# Citibike Analysis: A Deeper Look Into New York's Bike Sharing Program


## Author: Atharva Bhandarkar

#### Email: apb412@nyu.edu


Named after its chief sponsor, Citibank, Citibike is a privately owned public bike-sharing system in New York City. With 706 docking stations, 12000 bikes and attracting 163,000 annual subscribers in 2016, it is the largest bike sharing program in the United States.

Citibike hass seen a meteoric rise in popularity over the last few years, growing from under a mere 6 million trips in 2013 to over 14 million in 2016. This massive surge in people's interest in and usage of the program made me curious about it and thus, the goal of this project is to provide a deeper insight about citibike usage in New York City.

The project will analyze data published by NYC Citibikes; first on a general level (some descriptive analysis), and gradually build on that by trying to answer certain, more thought-provoking questions.

The preliminary analysis will use data from a single month (or some small, fixed time period) and the following is a non-comprehensive list of questions it will answer:

- Average distance travelled by citibikes
- Gender and statistics of citibike users
- Most popular locations in NYC (start points, end points etc)

Once these questions are answered, the project will delve deeper and try to understand some more implicit 'trends'. What follows are some examples of these questions:

- How does usage change over the course of a day (i.e what are peak times)
- What does a weekly trend look like?

At this point we know who rides citbikes, how they ride them, when they ride them. But I wanted to know why. One would assume that they're faster than walking, but are they faster than driving? I used the Google Maps API to find out.

# Data Report

Citibike publishes data about every single trip from 2012. As mentioned above, 2016 for example had 14 million trips; a datasize far too large (for computations sake). So I decided to focus on the September 2017 data.

The following is a link to where I obtained this zip file: https://www.citibikenyc.com/system-data/ (https://www.citibikenyc.com/system-data/)

However because it is 330 mb, it made sense to download and use it from my laptop instead. In that case the path became:

'/Users/atharvabhandarkar/Downloads/citi_trips_sep2017.csv'

# My Packages

In [1]:
```
import pandas as pd
from math import sin, cos, sqrt, atan2, radians
import numpy as np
import matplotlib.pyplot as plt
```

In [153]:
```
path = '/Users/atharvabhandarkar/Downloads/citi_trips_sep2017.csv'
```

In [154]:
```
df_full = pd.read_csv(path)
df = df_full#.sample(n=100)
```

In [155]:
```
df['bike_time'] = df['tripduration']/60 #creates new dataseries of trip t

df = df.rename(columns={'start station latitude':'start_lat',
                        'start station longitude':'start_long',
                        'end station latitude':'end_lat',
                        'end station longitude':'end_long',
                        'start station name': 'start_station_name',
                        'end station name': 'end_station_name'})
```

# Trip Distance Calculation

## Using a trig function to calculate distances between start and end latitudes and longitudes of a trip

In [5]:
```
def haversine(lon1, lat1, lon2, lat2):

    lon1, lat1, lon2, lat2 = map(np.radians, [df.start_long, df.start_lat

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2
    c = 2 * np.arcsin(np.sqrt(a))
    km = 6367 * c
    return km
```

In [6]:
```
for row in range(100): #doing range(100) because it is a 1.9million row d
                       #1.9mn times would be a unnecessarily time consumi
    df['dist_km'] = haversine(df.start_long.iloc[row],
                              df.start_lat.iloc[row],
                              df.end_long.iloc[row],
                              df.end_long.iloc[row])
```

In [7]:
```
df.head(2)
```

Out[7]:

| ie | start station id | start_station_name | start_lat | start_long | end station id | end_station_name | end_lat | end |
|---|---|---|---|---|---|---|---|---|
| 9- )1 19 | 3331 | Riverside Dr & W 104 St | 40.801343 | -73.971146 | 3328 | W 100 St & Manhattan Ave | 40.795000 | -73.9 |
| 9- )1 30 | 3101 | N 12 St & Bedford Ave | 40.720798 | -73.954847 | 3100 | Nassau Ave & Newell St | 40.724813 | -73.9 |

## Some descriptive statistics about the distances travelled

In [10]:
```
df.dist_km.mean()
```

Out[10]: 1.8863128567894845

In [12]:
```
df.dist_km.describe() #1.87 million trips in Sept 2017, with average trip
```

Out[12]:
```
count    1.878098e+06
mean     1.886313e+00
std      1.420290e+01
min      0.000000e+00
25%      8.755417e-01
50%      1.447624e+00
75%      2.423891e+00
max      8.662342e+03
Name: dist_km, dtype: float64
```

# Age and Gender Distribution

In [60]:
```
df['gender']= df['gender'].astype(str) #converting the numerical field of
                                        #so that they can be analyzed as c

grouped_genders = df.groupby('gender').count() #grouping by gender
sorted_gender = (grouped_genders.tripduration.sort_values(ascending = Fal
sorted_gender
```

Out[60]:
```
gender
1    1218524
2     444132
0     215442
Name: tripduration, dtype: int64
```

```
There are 1,218,524 males, 444,132 females and 215,442 customers who
aren't registered users hence their gender is unknown
```

In [37]:
```
df['age'] = 2017-df['birth year']
df.age.describe()
```

Out[37]:
```
count    1.670630e+06
mean     3.746549e+01
std      1.171327e+01
min      1.600000e+01
25%      2.800000e+01
50%      3.500000e+01
75%      4.500000e+01
max      1.320000e+02
Name: age, dtype: float64
```

