day in week:
        print("Day of the week:  {}   average_duaration:  {}".format(day
```

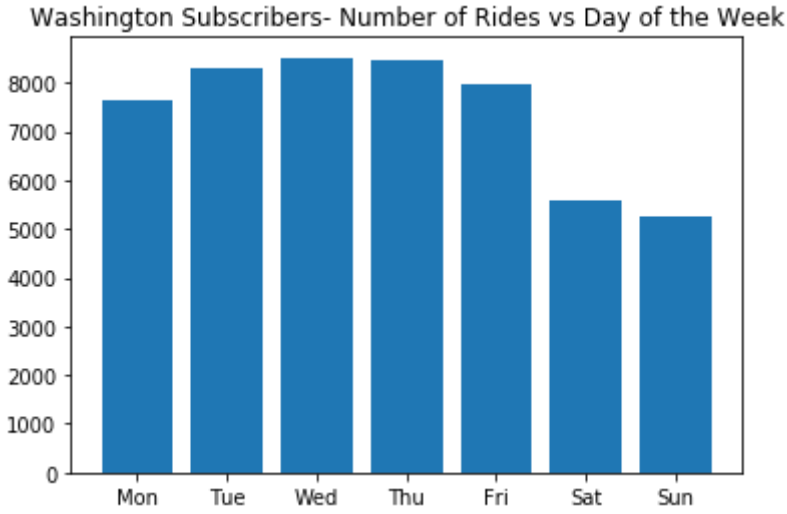
```
,str(average_durations[count])))
        count +=1

# Gathers the day of the week data from file

def day_of_the_week(filename,week):
    subscriber_days = []
    customer_days = []


    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)
        for row in reader:
            day = row['day_of_week']
            if row['user_type'] == 'Subscriber':
                subscriber_days.append(day)
            else:
                customer_days.append(day)

            all_days = customer_days + subscriber_days

    return all_days, customer_days, subscriber_days

def average_duration_by_day(filename,week):
    #subscriber_avg_duration_by_day = []
    #customer_avg_duration_by_day = []

    sub_avg_per_day = []
    cust_avg_per_day = []
    total_avg_per_day = []


    for day in week:
        subscriber_count = 0
        total_subscriber_minutes = 0
        customer_count = 0
        total_customer_minutes = 0

        with open(filename, 'r') as f_in:
        # set up csv reader object
            reader = csv.DictReader(f_in)
            for row in reader:
                testday = row['day_of_week']
                if testday == day:
                    if row['user_type'] == 'Subscriber':
                        subscriber_count +=1
                        total_subscriber_minutes += float(row['duration'
])
                    else:
                        customer_count +=1
                        total_customer_minutes += float(row['duration'])

        total_minutes = total_customer_minutes + total_subscriber_minute
s
        total_count = subscriber_count + customer_count
```

```
        sub_avg_per_day.append(round(total_subscriber_minutes/subscriber
_count,2))
        cust_avg_per_day.append(round(total_customer_minutes/customer_co
unt,2))
        total_avg_per_day.append(round(total_minutes/total_count,2))


    return total_avg_per_day,sub_avg_per_day,cust_avg_per_day
```

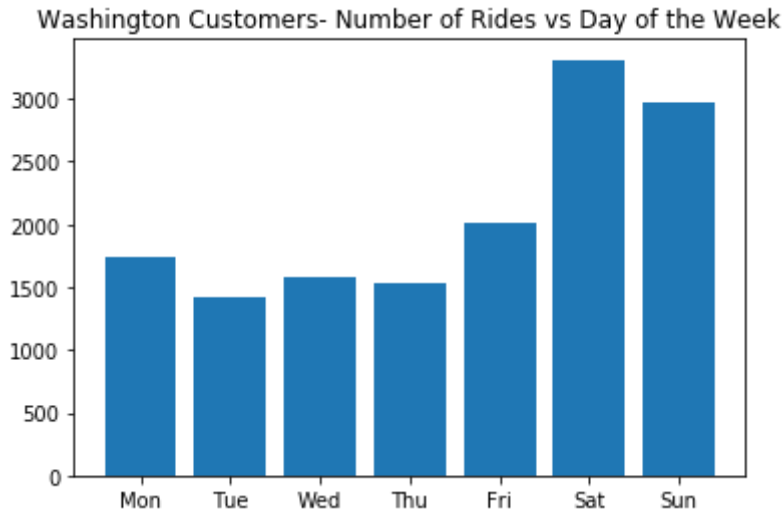
In [131]:
```
# Plots a graph and chart based on day of the week data from a city in t
he order specified for ride count

