Bikeshare Trip History

August 8, 2019

[2]:
```
import pandas as pd
import numpy as np
import sqlite3
import matplotlib.pyplot as plt
import requests
import json
from zipfile import ZipFile
import os
import seaborn as sns
from sklearn import preprocessing
import datetime
import math
Name = 'Catherine Bui'
```

1 Capital Bikeshare Trip History Analysis

Where do Capital Bikeshare riders go? When do they ride? How far do they go? Which stations are most popular? What days of the week are most rides taken on?

The data includes:

Duration – Duration of trip

Start Date – Includes start date and time

End Date – Includes end date and time

Start Station – Includes starting station name and number

End Station – Includes ending station name and number

Bike Number – Includes ID number of bike used for the trip

Member Type – Indicates whether user was a "registered" member (Annual Member, 30-Day Member or Day Key Member) or a "casual" rider (Single Trip, 24-Hour Pass, 3-Day Pass or 5-Day Pass)

2 Getting the Data

[5]:
```
# bike2010 = pd.read_csv('2010-capitalbikeshare-tripdata.csv')
# bike2011 = pd.read_csv('2011-capitalbikeshare-tripdata.csv')
# bike2012 = pd.concat([pd.read_csv('2012-capitalbikeshare-tripdata/'+f) for f␣
↪in os.listdir('2012-capitalbikeshare-tripdata')])
```

```
# bike2013 = pd.concat([pd.read_csv('2013-capitalbikeshare-tripdata/'+f) for f␣
↪in os.listdir('2013-capitalbikeshare-tripdata')])
# bike2014 = pd.concat([pd.read_csv('2014-capitalbikeshare-tripdata/'+f) for f␣
↪in os.listdir('2014-capitalbikeshare-tripdata')])
# bike2015 = pd.concat([pd.read_csv('2015-capitalbikeshare-tripdata/' + f) for f␣
↪in os.listdir('2015-capitalbikeshare-tripdata')])
# bike2016 = pd.concat([pd.read_csv('2016-capitalbikeshare-tripdata/'+f) for f␣
↪in os.listdir('2016-capitalbikeshare-tripdata')])
# bike2017 = pd.concat([pd.read_csv('2017-capitalbikeshare-tripdata/'+f) for f␣
↪in os.listdir('2017-capitalbikeshare-tripdata')])
# data = pd.
↪concat([bike2010,bike2011,bike2012,bike2013,bike2014,bike2015,bike2016,bike2017])
# data.to_csv('capitalbikeshare2010_2017.csv')
# data = pd.read_csv('capitalbikeshare2010_2017.csv')
```

3 Preliminary Data Cleaning

[ ]:
```
#Convert the duration from seconds to minutes
data['Duration(min)'] = data['Duration']/60
```

[ ]:
```
import datetime
#Convert date to datetime object
data['Start date object'] = data['Start date'].apply(
    lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
data['End date object'] = data['End date'].apply(
    lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

#Get weekday for date
data['Start day'] = data['Start date object'].apply(
    lambda x: x.weekday())
data['End day'] = data['End date object'].apply(
    lambda x: x.weekday())
data.to_csv('capitalbikeshare2010_2017_v2.csv',index=False)
```

4 Start here if you want to skip the other preliminary preparation

[4]:
```
# data with the duration and date object, start day and end day
data = pd.read_csv('capitalbikeshare2010_2017_v2.csv')
```

[5]:
```
# dropping these features because it's too memory heavy
data.drop(['Start date object', 'End date object', 'Unnamed: 0', 'Start day',␣
↪'End day'], axis=1, inplace=True)
```

5 Demographic Attributes of Station and Members

[5]:
```
# attributes of the station locations such as latitude and longitude
attributes = pd.read_csv('Capital_Bike_Share_Locations.csv')
```

[4]:
```
# number of docks in each station
# attributes['Number_of_docks'] = attributes['NUMBER_OF_BIKES'] +␣
↪attributes['NUMBER_OF_EMPTY_DOCKS']
```

[6]:
```
# removing the other features and keeping only terminal number and lat/lon for␣
↪census tract
attributes = attributes[['TERMINAL_NUMBER', 'LATITUDE', 'LONGITUDE']]
```

[7]:
```
# applying census tract to the data
#and using the census tract to understand demographic attributes of the members␣
↪and station



# finding the census tract code for the lat and lon
import requests
import urllib
def getcensustract(lat, lon):
    params = urllib.parse.urlencode({'latitude': lat, 'longitude':lon, 'format':
↪'json'})
    url = 'https://geo.fcc.gov/api/census/block/find?' + params
    response = requests.get(url)
    data = response.json()
    return data['Block']['FIPS']
#apply the census tract to the lat and lon of the attributes df
attributes['CensusTract'] = attributes.apply(lambda x:␣
↪getcensustract(x['LATITUDE'], x['LONGITUDE']), axis = 1)

# dictionary of station number and census tract
censustract = dict(zip(attributes['TERMINAL_NUMBER'], attributes['CensusTract']))

# apply census tract to the start and end station
data['CensusTractStart'] = data['Start station number'].map(censustract).
↪apply(lambda x: str(x)[0: len(str(x))-4])
data['CensusTractEnd'] = data['End station number'].map(censustract).
↪apply(lambda x: str(x)[0: len(str(x))-4])

# dataset with DC population demographics including total population,␣
↪percentages of racial and ages
# source2: http://opendata.dc.gov/datasets/census-blocks-centroid-in-2010
population= pd.read_csv('Census_Blocks_Centroid_in_2010.csv')



# dictionary of census tract and total population
```

```
population['tract#'] = population['GEOID'].apply(lambda h: str(h).split('␣
↪',3)[-1][0: len(str(h))-4])
total = population.groupby('tract#').agg({'P0010001': 'sum'}).reset_index()
censustractpop = dict(zip(total['tract#'], total['P0010001']))

data['StartTractPopulation'] = data['CensusTractStart'].map(censustractpop)
data['EndTractPopulation'] = data['CensusTractEnd'].map(censustractpop)
```

5.0.1 Population Analysis

[78]:

```
```

[78]:

```
Index(['Unnamed: 0', 'Duration', 'Start date', 'End date',
       'Start station number', 'Start station', 'End station number',
       'End station', 'Bike number', 'Member type', 'Duration(min)',
       'Start date object', 'End date object', 'Start day', 'End day',
       'Start Time', 'End Time', 'CensusTractStart', 'CensusTractEnd',
       'StartTractPopulation', 'EndTractPopulation'],
      dtype='object')
```

6 Data Cleaning & Feature Engineering Part 2

[49]:

```
# data with the duration and date object, start day and end day
data = pd.read_csv('capitalbikeshare2010_2017_v2.csv')
```