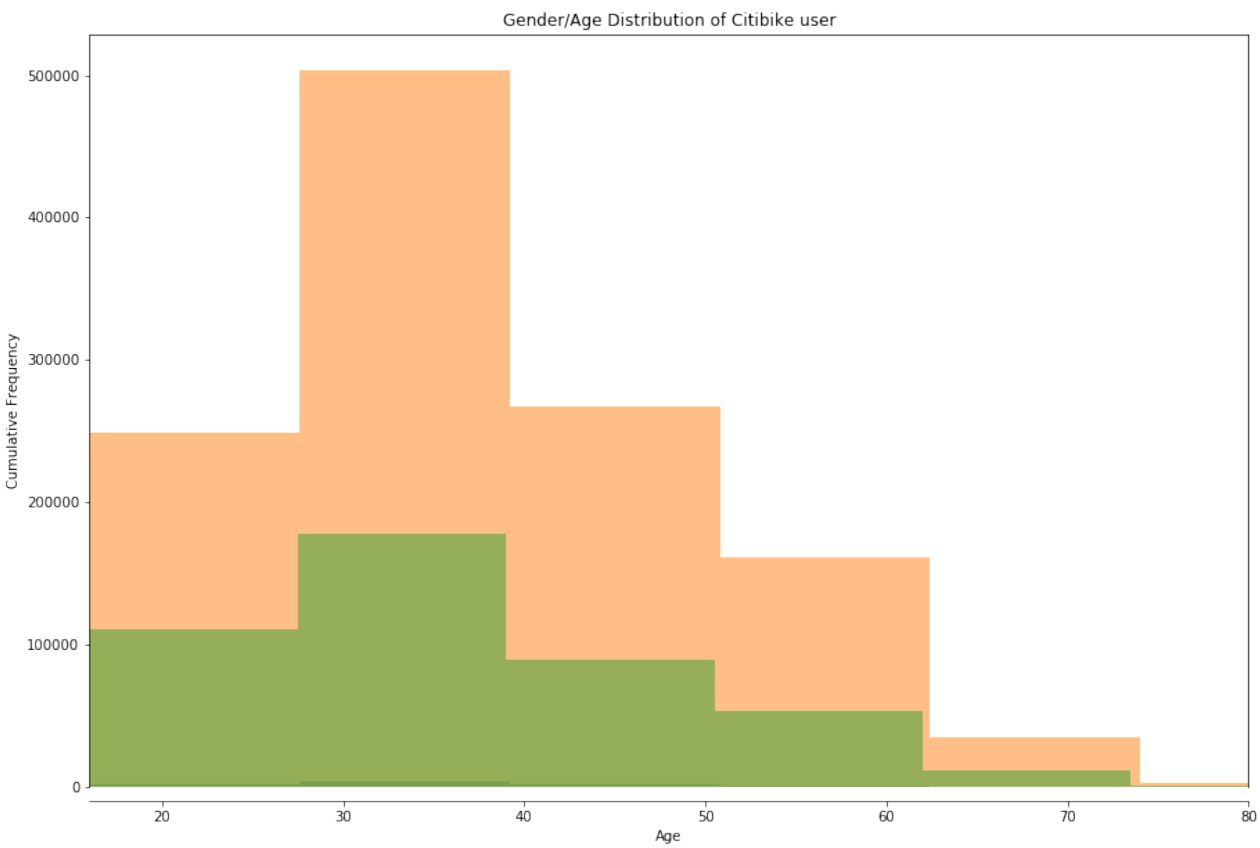
As the values in the series are floats (and attempts to convert them to integers failed due to NaN fields), summary statistics are dispayed in scientific notation here.

- Average age of a citibike subscriber seems to be 37.46 years old.
- The mid 50% of the subscribers range between 28 and 45 years old.
- Furthermore, a standard deviation of 17 years is actually pretty high.

(There seems to be an error in birthyear input int the data field because according to the max age, there was a 132 year old rider in the month of Sept, which of course, seems highly unlikely.)

This next graph shows the cumulatvive frequency in male and female citi riders over different age ranges. It gives us a good idea as to how users are distributed in terms of gender and age. "

In [40]:
```
fig, ax = plt.subplots()
df.groupby('gender').age.plot(ax = ax,
                             figsize = (15,10),
                            kind = 'hist',
                             alpha = 0.5,
                            title = 'Gender/Age Distribution of Citibike
ax.set_xlabel("Age")
ax.set_ylabel("Cumulative Frequency")
#ax.spines['left'].set_position(('outward, 10'))
ax.spines['bottom'].set_position(('outward', 10))
ax.spines['top'].set_visible('False')
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
# ax.text(55, 150, 'Green: Women', fontsize = 12)
# ax.text(80, 140, 'Orange: Men', fontsize = 12)
legend = True
ax.set_xlim(16,80)
plt.show()
```




# Most popular stations

In [41]:
```
g = df.groupby('start_station_name').count() #sorting by start station na
g_sorted = g.tripduration.sort_values(ascending=False)#arranding in desce
```

The 10 most popular start stations are:

In [42]:
```
g_sorted.head(10)
```

Out[42]:
```
start_station_name
Pershing Square North    17359
West St & Chambers St    13132
E 17 St & Broadway       12391
W 21 St & 6 Ave          11517
Broadway & E 22 St       11483
12 Ave & W 40 St         11356
8 Ave & W 31 St          11351
Broadway & E 14 St       10634
8 Ave & W 33 St          10226
W 38 St & 8 Ave           9901
Name: tripduration, dtype: int64
```

In [43]:
```
g1 = df.groupby('end_station_name').count() #sorting by end station names
g_sorted1 = g1.tripduration.sort_values(ascending=False) #arranging in de
```

The 10 most popular end stations are:

In [44]:
```
g_sorted1.head(10)
```

Out[44]:
```
end_station_name
Pershing Square North    17056
West St & Chambers St    14917
E 17 St & Broadway       12704
Broadway & E 22 St       11830
W 21 St & 6 Ave          11668
8 Ave & W 31 St          11487
12 Ave & W 40 St         11361
Broadway & E 14 St       10617
W 20 St & 11 Ave         10263
8 Ave & W 33 St           9917
Name: tripduration, dtype: int64
```

