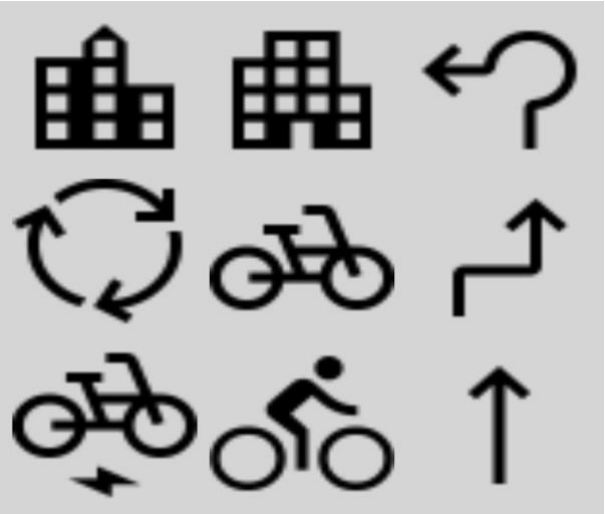
# NYC Citi Bike Analysis – Cyclo-paths

**Project Report for CS-GY - 6513 Big Data**
**Professor – Juan Rodriguez**



## Submitted by


| Full Name | Net Id | N Number |
|---|---|---|
| Chetan Ingle | cmi8525 | N16925270 |
| Sourabh Kumar Bhattacharjee | skb5275 | N15971782 |
| Supritha S Bhat | ssb9863 | N12795140 |

# Table of Contents

## Introduction

NYC as a city is rife with traffic issues which motivates citizens to make use of public transport like subways, buses, or personal two-wheeler rental services such as Citi bike, lime etc. These services provide a much-needed respite from a heavy traffic congestion and the more costly alternatives like Uber or Lyft by providing a cheap and easy way of commuting within the city.

## Problem Statement

However, with the rise of adoption of such services in the past few years, certain problems are being faced regularly by the users such as:
● Many docking stations have little to no bikes available during peak times
● Many docks are completely full creating issues for current riders to dock their rides and
end their trip resulting in additional costs to users.
● Many areas (e.g., Bay Ridge) are still not properly serviced e.g., lack of docking stations
and/or bikes.

## Proposed solution

We would like to design an analysis system which would allow the city officials or any stakeholder to properly analyze the deficits in the programmed in terms of high demand areas, availability of bikes and availability of docking stations, and to take necessary steps to combat these issues.

## Data Sets and Technologies Used

This work focuses on Citi bikes and the datasets which we plan to use for this project is:

1. Citi bike Trips Data
    a. Released Monthly
    b. Contains details of all trips taken

    c. Analysis Range: 2021 – 2022 (~10 GB)
    d. Demand Forecast Range: 2016 – 2022 (~30 GB)
2. Citi bike Live Station Feed API
    a. Refreshes every 30 seconds
    b.
3. NYC Weather Data
    a. Daily Weather features such as snowfall precipitation etc.
4. Citi Bike Monthly Operating Reports
    a. Monthly reports outlining fleet size and number of stations at the end of each month

These datasets provide us the information for each trip and the status of each docking station within a certain timeframe. Using the data, we plan to focus our work on the past 2 years due to computation nd logistic constraints. Even though this dataset is only a few GB in size it can be easily seen how this data scales to a big data problem due to the huge ridership of Citi bikes. We plan on using PySpark and/or Dask.

## Stakeholders

It is critical to identify potential stakeholders that a product is targeted at. Keeping this in mind, we believe that every New Yorker to be the primary stakeholder of this project.

Our detailed analysis & predictions would help these stakeholders in identifying the ongoing trends in the usage of Citi bikes. We can do this taking into consideration several factors such as the bike stations, the duration of the trip, the docker availability and so on.

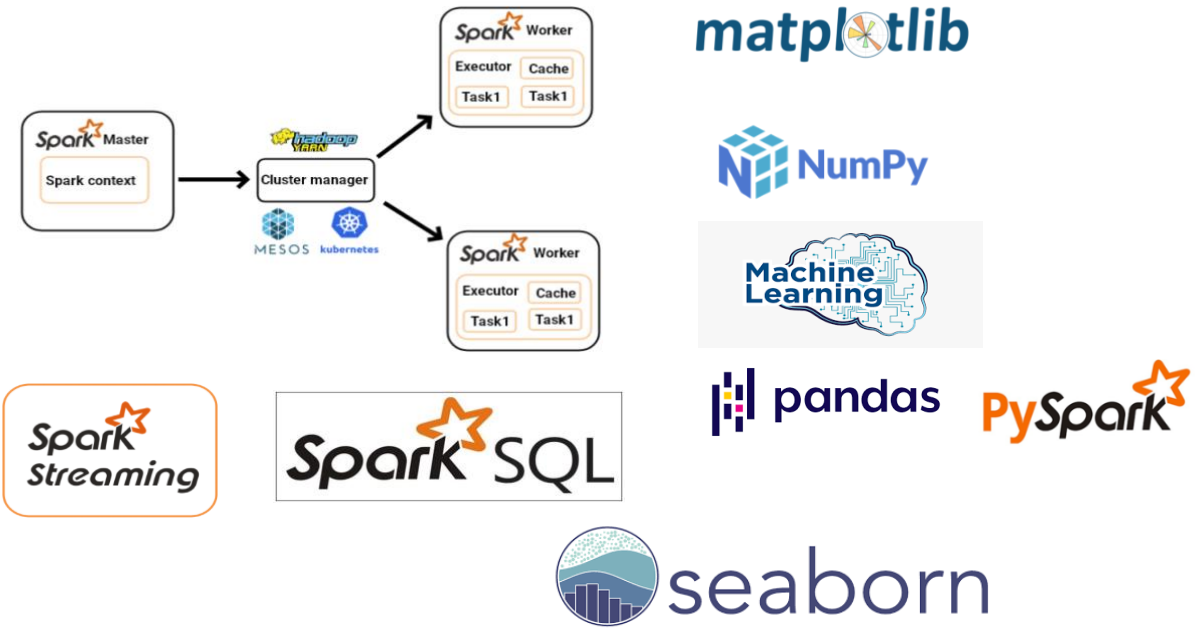
## Why is this a Big Data Problem?

Every day, thousands of people in New York city alone commute on Citi bikes. Considering the number of trips from each year we understand that the data we are dealing with is enormous. In our case, we considered the Citi bike trips data for just two years (2021 to 2022), and the size of the dataset was ~10 GBs with ~53 Million rows. Further, using Citibike's live station feed API, we were able to collect more than 300,000 documents within just 3 days. Hence this is a classic Big Data problem, with potential of

exponential data growth with increased adoption.

## Technology Framework

**PySpark :** At the heart of our analysis is PySpark. We concluded utilizing it because it is fast, has a distributed data processing engine which essentially breaks down data into chunks and distributes work among various workers.




Spark processing occurs completely in memory. It avoids the overhead of I/O calls by doing so. It supports libraries with APIs for several different use-cases. We made use of the following technologies/libraries for this project:

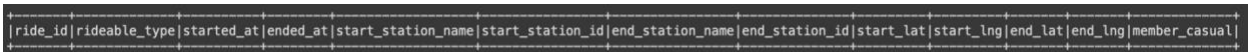
**PySpark SQL**: To process and query our Citi bike datasets.
**Sci-Kit Learn Pipeline**: To predict the daily ridership in 2022 using the data from 2016-2021.
**Matplotlib**: To visualize our findings to help provide better insights.
**Numpy:** To support large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
**Pandas:** In our project, to mainly divide and sort data values into categories.

**Folium:** To enable the binding of data visually to a geospatial data and passing markers on the map.
**Geopy:** To easily locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources.

## Approach

**Preliminary data analysis:** Before proceeding with detailed analysis, we first need to develop a high-level understanding of the data. We need to understand the columns, the fields. We observed that the data prior to 2020 is slightly different and decided to use the data after 2020 for our analysis. Since we have over 10GB of data and we will need more processing memory, we decided to switch from Jupyter notebook to Google Colab.

**Analysis:**
Post data cleaning and preprocessing of the Citi bike trips for 2021-22, we worked with the following to draw insights and predictions from the dataset see the following 13 columns and **53,146,269** rows/trips in the dataset we chose to consider.

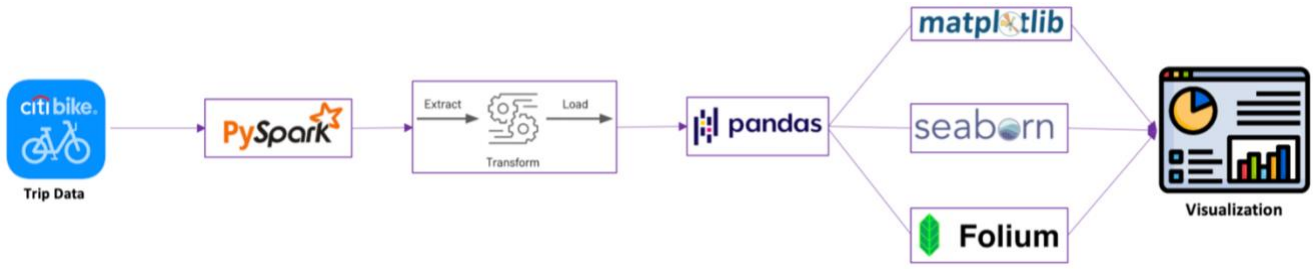



**Visualizations:**
Visualization is critical when conducting an in-depth analysis. They help in better understanding trends/stats that may have a relevant contribution to the outcome. In the project, Matplotlib is primarily used for plotting graph, along with Folium and Geopy to help us plot graph on the geospatial map.

# Analysis

We conducted a series of analyses on the dataset in the recent years (2013 to 2022) to receive insights that may be interesting/relevant to understand what kind of movies perform the best in the industry.

**Data Engineering Pipeline for Trips Analysis:**



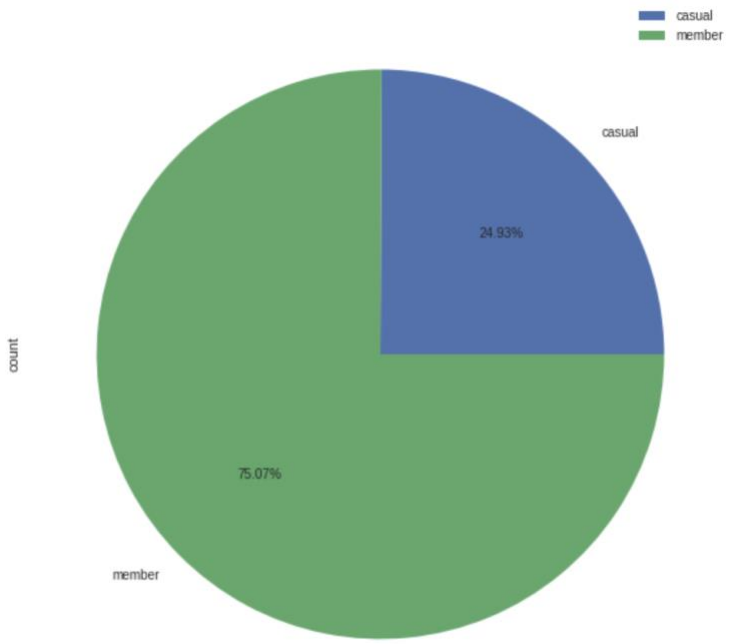

The Citi Bike trips dataset is loaded onto PySpark where we perform ETL (Extract, Transform, and Load) to get useful aggregations. These aggregations are then collected onto pandas which are then visualized using Matplotlib, Seaborn or Folium, to generate insights.

### 1. Members vs Casual riders

We first wanted to figure out what kinds of riders have been riding the Citi bikes. This was our first entry point to our analysis.

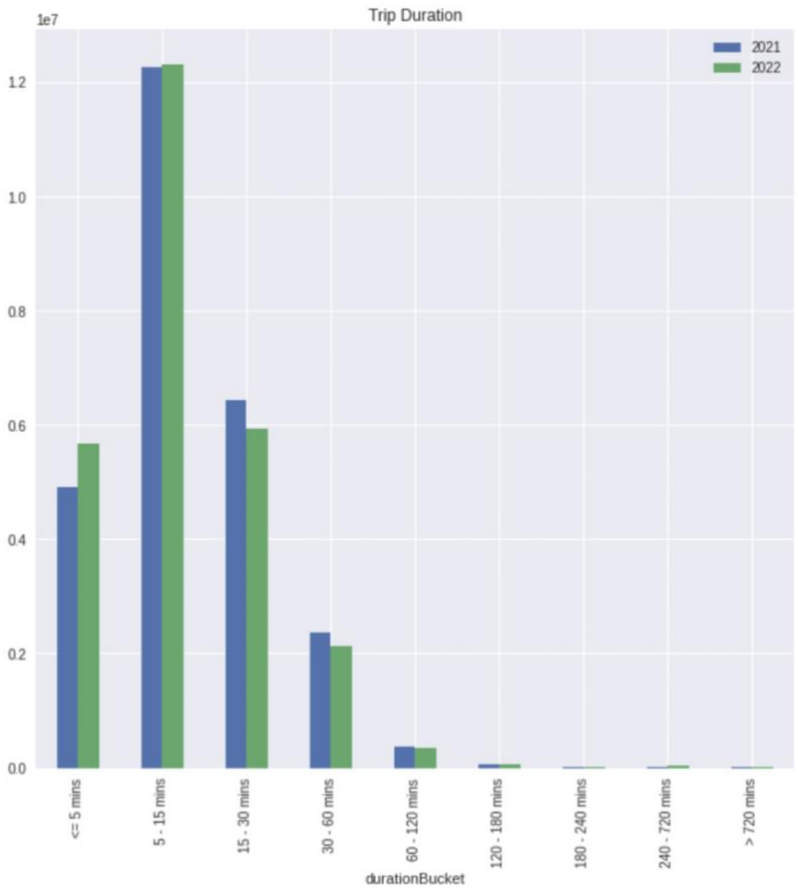
We saw that Citi bikes were mostly riders with Citi membership.

```
#get value counts of membership
mem_df = df.groupBy('member_casual').count().toPandas()

#Plot pie
mem_df.index = mem_df['member_casual']
plot = mem_df.plot.pie(y='count', figsize=(10, 10),autopct='%1.2f%%')
```