[8]:

```
#determine if a ride is at night or during the day
def time(x):
    '''AM PEAK (7AM-10AM)
       MIDDAY(10AM-4PM)
       PM PEAK (4PM-7PM)
       EVENING (7PM -12AM)
       EARLY MORNING (12AM-7AM)'''
    if x >= 7 and x < 10:
        return 'AM PEAK'
    elif x >= 10 and x < 16:
        return 'MIDDAY'
    elif x >= 16 and x < 19:
        return 'PM PEAK'
    elif x >= 19:
        return 'EVENING'
    elif x >= 0 and x < 7:
        return 'EARLY MORNING'
data['Start Time'] = data['Start date object'].apply(
    lambda x: time(datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S').hour))
data['End Time'] = data['End date object'].apply(
    lambda x: time(datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S').hour))
```

[9]:
```
#clean up the unknown member types
data = data[data['Member type'] != 'Unknown']
```

[89]:
```
data['Start Month'] = data['Start date object'].apply(
    lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S').month)
```

6.0.1 Average Duration

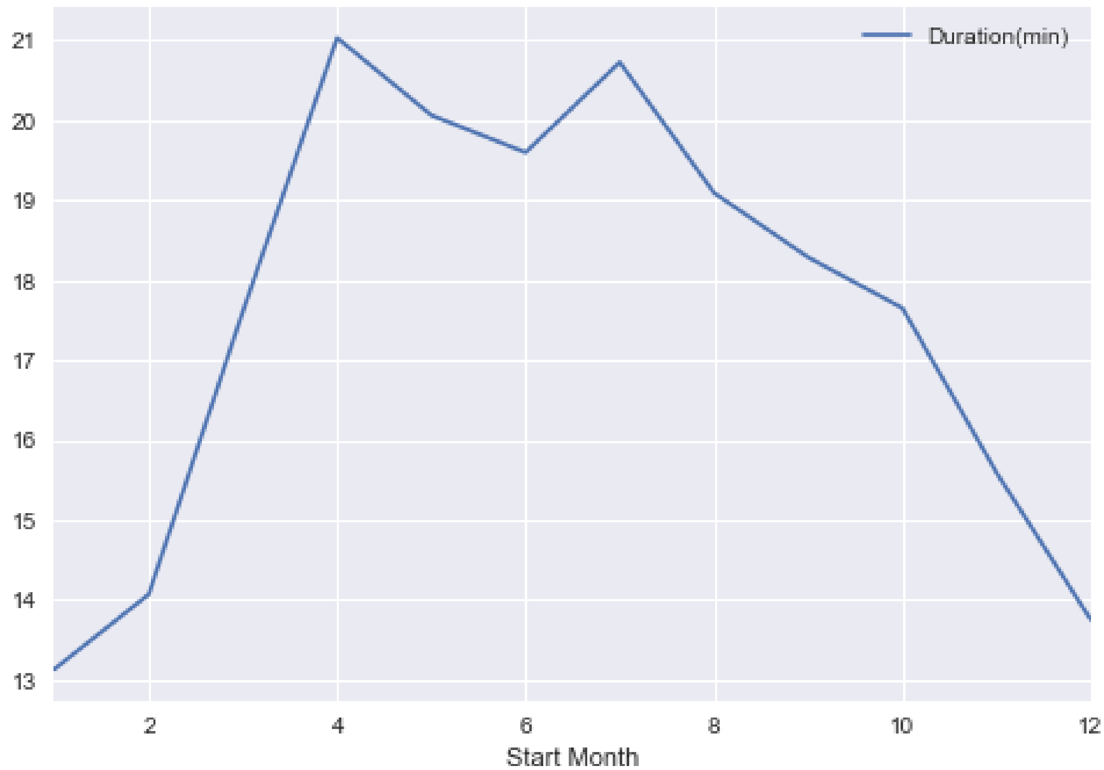
[90]:
```
#Find the average duration for start station and end station
averageduration = data.groupby(['Start station number', 'End station number']).
 ↪agg(
    {'Duration(min)': np.mean}).reset_index().rename(
    columns = {'Duration(min)': 'Duration(min)mean'})

# apply average duration to the station
data = data.merge(averageduration, on = ['Start station number', 'End station␣
 ↪number'], how = 'left')
# data['differenceduration'] = data['Duration(min)']- data['Duration(min)mean']
```

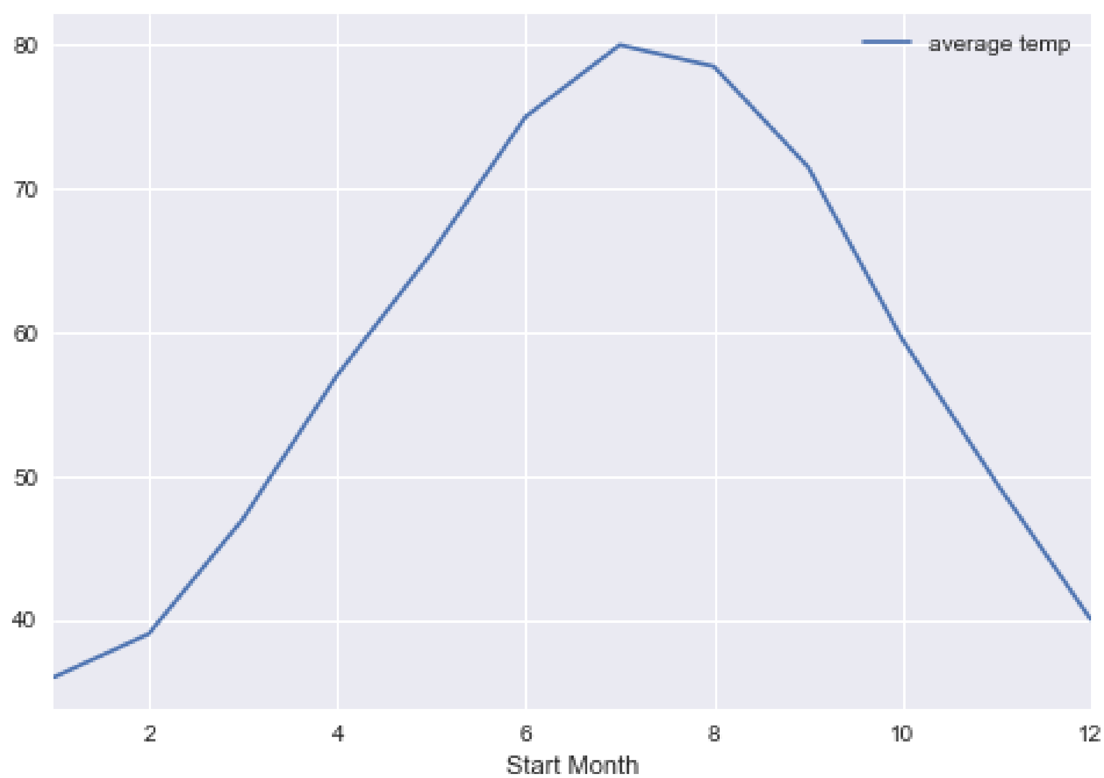
7 DATA AGGREGATION

7.1 Month

[107]:
```
#key is month
# values is [average low, average high]
weather = {1: [29,43],
          2: [31, 47],
          3: [38, 56],
          4: [47, 67],
          5:[56, 75],
          6: [66,84],
          7:[71, 89],
          8:[70, 87],
          9: [63, 80],
          10: [51, 68],
          11: [41, 58],
          12: [33, 47]}
```