What is interesting to observe is that almost all of these stations feature both in the most popular origination points and also destination points. And if the frequency for end station is higher, one can infer that there would have shortages of bikes at that dock at some point in the month.

```
For example Pershing Sq North has 17056 instances as destinations yet
17359 for instances of trip origination. This implies that the system
```

```
didnt replenish the bikes on its own and Citibike had to restock them
themselves. Contrastinly the dock at West St and Chambers St seeminly
maintained a healthy reserve of bikes with destinations outnumbering
origination point data by 1785 bikes.
```

Given that the most popular origination point is Pershing Sq North, it begs the question, where are these people going? We will try to find out the most popular destinations from Pershing Sq North (PSN)

In [56]:
```
PSN = df[df['start_station_name']== 'Pershing Square North'] #create df w
PSN_grouped = PSN.groupby('end_station_name').count().reset_index() #grou
PSN_grouped.end_station_name.sort_values(ascending = False).head() #and s
```

Out[56]:
```
485                York St & Jay St
484    Wythe Ave & Metropolitan Ave
483              Wyckoff St & 3 Ave
482         Willoughby St & Fleet St
481         Willoughby Ave & Hall St
Name: end_station_name, dtype: object
```

Above are the top 5 destinations that people go to when they embark from Pershing Sq North

## Most popular times; trends in usage

This section focuses on the timeseries aspect of the datafile. But the data itself is presente in strings. So the following manipulation brings us tot the desired state from where we can take off.

In [68]:
```
starttime = df.starttime.tolist() #list of start time series values as st
stoptime = df.stoptime.tolist()

##remove seconds field because it created too many unique instances and w

for i in range(len(starttime)):
    starttime[i] = starttime[i][:-3]

for i in range(len(stoptime)):
    stoptime[i] = stoptime[i][:-3]

df['starttime']= pd.to_datetime(starttime) #convert to datetime format
df['stoptime']= pd.to_datetime(stoptime) #convert to datetime format
```

### Usage of Citibikes on Sept 1

**How usage fluctuated at various points in the day**

- First we will group the trips by their start times (ie. 9.05am)
- Next we arrange them in descending order by the number of trips at each point in the day (ie. 43 trips at 9.05am)

Below is a plot of how citibike usage fluctuated throughout September 1st. The graph gives us a good idea of the peak times in the day.

In [71]:
```
df0901 = df[(df['starttime'] > '2017-09-01 00:00') & (df['starttime'] <=

df0901.starttime.describe() # shows number of trips on the 1st of Sept
```

Out[71]:
```
count                     54877
unique                     1390
top       2017-09-01 08:35:00
freq                        109
first     2017-09-01 00:01:00
last      2017-09-01 23:59:00
Name: starttime, dtype: object
```

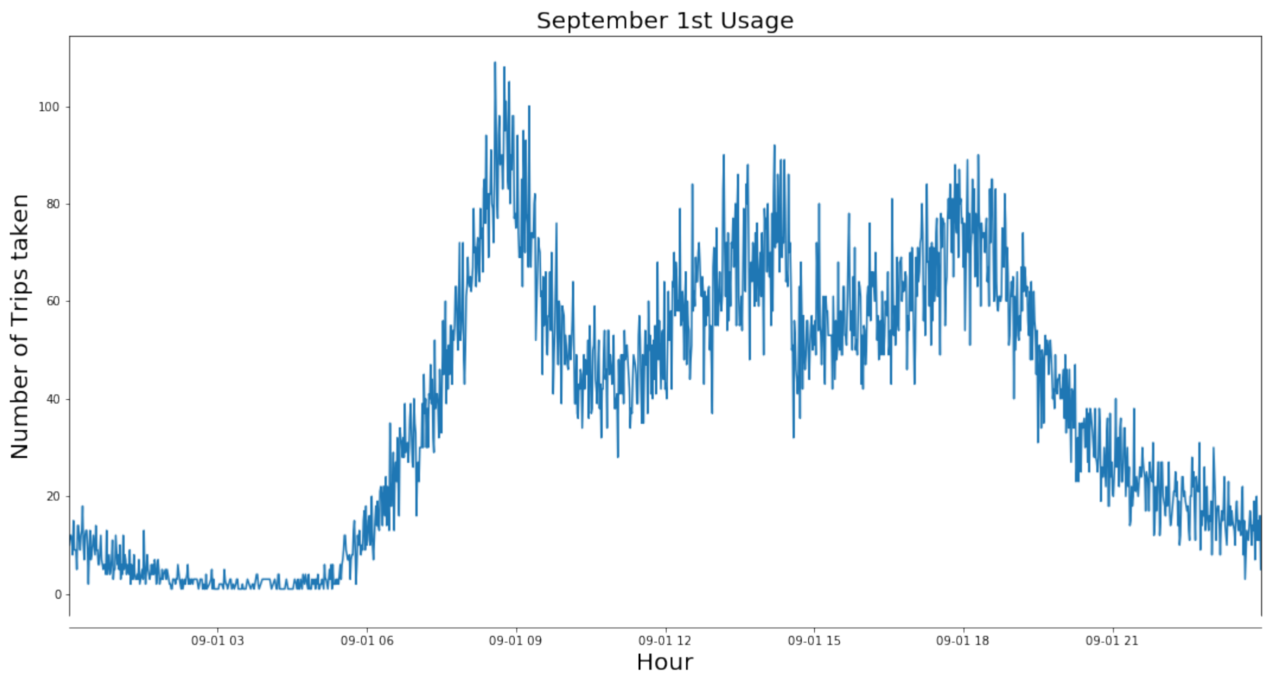
In [80]:
```
df0901 = df[(df['starttime'] > '2017-09-01 00:00') & (df['starttime'] <=
sept_grouped = df0901.groupby('starttime').count() #sorts by starttimes
(sept_grouped.tripduration.sort_values(ascending = False)).head()
```

Out[80]:
```
starttime
2017-09-01 08:35:00    109
2017-09-01 08:46:00    108
2017-09-01 08:52:00    105
2017-09-01 08:48:00    101
2017-09-01 09:16:00    100
Name: tripduration, dtype: int64
```

In [81]:
```
fig, ax = plt.subplots(figsize = (18,10))
sept_grouped.tripduration.plot(kind = 'line')

ax.set_xlabel("Hour", fontsize = 20)
ax.set_ylabel("Number of Trips taken", fontsize = 20)
ax.spines['bottom'].set_position(('outward', 10))
ax.spines['top'].set_visible('False')
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
ax.set_title("September 1st Usage", fontsize = 20)

plt.show()
```




- We can see that the peak point in the day is 8.35am with 109 trips originating at that point.
- We can observe a general peak usage period around 8.30-9.15am which is logical due to the morning rush hour.
- There is further spikes in activity during the afternoon with 1.11pm and 2.12pm having 90 and 93 trips respectively.
- Furthermore, the evening rush hour period starts at around 6.18pm.