city = 'Washington'
city_datafile = './data/Washington-2016-Summary.csv'
week = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','S
unday']
all_days, customer_days, subscriber_days = day_of_the_week(city_datafile
,week)
total_avg_per_day,sub_avg_per_day,cust_avg_per_day = average_duration_by
_day(city_datafile,week)


plot_day_graph(all_days, "All Washington Users",week)
plot_day_graph(subscriber_days, "Washington Subscribers",week)
plot_day_graph(customer_days, "Washington Customers",week)
```




```
All Washington Users
Day of the week:  Monday   Counts:  9394
Day of the week:  Tuesday   Counts:  9748
Day of the week:  Wednesday   Counts:  10103
Day of the week:  Thursday   Counts:  9984
Day of the week:  Friday   Counts:  9970
Day of the week:  Saturday   Counts:  8900
Day of the week:  Sunday   Counts:  8227
```

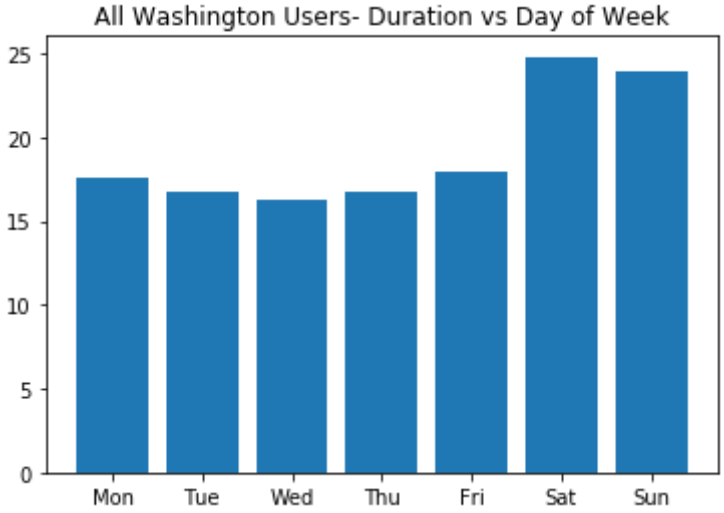



```
Washington Subscribers
Day of the week:  Monday   Counts:  7658
Day of the week:  Tuesday   Counts:  8322
Day of the week:  Wednesday   Counts:  8520
Day of the week:  Thursday   Counts:  8454
Day of the week:  Friday   Counts:  7958
Day of the week:  Saturday   Counts:  5589
Day of the week:  Sunday   Counts:  5252
```

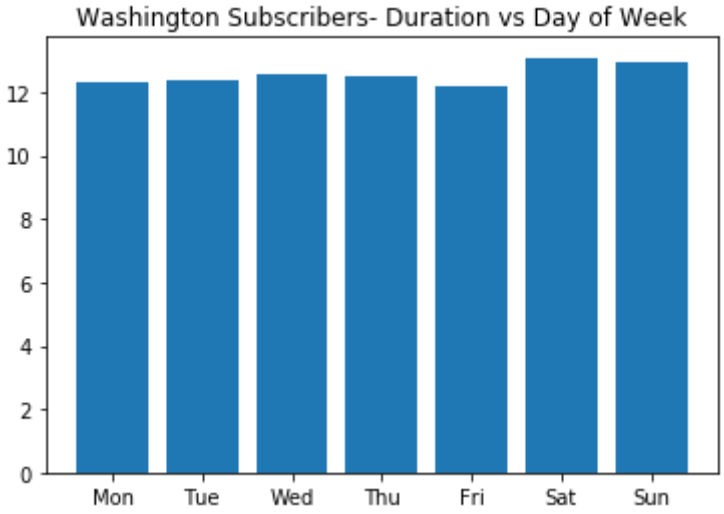