[119]:
```
monthduration = data.groupby('Start Month').agg({'Duration(min)': 'mean'}).
 ↪reset_index()
monthduration['temperature'] = monthduration['Start Month'].map(weather)
monthduration['average temp'] = monthduration['temperature'].apply(lambda x: np.
 ↪mean(x))
monthduration.plot(x ='Start Month', y ='Duration(min)')
monthduration.plot(x = 'Start Month', y = 'average temp')
plt.show()
```


Duration(min)
21
20
19
18
17
16
15
14
13
2
4
6
8
10
12
Start Month


average temp
80
70
60
50
40
2
4
6
8
10
12
Start Month

There are more trips when the weather is warmer than when the temperature is low.

7.1.1 Popular Stations

[27]:
```
#How many trips per start station? Which start station is popular?
data['ValueCountStart'] = data['Start station'].map(data['Start station'].
 ↪value_counts())
```

[28]:
```
data['ValueCountEnd'] = data['End station'].map(data['End station'].
 ↪value_counts())
```

[135]:
```
#stations not in the attributes
[station for station in data['Start station number'].unique() if station not in␣
 ↪attributes['TERMINAL_NUMBER'].unique()]
```

[135]:
```
[31008, 31709, 32009, 32202]
```

7.1.2 Time of Day and Night

[46]:
```
# how many trips in the midday, pm peak, am peak, and early morning
data['Start Time'].value_counts()
```

[46]:
```
MIDDAY           6006228
PM PEAK          5234441
EVENING          3524477
AM PEAK          3484474
EARLY MORNING     868023
Name: Start Time, dtype: int64
```

Comments: Many trips are during the Midday (10am-4pm) where there is the most sunlight. There's least amount of trips during the times where there is not alot of sunlight (Midnight-7AM)

[47]:
```
# how many trips in the midday, pm peak, am peak, and early morning
data['End Time'].value_counts()
```

[47]:
```
MIDDAY           5764552
PM PEAK          5256591
EVENING          3877360
AM PEAK          3382950
EARLY MORNING     836190
Name: End Time, dtype: int64
```

Comment: The same analysis above applies to End Time.

[48]:
```
data.groupby(['Start Time', 'End Time']).size().reset_index().
 ↪sort_values(['Start Time', 0])
```

[48]:
```
       Start Time       End Time      0
1         AM PEAK  EARLY MORNING     44
2         AM PEAK        EVENING    612
```

```
4          AM PEAK        PM PEAK     1828
3          AM PEAK         MIDDAY   198680
0          AM PEAK        AM PEAK  3283310
7    EARLY MORNING        EVENING      154
9    EARLY MORNING        PM PEAK      222
8    EARLY MORNING         MIDDAY     1287
5    EARLY MORNING        AM PEAK    95598
6    EARLY MORNING  EARLY MORNING   770762
14         EVENING        PM PEAK      407
13         EVENING         MIDDAY     1681
10         EVENING        AM PEAK     2210
11         EVENING  EARLY MORNING    64569
12         EVENING        EVENING  3455610
16          MIDDAY  EARLY MORNING      296
15          MIDDAY        AM PEAK      663
17          MIDDAY        EVENING     7383
19          MIDDAY        PM PEAK   436225
18          MIDDAY         MIDDAY  5561661
21         PM PEAK  EARLY MORNING      519
20         PM PEAK        AM PEAK     1169
23         PM PEAK         MIDDAY     1243
22         PM PEAK        EVENING   413601
24         PM PEAK        PM PEAK  4817909
```

Comments: The most trips start and end within the same time frame/group such as for example, if the trip starts at 7am, it would end within 7-10am.

There are less trips that start 7am-7pm and ends within 12am-7am.

There are less trips that start at 7pm-12am and ends within 4pm-7pm m(more than 16 hours). And there are less trips that start at 12am-7am and end at 7pm-12am (more than 12 hours).

[49]:
```
data.groupby(['Start Time', 'End Time']).agg({'Duration(min)': 'mean'}).
↪reset_index().sort_values(['Start Time', 'Duration(min)'])
```

[49]:
```
       Start Time       End Time  Duration(min)
0         AM PEAK        AM PEAK      11.806764
3         AM PEAK         MIDDAY      44.542693
4         AM PEAK        PM PEAK     505.534582
2         AM PEAK        EVENING     702.489679
1         AM PEAK  EARLY MORNING    1070.248864
6   EARLY MORNING  EARLY MORNING      12.096026
5   EARLY MORNING        AM PEAK      20.941507
8   EARLY MORNING         MIDDAY     522.535768
9   EARLY MORNING        PM PEAK     835.528003
7   EARLY MORNING        EVENING    1103.330519
12        EVENING        EVENING      14.669071
11        EVENING  EARLY MORNING      48.941991
10        EVENING        AM PEAK     675.110430
13        EVENING         MIDDAY     904.890432
14        EVENING        PM PEAK    1228.395414
```

```
18         MIDDAY         MIDDAY      19.397668
19         MIDDAY        PM PEAK      63.151950
17         MIDDAY        EVENING     349.573464
16         MIDDAY  EARLY MORNING     815.979673
15         MIDDAY        AM PEAK    1158.744495
24        PM PEAK        PM PEAK      14.811483
22        PM PEAK        EVENING      40.373120
21        PM PEAK  EARLY MORNING     551.252890
20        PM PEAK        AM PEAK     888.093385
23        PM PEAK         MIDDAY    1142.331215
```

Comments: As mentioned above, the most trips end in the same time frame/group.

These trips that end in the same time frame/group has the shortest duration (on average 14.53 minutes) compared to the other trips starting in one time frame ending in different time frame.