## Exploring the week of Sept 1st to Sept 7th

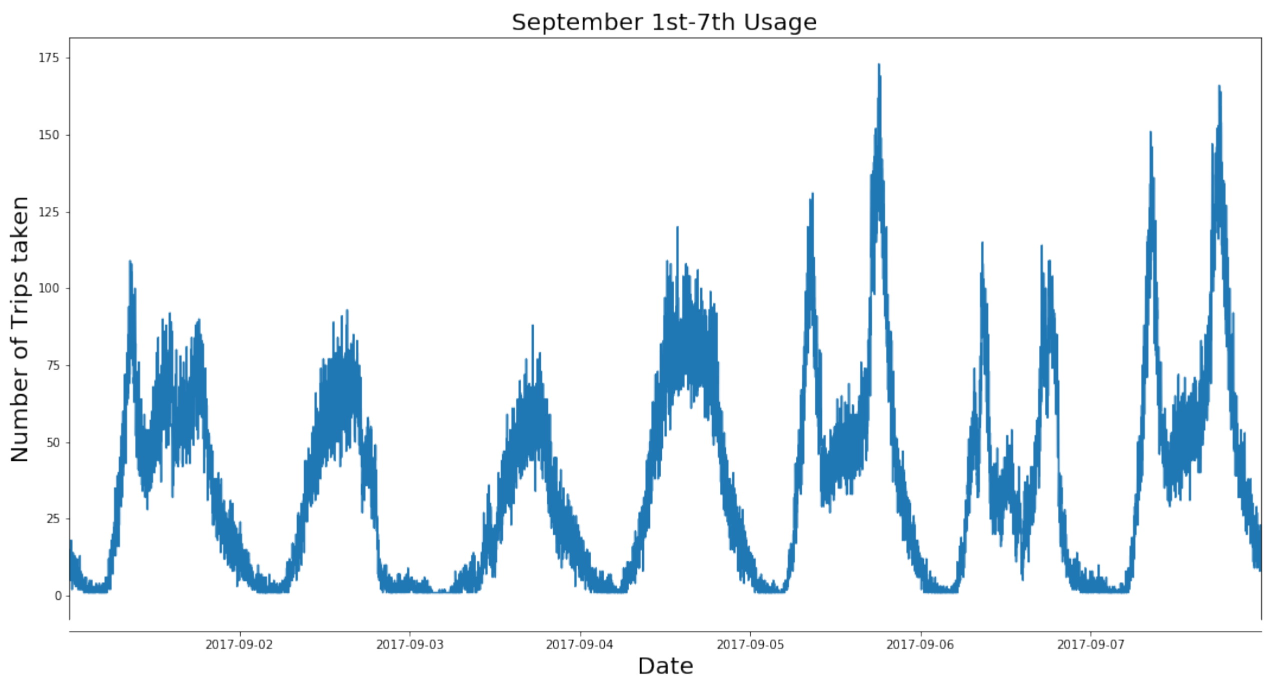
- We can then look at the trend throughout the week by changing the date parameters for the rows we import.
- Looking at the period between Sept 1st and Sept 7th, we can plot the following graph.
- We can clearly see a spike in the peak time activity on Sept 5th.

In [78]:
```
df0901_07 = df[(df['starttime'] > '2017-09-01 00:00') & (df['starttime']

grouped = df0901_07.groupby('starttime').count() #sorts by starttimes
grouped2 = (grouped.tripduration.sort_values(ascending = False)).head()
fig, ax = plt.subplots(figsize = (18,10))
grouped.tripduration.plot(kind = 'line')

ax.set_xlabel("Date", fontsize = 20)
ax.set_ylabel("Number of Trips taken", fontsize = 20)
ax.spines['bottom'].set_position(('outward', 10))
ax.spines['top'].set_visible('False')
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')
ax.set_title("September 1st-7th Usage", fontsize = 20)

plt.show()
```




In [79]:
```
grouped2.head()
```

Out[79]:
```
starttime
2017-09-05 18:08:00    173
2017-09-05 18:22:00    169
2017-09-05 18:13:00    167
2017-09-07 18:07:00    166
2017-09-07 18:18:00    164
Name: tripduration, dtype: int64
```

- A closer look at the peak times shows us that the peak was 6.08pm on Sept 5th with 173 trips.

- It is interesting to note that the peak period on Sept 5th was the evening rush hour unlike the morning rush hour we observed on Sept 1st.
- Also worth considering that Sept 4th is a Monday, and the days Sept 4th-7th (ie weekdays) show a gradual rise in activity over afternoon hours as compared to weekend days (Sept 2nd and 3rd), where they activity peaks in the afternoon.
- Interestingly, weekday peak usage periods are morning and evening rush hours (quite logically, of course) but what is strange is that peak usage on weekends is in the afternoon.
    - One could hypothesize that people are taking a leisurely bike ride around town or running errands in the afternoon.

## Speed

Given distances and duration of our trips, we can easily calculate the average speeds of every trip.