## 2. Trip duration

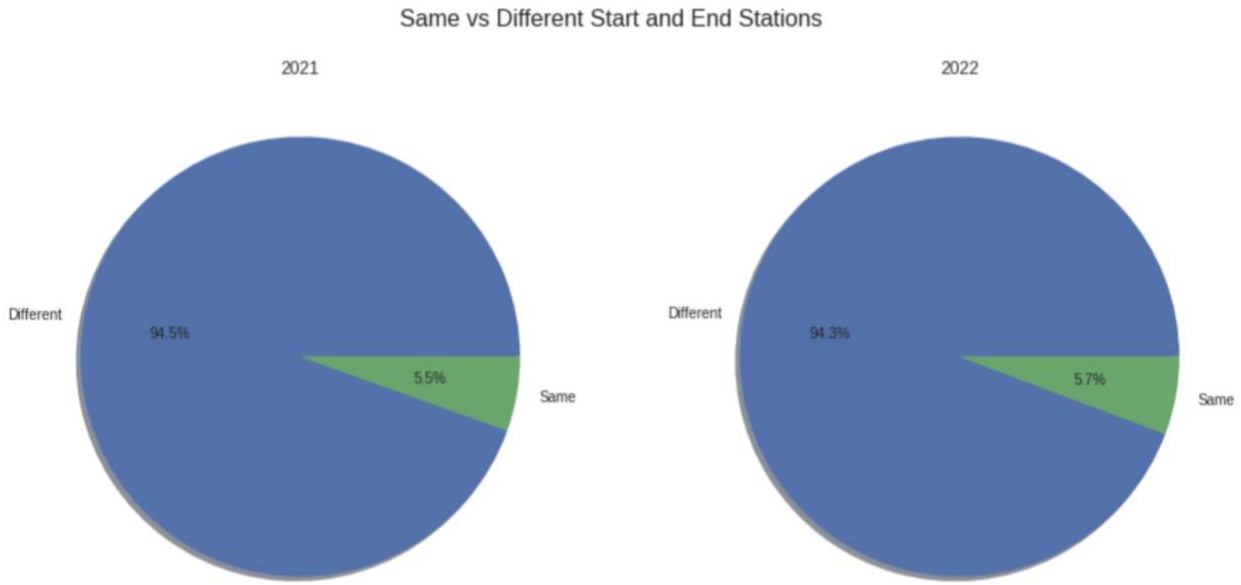
After understanding the type of riders. We decided to explore the trip duration. We get a thorough understanding of the purpose of using the Citi bikes. As we see from the peak of the graph, most riders ride for 5-15 mins. We can conclude that the purpose of using Citi bike is mostly for short trips of 5-15 mins.




```
#udf to calculate buckets

def buckets(x):
  if x <= 5:
    return '<= 5 mins'
  elif x <= 15:
    return '5 - 15 mins'
  elif x <= 30:
    return '15 - 30 mins'
  elif x <= 60:
    return '30 - 60 mins'
  elif x <= 120:
    return '60 - 120 mins'
  elif x <= 180:
    return '120 - 180 mins'
  elif x <= 240:
    return '180 - 240 mins'
  elif x <= 720:
    return '240 - 720 mins'
  elif x <= 99999999:
    return '> 720 mins'
  else:
    return 'None'

def duration(x):
  if x <= 5:
    return 5
  elif x <= 15:
    return 15
  elif x <= 30:
    return 30
  elif x <= 60:
    return 60
  elif x <= 120:
    return 120
  elif x <= 180:
    return 180
  elif x <= 240:
    return 240
  elif x <= 720:
    return 720
  elif x <= 99999999:
    return 721
  else:
    return 'None'

duration_bucket_udf =  udf(lambda dur: buckets(dur/60), StringType())
duration_udf =  udf(lambda dur: duration(dur/60), IntegerType())

duration_df = df.withColumn('tripDurationBucket', duration_bucket_udf(col('trip_duration'))) \
                .withColumn('duration', duration_udf(col('trip_duration'))) \
                .groupby('endYear', 'tripDurationBucket', 'duration').count().toPandas()



#Transform and Plot
dur21 = duration_df[duration_df['endYear'] == 2021][['tripDurationBucket', 'duration', 'count']] \
        .rename(columns = {'tripDurationBucket':'durationBucket', 'duration':'duration', 'count':'2021'}) \
        .sort_values('duration')

dur22 = duration_df[duration_df['endYear'] == 2022][['tripDurationBucket', 'duration', 'count']] \
        .rename(columns = {'tripDurationBucket':'durationBucket', 'duration':'duration', 'count':'2022'}) \
        .sort_values('duration')

dur = dur21.merge(dur22, left_on='durationBucket', right_on='durationBucket')[['durationBucket', '2021', '2022']]

dur.plot(x="durationBucket", y=['2021', '2022'], kind="bar", figsize=(10, 10), title = 'Trip Duration')
```

## 3. Round trip vs one-way trip

After understanding the purpose of the riders, we decided to check if the bikes were consistently used from the same start and end locations. This helps us give us an in-depth purpose of the riders. We analyzed if riders used the Citi bikes for a round-trip or one-way trip.



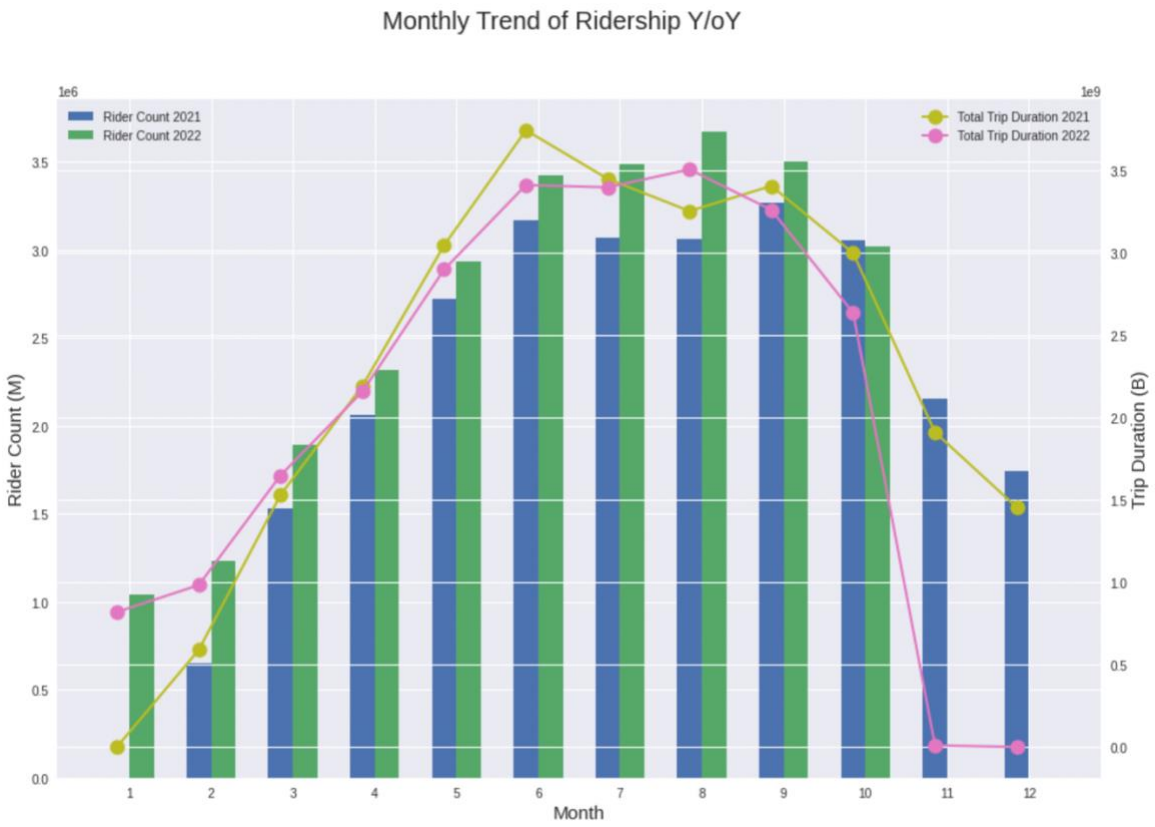

---

As we see, the riders mostly used the Citi bikes for one-way use, for both years with 0.3 points increase from 2021 - 2022. This indicates that an almost constant proportion of riders use Citi bikes for recreational purposes apart from the daily commute obligations

```
#Same start and end stations
sameStartEnd = udf(lambda st, end: 'Same' if st == end else 'Different', StringType())
sameStartEnd_df = df.withColumn('sameStartEnd', sameStartEnd(col('start_station_id'), col('end_station_id')))\
                    .groupBy('endYear', 'sameStartEnd').count().toPandas()

#get number of years
years = list(set(sameStartEnd_df['endYear'].sort_values().tolist()))

#create sublots
fig, axs = plt.subplots(1, len(years))
fig.suptitle('Same vs Different Start and End Stations', fontsize=16)
fig.set_figheight(7)
fig.set_figwidth(15)

#Plot pie charts for each year
for i, year in enumerate(years):
  fracs = sameStartEnd_df[sameStartEnd_df['endYear'] == year]['count'].tolist()
  labels = sameStartEnd_df[sameStartEnd_df['endYear'] == year]['sameStartEnd'].tolist()

  axs[i].pie(fracs, labels=labels, autopct='%1.1f%%', shadow=True)

  axs[i].set_title(year)
```

## 4. Monthly trend

For the next phase of analyses, we decided to check the monthly ridership trend. This helps us understand the demand across the year at a month level along with a Y2Y comparison of the same.

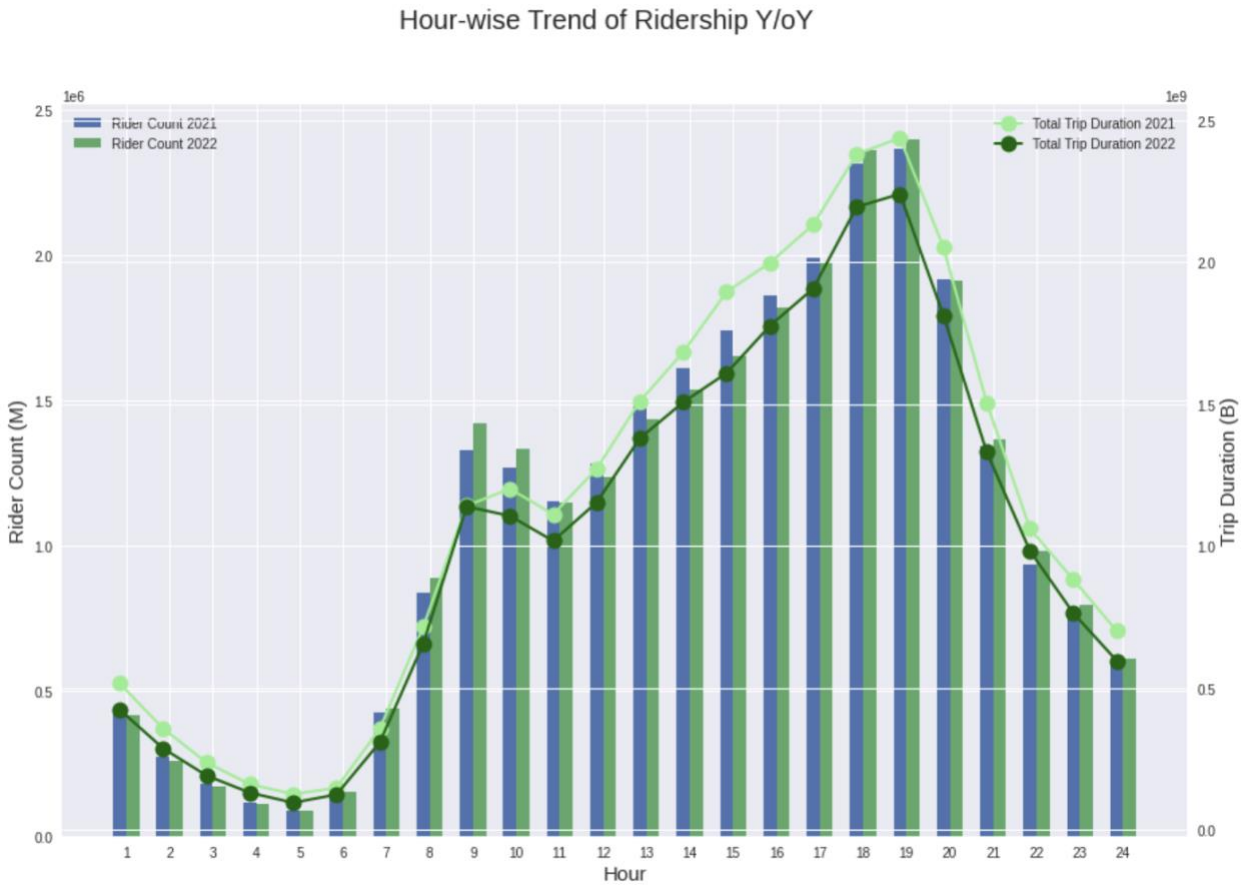



We can see that summer and fall months have a higher ridership than the rest of the year with August and September being the highest for the year 2021 and 2022 respectively. We believe that weather plays an important role in the ridership count and the trip duration

```
from pyspark.sql.functions import sum, count, mean

# get data for each month for all years

month_df = df.groupBy('endYear', 'endMonth').agg(sum('trip_duration').alias('total_trip_duration'), count('ride_id').alias('count')).toPandas().sort_values(['endYear', 'endMonth'])


#get years and months
years = list(set(month_df['endYear'].tolist()))
months = list(set(month_df['endMonth'].tolist()))

#set figure
fig,ax=plt.subplots(figsize=(15,10))
fig.suptitle('Monthly Trend of Ridership Y/oY', fontsize = 20)
ax2=ax.twinx()

#set width of bar
width = 0.3

#loop through each year plot results
for i, year in enumerate(years):
  month_riders_data = month_df[month_df['endYear'] == year]['count'].tolist()
  month_trip_durations = month_df[month_df['endYear'] == year]['total_trip_duration'].tolist()
  month = month_df[month_df['endYear'] == year]['endMonth'].tolist()

  #replace missing values with 0
  for j, m in enumerate(months):
    if m not in month:
      month_riders_data.insert(j, 0)
      month_trip_durations.insert(j, 0)

  #add bar for trip count
  ax.bar(np.array(months) + i*width, height=month_riders_data, width = width, label='Rider Count ' + str(year))

  #add line for trip duration
  ax2.plot(months,
           month_trip_durations,
           label='Total Trip Duration ' + str(year),
           color = colors[random.randint(0, len(colors)-1)],
           marker='o',
           linewidth=2,
           markersize=12)

plt.xticks(np.array(months) + width / 2, months)

ax.set_ylabel('Rider Count (M)', fontsize=15)
ax.set_xlabel('Month', fontsize=15)
ax2.set_ylabel('Trip Duration (B)', fontsize=15)
ax.legend(loc='upper left')
ax2.legend(loc='best')
```

## 5. Hourly trend (Total)

Like the monthly trend, we analyze the hourly riding trend. This helps us understand which hour of the day, Citi bikes are used more often.