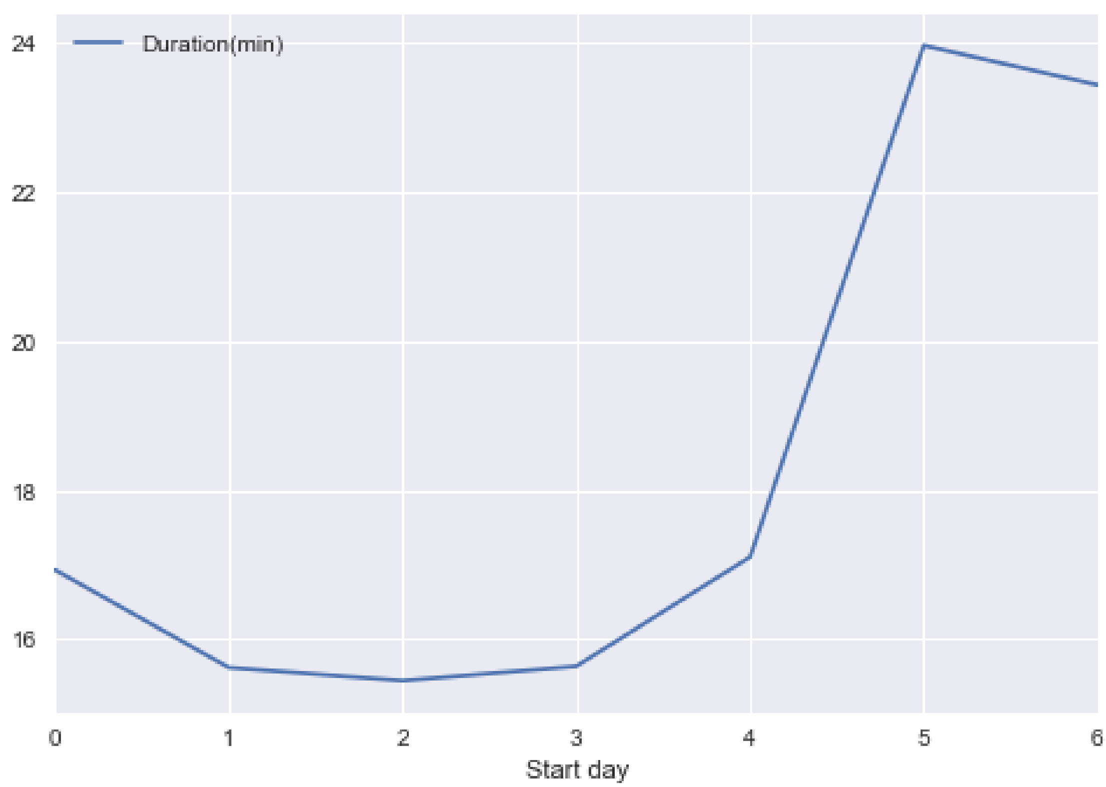
The trips with the longest duration are on average approximately 17 hours.

Question: What kind of passes affect the duration of these trips?

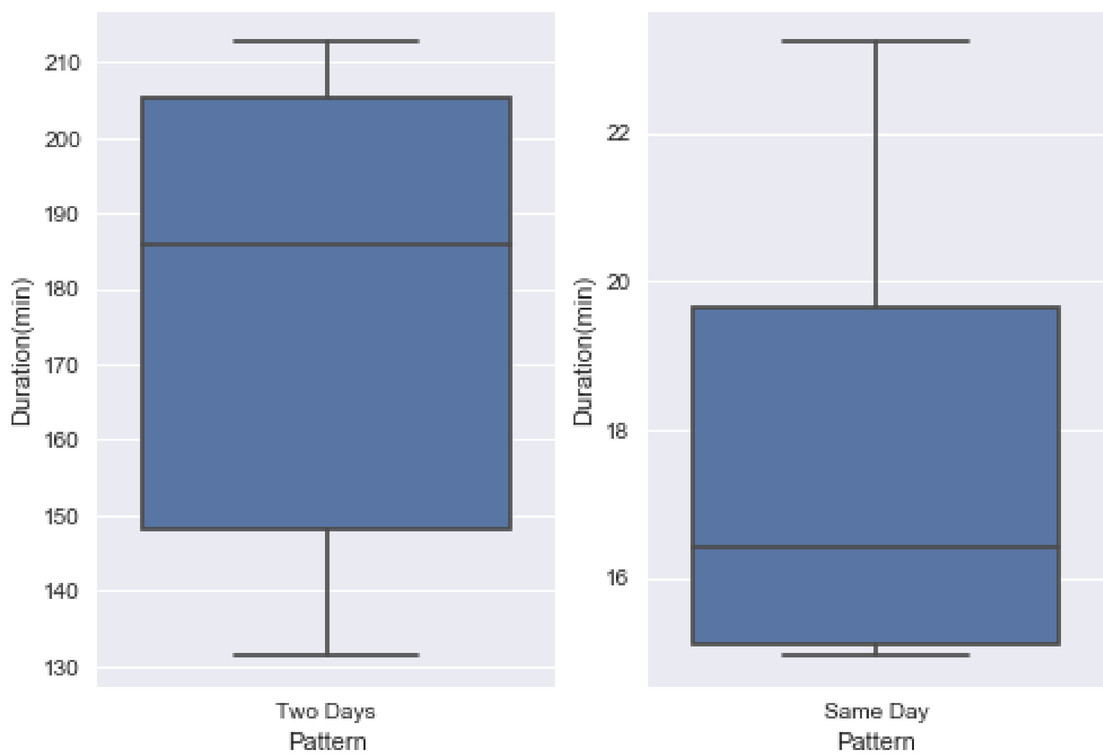
7.2 Day of the Trip

[33]:
```python
durationday = data.groupby(['Start day']).agg({'Duration(min)': 'mean'})
```

[36]:
```python
# mean duration over the days
durationday.plot()
plt.show()
```



[100]:
```
# the mean duration distribution if the trip starts and ends in the same day
# the mean duration distribution if the trip starts and end in two consecutive␣
 ↪days
durationstartend = data.groupby(['Start day', 'End day']).agg({'Duration(min)':␣
 ↪'mean'}).reset_index()
durationstartend['Pattern'] = durationstartend.apply(lambda x: 'Same Day' if␣
 ↪x['Start day'] == x['End day'] else 'Two Days', axis = 1)
twodays = durationstartend[durationstartend['Pattern'] == 'Two Days']
sameday = durationstartend[durationstartend['Pattern'] == 'Same Day']
plt.subplot(1,2,1)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = twodays)
plt.subplot(1,2,2)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = sameday)
plt.show()
```