In [124]:
```
for row in range(100):
    df0901['dist_km'] = haversine(df0901.start_long.iloc[row],
                                  df0901.start_lat.iloc[row],
                                  df0901.end_long.iloc[row],
                                  df0901.end_long.iloc[row])

df0901['bike_time'] = df0901['tripduration']/60
```

In [125]:
```
df0901['speed_kmh'] = df0901['dist_km']/(df0901['bike_time']/60)
```

In [126]:
```
df0901.head()
```

Out[126]:

| station_name | start_lat | start_long | end station id | end_station_name | end_lat | end_long | bikeid | u |
|---|---|---|---|---|---|---|---|---|
| Lafayette St & Jersey St | 40.724305 | -73.996010 | 3431 | E 35 St & 3 Ave | 40.746524 | -73.977885 | 25413 | Su |
| ıt Ave & N 7 St | 40.720368 | -73.961651 | 3358 | Garfield Pl & 8 Ave | 40.671198 | -73.974841 | 17584 | Su |
| W 106 St & msterdam Ave | 40.800836 | -73.966449 | 3343 | W 107 St & Columbus Ave | 40.799757 | -73.962113 | 28581 | Su |
| ' 20 St & 8 Ave | 40.743453 | -74.000040 | 388 | W 26 St & 10 Ave | 40.749718 | -74.002950 | 30470 | Su |
| 56 St & 10 Ave | 40.768254 | -73.988639 | 529 | W 42 St & 8 Ave | 40.757570 | -73.990985 | 18150 | Su |

# Focus on the most popular citibike

## This section will analyze the most popular bike in September using its bike-id

In [192]:
```
df['bikeid']= df['bikeid'].astype(str)
df['bikeid'].describe()
```

Out[192]:
```
count     1878098
unique      11615
top         30458
freq          419
Name: bikeid, dtype: object
```

As we can see, although there have been 1,879,098 Citibike trips in the month of September, there are only 11,615 active bikes around the city. The most popular one seems to be bike nymber '30458' and has shown up in 419 different trips. This is the bike we will track.

In [194]:
```
df30458 = df[df['bikeid']== '30458'] #create df with only rows that have
x = df30458.groupby('start_station_name').count()
x.sort_values(by= 'tripduration', ascending = False).head() # we use trip
                                                              #in descendin
```

Out[194]:

| start_station_name | tripduration | starttime | stoptime | start station id | start_lat | start_long | end station id | end_s |
|---|---|---|---|---|---|---|---|---|
| 1 Ave & E 62 St | 7 | 7 | 7 | 7 | 7 | 7 | 7 | |
| 5 Ave & E 88 St | 5 | 5 | 5 | 5 | 5 | 5 | 5 | |
| Carmine St & 6 Ave | 5 | 5 | 5 | 5 | 5 | 5 | 5 | |
| Barrow St & Hudson St | 4 | 4 | 4 | 4 | 4 | 4 | 4 | |
| W 18 St & 6 Ave | 4 | 4 | 4 | 4 | 4 | 4 | 4 | |

Above are the 5 most frequent points that bike 30458 originates from.

In [196]:
```
df30458 = df[df['bikeid']== '30458'] #create df with only rows that have
x = df30458.groupby('end_station_name').count()
x.sort_values(by= 'tripduration', ascending = False).head() # we use trip
                                                              #in descendin
```

Out[196]:

| end_station_name | tripduration | starttime | stoptime | start station id | start_station_name | start_lat | start_l |
|---|---|---|---|---|---|---|---|
| 1 Ave & E 62 St | 7 | 7 | 7 | 7 | | 7 | 7 |
| Carmine St & 6 Ave | 5 | 5 | 5 | 5 | | 5 | 5 |
| 5 Ave & E 88 St | 5 | 5 | 5 | 5 | | 5 | 5 |
| W 18 St & 6 Ave | 4 | 4 | 4 | 4 | | 4 | 4 |
| Greenwich Ave & 8 Ave | 4 | 4 | 4 | 4 | | 4 | 4 |

Above are the 5 most frequent locations that bike 30458 ends up at.

Interesting that 4 out of the 5 stations are the same as the top origination points. Could imply that many people just shuttle between those points and bike 30458 seems to be the most frequently used bike for this short trip.

# Googlemaps Directions API

## Using the API to figure out driving and walking times for the routes taken by the citibikes

Now for the biggest question; it it worth our while to take a city bike? To find out, we use to Google Maps API.

Sending the API the start and end station names for the trips in our dataset, the functions 'walking_time' and 'driving_time' will return the walking and driving times for the specific trips and enter these into our existing dataframe.