```
Washington Customers
Day of the week:  Monday   Counts:  1736
Day of the week:  Tuesday   Counts:  1426
Day of the week:  Wednesday   Counts:  1583
Day of the week:  Thursday   Counts:  1530
Day of the week:  Friday   Counts:  2012
Day of the week:  Saturday   Counts:  3311
Day of the week:  Sunday   Counts:  2975
```

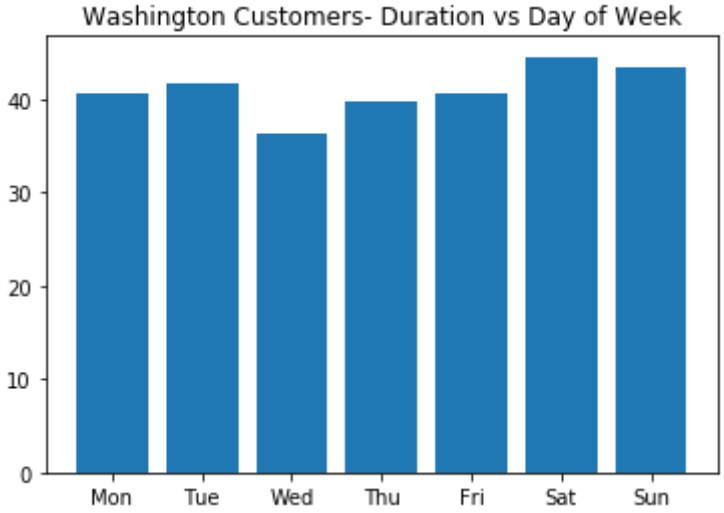
In [132]:
```
# Plots a graph and chart based on day of the week data from a city in t
he order specified for ride duration

city = 'Washington'
city_datafile = './data/Washington-2016-Summary.csv'
week = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','S
unday']
all_days, customer_days, subscriber_days = day_of_the_week(city_datafile
,week)
total_avg_per_day,sub_avg_per_day,cust_avg_per_day = average_duration_by
_day(city_datafile,week)

plot_duration_by_day_graph(total_avg_per_day, "All Washington Users",wee
k)
plot_duration_by_day_graph(sub_avg_per_day, "Washington Subscribers",wee
k)
plot_duration_by_day_graph(cust_avg_per_day, "Washington Customers",week
)
```




```
All Washington Users
Day of the week:  Monday   average_duaration:  17.56
Day of the week:  Tuesday   average_duaration:  16.69
Day of the week:  Wednesday   average_duaration:  16.29
Day of the week:  Thursday   average_duaration:  16.69
Day of the week:  Friday   average_duaration:  17.93
Day of the week:  Saturday   average_duaration:  24.81
Day of the week:  Sunday   average_duaration:  23.97
```




```
Washington Subscribers
Day of the week:  Monday   average_duaration:  12.31
Day of the week:  Tuesday   average_duaration:  12.39
Day of the week:  Wednesday   average_duaration:  12.56
Day of the week:  Thursday   average_duaration:  12.52
Day of the week:  Friday   average_duaration:  12.17
Day of the week:  Saturday   average_duaration:  13.09
Day of the week:  Sunday   average_duaration:  12.95
```




```
Washington Customers
Day of the week:  Monday   average_duaration:  40.75
Day of the week:  Tuesday   average_duaration:  41.78
Day of the week:  Wednesday   average_duaration:  36.37
Day of the week:  Thursday   average_duaration:  39.69
Day of the week:  Friday   average_duaration:  40.72
Day of the week:  Saturday   average_duaration:  44.59
Day of the week:  Sunday   average_duaration:  43.43
```

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:

I'd be interested in analyzing goverment data or medical data.

For government data - it would be interesting to examine data around educational spending. See if there are any that jump out. Maybe look at students where the spending is the same, but the outcomes are different.

For medical data - If would be interesting to look for patterns across datapoints... service providers, age, race, etc.

> **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.