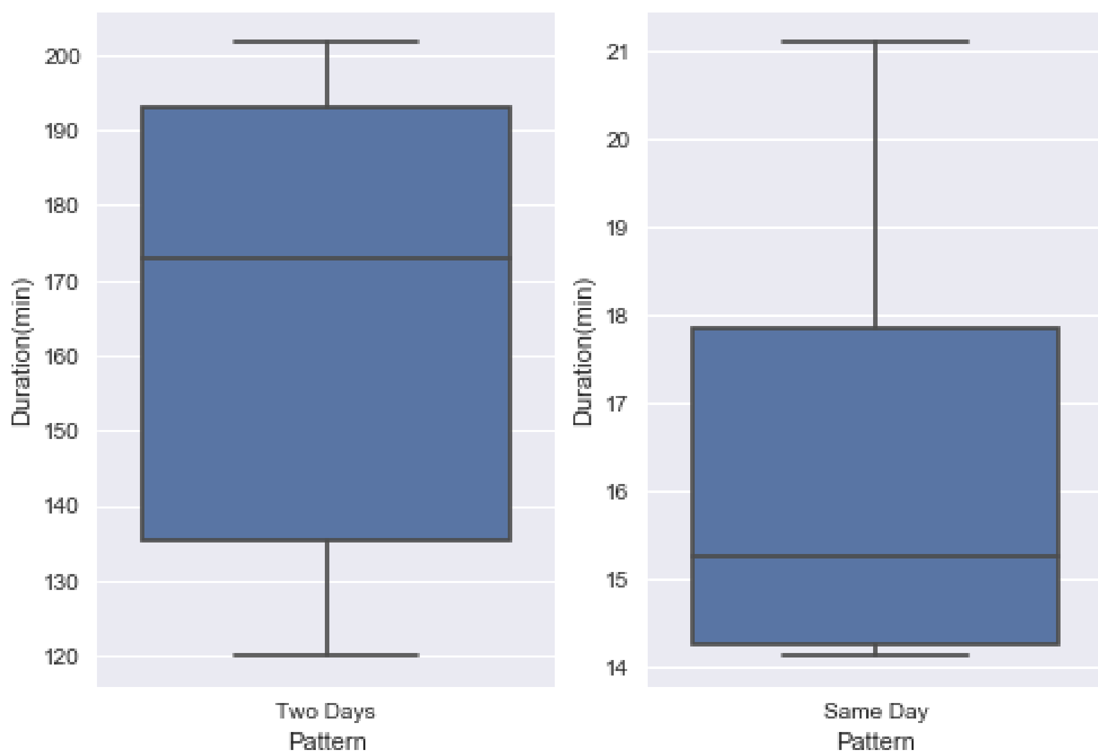
Comments: The duration is longer on the weekend than at the beginning of the week. In the middle of the week, it is on average around 15.5 minutes. This could be because people use these bikes to go on adventures and activities on the weekend.

Question: Do they use it to go to school or work on the weekdays?

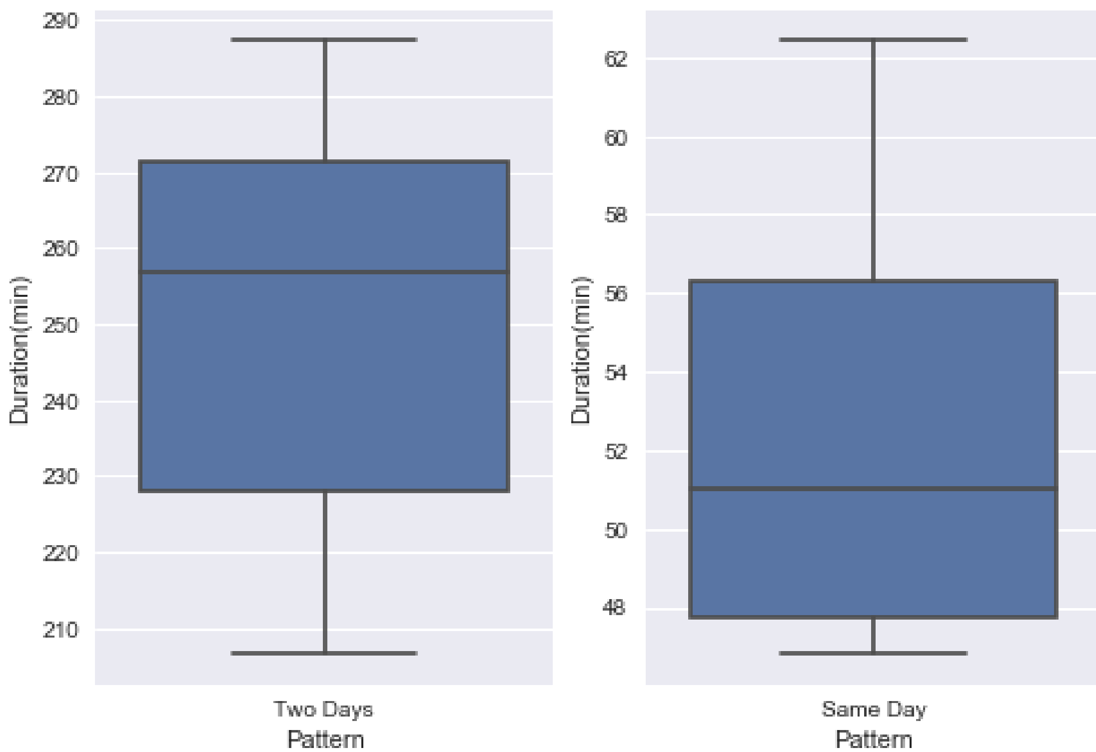
When the trips goes from one day to the next day, the duration is longer than the duration of trips done in a single day. This could be because they use it for longer activities/events or they take it home and return it tomorrow.

Question: Do the users who take trips in two consecutive days go to different stations or the same station when they return their bike? Answer: Users who take trips in two consecutive days take longer to go to different stations than users who take trips in two consecutive days ending in the same station as the start. It doesn't matter if they go to a different end station or the same end station.