In [147]:

```
# we use our start and end locations, the standard API format and an API
# we then query the API with this URL and extract a detailed set of direc
origin = df.start_station_name[0].replace(' ', '+') #remove whitespaces
destination = df.end_station_name[0].replace(' ', '+')
endpoint = 'https://maps.googleapis.com/maps/api/directions/json?'
api_key = 'AIzaSyCLNO5mol_LjqDuOkTKLBke4Q9de-6GVy4'

nav_request = 'origin={}&destination={}&key={}'.format(origin,destination
request = endpoint + nav_request
response = urllib.request.urlopen(request).read()
directions = json.loads(response)

print (directions)
```


```
{'geocoded_waypoints': [{'geocoder_status': 'OK', 'place_id': 'ChIJ8aY
qYkH2wokRccUf0WRGIVE', 'types': ['route']}, {'geocoder_status': 'OK',
'place_id': 'ChIJfw68iyb2wokRjDwqmYkl9Mo', 'types': ['route']}], 'rout
es': [{'bounds': {'northeast': {'lat': 40.8162123, 'lng': -73.9620394}
, 'southwest': {'lat': 40.7960836, 'lng': -73.9700112}}, 'copyrights':
'Map data ©2017 Google', 'legs': [{'distance': {'text': '1.7 mi', 'val
ue': 2669}, 'duration': {'text': '8 mins', 'value': 494}, 'end_address
': 'W 100th St, New York, NY 10025, USA', 'end_location': {'lat': 40.7
960836, 'lng': -73.967017}, 'start_address': 'Riverside Dr, New York,
NY, USA', 'start_location': {'lat': 40.8162123, 'lng': -73.9620394}, '
steps': [{'distance': {'text': '1.0 mi', 'value': 1541}, 'duration': {
'text': '3 mins', 'value': 200}, 'end_location': {'lat': 40.8037, 'lng
': -73.9693882}, 'html_instructions': 'Head <b>southwest</b> on <b>Riv
erside Dr</b>', 'polyline': {'points': 'i|bxFvulbMNRLNPLRL\\TD@\\N|EpB
vAn@lAh@f@TZJLDJBFB\\HVD\\BV@V?~AAX?H@L@N@RFFBVHLHxAbAJFFD|CpBpBrAtA~@
XPRXDF`@h@h@n@FFJHVR`@Xf@ZRLHFbB`APJNHrBz@~@b@VLXJz@^pAj@bAd@vAt@dBfAH
```

```
D'}, 'start_location': {'lat': 40.8162123, 'lng': -73.9620394}, 'trave
l_mode': 'DRIVING'}, {'distance': {'text': '0.1 mi', 'value': 165}, 'd
uration': {'text': '1 min', 'value': 45}, 'end_location': {'lat': 40.8
029763, 'lng': -73.96767899999999}, 'html_instructions': 'Turn <b>left
</b> onto <b>W 108th St</b>', 'maneuver': 'turn-left', 'polyline': {'p
oints': 'cn`xFtcnbMFSP]tBcH'}, 'start_location': {'lat': 40.8037, 'lng
': -73.9693882}, 'travel_mode': 'DRIVING'}, {'distance': {'text': '0.4
mi', 'value': 677}, 'duration': {'text': '3 mins', 'value': 158}, 'end
_location': {'lat': 40.7972537, 'lng': -73.9700112}, 'html_instruction
s': 'Turn <b>right</b> at the 1st cross street onto <b>Broadway</b>',
'maneuver': 'turn-right', 'polyline': {'points': 'si`xF~xmbM`A^TDPBL?J
?LBPBbCLN@\\@vBNRAN@J@L@RDPDZHZFJDNDhBv@LDpBnALFJHdBfALHDDhBnAJF'}, 's
tart_location': {'lat': 40.8029763, 'lng': -73.96767899999999}, 'trave
l_mode': 'DRIVING'}, {'distance': {'text': '0.2 mi', 'value': 286}, 'd
uration': {'text': '2 mins', 'value': 91}, 'end_location': {'lat': 40.
7960836, 'lng': -73.967017}, 'html_instructions': 'Turn <b>left</b> on
to <b>W 100th St</b>', 'maneuver': 'turn-left', 'polyline': {'points':
'ye_xFpgnbMLa@bBoFJ]?Q?G?E?C@C`BmFFO'}, 'start_location': {'lat': 40.7
972537, 'lng': -73.9700112}, 'travel_mode': 'DRIVING'}], 'traffic_spee
d_entry': [], 'via_waypoint': []}], 'overview_polyline': {'points': 'i
|bxFvulbM\\b@d@Zb@VzF`ChFzB~@Vt@Hn@@xBAVBb@H^LzBzA~JvGdBzBlA~@hDrB`@Tr
D~A~DdBzCzAnBlAFSP]tBcH`A^f@HX?^FrCNtCPb@?~@NrA\\vB|@~BvAdC`BtBvApBqGJ
]?Q?MjBeG'}, 'summary': 'Riverside Dr and Broadway', 'warnings': [], '
waypoint_order': []}], 'status': 'OK'}
```


In [148]:
```
# the challenge here is to decypher the data dump google returns
# and to make sense of it so we can extract exactly what we need
# taking a closer glance at this chunk of text, we find that what we're r
# so we create a sublist called routes
routes = directions['routes']
routes
```