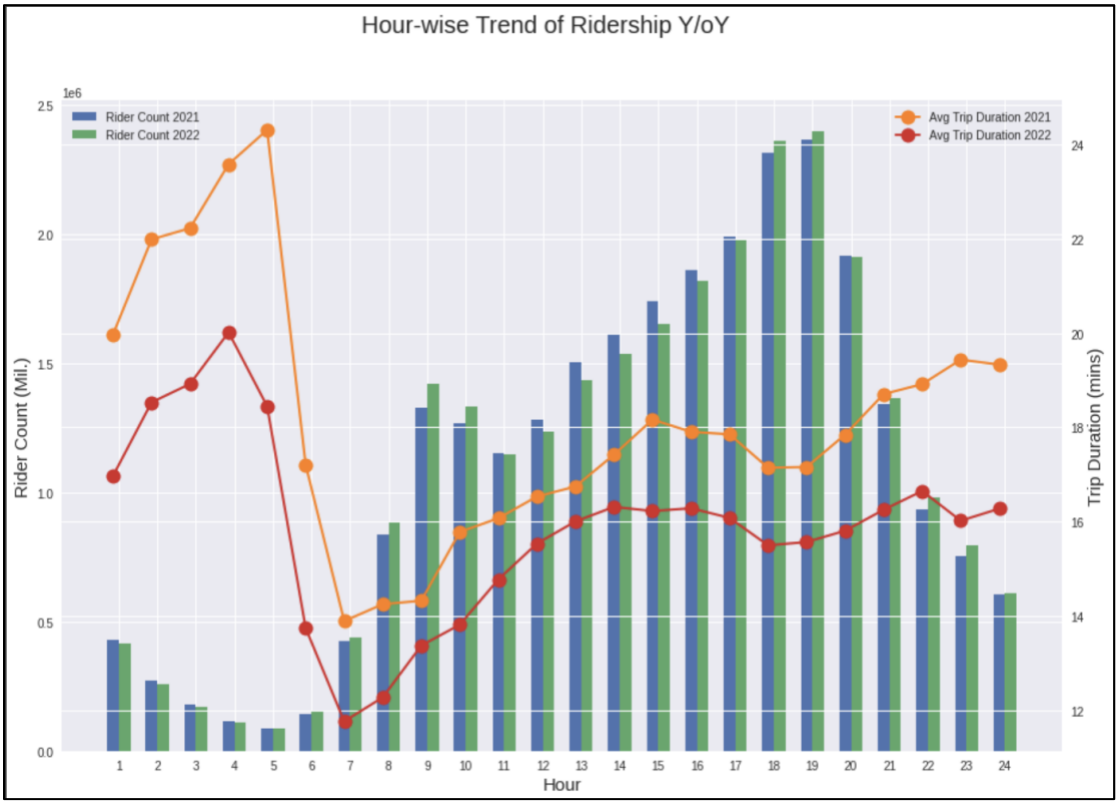

We see that there are two distinct peaks in the graph. One at 6pm in the evening and the other is at 9am in the morning. This is when both the number of riders and the total duration of the rides evidently peak.

```
#get years and hour
years = list(set(hour_df['endYear'].tolist()))
hours = list(range(24))

#set figure
fig,ax=plt.subplots(figsize=(15,10))
fig.suptitle('Hour-wise Trend of Ridership Y/oY', fontsize = 20)
ax2=ax.twinx()

#set width of bar
width = 0.3

#loop through each year plot results
for i, year in enumerate(years):
  hour_riders_data = hour_df[hour_df['endYear'] == year]['count'].tolist()
  hour_trip_durations = hour_df[hour_df['endYear'] == year]['total_trip_duration'].tolist()
  hour = hour_df[hour_df['endYear'] == year]['endHour'].tolist()

  #replace missing values with 0
  for j, h in enumerate(hours):
    if h not in hour:
      hour_riders_data.insert(j, 0)
      hour_trip_durations.insert(j, 0)

  #add bar for trip count
  ax.bar(np.array(hours) + i*width, height=hour_riders_data, width = width, label='Rider Count ' + str(year))

  #add line for trip duration
  ax2.plot(hours,
           hour_trip_durations,
           label='Total Trip Duration ' + str(year),
           color = colors[random.randint(0, len(colors)-1)],
           marker='o',
           linewidth=2,
           markersize=12)

plt.xticks(np.array(hours) + width / 2, list(np.array(hours)+1))

ax.set_ylabel('Rider Count (M)', fontsize=15)
ax.set_xlabel('Hour', fontsize=15)
ax2.set_ylabel('Trip Duration (B)', fontsize=15)
ax.legend(loc='upper left')
ax2.legend(loc='best')

plt.savefig('/content/drive/MyDrive/BigDataProject/yoy_TODRidership.png')
plt.show()
```

Therefore, we believe that the general NYC road traffic has a significant impact on the Citi bike trend.

## 6. Hourly Trend Average

Similar to Hourly Trend (Total), we plot the average ride durations against each hour.



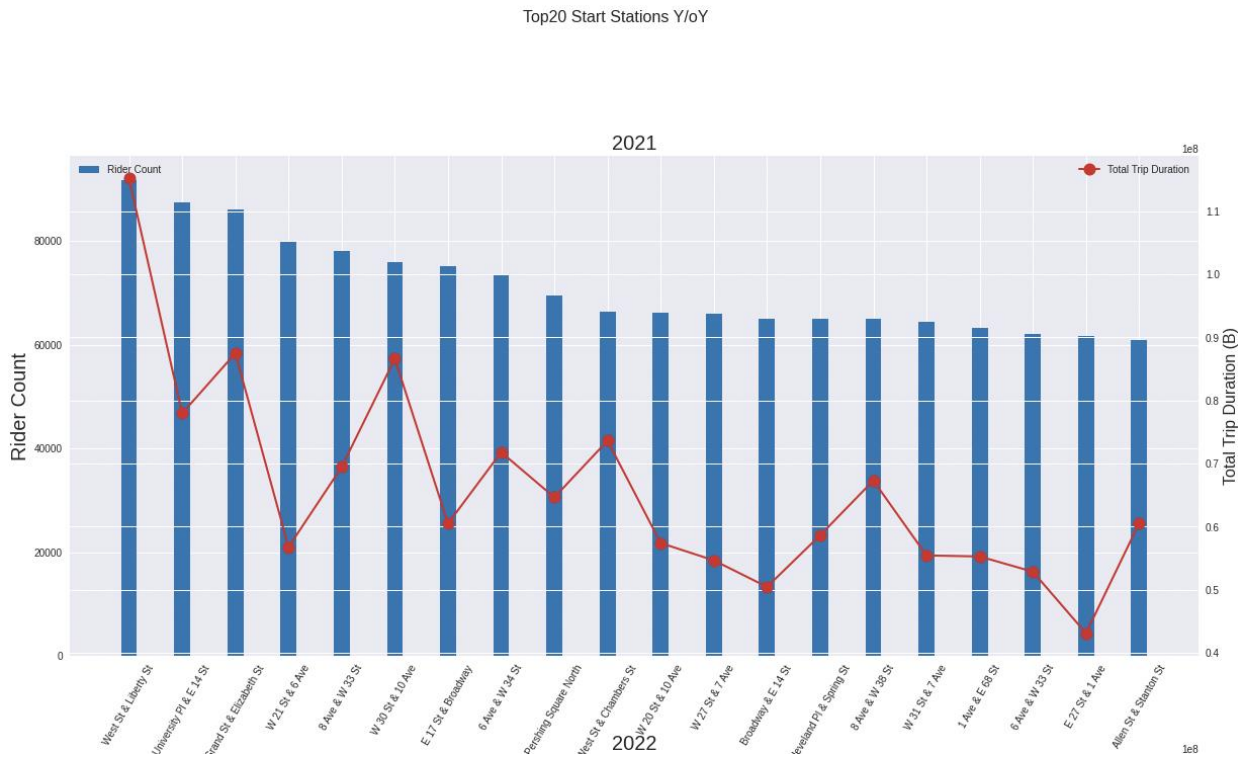

We can see that the average ride duration increases steadily between the two peaks, however, the average ride time is highest at 5 AM in the morning followed by a sharp dip, between 5-7 am.

This may be attributed to the fact that late night workers might be using Citibike to get back home up until 5 AM, post which the number of riders are pretty less.

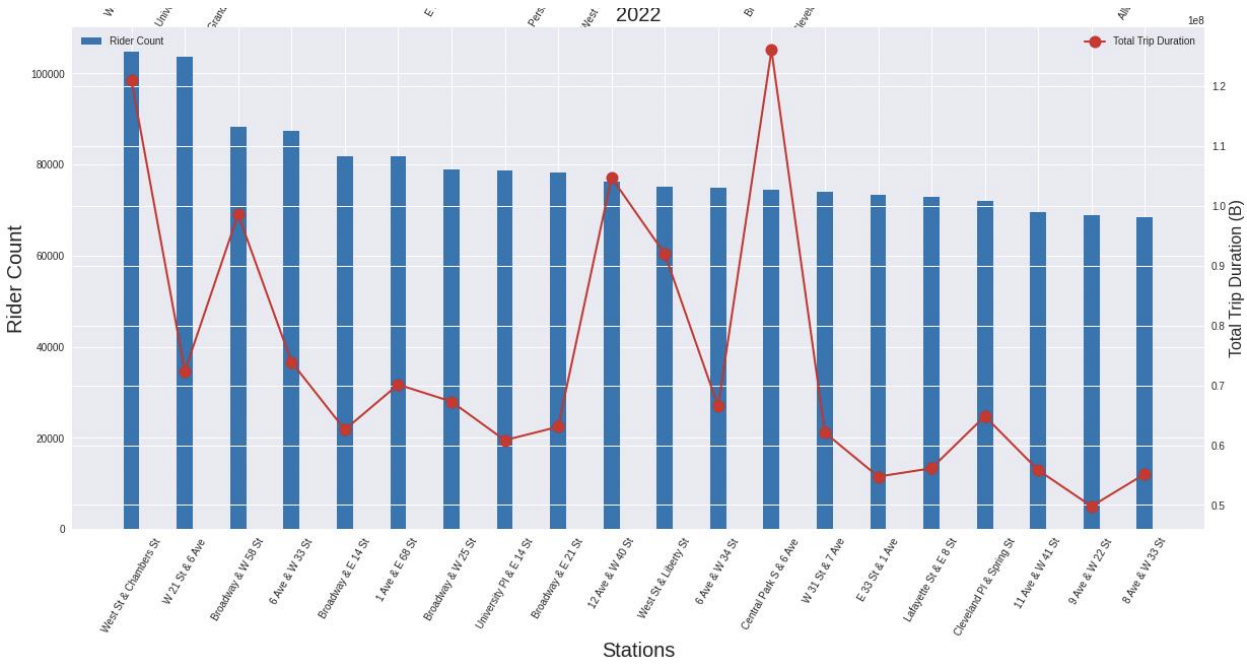
```
#get years and hour
years = list(set(hour_df['endYear'].tolist()))
hours = list(range(24))

#set figure
fig,ax=plt.subplots(figsize=(15,10))
fig.suptitle('Hour-wise Trend of Ridership Y/oY', fontsize = 20)
ax2=ax.twinx()

#set width of bar
width = 0.3

#loop through each year plot results
for i, year in enumerate(years):
  hour_riders_data = hour_df[hour_df['endYear'] == year]['count'].tolist()
  hour_trip_durations = (np.array(hour_df[hour_df['endYear'] == year]['avg_trip_duration'].tolist())/60.0).tolist()
  hour = hour_df[hour_df['endYear'] == year]['endHour'].tolist()

  #replace missing values with 0
  for j, h in enumerate(hours):
    if h not in hour:
      hour_riders_data.insert(j, 0)
      hour_trip_durations.insert(j, 0)

  #add bar for trip count
  ax.bar(np.array(hours) + i*width, height=hour_riders_data, width = width, label='Rider Count ' + str(year))

  #add line for trip duration
  ax2.plot(hours,
           hour_trip_durations,
           label='Avg Trip Duration ' + str(year),
           color = colors[random.randint(0, len(colors)-1)],
           marker='o',
           linewidth=2,
           markersize=12)

plt.xticks(np.array(hours) + width / 2, list(np.array(hours)+1))

ax.set_ylabel('Rider Count (Mil.)', fontsize=15)
ax.set_xlabel('Hour', fontsize=15)
ax2.set_ylabel('Trip Duration (mins)', fontsize=15)
ax.legend(loc='upper left')
ax2.legend(loc='best')

# plt.savefig('/content/drive/MyDrive/BigDataProject/yoy_TODRidership.png')
plt.show()
```

## 7. Most Used Bike Station

We thought it may be interesting to see which station was the most used.




We found that in the year 2021, West St & Liberty St were the most popular.



Whereas in the year 2022, though the number of riders is still high at the West St & Chamber St, the total trip duration is high at Central Park S & 6 Ave.

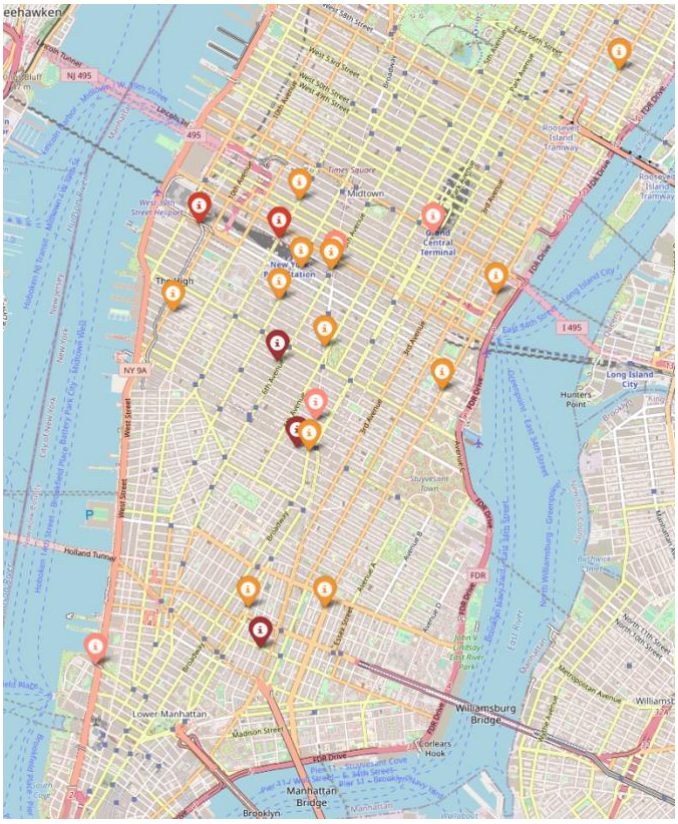
```
from pyspark.sql.functions import sum, count, mean
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, rank, dense_rank

#Define window function
windowSpec  = Window.partitionBy("startYear").orderBy(col("num_trips").desc())

#Get Station Ridership counts for each year, and station
startStationRidership = df.select("ride_id", "start_station_name", "start_lng", "start_lat", "startYear", "trip_duration")\
                          .groupby("startYear", "start_station_name", "start_lng", "start_lat") \
                          .agg(sum('trip_duration').alias('total_trip_duration'), \
                               mean('trip_duration').alias('avg_trip_duration'), \
                               count('ride_id').alias('num_trips')) \
                          .withColumn('stationTripRank', row_number().over(windowSpec)) \
                          .toPandas() \
                          .sort_values(['startYear', 'stationTripRank'])


#set width of bar
width = 0.3

#Set number of top stations needed
top = 20

#create sublots
fig, axs = plt.subplots(len(years), figsize=(20, 20))
fig.suptitle('Top ' + str(top) + ' Start Stations Y/oY', fontsize = 25, y = 0.93)
#fig.set_figheight(7)
#fig.set_figwidth(15)


for i, year in enumerate(years):

  # if i > 0:
  #   box = axs[i].get_position()
  #   box.y0 = box.y0 - 0.05
  #   box.y1 = box.y1 - 0.05
  #   axs[i].set_position(box)


  #Get top stations for given year
  topStations = startStationRidership.loc[(startStationRidership.startYear == year) & (startStationRidership.stationTripRank <= top)]

  #Configure axes
  axs[i].set_title(year, fontsize = 20)
  ax2=axs[i].twinx()

  #plot num_trips
  axs[i].bar(np.array(range(top)),
             height=topStations['num_trips'].tolist(),
             width = width,
             label='Rider Count',
             color='#1f77b4')



  #add line for trip duration
  ax2.plot(topStations['start_station_name'].tolist(),
           topStations['total_trip_duration'].tolist(),
           label='Total Trip Duration',
           color = '#d62728',
           marker='o',
           linewidth=2,
           markersize=12)

  #Set the xLabel and ticks
  axs[i].set_xlabel('Stations', fontsize = 20)
  axs[i].set_xticks(ticks = np.array(range(top))) # + width/2)
  axs[i].set_xticklabels(topStations['start_station_name'].tolist(), rotation = 60, fontsize = 10)

  #Set y labels
  axs[i].set_ylabel('Rider Count', fontsize = 20)
  ax2.set_ylabel('Total Trip Duration (B)', fontsize = 16)

  #set legend
  axs[i].legend(loc='upper left')
  ax2.legend(loc='upper right')

plt.savefig('/content/drive/MyDrive/BigDataProject/yoy_topstartstations.png')
```

This helps us understand the trends in the station.