[102]:
```
# When the start station is not the same as end station,
#what is the mean duration for trips ending in the same day or trips in the next␣
↪day?

df = data[data['Start station'] != data['End station']].groupby(
    ['Start day', 'End day']).agg({'Duration(min)': 'mean'}).reset_index()
df['Pattern'] = df.apply(
    lambda x: 'Same Day' if x['Start day'] == x['End day'] else 'Two Days', axis␣
↪= 1)
twodays = df[df['Pattern'] == 'Two Days']
sameday = df[df['Pattern'] == 'Same Day']
plt.subplot(1,2,1)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = twodays)
plt.subplot(1,2,2)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = sameday)
plt.show()
```



[103]:
```
df = data[data['Start station'] == data['End station']].groupby(
    ['Start day', 'End day']).agg({'Duration(min)': 'mean'}).reset_index()
df['Pattern'] = df.apply(
    lambda x: 'Same Day' if x['Start day'] == x['End day'] else 'Two Days', axis␣
 ↪= 1)
twodays = df[df['Pattern'] == 'Two Days']
sameday = df[df['Pattern'] == 'Same Day']
plt.subplot(1,2,1)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = twodays)
plt.subplot(1,2,2)
sns.boxplot(x = 'Pattern', y = 'Duration(min)', data = sameday)
plt.show()
```



7.3 DAY AND TIME

[74]:
```
data[data['Start day'] != data['End day']].groupby(
    ['Start Time', 'End Time']).agg(
    {'Duration(min)': 'mean'}).reset_index().sort_values(['Start Time',␣
 ↪'Duration(min)'])
```

[74]:
```
     Start Time       End Time  Duration(min)
1       AM PEAK  EARLY MORNING    1070.248864
0       AM PEAK        AM PEAK    1383.917059
```

```
2   EARLY MORNING  EARLY MORNING    1308.032051
4         EVENING  EARLY MORNING      48.941991
3         EVENING        AM PEAK     675.110430
6         EVENING         MIDDAY     904.890432
7         EVENING        PM PEAK    1228.395414
5         EVENING        EVENING    1341.019231
9          MIDDAY  EARLY MORNING     815.979673
8          MIDDAY        AM PEAK    1158.744495
10         MIDDAY         MIDDAY    1315.364848
12        PM PEAK  EARLY MORNING     551.252890
11        PM PEAK        AM PEAK     888.093385
13        PM PEAK         MIDDAY    1142.331215
14        PM PEAK        PM PEAK    1380.066831
```

Comments: Since these are the trips that took two consecutive days, the trips took hours except for trips that started in evening and ended in early morning. On average, it doesn't take more than 24 hours.

[75]:
```
data[data['Start day'] == data['End day']].groupby(
    ['Start Time', 'End Time']).agg(
    {'Duration(min)': 'mean'}).reset_index().sort_values(['Start Time',␣
↪'Duration(min)'])
```

[75]:
```
       Start Time       End Time  Duration(min)
0         AM PEAK        AM PEAK      11.771244
2         AM PEAK         MIDDAY      44.542887
3         AM PEAK        PM PEAK     505.534582
1         AM PEAK        EVENING     702.489679
5   EARLY MORNING  EARLY MORNING      12.052276
4   EARLY MORNING        AM PEAK      20.941507
7   EARLY MORNING         MIDDAY     522.535768
8   EARLY MORNING        PM PEAK     835.528003
6   EARLY MORNING        EVENING    1103.330519
9         EVENING        EVENING      14.594237
11         MIDDAY         MIDDAY      19.177681
12         MIDDAY        PM PEAK      63.152173
10         MIDDAY        EVENING     349.573464
14        PM PEAK        PM PEAK      14.753956
13        PM PEAK        EVENING      40.373180
```

Question: Why do same day trips occur at certain times compared to two day trips? Perhaps we can divide the days into weekday and weekend

7.4 BIKE MEMBERS

[60]:
```
data.groupby('Member type').size()
```

[60]:
```
Member type
Casual      4175473
Member     14942112
```

```
dtype: int64
```

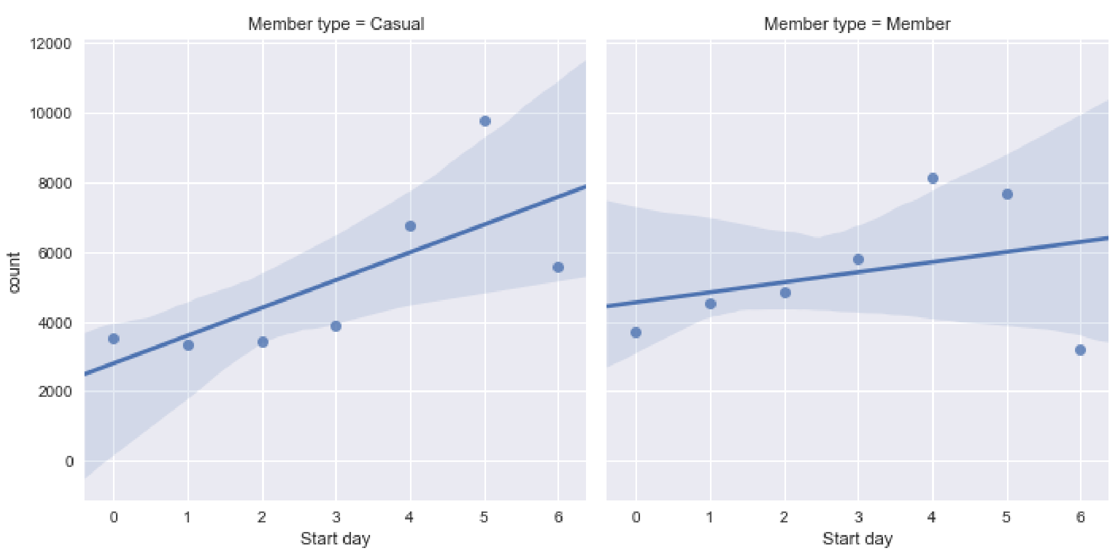
[67]:
```
# duration by two consecutive days for member type
memberduration = data.groupby(
    ['Member type','Start day', 'End day']).agg(
    {'Duration(min)': 'mean'}).reset_index()
memberduration[memberduration['Start day'] != memberduration['End day']].groupby(
    'Member type').agg({'Duration(min)': 'mean'})
```

[67]:
```
             Duration(min)
Member type
Casual          226.140183
Member          128.118178
```

[68]:
```
# duration on the same day for member type
memberduration = data.groupby(
    ['Member type','Start day', 'End day']).agg(
    {'Duration(min)': 'mean'}).reset_index()
memberduration[memberduration['Start day'] == memberduration['End day']].groupby(
    'Member type').agg({'Duration(min)': 'mean'})
```

[68]:
```
             Duration(min)
Member type
Casual           37.895365
Member           11.836872
```