Out[148]:

```
[{'bounds': {'northeast': {'lat': 40.8162123, 'lng': -73.9620394},
   'southwest': {'lat': 40.7960836, 'lng': -73.9700112}},
  'copyrights': 'Map data ©2017 Google',
  'legs': [{'distance': {'text': '1.7 mi', 'value': 2669},
    'duration': {'text': '8 mins', 'value': 494},
    'end_address': 'W 100th St, New York, NY 10025, USA',
    'end_location': {'lat': 40.7960836, 'lng': -73.967017},
    'start_address': 'Riverside Dr, New York, NY, USA',
    'start_location': {'lat': 40.8162123, 'lng': -73.9620394},
    'steps': [{'distance': {'text': '1.0 mi', 'value': 1541},
      'duration': {'text': '3 mins', 'value': 200},
      'end_location': {'lat': 40.8037, 'lng': -73.9693882},
      'html_instructions': 'Head <b>southwest</b> on <b>Riverside Dr</
b>',
      'polyline': {'points': 'i|bxFvulbMNRLNPLRL\\TD@\\N|EpBvAn@lAh@f@
TZJLDJBFB\\HVD\\BV@V?~AAX?H@L@N@RFFBVHLHxAbAJFFD|CpBpBrAtA~@XPRXDF`@h@
h@n@FFJHVR`@Xf@ZRLHFbB`APJNHrBz@~@b@VLXJz@^pAj@bAd@vAt@dBfAHD'},
      'start_location': {'lat': 40.8162123, 'lng': -73.9620394},
```

```
       'travel_mode': 'DRIVING'},
      {'distance': {'text': '0.1 mi', 'value': 165},
       'duration': {'text': '1 min', 'value': 45},
       'end_location': {'lat': 40.8029763, 'lng': -73.96767899999999},
       'html_instructions': 'Turn <b>left</b> onto <b>W 108th St</b>',
       'maneuver': 'turn-left',
       'polyline': {'points': 'cn`xFtcnbMFSP]tBcH'},
       'start_location': {'lat': 40.8037, 'lng': -73.9693882},
       'travel_mode': 'DRIVING'},
      {'distance': {'text': '0.4 mi', 'value': 677},
       'duration': {'text': '3 mins', 'value': 158},
       'end_location': {'lat': 40.7972537, 'lng': -73.9700112},
       'html_instructions': 'Turn <b>right</b> at the 1st cross street
 onto <b>Broadway</b>',
       'maneuver': 'turn-right',
       'polyline': {'points': 'si`xF~xmbM`A^TDPBL?J?LBPBbCLN@\\@vBNRAN@
J@L@RDPDZHZFJDNDhBv@LDpBnALFJHdBfALHDDhBnAJF'},
       'start_location': {'lat': 40.8029763, 'lng': -73.96767899999999}
,
       'travel_mode': 'DRIVING'},
      {'distance': {'text': '0.2 mi', 'value': 286},
       'duration': {'text': '2 mins', 'value': 91},
       'end_location': {'lat': 40.7960836, 'lng': -73.967017},
       'html_instructions': 'Turn <b>left</b> onto <b>W 100th St</b>',
       'maneuver': 'turn-left',
       'polyline': {'points': 'ye_xFpgnbMLa@bBoFJ]?Q?G?E?C@C`BmFFO'},
       'start_location': {'lat': 40.7972537, 'lng': -73.9700112},
       'travel_mode': 'DRIVING'}],
     'traffic_speed_entry': [],
     'via_waypoint': []}],
   'overview_polyline': {'points': 'i|bxFvulbM\\b@d@Zb@VzF`ChFzB~@Vt@Hn
@@xBAVBb@H^LzBzA~JvGdBzBlA~@hDrB`@TrD~A~DdBzCzAnBlAFSP]tBcH`A^f@HX?^Fr
CNtCPb@?~@NrA\\vB|@~BvAdC`BtBvApBqGJ]?Q?MjBeG'},
   'summary': 'Riverside Dr and Broadway',
   'warnings': [],
   'waypoint_order': []}]
```


In [149]:
```python
# now its slighly less messy, but we still need to extract the exact time
# turns out we're intersted in the values associated with 'legs'
# so we create a list of 'legs' values

routes[0]['legs']
```

Out[149]:


```
[{'distance': {'text': '1.7 mi', 'value': 2669},
  'duration': {'text': '8 mins', 'value': 494},
  'end_address': 'W 100th St, New York, NY 10025, USA',
  'end_location': {'lat': 40.7960836, 'lng': -73.967017},
  'start_address': 'Riverside Dr, New York, NY, USA',
  'start_location': {'lat': 40.8162123, 'lng': -73.9620394},
  'steps': [{'distance': {'text': '1.0 mi', 'value': 1541},
```

```
    'duration': {'text': '3 mins', 'value': 200},
    'end_location': {'lat': 40.8037, 'lng': -73.9693882},
    'html_instructions': 'Head <b>southwest</b> on <b>Riverside Dr</b>
',
    'polyline': {'points': 'i|bxFvulbMNRLNPLRL\\TD@\\N|EpBvAn@lAh@f@TZ
JLDJBFB\\HVD\\BV@V?~AAX?H@L@N@RFFBVHLHxAbAJFFD|CpBpBrAtA~@XPRXDF`@h@h@
n@FFJHVR`@Xf@ZRLHFbB`APJNHrBz@~@b@VLXJz@^pAj@bAd@vAt@dBfAHD'},
    'start_location': {'lat': 40.8162123, 'lng': -73.9620394},
    'travel_mode': 'DRIVING'},
   {'distance': {'text': '0.1 mi', 'value': 165},
    'duration': {'text': '1 min', 'value': 45},
    'end_location': {'lat': 40.8029763, 'lng': -73.96767899999999},
    'html_instructions': 'Turn <b>left</b> onto <b>W 108th St</b>',
    'maneuver': 'turn-left',
    'polyline': {'points': 'cn`xFtcnbMFSP]tBcH'},
    'start_location': {'lat': 40.8037, 'lng': -73.9693882},
    'travel_mode': 'DRIVING'},
   {'distance': {'text': '0.4 mi', 'value': 677},
    'duration': {'text': '3 mins', 'value': 158},
    'end_location': {'lat': 40.7972537, 'lng': -73.9700112},
    'html_instructions': 'Turn <b>right</b> at the 1st cross street on
to <b>Broadway</b>',
    'maneuver': 'turn-right',
    'polyline': {'points': 'si`xF~xmbM`A^TDPBL?J?LBPBbCLN@\\@vBNRAN@J@
L@RDPDZHZFJDNDhBv@LDpBnALFJHdBfALHDDhBnAJF'},
    'start_location': {'lat': 40.8029763, 'lng': -73.96767899999999},
    'travel_mode': 'DRIVING'},
   {'distance': {'text': '0.2 mi', 'value': 286},
    'duration': {'text': '2 mins', 'value': 91},
    'end_location': {'lat': 40.7960836, 'lng': -73.967017},
    'html_instructions': 'Turn <b>left</b> onto <b>W 100th St</b>',
    'maneuver': 'turn-left',
    'polyline': {'points': 'ye_xFpgnbMLa@bBoFJ]?Q?G?E?C@C`BmFFO'},
    'start_location': {'lat': 40.7972537, 'lng': -73.9700112},
    'travel_mode': 'DRIVING'}],
  'traffic_speed_entry': [],
  'via_waypoint': []}]
```


In [150]:
```
#it is clear from this that the exact time is under duration
# we pin point the list by slicing it further and convert the seconds fie
walk_secs = routes[0]['legs'][0]['duration']['value']/60
walk_secs
```

Out[150]: 8.233333333333333

In [151]:
```
walk_mins=round(walk_secs,2) #rounding to 2 dp we get walk_mins
walk_mins
```