We now tried plotting the same on a geospatial map.

We see the most popular stations in a darker shade of red and the less popular stations are a lighter orange. We divided the markers into four groups based on the number of trips and colored the markers in the increasing order as orange, light red, red and dark red.

When we click on these markers, we can see more data related to that station like how popular the station is, number of trips, etc.




Station: Grand St & Elizabeth St
Top Station Rank: 3
Trips initiated in 2021: 86116
Total Trip Duration: 1457947 mins
Average Trip Duration: 16 mins
Midtown
Times Square
Grand Central Terminal
Lower Manhattan
Williamsburg Bridge
Manhattan Bridge
Brooklyn Bridge
Holland Tunnel
Lincoln Tunnel
Madison Street
Broadway
West Street
FDR Drive
Stuyvesant Town
Corlears Hook
John V. Lindsay East River Park
East River
Brooklyn

```
#Plot 2021 on Map
try:

  year = int(input('Select Year for which you want to visualize (2021/2022): '))
  topRank = int(input('Enter Number of top Start Stations you want to visualize: '))

  #Filter Data based on input
  tmp = startStationRidership.loc[(startStationRidership.startYear == year) & (startStationRidership.stationTripRank <= topRank)].reset_index(drop = True)
  #Create lists of latitude, longitude, station names and number of trips
  lats = tmp['start_lat']
  longs = tmp['start_lng']
  names = tmp['start_station_name']
  numTrips = tmp['num_trips']
  totalDur = tmp['total_trip_duration']
  avgDur = tmp['avg_trip_duration']
  ranks = tmp['stationTripRank']
  colors = pd.cut(tmp['num_trips'], bins=4, labels=['orange', 'lightred', 'red', 'darkred']) # ['darkgreen', 'green', 'lightgreen', 'yellow', 'orange', 'lightred', 'red', 'darkred']
  places = [[x[0],x[1]] for x in zip(lats,longs)]

  #Render map using first station in list
  map = folium.Map(places[random.randint(0, len(places)-1)], tiles='cartodbpositron', zoom_start=13)
  #Title
  map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Top {topRank} Start Stations of {year}</b></h3>'))

  #Plot and label each subsequent station
  for i, loc in enumerate(places[1:]):
      #Define popup
      popup = folium.Popup(f'''<b>Station</b>: {names[i]}
                           <br><b>Top Station Rank</b>: {ranks[i]}
                           <br><b>Trips initiated in {year}</b>:\n{numTrips[i]}
                           <br><b>Total Trip Duration</b>: {int(totalDur[i]/60.0)} mins
                           <br><b>Average Trip Duration</b>: {int(avgDur[i]/60.0)} mins''',
                       min_width=300, max_width=300)
      folium.Marker(loc, icon=folium.Icon(color=colors[i]), popup=popup).add_to(map)

  display(map)
except:
  print('Invalid Input')
```

Below is the graph plotted with the data from the year 2022.

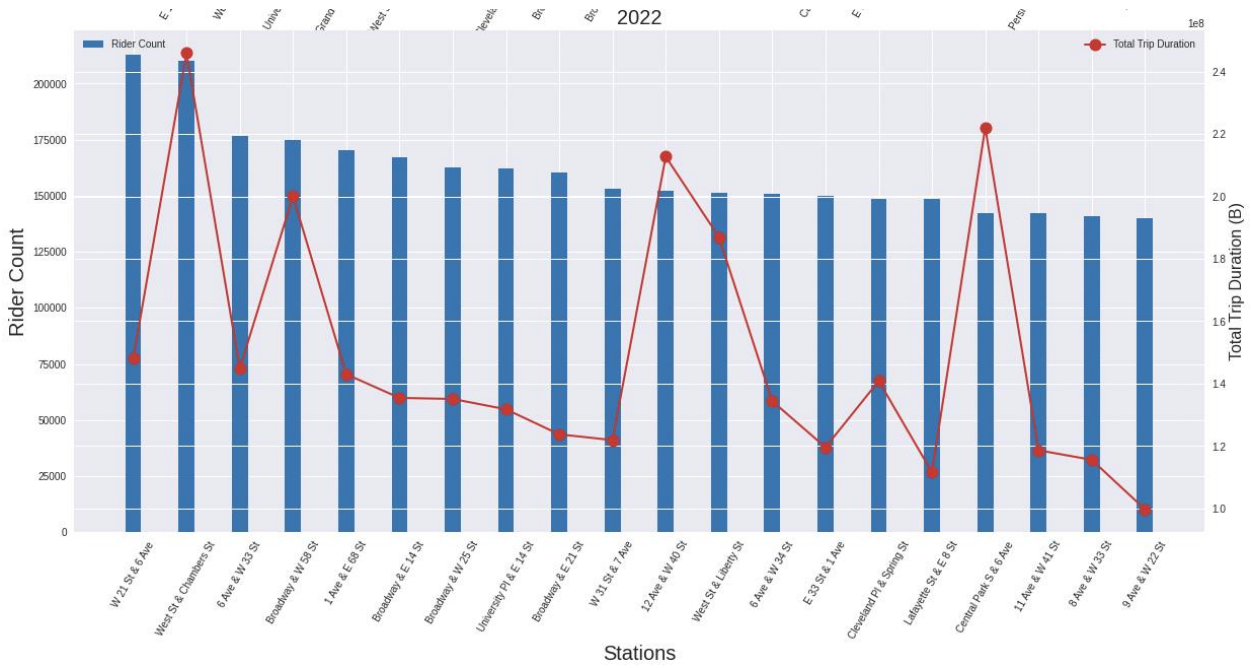
```
#Plot 2022 on Map
try:

  year = int(input('Select Year for which you want to visualize (2021/2022): '))
  topRank = int(input('Enter Number of top Start Stations you want to visualize: '))

  #Filter Data based on input
  tmp = startStationRidership.loc[(startStationRidership.startYear == year) & (startStationRidership.stationTripRank <= topRank)].reset_index(drop = True)
  #Create lists of latitude, longitude, station names and number of trips
  lats = tmp['start_lat']
  longs = tmp['start_lng']
  names = tmp['start_station_name']
  numTrips = tmp['num_trips']
  totalDur = tmp['total_trip_duration']
  avgDur = tmp['avg_trip_duration']
  ranks = tmp['stationTripRank']
  colors = pd.cut(tmp['num_trips'], bins=4, labels=['orange', 'lightred', 'red', 'darkred']) # ['darkgreen', 'green', 'lightgreen', 'yellow', 'orange', 'lightred', 'red', 'darkred']
  places = [[x[0],x[1]] for x in zip(lats,longs)]

  #Render map using first station in list
  map = folium.Map(places[random.randint(0, len(places)-1)], tiles='cartodbpositron', zoom_start=13)
  #Title
  map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Top {topRank} Start Stations of {year}</b></h3>'))

  #Plot and label each subsequent station
  for i, loc in enumerate(places[1:]):
      #Define popup
      popup = folium.Popup(f'''<b>Station</b>: {names[i]}
                               <br><b>Top Station Rank</b>: {ranks[i]}
                               <br><b>Trips initiated in {year}</b>:\n{numTrips[i]}
                               <br><b>Total Trip Duration</b>: {int(totalDur[i]/60.0)} mins
                               <br><b>Average Trip Duration</b>: {int(avgDur[i]/60.0)} mins''',
                           min_width=300, max_width=300)
      folium.Marker(loc, icon=folium.Icon(color=colors[i]), popup=popup).add_to(map)

  display(map)
except:
  print('Invalid Input')
```

The graph is generated dynamically. The user can select the year they want to plot.

We can also allow the user to choose the graph for the top stations instead. The markers for the year 2022 go in the increasing order of the number of trips as light red, red and dark red and the markers for the year 2021 go in the increasing order of the number of trips as light green, green and dark green.

```
#Plot YoY Comparison on Map
try:

  # year = int(input('Select Year for which you want to visualize (2021/2022): '))
  topRank = int(input('Enter Number of top End Stations you want to visualize: '))

  #Filter Data based on input for 2021
  tmp21 = endStationRidership.loc[(endStationRidership.endYear == 2021) & (endStationRidership.stationTripRank <= topRank)].reset_index(drop=True)
  #Create lists of latitude, longitude, station names and number of trips
  lats21 = tmp21['end_lat']
  longs21 = tmp21['end_lng']
  names21 = tmp21['end_station_name']
  numTrips21 = tmp21['num_trips']
  totalDur21 = tmp21['total_trip_duration']
  avgDur21 = tmp21['avg_trip_duration']
  rank21 = tmp21['stationTripRank']
  colors21 = pd.cut(tmp21['num_trips'], bins=3, labels=['lightred', 'red', 'darkred']) # ['darkgreen', 'green', 'lightgreen', 'yellow', 'orange', 'lightred', 'red', 'darkred']
  places21 = [[x[0],x[1]] for x in zip(lats21,longs21)]


  #Filter Data based on input for 2022
  tmp22 = endStationRidership.loc[(endStationRidership.endYear == 2022) & (endStationRidership.stationTripRank <= topRank)].reset_index(drop=True)
  #Create lists of latitude, longitude, station names and number of trips
  lats22 = tmp22['end_lat']
  longs22 = tmp22['end_lng']
  names22 = tmp22['end_station_name']
  numTrips22 = tmp22['num_trips']
  totalDur22 = tmp22['total_trip_duration']
  avgDur22 = tmp22['avg_trip_duration']
  rank22 = tmp22['stationTripRank']
  colors22 = pd.cut(tmp22['num_trips'], bins=3, labels=['lightgreen', 'green', 'darkgreen']) # ['darkgreen', 'green', 'lightgreen', 'yellow', 'orange', 'lightred', 'red', 'darkred']
  places22 = [[x[0],x[1]] for x in zip(lats22,longs22)]

  #Render map using first station in list
  map = folium.Map(places21[random.randint(0, len(places21)-1)], tiles='cartodbpositron', zoom_start=13)
  #Title
  map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>YoY Comparison of Top End Stations</b></h3>'))

  #Plot and label each subsequent station for 2021
  for i, loc in enumerate(places21[1:]):
      #Define popup
      popup1 = folium.Popup(f'''<b>Year</b>: 2021
                            <br><b>Station</b>: {names21[i]}
                            <br><b>Top Station Rank</b>: {rank21[i]}
                            <br><b>Trips culminated in</b>:{numTrips21[i]}
                            <br><b>Total Trip Duration</b>: {int(totalDur21[i]/60.0)} mins
                            <br><b>Average Trip Duration</b>: {int(avgDur21[i]/60.0)} mins''',
                          min_width=300, max_width=300)
      folium.Marker(loc, icon=folium.Icon(color=colors21[i]), popup=popup1).add_to(map)

  #Plot and label each station for 2022
  for i, loc in enumerate(places22):
      #Define popup
      popup2 = folium.Popup(f'''<b>Year</b>: 2022
                            <br><b>Station</b>: {names22[i]}
                            <br><b>Top Station Rank</b>: {rank21[i]}
                            <br><b>Trips culminated in</b>: {numTrips22[i]}
                            <br><b>Total Trip Duration</b>: {int(totalDur22[i]/60.0)} mins
                            <br><b>Average Trip Duration</b>: {int(avgDur22[i]/60.0)} mins''',
                          min_width=300, max_width=300)
      folium.Marker(loc, icon=folium.Icon(color=colors22[i]), popup=popup2).add_to(map)

  # Render map
  display(map)
except:
  print('Invalid Input')
```

We plot the graph for the top station adding the rides from a round trip. The below graph is for the year 2022. We see a difference when the round trip was excluded.

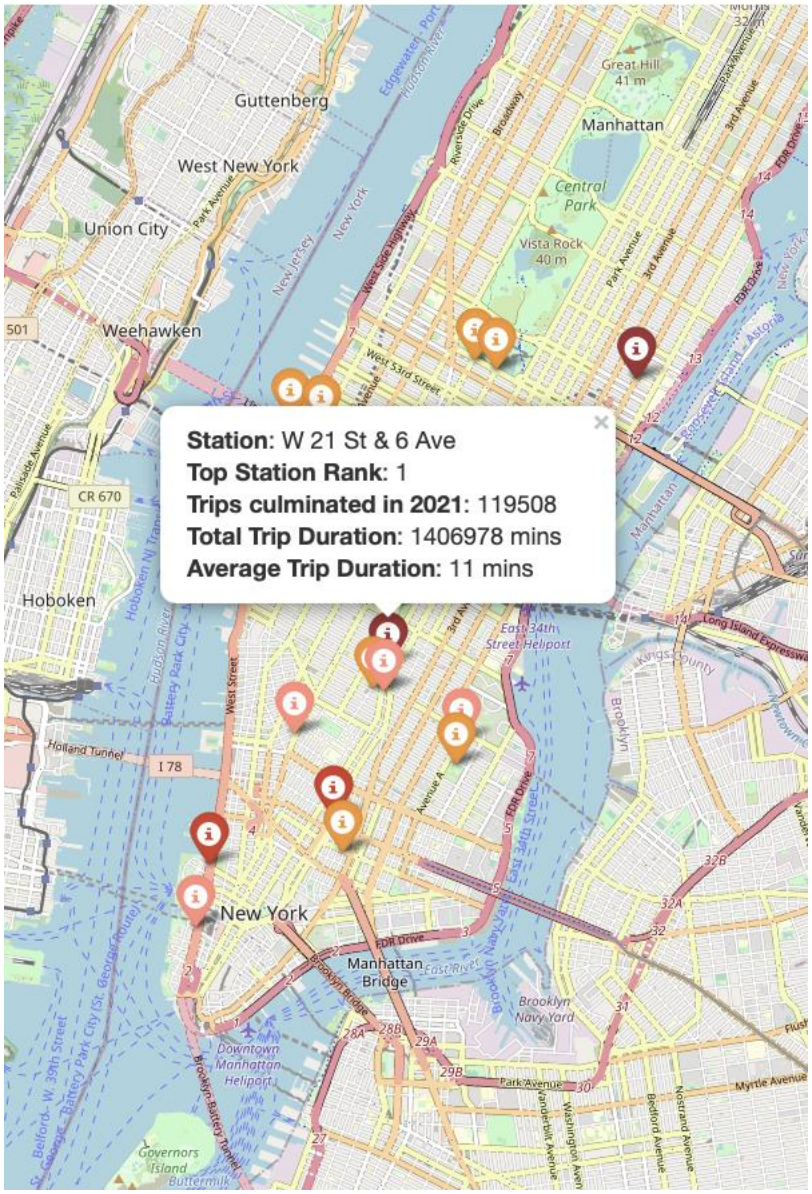


```
from pyspark.sql.functions import sum, count, mean
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number, rank, dense_rank
from pyspark.sql.functions import col,isnan, when, count
from pyspark.sql.functions import col,isnan, when, count

#Get trips starting from each stations which end at a different station
startStationTrips = df.select("ride_id", \
                              col("start_station_name").alias('station_name'), \
                              col("start_lng").alias('lng'), \
                              col("start_lat").alias('lat'), \
                              col("startYear").alias('year'), \
                              "trip_duration" \
                     ) \
                     .filter(col('start_station_name') != col('end_station_name'))

#Get trips ending at each stations which start at a different station
endStationTrips = df.select("ride_id", \
                              col("end_station_name").alias('station_name'), \
                              col("end_lng").alias('lng'), \
                              col("end_lat").alias('lat'), \
                              col("endYear").alias('year'), \
                              "trip_duration" \
                    )\
                    .filter(col('start_station_name') != col('end_station_name'))

#Get trips which start and end at same station
circularTrips = df.select("ride_id", \
                              col("start_station_name").alias('station_name'), \
                              col("start_lng").alias('lng'), \
                              col("start_lat").alias('lat'), \
                              col("startYear").alias('year'), \
                              "trip_duration" \
                     ) \
                     .filter(col('start_station_name') == col('end_station_name')) \
                     .groupby('station_name', 'lat', 'lng', 'year') \
                     .agg(sum('trip_duration').alias('total_trip_duration_c'), \
                          mean('trip_duration').alias('avg_trip_duration_c'), \
                          count('ride_id').alias('num_trips_c'))

# Take union of start and end trips
singleStationTrips = startStationTrips.union(endStationTrips)
#Group and aggregate based on station, and year
singleStationTrips = singleStationTrips.groupby('station_name', 'lng', 'lat', 'year') \
                                       .agg(sum('trip_duration').alias('total_trip_duration_nc'), \
                                            mean('trip_duration').alias('avg_trip_duration_nc'), \
                                            count('ride_id').alias('num_trips_nc'))

#join single station trips with circular trips
totalStationTrips = singleStationTrips.join(circularTrips,
                                            (singleStationTrips.station_name == circularTrips.station_name) \
                                            & (singleStationTrips.lng == circularTrips.lng) \
                                            & (singleStationTrips.lat == circularTrips.lat) \
                                            & (singleStationTrips.year == circularTrips.year), \
                                            'left_outer') \
                                       .drop(circularTrips.station_name) \
                                       .drop(circularTrips.lng) \
                                       .drop(circularTrips.lat) \
                                       .drop(circularTrips.year)


#Define window function
windowSpec  = Window.partitionBy("year").orderBy(col("num_trips").desc())

#Fill na values after join and calculate total duration, avg, and total rides
totalStationTrips = totalStationTrips.na.fill(value=0,subset=["total_trip_duration_c", 'avg_trip_duration_c', 'num_trips_c']) \
                                     .withColumn('num_trips', col('num_trips_nc') + col('num_trips_c')) \
                                     .withColumn('total_trip_duration', col('total_trip_duration_nc') + col('total_trip_duration_c')) \
                                     .withColumn('avg_trip_duration', (col('total_trip_duration_nc') + col('total_trip_duration_c'))/col('num_trips').cast(DecimalType())) \
                                     .select('station_name', 'lng', 'lat', 'year', 'num_trips', 'total_trip_duration', 'avg_trip_duration') \
                                     .withColumn('stationTripRank', row_number().over(windowSpec)) \
                                     .toPandas() \
                                     .sort_values(['year', 'stationTripRank'])
```

```
#set width of bar
width = 0.3

#Set number of top stations needed
top = 20

#create sublots
fig, axs = plt.subplots(len(years), figsize=(20, 20))
fig.suptitle('Top' + str(top) + ' Stations (Overall) Y/oY', fontsize = 25, y = 0.93)
#fig.set_figheight(7)
#fig.set_figwidth(15)


for i, year in enumerate(years):

  # if i > 0:
  #   box = axs[i].get_position()
  #   box.y0 = box.y0 - 0.05
  #   box.y1 = box.y1 - 0.05
  #   axs[i].set_position(box)


  #Get top stations for given year
  topStations = totalStationTrips.loc[(totalStationTrips.year == year) & (totalStationTrips.stationTripRank <= top)]

  #Configure axes
  axs[i].set_title(year, fontsize = 20)
  ax2=axs[i].twinx()

  #plot num_trips
  axs[i].bar(np.array(range(top)),
             height=topStations['num_trips'].tolist(),
             width = width,
             label='Rider Count',
             color='#1f77b4')



  #add line for trip duration
  ax2.plot(topStations['station_name'].tolist(),
           topStations['total_trip_duration'].tolist(),
           label='Total Trip Duration',
           color = '#d62728',
           marker='o',
           linewidth=2,
           markersize=12)

  #Set the xLabel and ticks
  axs[i].set_xlabel('Stations', fontsize = 20)
  axs[i].set_xticks(ticks = np.array(range(top))) # + width/2)
  axs[i].set_xticklabels(topStations['station_name'].tolist(), rotation = 60, fontsize = 10)

  #Set y labels
  axs[i].set_ylabel('Rider Count', fontsize = 20)
  ax2.set_ylabel('Total Trip Duration (B)', fontsize = 16)

  #set legend
  axs[i].legend(loc='upper left')
  ax2.legend(loc='upper right')

plt.savefig(f'/content/drive/MyDrive/BigDataProject/yoy_top{top}Stations.png')
plt.show()
```

We plot the graph on the geospatial map with added information. We divided the markers into four groups based on the number of trips and colored the markers in the increasing order as orange, light red, red and dark red.

**Top End Stations of 2021**




| | start_station_name | start_lng | start_lat | end_station_name | end_lng | end_lat | startYear | total_trip_duration | avg_trip_duration | num_trips | routeRank |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3662182 | Central Park S & 6 Ave | -73.9763415100 | 40.7659093600 | Central Park S & 6 Ave | -73.9763415100 | 40.7659093600 | 2021 | 15674342 | 1998.513579 | 7843 | 1 |
| 3662183 | Roosevelt Island Tramway | -73.9536000000 | 40.7572840000 | Roosevelt Island Tramway | -73.9536000000 | 40.7572840000 | 2021 | 18407886 | 2361.499166 | 7795 | 2 |
| 3662184 | 7 Ave & Central Park South | -73.9790689945 | 40.7667405591 | 7 Ave & Central Park South | -73.9790689945 | 40.7667405591 | 2021 | 13157082 | 2083.465083 | 6315 | 3 |
| 3662185 | Central Park North & Adam Clayton Powell Blvd | -73.9556130000 | 40.7994840000 | Central Park North & Adam Clayton Powell Blvd | -73.9556130000 | 40.7994840000 | 2021 | 8272909 | 1369.233532 | 6042 | 4 |
| 3662186 | S 5 Pl & S 5 St | -73.9608760000 | 40.7104510000 | S 5 Pl & S 5 St | -73.9608760000 | 40.7104510000 | 2021 | 2457618 | 461.784667 | 5322 | 5 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3662177 | St Ann's Ave & Westchester Ave | -73.9119552370 | 40.8158800600 | Melrose Ave & E 150 St | -73.9173380000 | 40.8168270000 | 2022 | 429 | 429.000000 | 1 | 3662178 |
| 3662178 | Steinway St & 21 Ave | -73.9036470650 | 40.7748835090 | Ditmars Blvd & 73 St | -73.8962930000 | 40.7681700000 | 2022 | 286 | 286.000000 | 1 | 3662179 |
| 3662179 | E 41 St & Madison Ave (SE corner) | -73.9798657890 | 40.7518806460 | West End Ave & W 60 St | -73.9900500000 | 40.7723700000 | 2022 | 682 | 682.000000 | 1 | 3662180 |
| 3662180 | Loring Pl North & W 183 St | -73.9093483690 | 40.8603944780 | Bergen Ave & E 152 St | -73.9147370000 | 40.8171660000 | 2022 | 853 | 853.000000 | 1 | 3662181 |
| 3662181 | W 44 St & 5 Ave | -73.9803220030 | 40.7549124960 | West End Ave & W 60 St | -73.9900500000 | 40.7723700000 | 2022 | 4713 | 4713.000000 | 1 | 3662182 |

4970568 rows × 11 columns

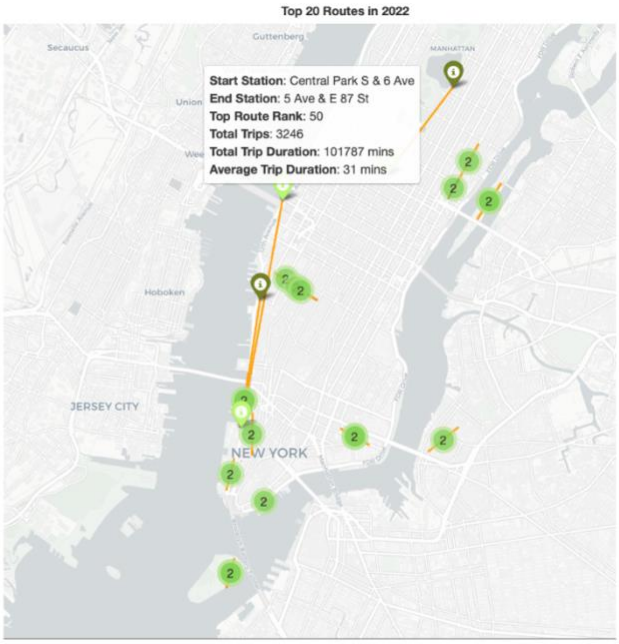
## 8. Routes from Start Station to the End Station

Finding out the top stations lead us to plotting them on the geospatial map. We decided to take it a step further and plot the path taken from a given start station to the end station.

We have used the window function to assign ranks to the routes in desc order from 1.

```
[ ] from pyspark.sql.functions import sum, count, mean
    from pyspark.sql.window import Window
    from pyspark.sql.functions import row_number, rank, dense_rank

    # Get data by aggregating on routes
    #Define window function
    windowSpec  = Window.partitionBy("startYear").orderBy(col("num_trips").desc())

    routes = df.groupBy('start_station_name', 'start_lng', 'start_lat', 'end_station_name', 'end_lng', 'end_lat', 'startYear') \
              .agg(sum('trip_duration').alias('total_trip_duration'),\
                   mean('trip_duration').alias('avg_trip_duration'), \
                   count('ride_id').alias('num_trips')) \
              .withColumn('routeRank', row_number().over(windowSpec)) \
              .toPandas() \
              .sort_values(['startYear', 'routeRank'])
```

We plot these routes on the geospatial map. We see the route connecting the two stations.

```
#get year and number of routes needed
year = int(input('Select Year for which you want to visualize (2021/2022): '))
topRank = int(input('Enter Number of top End Stations you want to visualize: '))

#Filter routes data
tmp = routes.loc[(routes.startYear == year) \
                 & (routes.start_station_name != routes.end_station_name) \
                 & (routes.start_lng != routes.end_lng) \
                 & (routes.start_lat != routes.end_lat) \
                 ][:topRank].reset_index(drop=True)

#create series for each column
latstart = tmp['start_lat']
longstart = tmp['start_lng']
namestart = tmp['start_station_name']
latsend = tmp['end_lat']
longsend = tmp['end_lng']
nameend = tmp['end_station_name']
numTrips = tmp['num_trips']
ranks = tmp['routeRank']
totalDur = tmp['total_trip_duration']
avgDur = tmp['avg_trip_duration']
#colors = ['red', 'blue', 'green', 'purple', 'orange', 'darkred',\
#          'lightred', 'beige', 'darkblue', 'darkgreen', 'cadetblue',\
#          'darkpurple', 'pink', 'lightblue', 'lightgreen', 'gray', 'black', 'lightgray']

colors = ['lightgreen', 'darkgreen']

colorsRoutes = pd.cut(tmp['num_trips'], bins=4, labels=['orange', 'darkorange', 'red', 'darkred'])
placestart = [[x[0],x[1]] for x in zip(latstart, longstart)]
placesend = [[x[0],x[1]] for x in zip(latsend, longsend)]
places = list(zip(placestart, placesend))

map = folium.Map(places[0][0], tiles='cartodbpositron', zoom_start=13)
map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Top {topRank} Routes in {year}</b></h3>'))


for i, pair in enumerate(places):
    marker_cluster = plugins.MarkerCluster().add_to(map)

    startTooltip = folium.Tooltip(f'''<b>Start Station</b>: {namestart[i]}''') #,  \
                                  #min_width=300, max_width=300)

    endTooltip = folium.Tooltip(f'''<b>End Station</b>: {nameend[i]}''') #, \
                                #min_width=300, max_width=300)

    routeTooltip = folium.Tooltip(f'''<b>Start Station</b>: {namestart[i]} \
                                      <br><b>End Station</b>: {nameend[i]} \
                                      <br><b>Top Route Rank</b>: {ranks[i]}
                                      <br><b>Total Trips</b>: {numTrips[i]}
                                      <br><b>Total Trip Duration</b>: {int(totalDur[i]/60.0)} mins
                                      <br><b>Average Trip Duration</b>: {int(float(avgDur[i])/60.0)} mins''') #,
                                  #min_width=300, max_width=300)


    folium.Marker(pair[0], icon=folium.Icon(color=colors[0]), tooltip = startTooltip).add_to(marker_cluster)
    folium.Marker(pair[1], icon=folium.Icon(color=colors[1]), tooltip = endTooltip).add_to(marker_cluster)
    folium.PolyLine(pair, color=colorsRoutes[i], tooltip = routeTooltip).add_to(map)

map
```

We plot the graph on the geospatial map with added information by dividing the markers into four groups based on the number of trips and colored the markers in the increasing order as orange, light red, red and dark red.

We plotted the top 20 routes on the geospatial map for the hour that had the highest traffic, i.e., 6pm.

We have colored the routes in the increasing order of the number of routes as orange, dark orange, red and dark red.

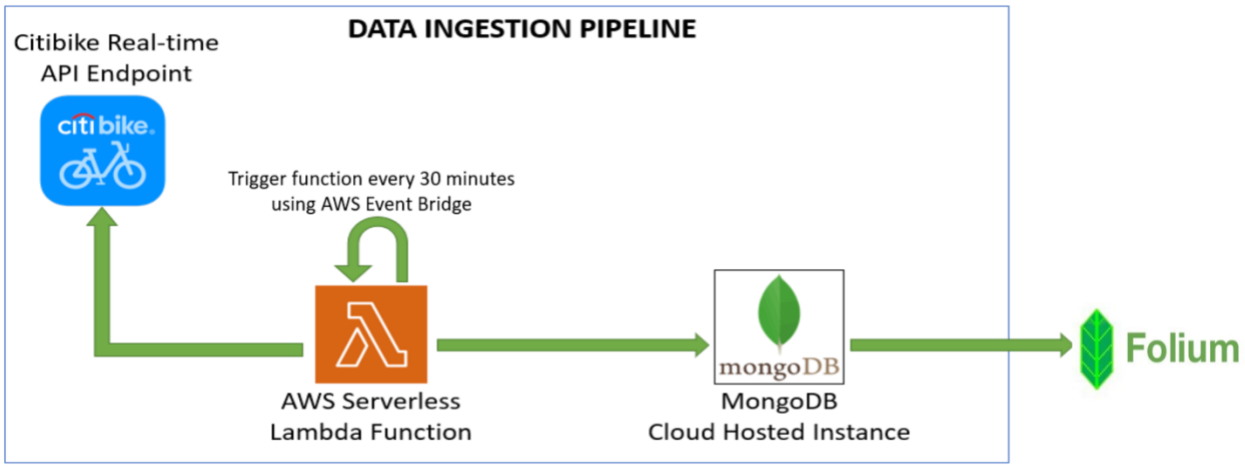
**Top 20 Routes 18:00 Hours in 2022**

```
#get hours
hours = list(range(24))

# Get data by aggregating on routes
#Define window function
windowSpec  = Window.partitionBy("startYear").orderBy(col("num_trips").desc())


#Enter hour and year
hour = input('Which hour of day do you want to explore (1-24)?: ')
year = input('Select year (2021/2022): ')
#take number of routes as input
topRank = int(input('How many top routes do you wish to visualize?: '))


routesHour = df.filter((col('start_station_name') != col('end_station_name')) \
                       & (col('startHour') == int(hour)) \
                       & (col('startYear') == int(year))) \
               .groupBy('start_station_name', 'start_lng', 'start_lat', 'end_station_name', 'end_lng', 'end_lat', 'startYear', 'startHour') \
               .agg(sum('trip_duration').alias('total_trip_duration'), \
                    mean('trip_duration').alias('avg_trip_duration'), \
                    count('ride_id').alias('num_trips')) \
               .withColumn('routeRank', row_number().over(windowSpec)) \
               .toPandas() \
               .sort_values(['startYear', 'routeRank'])


#filter based on number of routes
tmp = routesHour[:topRank].reset_index(drop = True)

#create series for each column
latstart = tmp['start_lat']
longstart = tmp['start_lng']
namestart = tmp['start_station_name']
latsend = tmp['end_lat']
longsend = tmp['end_lng']
nameend = tmp['end_station_name']
numTrips = tmp['num_trips']
ranks = tmp['routeRank']
totalDur = tmp['total_trip_duration']
avgDur = tmp['avg_trip_duration']
#colors = ['red', 'blue', 'green', 'purple', 'orange', 'darkred',\
#          'lightred', 'beige', 'darkblue', 'darkgreen', 'cadetblue',\
#          'darkpurple', 'pink', 'lightblue', 'lightgreen', 'gray', 'black', 'lightgray']

colors = ['lightgreen', 'darkgreen']

colorsRoutes = pd.cut(tmp['num_trips'], bins=4, labels=['orange', 'darkorange', 'red', 'darkred'])
placestart = [[x[0],x[1]] for x in zip(latstart, longstart)]
placesend = [[x[0],x[1]] for x in zip(latsend, longsend)]
places = list(zip(placestart, placesend))

map = folium.Map(places[0][0], tiles='cartodbpositron', zoom_start=13)
map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Top {topRank} Routes {hour}:00 Hours in {year}</b></h3>'))


for i, pair in enumerate(places):
    marker_cluster = plugins.MarkerCluster().add_to(map)

    startTooltip = folium.Tooltip(f'''<b>Start Station</b>: {namestart[i]}''') #,  \
                                  #min_width=300, max_width=300)


    endTooltip = folium.Tooltip(f'''<b>End Station</b>: {nameend[i]}''') #, \
                                #min_width=300, max_width=300)

    routeTooltip = folium.Tooltip(f'''<b>Start Station</b>: {namestart[i]} \
                                      <br><b>End Station</b>: {nameend[i]} \
                                      <br><b>Top Route Rank</b>: {ranks[i]}
                                      <br><b>Total Trips</b>: {numTrips[i]}
                                      <br><b>Total Trip Duration</b>: {int(totalDur[i]/60.0)} mins
                                      <br><b>Average Trip Duration</b>: {int(float(avgDur[i])/60.0)} mins''') #,
                                  #min_width=300, max_width=300)


    folium.Marker(pair[0], icon=folium.Icon(color=colors[0]), tooltip = startTooltip).add_to(marker_cluster)
    folium.Marker(pair[1], icon=folium.Icon(color=colors[1]), tooltip = endTooltip).add_to(marker_cluster)
    folium.PolyLine(pair, color=colorsRoutes[i], tooltip = routeTooltip).add_to(map)
```

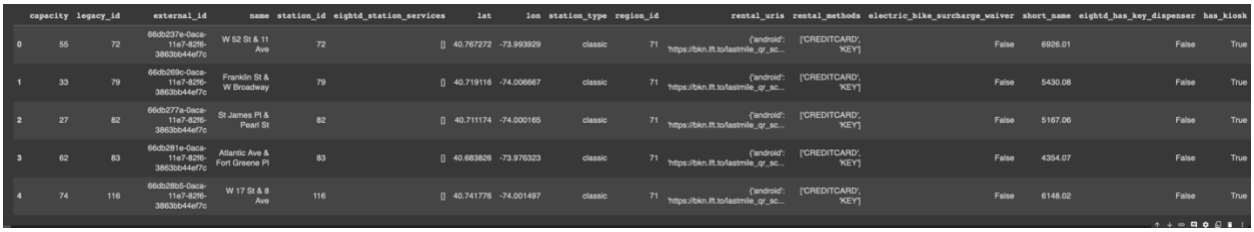
## 9. Station Data Analysis

We will describe our analysis of the Citi Bike station information based on the data collected in the last 3 days.

From the analysis we have done so far, we have found the top start stations, top end stations and the top overall stations. Keeping these in mind, we decided to dive into the live data to obtain the live information.




We setup our AWS Serverless Lambda Function to trigger an API request to fetch the Citi bike real-time data every 30 seconds. We fetch this data from Mongo DB into our notebook.



| | capacity | legacy_id | external_id | name | station_id | eightd_station_services | lat | lon | station_type | region_id | rental_uris | rental_methods | electric_bike_surcharge_waiver | short_name | eightd_has_key_dispenser | has_kiosk |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 55 | 72 | 66db237e-0aca-11e7-82f6-3863bb44ef7c | W 52 St & 11 Ave | 72 | [] | 40.767272 | -73.993929 | classic | 71 | {'android': 'https://bkn.lft.to/lastmile_qr_sc... | ['CREDITCARD', 'KEY'] | False | 6926.01 | False | True |
| 1 | 33 | 79 | 66db269c-0aca-11e7-82f6-3863bb44ef7c | Franklin St & W Broadway | 79 | [] | 40.719116 | -74.006667 | classic | 71 | {'android': 'https://bkn.lft.to/lastmile_qr_sc... | ['CREDITCARD', 'KEY'] | False | 5430.08 | False | True |
| 2 | 27 | 82 | 66db277a-0aca-11e7-82f6-3863bb44ef7c | St James Pl & Pearl St | 82 | [] | 40.711174 | -74.000165 | classic | 71 | {'android': 'https://bkn.lft.to/lastmile_qr_sc... | ['CREDITCARD', 'KEY'] | False | 5167.06 | False | True |
| 3 | 62 | 83 | 66db281e-0aca-11e7-82f6-3863bb44ef7c | Atlantic Ave & Fort Greene Pl | 83 | [] | 40.683826 | -73.976323 | classic | 71 | {'android': 'https://bkn.lft.to/lastmile_qr_sc... | ['CREDITCARD', 'KEY'] | False | 4354.07 | False | True |
| 4 | 74 | 116 | 66db28b5-0aca-11e7-82f6-3863bb44ef7c | W 17 St & 8 Ave | 116 | [] | 40.741776 | -74.001497 | classic | 71 | {'android': 'https://bkn.lft.to/lastmile_qr_sc... | ['CREDITCARD', 'KEY'] | False | 6148.02 | False | True |

We analyze the data in real-time. The data has 16 columns with information about the rides and the stations.

```
[ ] # station info
    station_info_df1 = pd.read_csv('s3://citibike-realtime/station-information/data1.csv')
    station_info_df1.head()
```

We connect to our database and fetch the columns needed for our analysis.

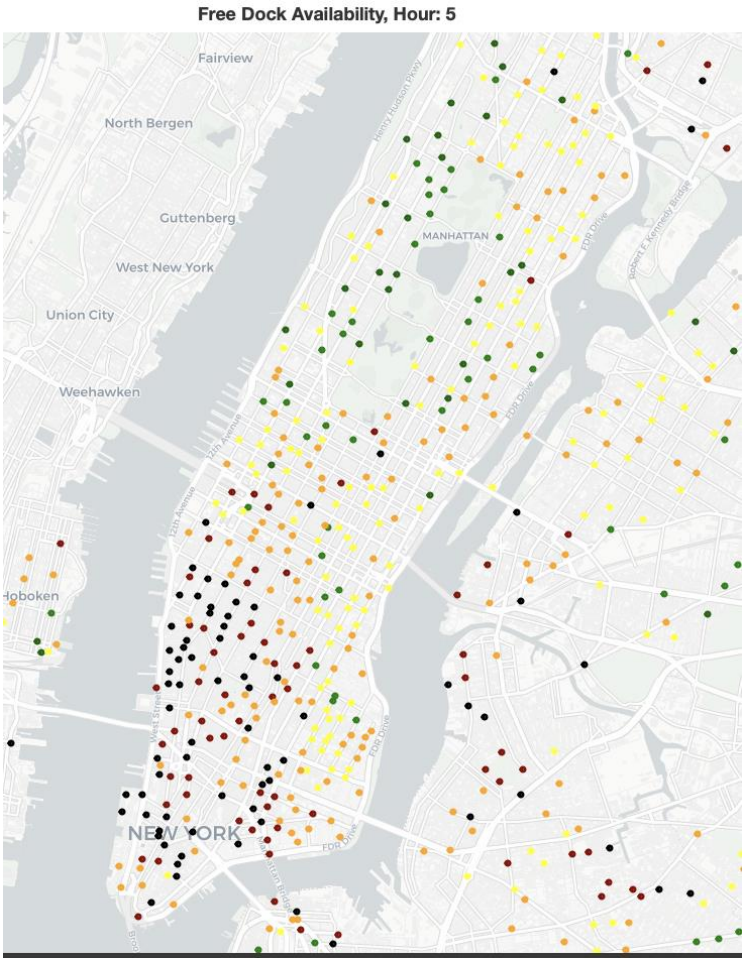
```
+----------+------------------+------------------+------------------------------+
|station_id|lat               |lon               |station_name                  |
+----------+------------------+------------------+------------------------------+
|72        |40.76727216       |-73.99392888      |W 52 St & 11 Ave              |
|79        |40.71911552       |-74.00666661      |Franklin St & W Broadway      |
|82        |40.71117416       |-74.00016545      |St James Pl & Pearl St        |
|83        |40.68382604       |-73.97632328      |Atlantic Ave & Fort Greene Pl|
|116       |40.74177603       |-74.00149746      |W 17 St & 8 Ave               |
|119       |40.69608941       |-73.97803415      |Park Ave & St Edwards St      |
|120       |40.68676793       |-73.95928168      |Lexington Ave & Classon Ave   |
|127       |40.73172428       |-74.00674436      |Barrow St & Hudson St         |
|128       |40.72710258       |-74.00297088      |MacDougal St & Prince St      |
|143       |40.69239502       |-73.99337909      |Clinton St & Joralemon St     |
|144       |40.69839895       |-73.98068914      |Nassau St & Navy St           |
|146       |40.71625008       |-74.0091059       |Hudson St & Reade St          |
|150       |40.7208736        |-73.98085795      |E 2 St & Avenue C             |
|151       |40.722103786686034|-73.99724900722504|Cleveland Pl & Spring St      |
|152       |40.71473993       |-74.00910627      |Warren St & W Broadway        |
|153       |40.752062307      |-73.9816324043    |E 40 St & 5 Ave               |
|157       |40.69089272       |-73.99612349      |Henry St & Atlantic Ave       |
|161       |40.72917025       |-73.99810231      |LaGuardia Pl & W 3 St         |
|164       |40.75323098       |-73.97032517      |E 47 St & 2 Ave               |
|168       |40.73971301       |-73.99456405      |W 18 St & 6 Ave               |
+----------+------------------+------------------+------------------------------+
only showing top 20 rows
```

```
[ ] client = pymongo.MongoClient("mongodb+srv://chetan:IWsL1aILKbTxIBsq@cluster0.wjaj68m.mongodb.net/?retryWrites=true&w=majority")
    db = client.station_data

[ ] collection = db["information_data"]
    station_info_items = []
    for record in collection.find({},{'station_id':1, 'lat':1, 'lon':1, 'station_name':1}):
        station_info_items.append(record)

    # print(station_info_items[0])

    station_information_schema = StructType([
        StructField('station_id', StringType(), True),
        StructField('lat', StringType(), True),
        StructField('lon', StringType(), True),
        StructField('station_name', StringType(), True),
    ])

    station_information_data = spark.createDataFrame(data=station_info_items, schema = station_information_schema)
    # station_information_data.printSchema()
    station_information_data.show(truncate=False)
```

## 10. Availability of dock stations on an hourly basis

We now dive deep into the real-time data to analyze the availability of the free docks. We let the user dynamically choose the hour to get the available docks. We have plotted the graph below for 5am.

We have the markers divided into intervals of 10 dock stations in the increasing order as white, black, dark red, orange, yellow, green, and dark green.