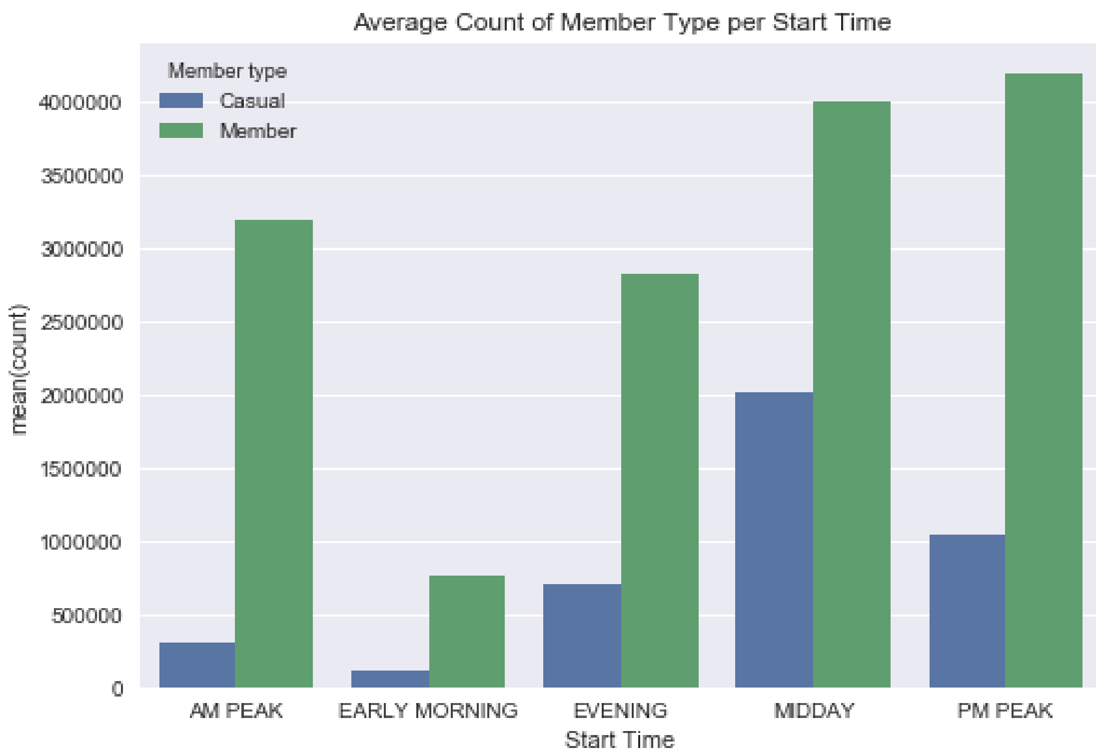
[54]:
```
# how many casual/member per day
#eliminating two consecutive days to keep it consistent
sameday = data[data['Start day'] != data['End day']]
memberday = sameday.groupby(['Member type', 'Start day']).size().to_frame().
 ↪reset_index().rename(columns = {0: 'count'})
sns.lmplot(x = 'Start day', y = 'count', data = memberday, col = 'Member type')
plt.show()
```



Casual bikers tend to go on more trips during Friday to Sunday than the rest of the week. Saturday is a popular day for casual bikers.

Member bikeres tend to increasely go on more trips starting from Monday to Saturday, but on Sunday, they go on less trips. Friday is a popular day for Member bikers.

[120]:
```
memberstartime= data.groupby(['Member type', 'Start Time']).count().
  ↪reset_index().rename(columns = {'Unnamed: 0': 'count'})[['Member type', 'Start␣
  ↪Time', 'count']]
sns.barplot(x = 'Start Time', y = 'count', hue = 'Member type', data =␣
  ↪memberstartime)
plt.title("Average Count of Member Type per Start Time")
plt.show()
```



8 Visualizing the network

8.1 DISTANCE BETWEEN START AND END STATION

[61]:
```
# calculate the great circle distance between two points
def great_circle_distance(lat1, lon1, lat2, lon2):
    if any(l is None for l in [lat1, lon1, lat2,lon2]):
        return None
```

```
    '''return distance in miles'''
    Radius = 6371
    phi1 = math.radians(lat1)
    phi2 = math.radians(lat2)
    Deltaphi = math.radians(lat2-lat1)
    Deltalambda = math.radians(lon2 - lon1)

    a = math.sin(Deltaphi/2)* math.sin(Deltaphi/2) + math.cos(phi1)*math.
 ↪cos(phi2)*math.sin(Deltalambda/2)*math.sin(Deltalambda/2)
    c = 2*math.atan2(math.sqrt(a), math.sqrt(1-a))
    d = Radius*c
    #1 kilometer (km) = 0.621371192 miles (m).
    d = d*0.621371192
    return d
```

[62]:
```
latlon = {}
for row in range(attributes.shape[0]):
    s = attributes.iloc[row]
    latlon[s['TERMINAL_NUMBER']] = [s['LATITUDE'], s['LONGITUDE']]
```

[66]:
```
data['Start coord'] = data['Start station number'].apply(lambda x: latlon.get(x))
data['End coord'] = data['End station number'].apply(lambda x: latlon.get(x))
```

[67]:
```
%%time
data['distance'] = data.apply(
    lambda r: great_circle_distance(
        r['Start coord'][0],
        r['Start coord'][1],
        r['End coord'][0],
        r['End coord'][1]) if r['Start coord'] is not None and r['End coord'] is␣
 ↪not None else np.nan, axis = 1)
```

```
CPU times: user 27min 56s, sys: 4min 7s, total: 32min 4s
Wall time: 4h 10min 9s
```

[70]:
```
#What stations goes to other stations often?
distanceduration = data.groupby(['Start station', 'End station']).
 ↪agg({'distance': 'mean', 'Duration(min)': 'mean'}).reset_index().
 ↪sort_values(['Start station', 'distance'])
```

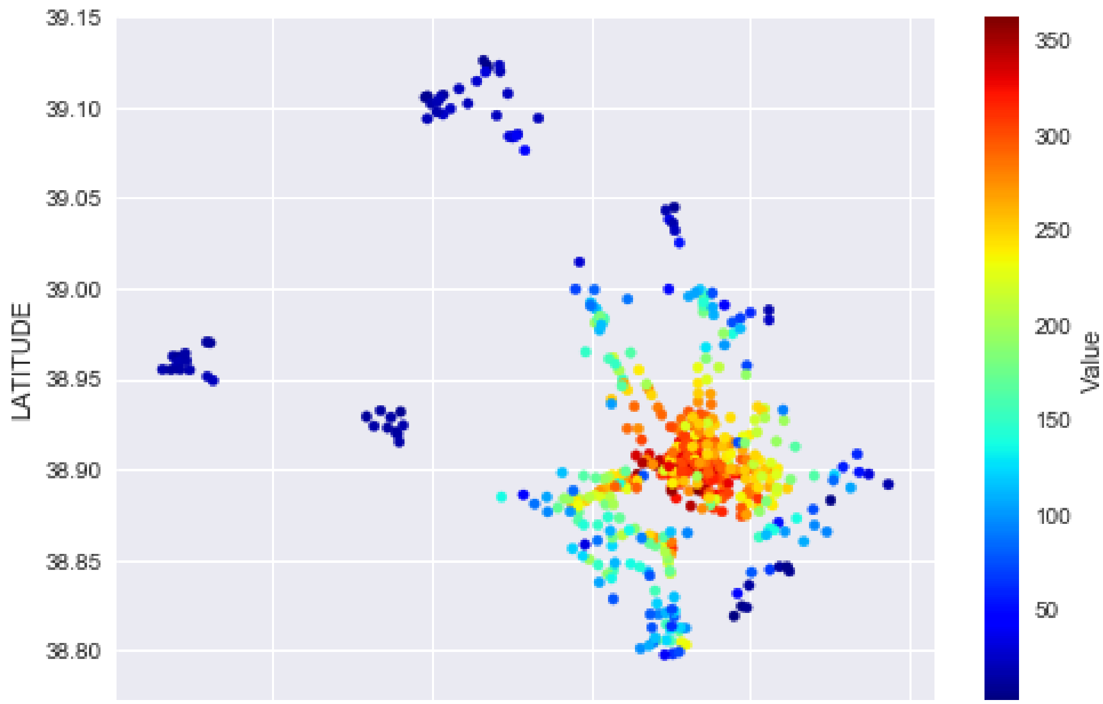
[71]:
```
distanceduration['distance'].corr(distanceduration['Duration(min)'])
```