Out[151]: 8.23

In [133]:
```
import urllib.request, json

# now we can create a function to automate this process a bit more
def walking_time(origin, destination): #give it start and end locations
    origin = df.start_station_name[row].replace(' ', '+') #remove whitesp
    destination = df.end_station_name[row].replace(' ', '+')
    endpoint = 'https://maps.googleapis.com/maps/api/directions/json?'
    api_key = 'AIzaSyCLNO5mol_LjqDuOkTKLBke4Q9de-6GVy4'

    nav_request = 'origin={}&destination={}&key={}'.format(origin,destina
    request = endpoint + nav_request
    response = urllib.request.urlopen(request).read()
    directions = json.loads(response)
    routes = directions['routes']
    walk_secs = routes[0]['legs'][0]['duration']['value']/60
    walk_mins=round(walk_secs,2)

    return walk_mins
```

In [134]:
```
walk_time= [] #creating a empty list of walk times where the values will
for row in range(len(df)):
    element = walking_time(df['start_station_name'][row], df['end_station
    walk_time.append(element) #add this value to the list
#walk_time
df['walk_time']= walk_time #create data series called walk_time
df.head()
```

Out[134]:

| | tripduration | starttime | stoptime | start station id | start_station_name | start_lat | start_long | end station id |
|---|---|---|---|---|---|---|---|---|
| **0** | 362 | 9/1/17 0:00 | 9/1/17 0:06 | 3331 | Riverside Dr & W 104 St | 40.801343 | -73.971146 | 3328 |
| **1** | 188 | 9/1/17 0:00 | 9/1/17 0:03 | 3101 | N 12 St & Bedford Ave | 40.720798 | -73.954847 | 3100 |
| **2** | 305 | 9/1/17 0:00 | 9/1/17 0:05 | 3140 | 1 Ave & E 78 St | 40.771404 | -73.953517 | 3141 |
| **3** | 223 | 9/1/17 0:00 | 9/1/17 0:04 | 236 | St Marks Pl & 2 Ave | 40.728419 | -73.987140 | 473 |
| **4** | 758 | 9/1/17 0:01 | 9/1/17 0:13 | 3427 | Lafayette St & Jersey St | 40.724305 | -73.996010 | 3431 |

In [152]:
```
# we do the exact same thing for driving times
# the codes here are slighly different
    #for example here instead of legs, we want legs[steps] and that retur
import urllib.request, json
## Driving Time Calculation
def driving_time(origin, destination):
    origin = df.start_station_name[row].replace(' ', '+')
    destination = df.end_station_name[row].replace(' ', '+')
    endpoint = 'https://maps.googleapis.com/maps/api/directions/json?'
    api_key = 'AIzaSyCLNO5mol_LjqDuOkTKLBke4Q9de-6GVy4'

    nav_request = 'origin={}&destination={}&key={}'.format(origin,destina
    request = endpoint + nav_request
    response = urllib.request.urlopen(request).read()
    directions = json.loads(response)
    routes = directions['routes']
    drive_secs = routes[0]['legs'][0]['steps'][0]['duration']['value']
    drive_mins =round(drive_secs/60,2)
    return drive_mins
```

In [136]:
```
drive_time= [] #empty list of drive times
for row in range(len(df)):
    element = driving_time(df.start_station_name[row], df.end_station_nam
    drive_time.append(element) #append to empty list
df['drive_time'] = drive_time #create data series called walk_time
```

In [114]:
```
df.head(10)
```

Out[114]:

| tart_station_name | start_lat | start_long | end station id | end_station_name | end_lat | end_long | bikeid |
|---|---|---|---|---|---|---|---|
| Riverside Dr & W 104 St | 40.801343 | -73.971146 | 3328 | W 100 St & Manhattan Ave | 40.795000 | -73.964500 | 14530 |
| N 12 St & Bedford Ave | 40.720798 | -73.954847 | 3100 | Nassau Ave & Newell St | 40.724813 | -73.947526 | 15475 |
| 1 Ave & E 78 St | 40.771404 | -73.953517 | 3141 | 1 Ave & E 68 St | 40.765005 | -73.958185 | 30346 |
| St Marks Pl & 2 Ave | 40.728419 | -73.987140 | 473 | Rivington St & Chrystie St | 40.721101 | -73.991925 | 28056 |
| Lafayette St & Jersey St | 40.724305 | -73.996010 | 3431 | E 35 St & 3 Ave | 40.746524 | -73.977885 | 25413 |
| Kent Ave & N 7 St | 40.720368 | -73.961651 | 3358 | Garfield Pl & 8 Ave | 40.671198 | -73.974841 | 17584 |
| W 106 St & Amsterdam Ave | 40.800836 | -73.966449 | 3343 | W 107 St & Columbus Ave | 40.799757 | -73.962113 | 28581 |
| W 20 St & 8 Ave | 40.743453 | -74.000040 | 388 | W 26 St & 10 Ave | 40.749718 | -74.002950 | 30470 |
| W 56 St & 10 Ave | 40.768254 | -73.988639 | 529 | W 42 St & 8 Ave | 40.757570 | -73.990985 | 18150 |
| Washington St & Gansevoort St | 40.739323 | -74.008119 | 358 | Christopher St & Greenwich St | 40.732916 | -74.007114 | 28200 |

The above table neatly shows the Google Maps' values in the walk_time and drive_time columns.

It is worth mentioning that the reason I didnt do the entire dataframe is because as we know the dataframe is 1.87mn rows and the API has a per day limit of 100,000. But even then, the code itself takes time to run and 10 rows is enough to give an idea of the point we're trying to make.

So due to this stipulation and also simplicity sake we dont iterate through the entire dataframe.

# Conclusion/ Final Words

As seen by the Google Maps API section, drivng (unfortunately) does seem to win every race against the citibikes. But as predicted citibikes are obviously faster than walking. A further layer of complexity to this investigation could have been analyzing how this changes during peak times. For example would driving still be the fast option during peak times of the day?

In [ ]:

```

```