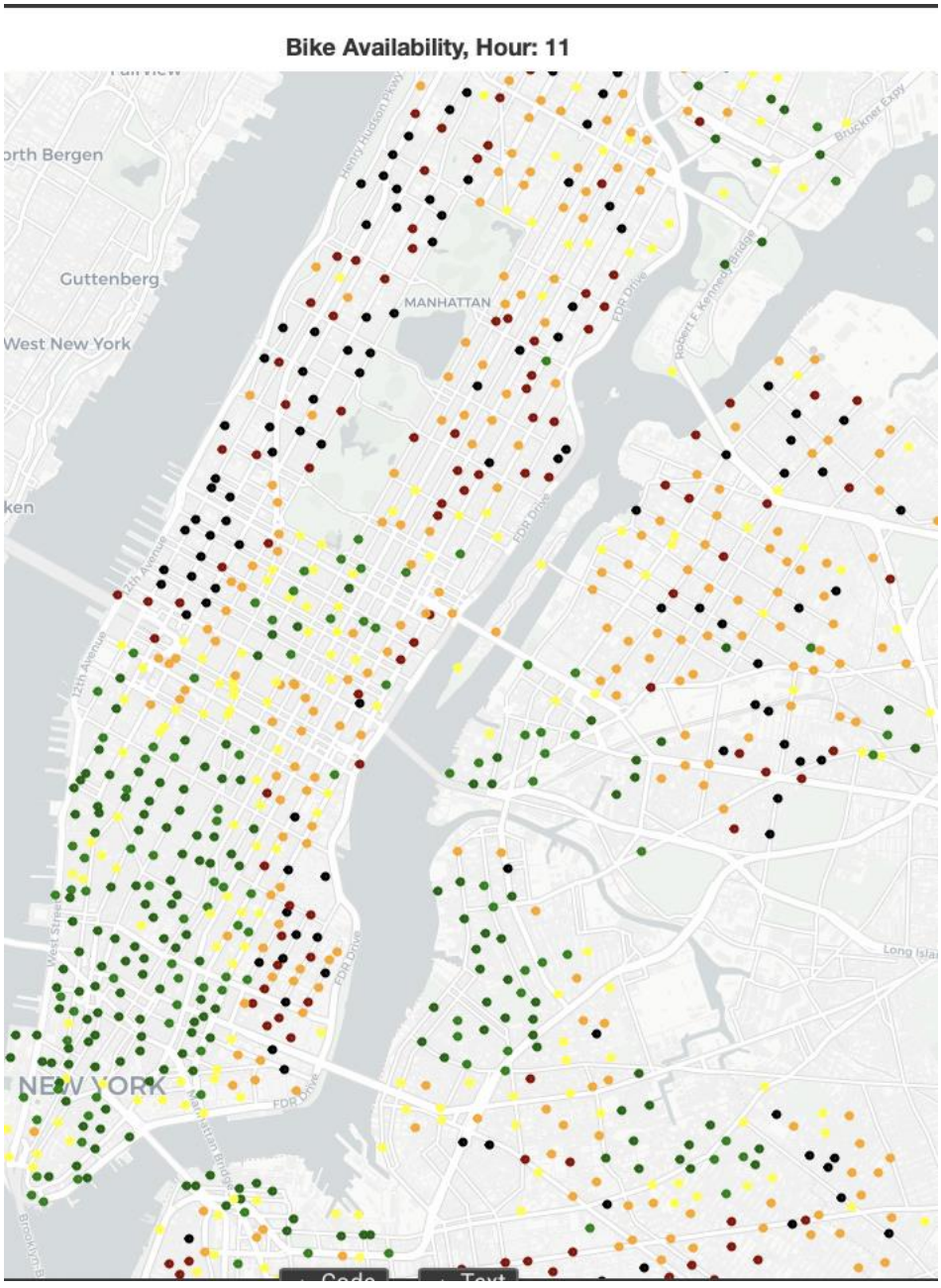


```
curr_items = []
station_dock_availability_map = folium.Map(location=[40.7075159327968, -73.98505617283716 ],tiles='cartodbpositron',
                                zoom_start=12, min_zoom = 10)
# print("Time selected: %d : 00" % (hour))

#Getting the station status data for a particular hour
# for record in collection.find({'reported_hour':hour},{'station_id':1, 'station_status':1, 'num_bikes_available':1, 'num_docks_available':1, 'num_ebikes_available':1, 'num_docs_disabled':1}):
#   curr_items.append(record)

# print(curr_items[0]) # Return Value: 6

##############################  Creating pyspark dataframe from array   ########################################

station_status_data = spark.createDataFrame(data=variables['result%s' % hour], schema = station_status_schema)
# station_status_data.printSchema()
# station_status_data.show(truncate=False)

##############################  Computing necessary columns in the pyspark dataframe   ########################################

station_status_data = station_status_data\
                    .withColumn("total_bikes_available", col("num_bikes_available")+col("num_ebikes_available"))\
                    .withColumn("capacity", col("num_bikes_available")+col("num_docks_available")+col("num_ebikes_available"))\
                    .withColumn("availability_of_docks(%)", format_number( 100*col("num_docks_available")/ col("capacity"),2))\
                    # .withColumn("occupancy_of_docks(%)", 100 - col("availability_of_docks(%)")) # X % of the station is full

# availability_of_docks(%) shows how much % of the station is available to park bikes
# occupancy_of_docks(%) shows how much the dock is occupied. Or it shows how much % of bikes are available
# station_status_data.show(10)

##############################  Finding average of the availability col   ########################################

station_status_average_avail_data = station_status_data.groupBy("station_id").agg({"availability_of_docks(%)":"avg"})
# station_status_average_avail_data.show(5)

##############################  Joining the pyspark dataframe with location dataframe   ########################################

final_availability_dataframe = station_status_average_avail_data.join(station_information_data, on='station_id')
final_availability_df = final_availability_dataframe.toPandas()
# final_availability_dataframe.show(5)

##############################  Plotting the map for the given data   ########################################

for each in final_availability_df[:].iterrows():
    # date_time = datetime.datetime.fromtimestamp( stationData['last_reported'] )
      name = each[1]['station_name']
      availability = each[1]['avg(availability_of_docks(%))']
      reportedAt = hour

      markerColor = 'white'
      if availability <= 10:
        markerColor = 'black'
      elif availability <= 20:
        markerColor = 'darkred'
      elif availability <= 50:
        markerColor = 'orange'
      elif availability <= 80:
        markerColor = 'yellow'
      elif availability <= 90:
        markerColor = 'green'
      elif availability <= 100 :
        markerColor = 'darkgreen'

      folium.CircleMarker(
          location = [each[1]['lat'],each[1]['lon']],
          radius=2,
          color=markerColor,
          popup=str(each[1]['lat'])+','+str(each[1]['lon']),
          fill_color="#0A3260",
          tooltip = (
              "<strong>Name:</strong> "+ name +"<br>"
              "<strong>Dock Availability(%):</strong> "+str(availability)+"<br>"
              "<strong>Reported Hour:</strong> "+str(reportedAt)+"<br>"
              )
          ).add_to(station_dock_availability_map)


station_dock_availability_map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Free Dock Availability, Hour: '+str(hour) +'</b></h3>'))

station_dock_availability_map.save('DockAvailabilitv/dock availability '+str(hour)+'.html')
```

## 11. Availability of bikes on an hourly basis

We now analyze the availability of the bikes based on the hour. We let the user dynamically choose the hour to get the available bikes. We have plotted the graph below for 11am.

We have the markers divided into intervals of 10 stations in the increasing order of the available bikes as white, black, dark red, orange, yellow, green, and dark green.



```
station_bike_availability_map = folium.Map(location=[40.7075159327968, -73.98505617283716 ],tiles='cartodbpositron',
                                           zoom_start=12, min_zoom = 10)
# print("Time selected: %d : 00" % (hour))

#Getting the station status data for a particular hour
# for record in collection.find({'reported_hour':hour},{'station_id':1, 'station_status':1, 'num_bikes_available':1, 'num_docks_available':1, 'num_ebikes_available':1, 'num_docs_disabled':1}):
#   curr_items.append(record)

# print(curr_items[0]) # Return Value: 6

##############################  Creating pyspark dataframe from array   ########################################

station_status_data = spark.createDataFrame(data=variables['result%s' % hour], schema = station_status_schema)
# station_status_data.printSchema()
# station_status_data.show(truncate=False)

##############################  Computing necessary columns in the pyspark dataframe   ########################################

station_status_data = station_status_data\
                      .withColumn("total_bikes_available", col("num_bikes_available")+col("num_ebikes_available"))\
                      .withColumn("capacity", col("num_bikes_available")+col("num_docks_available")+col("num_ebikes_available"))\
                      .withColumn("availability_of_docks(%)", format_number( 100*col("num_docks_available")/ col("capacity"),2))\
                      .withColumn("availability_of_bikes(%)", 100 - col("availability_of_docks(%)")) # X % of the station is full

# availability_of_docks(%) shows how much % of the station is available to park bikes
# availability_of_bikes(%) shows how much bikes are available at the station.
# station_status_data.show(10)

##############################  Finding average of the availability col   ########################################

station_status_average_avail_data = station_status_data.groupBy("station_id").agg({"availability_of_bikes(%)":"avg"})
# station_status_average_avail_data.show(5)

##############################  Joining the pyspark dataframe with location dataframe   ########################################

final_availability_dataframe = station_status_average_avail_data.join(station_information_data, on='station_id')
final_availability_df = final_availability_dataframe.toPandas()
# final_availability_dataframe.show(5)

##############################  Plotting the map for the given data   ########################################

for each in final_availability_df[:].iterrows():
    # date_time = datetime.datetime.fromtimestamp( stationData['last_reported'] )
      name = each[1]['station_name']
      availability = each[1]["avg(availability_of_bikes(%))"]
      reportedAt = hour

      markerColor = 'white'
      if availability <= 10:
        markerColor = 'black'
      elif availability <= 20:
        markerColor = 'darkred'
      elif availability <= 50:
        markerColor = 'orange'
      elif availability <= 80:
        markerColor = 'yellow'
      elif availability <= 90:
        markerColor = 'green'
      elif availability <= 100 :
        markerColor = 'darkgreen'

      folium.CircleMarker(
          location = [each[1]['lat'],each[1]['lon']],
          radius=2,
          color=markerColor,
          popup=str(each[1]['lat'])+','+str(each[1]['lon']),
          fill_color='#0A3260',
          tooltip = (
              "<strong>Name:</strong> "+ name +"<br>"
              "<strong>Dock Availability(%):</strong> "+str(availability)+"<br>"
              "<strong>Reported Hour:</strong> "+str(reportedAt)+"<br>"
              )
          ).add_to(station_bike_availability_map)


station_bike_availability_map.get_root().html.add_child(folium.Element(f'<h3 align="center" style="font-size:16px"><b>Bike Availability, Hour: '+str(hour) +'</b></h3>'))
station_bike_availability_map.save('BikeAvailability/bike_availability_'+str(hour)+'.html')

display(station_bike_availability_map)
```

### 12. Top stations with low bike availability

We assign the ranks of the stations based on the low availability of bikes and plot it on a geospatial map.

On hover over the markers, we can see more information such as the station name and the % of bikes available. On click we get the exact latitude and longitude of the dock station.


Edgewater
Cliffside Park
Fairview
North Bergen
Guttenberg
West New York
Union City
Weehawken
MANHATTAN
NEW YORK
Henry Hudson Pkwy
FDR Drive
12th Avenue
West Street
Manhattan Br

```
startStnsDF = low_availability_start_stations.toPandas()
station_availability_map = folium.Map(location=[40.7075159327968, -73.98505617283716 ],tiles='cartodbpositron',
                      zoom_start=12,
                      min_zoom = 12,
                     )

    ##############################  Plotting the map for the given data   ##########################################

for each in startStnsDF[:].iterrows():
    # date_time = datetime.datetime.fromtimestamp( stationData['last_reported'] )
      name = each[1]['station_name']
      availability = each[1]['avg(availability_of_docks(%))']
      # reportedAt = hour

      # markerColor = 'darkred'

      folium.Marker(
          location = [each[1]['lat'],each[1]['lon']],
          # radius=2,
          icon=folium.Icon(color='red'),
          popup=str(each[1]['lat'])+','+str(each[1]['lon']),
          fill_color='#0A3260',
          tooltip = (
              "<strong>Name:</strong> "+ name +"<br>"
              "<strong>Dock Availability(%):</strong> "+str(availability)+"<br>"
              )
          ).add_to(station_availability_map)

display(station_availability_map)
```

**13. Top stations with low free dock stations**

We assign the ranks to the stations based on the availability of free docks.

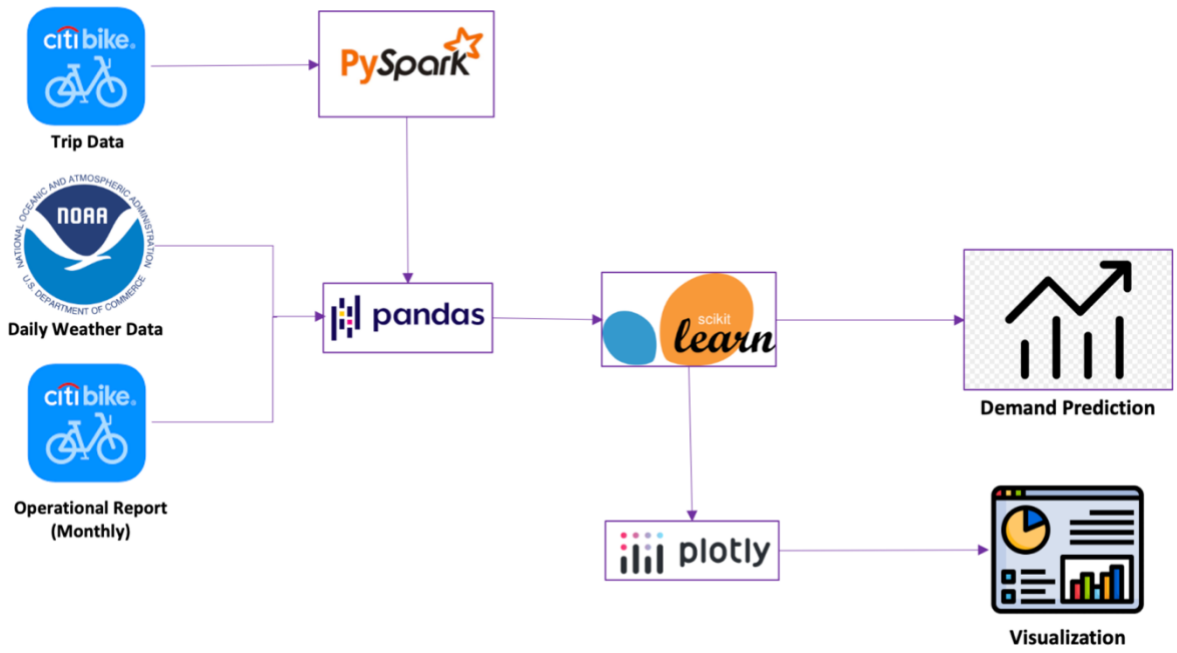
We assign the ranks of the stations based on the free availability of dock stations and plot it on a geospatial map. We have the top 50 stations with low free dock stations available.


40.74238787,-73.99726235
HELL'S KITCHEN
MIDTOWN EAST
THEATER DISTRICT
HUDSON YARDS
TURTLE BAY
TUDOR CITY
HUNTERS POINT
KIPS BAY
MEATPACKING DISTRICT
WEST VILLAGE
GRAMERCY
EAST VILLAGE
HUDSON SQUARE
ALPHABET CITY
TRIBECA
CHINATOWN
LOWER EAST SIDE
CIVIC CENTER
NEW YORK
BATTERY PARK CITY
TWO BRIDGES
DUMBO
LINCOLN HARBOR
Hudson River
East River
West Street
FDR Drive
Holland Tunnel
Lincoln Tunnel
Williamsburg Bridge
Manhattan Bridge
Brooklyn Queens Expy
Pulitzer Fountain
Wading Pool
Bosque Fountain
Dry Dock 6

```
endStnsDF = high_occupancy_end_stations.toPandas()
station_availability_map = folium.Map(location=[40.7075159327968, -73.98505617283716 ],tiles='cartodbpositron',
                      zoom_start=12,
                     min_zoom = 12,
                    )

    ############################  Plotting the map for the given data   ########################################

for each in endStnsDF[:].iterrows():
    # date_time = datetime.datetime.fromtimestamp( stationData['last_reported'] )
      name = each[1]['station_name']
      availability = each[1]['avg(availability_of_docks(%))']
      # reportedAt = hour

      # markerColor = 'darkred'

      folium.Marker(
          location = [each[1]['lat'],each[1]['lon']],
          # radius=2,
          icon=folium.Icon(color='red'),
          popup=str(each[1]['lat'])+','+str(each[1]['lon']),
          fill_color='#0A3260',
          tooltip = (
              "<strong>Name:</strong> "+ name +"<br>"
              "<strong>Dock Availability(%):</strong> "+str(availability)+"<br>"
              )
          ).add_to(station_availability_map)

display(station_availability_map)
```

## 14. Demand Forecasting:

### Data Engineering Pipeline:




Here we considered the date aggregated Citi bike Trips dataset from 2016 to the Present (~30 GBs). Also for feature generated we consider the NYC Daily Weather dataset and the Citi bike Monthly Operating Reports. Trips data is ingested and aggregated through PySpark and then collected onto Pandas. The weathers and monthly datasets aren't big data and are hence ingested into pandas directly. Then a combination of these datasets are used by Sci-Kit Learn to build and train a demand forecasting model, and Plotly is used to plot our time-series data and

results.

**Preprocessing:** For the machine learning aspect of our analysis, we decided to include all the data starting from 2013 to 2022.

Since the data is slightly different from before 2020, we load the data from after 2021 and before 2021 separately. Also, we focus on data from 2016 and later, so we filter that out as well.

```
#get list of files
path = '/content/data/2021_22/'
dir_list = os.listdir(path)
dir_list.sort()
if dir_list[-1] == '__MACOSX':
  dir_list = dir_list[:-1]

filePaths =[path + fileName for fileName in dir_list]
filePaths
```

```
#read data into spark

start = time()

df = spark.read\
     .option('inferSchema', 'true')\
     .option('header', 'true')\
     .csv(filePaths)

load_end = time()

print(f'''2021-22 Data loaded to spark in {(load_end - start)/60.0} mins''')

#remove nulls
df = df.na.drop("any")

from pyspark.sql.types import IntegerType,BooleanType,DateType, TimestampType, DecimalType, StringType
from pyspark.sql.functions import udf, col, dayofweek, dayofmonth, dayofyear, hour, month, quarter, year, to_date, weekofyear
from pyspark.sql.functions import sum, count, mean

#define udf to reverse geolocate buroughs, counties, neighbourhoods
manhattan_neighbourhood_udf = udf(lambda lat, lng, b: geolocator.reverse([lat, lng])[0].split(', ')[-7] if b=='Manhattan'  else '', StringType())
burough_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-6], StringType())
county_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-5], StringType())
city_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-4], StringType())

#Get informations from the start and end dates
df = df.withColumn("startDate", to_date(col('started_at'))) \
       .withColumn('startDOW', dayofweek(col('started_at'))) \
       .withColumn('startDOM', dayofmonth(col('started_at'))) \
       .withColumn('startDOY', dayofyear(col('started_at'))) \
       .withColumn('startHour', hour(col('started_at'))) \
       .withColumn('startMonth', month(col('started_at'))) \
       .withColumn('startQuarter', quarter(col('started_at'))) \
       .withColumn('startWOY', weekofyear(col('started_at'))) \
       .withColumn('startYear', year(col('started_at'))) \
       .groupby(col('startDate').alias('date'), \
                col('startYear').alias('year'), \
                col('startMonth').alias('month'), \
                col('startWOY').alias('week'), \
                col('startDOM').alias('Day'), \
                col('startDOW').alias('DOW'), \
                col('startDOY').alias('DOY')) \
       .agg(count(col('ride_id'))) \
       .withColumnRenamed('count(ride_id)', 'count') \
       .toPandas()

df.to_csv('/content/data/aggRidership.csv')

save_end = time()

print(f'''2021-22 Aggregated Data Processed and Saved in {(save_end - load_end)/60.0} mins''')
```

```
2021-22 Data loaded to spark in 2.177092456817627 mins
2021-22 Aggregated Data Processed and Saved in 1.5878947814305624 mins
```

```
#get list of files
path = '/content/data/2013_20/'
# dir_list = os.listdir(path)
# dir_list.sort()
# if dir_list[-1] == '__MACOSX':
#   dir_list = dir_list[:-1]

filePaths = glob.glob(path + '*/*.csv', recursive=False)

#filePaths =[path + fileName for fileName in dir_list]
filePaths
```

```
#read data into spark

start = time()

df = spark.read\
     .option('inferSchema', 'true')\
     .option('header', 'true')\
     .csv(filePaths)

load_end = time()

print(f'''2013-20 Data loaded to spark in {(load_end - start)/60.0} mins''')

from pyspark.sql.types import IntegerType,BooleanType,DateType, TimestampType, DecimalType, StringType
from pyspark.sql.functions import udf, col, dayofweek, dayofmonth, dayofyear, hour, month, quarter, year, to_date, weekofyear
from pyspark.sql.functions import sum, count, mean

#define udf to reverse geolocate buroughs, counties, neighbourhoods
manhattan_neighbourhood_udf = udf(lambda lat, lng, b: geolocator.reverse([lat, lng])[0].split(', ')[-7] if b=='Manhattan'  else '', StringType())
burough_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-6], StringType())
county_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-5], StringType())
city_udf = udf(lambda lat, lng: geolocator.reverse([lat, lng])[0].split(', ')[-4], StringType())

#Get informations from the start and end dates
df = df.withColumn("startDate", to_date(col('starttime'))) \
       .withColumn('startDOW', dayofweek(col('starttime'))) \
       .withColumn('startDOM', dayofmonth(col('starttime'))) \
       .withColumn('startDOY', dayofyear(col('starttime'))) \
       .withColumn('startHour', hour(col('starttime'))) \
       .withColumn('startMonth', month(col('starttime'))) \
       .withColumn('startQuarter', quarter(col('starttime'))) \
       .withColumn('startWOY', weekofyear(col('starttime'))) \
       .withColumn('startYear', year(col('starttime'))) \
       .groupby(col('startDate').alias('date'), \
                col('startYear').alias('year'), \
                col('startMonth').alias('month'), \
                col('startWOY').alias('week'), \
                col('startDOM').alias('day'), \
                col('startDOW').alias('DOW'), \
                col('startDOY').alias('DOY')) \
       .agg(count('bikeid')) \
       .withColumnRenamed('count(bikeid)', 'count') \
       .toPandas() \

df.to_csv('/content/data/aggRidership.csv', mode = 'a', header = False)

save_end = time()

print(f'''2013-20 Aggregated Data Processed and Saved in {(save_end - load_end)/60.0} mins''')
```

```
2013-20 Data loaded to spark in 4.836398681004842 mins
Index(['date', 'year', 'month', 'week', 'day', 'DOW', 'DOY', 'count'], dtype='object')
2013-20 Aggregated Data Processed and Saved in 2.534510691960653 mins
```

We already saw from the previous analysis that our ridership data is affected by various factors, such as the time of the day and weather.

Therefore, we decided to include the weather data into our analysis.

```
#read the weather and bike station data
bikeStation_df = pd.read_csv('/content/drive/MyDrive/BigDataProject/bikes_stations_data_21_22.csv')
weatherNYC_df = pd.read_csv('/content/drive/MyDrive/BigDataProject/NYC_weather_2013-22.csv')

#convert date to str
weatherNYC_df['date'] = weatherNYC_df['DATE'].astype(str)

#create new columns for bikestations
bikeStation_df['month'] = bikeStation_df['Month'].apply(lambda x: int(x.split('-')[1]))
bikeStation_df['year'] = bikeStation_df['Month'].apply(lambda x: int(x.split('-')[0]))
```

weatherNYC_df

| | STATION | NAME | LATITUDE | LONGITUDE | ELEVATION | DATE | AWND | AWND_ATTRIBUTES | PGTM | PGTM_ATTRIBUTES | ... | WT13_ATTRIBUTES | WT16 | WT16_ATTRIBUTES | WT18 | WT18_ATTRIBUTES | WT19 | WT19_ATTRIBUTES | WT22 | WT22_ATTRIBUTES | date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2013-01-01 | 6.93 | ,,X | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013-01-01 |
| 1 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2013-01-02 | 5.82 | ,,X | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013-01-02 |
| 2 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2013-01-03 | 4.47 | ,,X | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013-01-03 |
| 3 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2013-01-04 | 8.05 | ,,X | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013-01-04 |
| 4 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2013-01-05 | 6.71 | ,,X | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2013-01-05 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3629 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2022-12-09 | 6.71 | ,,W | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2022-12-09 |
| 3630 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2022-12-10 | 7.16 | ,,W | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2022-12-10 |
| 3631 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2022-12-11 | 8.05 | ,,W | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2022-12-11 |
| 3632 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2022-12-12 | 5.82 | ,,W | 1319.0 | ,,W | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2022-12-12 |
| 3633 | USW00094728 | NY CITY CENTRAL PARK, NY US | 40.77898 | -73.96925 | 42.7 | 2022-12-13 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 2022-12-13 |

3634 rows × 57 columns

We see that the weather data has various attributes such as location, peak gust time, average daily wind speed, etc.

```
def add_calendar_features(df):
    df_new = df.copy()
    days_in_year = np.where(df["year"].apply(lambda y: calendar.isleap(y)), 366, 365)
    df_new['sin_doy'] = np.sin(2*np.pi*df_new["DOY"]/days_in_year)
    df_new['cos_doy'] = np.cos(2*np.pi*df_new["DOY"]/days_in_year)
    df_new['sin_dow'] = np.sin(2*np.pi*df_new["DOW"]/7)
    df_new['cos_dow'] = np.cos(2*np.pi*df_new["DOW"]/7)
    df_new = pd.concat([df_new, pd.get_dummies(df_new["DOW"], "dow")], axis=1)
    df_new = pd.concat([df_new, pd.get_dummies(df_new["month"], "month")], axis=1)
    df_new = pd.concat([df_new, pd.get_dummies(df_new["Day"], "day")], axis=1)
    df_new["holiday"] = [1 if d in us_holidays else 0 for d in df_new["date"]]
    df_new['date'] = df_new['date'].astype(str)
    return df_new

df_nyc = add_calendar_features(dailyRidership_df)

df_nyc
```

**Calendar Features:** We now include the calendar feature for our analysis. Our feature includes the day of the month and holidays. As seen in the previous trip analysis, ridership has seasonality and a cyclical nature with respect to Day-of-Year and Day-of-Week. Hence, we have Sin and Cos transformed these attributes to generate features from them. Rest were either featurized by categorical dummies or boolean variables.

We build a regression model for the number of trips. We consider Linear Regression, Polynomial Regression, and Polynomial Regression with Ridge

Regularization, and chose the best of the lot. The implementation of our models and metrics is given below:

```python
def mape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def fit_predict(
    df,
    features,
    model="linear",
    scale=True,
    log=False,
    train_year_to=2022,
    #train_month_to=8,
    test_year=2022,
    #test_month_from = 9,
    degree=2,
    alphas=np.logspace(-2,2)
):
    df_train = df.loc[(df["year"] < train_year_to)]
    df_test = df.loc[(df["year"] >= test_year)]

    print(df_train.shape)
    print(df_test.shape)

    X = df_train[features]
    y = df_train["count"]

    if scale:
        scaler = StandardScaler()
        scaler.fit(X)
        X = scaler.transform(X)
    if log: y = np.log(y)

    if model == "linear":
        regr = LinearRegression()
    elif model == "polynomial":
        regr = make_pipeline(
            PolynomialFeatures(degree=degree),
            LinearRegression()
            )
    elif model == "ridge":
        regr = make_pipeline(
            PolynomialFeatures(degree=degree),
            RidgeCV(alphas=alphas, cv=TimeSeriesSplit())
            )
    else:
        assert False, f"Unknown model {model}"

    regr.fit(X, y)

    if model == "ridge":
        print(f"Best alpha: {regr['ridgecv'].alpha_}")

    X_test = df_test[features]
    if scale: X_test = scaler.transform(X_test)
    y_train = regr.predict(X)
    y_test = regr.predict(X_test)
    if log: y_test, y_train = np.exp(y_test), np.exp(y_train)

    # prevent model from predicting negative values
    y_test, y_train = np.clip(y_test, 0, np.max(y_test)), np.clip(y_train, 0, np.max(y_train))

    print(f"Training set R2 score: {r2_score(df_train['count'], y_train)}")
    print(f"Test set R2 core: {r2_score(df_test['count'], y_test)}")
    print(f"Test set MAPE: {mape(df_test['count'], y_test)}")

    return df_train, df_test, y_test, y_train, regr
```

We first train the best models using the Calendar features defined above. We see that the training set results in 0.69 whereas the test set score is at 0.47.

```
(2391, 64)
(307, 64)
Training set R2 score: 0.6935678655437624
Test set R2 core: 0.47580525656913064
Test set MAPE: 21556.443439829978
```

To improve on this we add new features to our data set.

**Weather Features:** We add the following features from the NYC weathers data set for each day to our aggregated trips dataset:

1. SNOW_log – Log of Difference (in mm) b/w Last Snowfall and current snowfall.
2. SNWD_log – Log of Number of total inches of snow on the ground at a given day
3. PREC_log – Precipitation (in mm)
4. TMAX – Max Temperature of a given day
5. TMIN – Min Temperature of a minimum day

```
#Join Weather details
df_nyc = pd.merge(left=df_nyc, right=weatherNYC_df, left_on='date', right_on='date', how= 'inner')
df_nyc["PRCP_log"] = np.log(df_nyc["PRCP"] + 1)
df_nyc["SNOW_log"] = np.log(df_nyc["SNOW"] + 1)
df_nyc["SNWD_log"] = np.log(df_nyc["SNWD"] + 1)
```

Now we trained the best possible model with the above calendar features and the newly computed Weather features. The best model turned out to be a Polynomial Regression Model of degree 2, and a Ridge Regularizer with alpha 20.69.

```
weather_features = ["TMAX", "TMIN", "PRCP_log", "SNOW_log", "SNWD_log"]
wt_columns = ["WT01", "WT08"]
features = [*calendar_features, *weather_features]

df_train, df_test, y_test, y_train, regr = fit_predict(df_nyc, features, model="ridge", degree=2, alphas=np.logspace(1,3,20))
```

```
(1913, 123)
(304, 123)
Best alpha: 20.6913808111479
Training set R2 score: 0.8742445507680018
Test set R2 core: 0.864312438840875
Test set MAPE: 17.770484780541686
```

We see that this model results in training set R^2 score is 0.8742 and the test set is close to the testing set R^2 score of 0.8643.

We tried to further improve these results by enriching our features.

**Bike Stations Features**: We add 'EoM Fleet' feature to our dataset, which denotes the number of bikes available at the end of each month.

Then we train another degree 2 polynomial regression model with ridge regularization.

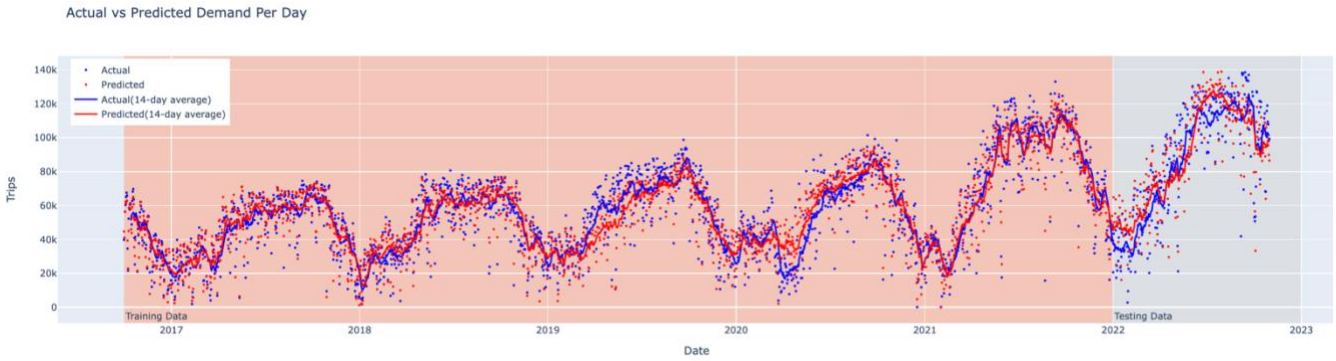
```
features = [*calendar_features, *weather_features, "EoM fleet"]
df_train, df_test, y_test, y_train, regr = fit_predict(df_nyc, features, model="ridge", log=False, degree=2, alphas=np.logspace(0,2))
```

```
(1913, 128)
(304, 128)
Best alpha: 100.0
Training set R2 score: 0.9087981634630531
Test set R2 core: 0.8500143716039421
Test set MAPE: 19.280609298587024
```

We see that the model trained on Calendar, Weather and Bike station features, obtains a training R^2 of 0.9087, and Testing R^2 of 0.8500.

Hence we can see that the calendar and weather features combined give us the best results.

So We plot a graph with our results from the model trained on just the Weather and Calendar features. We see that the prediction runs along with the actual number of trips daily.



# References & Citations


1. Citi bike Trips Data
2. Citi bike Real-time Station Feed API
3. NYC Weather Data
4. Citi bike Monthly Operating Reports
5. Reverse Geocoding with NYC Bike Share Data – Clif Kranish, 2021
6. Exploring NYC Bike Share Data – Clif Kranish, 2021