[71]:
```
0.42528473797412109
```

Duration and distance are not strongly correlated.

[82]:
```
# the map of how many stations are connected to other stations
#Red means these are the stations are highly connected.
#Blue means there are a few exclusive stations traveling to it.
r = data[~data['Start station number'].isin([31008, 31709, 32009, 32202])]
df  = r.groupby(
```

```
    ['Start station number', 'End station number']).agg(
    {'distance': 'mean'}).reset_index().groupby('Start station number').count().
↪reset_index()[['Start station number', 'End station number']].rename(
    columns = {'End station number': 'count'}).rename(columns = {'Start station␣
↪number': 'TERMINAL_NUMBER'})
values = dict(zip(df['TERMINAL_NUMBER'], df['count']))
attributes['Value'] = attributes['TERMINAL_NUMBER'].map(values)
attributes.plot(kind="scatter", x="LONGITUDE", y="LATITUDE", c = 'Value',␣
↪cmap=plt.get_cmap("jet"), colorbar=True)
plt.show()
```



[ ]:
```
stationtostation= data.groupby(['Start station number', 'End station number']).
↪agg({'distance': 'mean'}).reset_index()
r = stationtostation[stationtostation['distance'] < 1]
df  = r.groupby(
    ['Start station number', 'End station number']).agg(
    {'distance': 'mean'}).reset_index().groupby('Start station number').count().
↪reset_index()[['Start station number', 'End station number']].rename(
    columns = {'End station number': 'count'}).rename(columns = {'Start station␣
↪number': 'TERMINAL_NUMBER'})
values = dict(zip(df['TERMINAL_NUMBER'], df['count']))
attributes['Value'] = attributes['TERMINAL_NUMBER'].map(values)
attributes.plot(kind="scatter", x="LONGITUDE", y="LATITUDE", c = 'Value',␣
↪cmap=plt.get_cmap("jet"), colorbar=True)
```

```
plt.show()
```

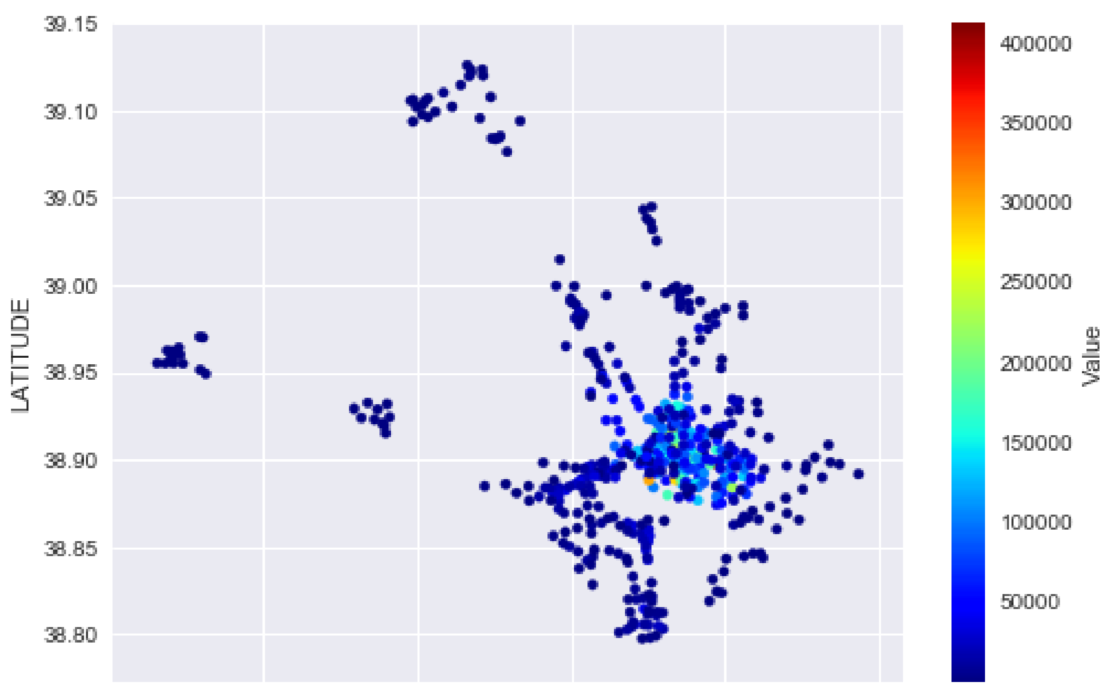
Questions: 1. Do bikers go to closeby end station from the start station? 2. Do start stations have commonly occured end stations? 3. Do some riders from one station never visit other stations due to distance?

[85]:
```
%%time
#eliminate the terminals that do not have attributes
r = data[~data['Start station number'].isin([31008, 31709, 32009, 32202])]
values = r[~r['End station number'].isin([31008, 31709, 32009, 32202])]['Start␣
↪station number'].value_counts()
```

```
CPU times: user 21.7 s, sys: 1min 32s, total: 1min 54s
Wall time: 4min 12s
```

[86]:
```
attributes['Value'] = attributes['TERMINAL_NUMBER'].map(values)
```

[87]:
```
# this is plot of the value count of each start station
attributes.plot(kind="scatter", x="LONGITUDE", y="LATITUDE", c= 'Value',␣
↪cmap=plt.get_cmap("jet"),colorbar=True)
plt.show()
```



There are very few stations that are popular (most bikers take trips from there often)

[ ]: