Citi Bike Data Analysis Using PySpark

Meher Kaur Banga
University of Colorado
meba8689@colorado.edu

Meaza Feyso
University of Colorado
mefe3861@colorado.edu

Alan Jordan
University of Colorado
aljo6807@colorado.edu

Maddie Wallace
University of Colorado
mawa9164@colorado.edu

Abstract

Public bike-share services are becoming increasingly popular across the nation. The emergence of bike-share systems has revolutionized the way people commute, and with the rise of climate change and environmental awareness, it has become a crucial part of urban transportation. Citi Bike is the largest bike-share program in the nation with over 25,000 bikes [1]. The availability of large amounts of data on bike-share usage provides an opportunity to analyze usage patterns and explore ways to optimize bike-share systems to improve accessibility for all users. We want to analyze how riders use these bikes in their lives, which can impact where new stations should be set up and how to maintain current ones. This project aims to analyze bike-share data to gain insights into usage patterns and explore ways to improve accessibility. Our analysis techniques include the use of Pyspark, Python, Matplotlib, and Folium.

1. Introduction/Background

Citi Bike is the largest bike-share program in the nation "with 25,000 bikes and over 1,500 stations across Manhattan, Brooklyn, Queens, the Bronx, Jersey City, and Hoboken" [1]. Users are able to pick up bikes from one station and return them to any other station. This enables eco-friendly transportation to be accessible in highly populated urban areas. Citi Bike offers two kinds of bike options: classic and electric. Users can ride casually (pay per ride or purchase a day pass) or can purchase a membership. Memberships provide "unlimited 45-minute rides on classic bikes, reduced e-bike prices, free unlocks" [1], and more.

The wide availability of Citi Bikes results in immense amounts of data available for analysis via the Citi Bike website. Data is available in the form of zip file downloads, organized by year and month. Some of these files contain millions of rows of data, equating to millions of rides per month.

1.1. Objectives

We analyzed four main objectives for this project. The first objective was to determine and examine patterns of the most popular bike routes (between stations) around Jersey City. The second objective was to identify areas of high usage and low usage to optimize the placement of bike-share stations, as well as bikes. The third objective was to establish the busiest times of day when bike-share was most used, as well as the busiest day of the week so that stations are able to be checked and properly stocked for busy times, enabling everyone who wishes to use the service to be able to.

An overall examination of the objectives gives a holistic view of the way Citi Bike is used by the community. It also provides insights for bike-sharing companies to optimize their resources, ways to increase the number of users, and ways to reduce costs to improve their bottom line.

1.2. Pre- and post-COVID 19

Our last objective was to deduce a variation in bike-share trends pre-Covid and post-Covid. This was important because much has changed in the ways that we as humans live our lives since COVID-19 started, and we wanted to observe those changes. Some results specifically based on this objective leave scope for further analysis of changing trends that can be observed ever since the pandemic hit. This allows the companies to better optimize their resources to cater to the new ways of bike-sharing.

Since this objective requires a comparison between two sets of data, the data chosen was the month of July 2019 to study pre-Covid trends, and the month of July 2022 to study post-Covid trends. These months were chosen based on higher usage and lower difference in the number of rides taken - unlike other months where there was a vast difference in the number of rides taken in 2019 versus the number of rides taken in 2022 for the same month.

2. Data

2.1. Obtaining the data

The data we used in this analysis was obtained from the Citi Bike official website [1]. The Citi Bike website contains data from 2013 to the present, broken down by month. There are larger datasets, containing all the data for every borough in New York City. There are also datasets containing only the data for Jersey City (JC) with file names beginning with the letter JC. The size of these datasets shows how much of a necessity Citi Bike has become, with some months containing millions of rows, meaning millions of rides being taken. For this project and the scope we were aiming to achieve, we decided to limit the data to the Jersey City datasets. This way we would be able to manage the size of the data better but still return an accurate representation of trends.

Within the Jersey City data, we used data from the months of April 2022 until March 2023, in order to analyze the most recent hence relevant trends.

The columns in the data before preprocessing included: ride ID, rideable type, started at, ended at, start station name, start station ID, end station name, end station ID, start latitude, start longitude, end latitude, end longitude, member or casual ride, as shown in *Figure 1*.

```
root
 |-- ride_id: string (nullable = true)
 |-- rideable_type: string (nullable = true)
 |-- started_at: timestamp (nullable = true)
 |-- ended_at: timestamp (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- start_station_id: string (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- end_station_id: string (nullable = true)
 |-- start_lat: double (nullable = true)
 |-- start_lng: double (nullable = true)
 |-- end_lat: double (nullable = true)
 |-- end_lng: double (nullable = true)
 |-- member_casual: string (nullable = true)
```

Figure 1: A schema of the dataset before preprocessing.

2.2. Preprocessing the data

We took multiple steps in the preprocessing of the data. The initial step was combining the twelve different datasets into one singular dataset. The joined dataset consisted of 955,591 rows and 13 columns. Then, all null values were dropped from the dataset.

The next step required us to do some feature engineering. The first step in this process was creating a new column containing the Unix timestamp for the started_at and ended_at columns. End time was then subtracted from the start time to obtain the trip duration in minutes in a new column. The Unix timestamp was helpful for this process because "Unix time is a way of representing a timestamp by representing the time as the number of seconds since January 1st, 1970 at 00:00:00 UTC" [2] and enabled us to create the trip duration column with ease. Once this column was created in the data frame, the Unix time columns were dropped, as they weren't needed for the analysis anymore.

Our final step in pre-processing our data was to drop all rides with trip durations of less than one minute and greater than thirty minutes[1], as well as any trips that had durations of negative times. This left us with a final dataset consisting of 919,107 rows and 14 columns that we used for our general analytics. The final dataset included the following columns: ride ID, rideable type, started at, ended at, start station name, start station ID, end station name, end station ID, start latitude, start longitude, end latitude, end longitude, member or casual ride, trip duration, as shown in *Figure 2*.

```
root
 |-- ride_id: string (nullable = true)
 |-- rideable_type: string (nullable = true)
 |-- started_at: timestamp (nullable = true)
 |-- ended_at: timestamp (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- start_station_id: string (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- end_station_id: string (nullable = true)
 |-- start_lat: double (nullable = true)
 |-- start_lng: double (nullable = true)
 |-- end_lat: double (nullable = true)
 |-- end_lng: double (nullable = true)
 |-- member_casual: string (nullable = true)
 |-- trip_duration: double (nullable = true)
```

Figure 2: A schema of the final dataset after preprocessing that is used for analysis.

2.3. Preprocessing the data, pre- and post-COVID-19

As mentioned in section 1.2, we acquired different months of data for our pre- and post-Covid analysis. Our processing for this data followed the same steps as mentioned in section 2.2, as well as some additional steps.

The data from July 2019 was provided with the following columns: trip duration (seconds), start time and date, stop time and date, start station name, end station name, station ID, station lat/long, bike ID, user type (customer or subscriber), gender, and year of birth. This dataset had 43,746 rows and 15 columns as shown in *Figure 3*.

```
root
 |-- tripduration: integer (nullable = true)
 |-- starttime: timestamp (nullable = true)
 |-- stoptime: timestamp (nullable = true)
 |-- start station id: integer (nullable = true)
 |-- start station name: string (nullable = true)
 |-- start station latitude: double (nullable = true)
 |-- start station longitude: double (nullable = true)
 |-- end station id: integer (nullable = true)
 |-- end station name: string (nullable = true)
 |-- end station latitude: double (nullable = true)
 |-- end station longitude: double (nullable = true)
 |-- bikeid: integer (nullable = true)
 |-- usertype: string (nullable = true)
 |-- birth year: integer (nullable = true)
 |-- gender: integer (nullable = true)
```

Figure 3: Schema for July 2019 before preprocessing.

[1] This step may vary for some objectives and is discussed later in the report.

The data from July 2022 was provided with the same column names as the data in section 2.1. This dataset consisted of 108,502 rows and 13 columns as shown in *Figure 4*.

```
root
 |-- ride_id: string (nullable = true)
 |-- rideable_type: string (nullable = true)
 |-- started_at: timestamp (nullable = true)
 |-- ended_at: timestamp (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- start_station_id: string (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- end_station_id: string (nullable = true)
 |-- start_lat: double (nullable = true)
 |-- start_lng: double (nullable = true)
 |-- end_lat: double (nullable = true)
 |-- end_lng: double (nullable = true)
 |-- member_casual: string (nullable = true)
```

Figure 4: Schema for July 2022 before preprocessing.

We first started by addressing the problem of inconsistent column names. For the pre-Covid (2019) dataset, we changed the following column names to match the column names in the post-Covid (2022) dataset: start time to started_at, stop time to ended_at, start station name to start_station_name, end station name to end_station_name, start station id to start_station_id, end station id to end_station_id, start station latitude to start_lat, end station latitude to end_lat, start station longitude to start_lng, and end station longitude to end_lng. Then, from the 2019 data set, we dropped the following: birth year, gender, bike ID, and trip duration. From the 2022 dataset, we dropped the ride_ID and rideable_type columns because the 2019 dataset lacked this data, and we could not engineer it ourselves. Then we had to change the user type column from containing 'subscriber' or 'customer' to match the 2022 data containing 'member' or 'casual', and changed the column name to member_casual.

An index column was also added to both datasets since we had to get rid of the ride_id column from 2022 and there was no identifier column in 2019.

We put both the 2019 and 2022 datasets through the same preprocessing steps mentioned in section 2.2. After preprocessing, the July 2019 data consisted of 43,746 rows and July 2022 consisted of 108,115 rows. Both these datasets had 15 columns for the final analysis as shown in *Figure 5*.

```
root
 |-- started_at: timestamp (nullable = true)
 |-- ended_at: timestamp (nullable = true)
 |-- start_station_id: integer (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- start_lat: double (nullable = true)
 |-- start_lng: double (nullable = true)
 |-- end_station_id: integer (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- end_lat: double (nullable = true)
 |-- end_lng: double (nullable = true)
 |-- member_casual: string (nullable = true)
 |-- index: long (nullable = false)
 |-- started_at_unix: long (nullable = true)
 |-- ended_at_unix: long (nullable = true)
 |-- trip_duration: double (nullable = true)

root
 |-- started_at: timestamp (nullable = true)
 |-- ended_at: timestamp (nullable = true)
 |-- start_station_name: string (nullable = true)
 |-- start_station_id: string (nullable = true)
 |-- end_station_name: string (nullable = true)
 |-- end_station_id: string (nullable = true)
 |-- start_lat: double (nullable = true)
 |-- start_lng: double (nullable = true)
 |-- end_lat: double (nullable = true)
 |-- end_lng: double (nullable = true)
 |-- member_casual: string (nullable = true)
 |-- index: long (nullable = false)
 |-- started_at_unix: long (nullable = true)
 |-- ended_at_unix: long (nullable = true)
 |-- trip_duration: double (nullable = true)
```

Figure 5: Schemas for July 2019 and July 2022 respectively, after going through preprocessing steps.

3. Methods

The code for our analysis was written using Python 3 programming language. All of our analysis was done using the Python API for Apache Spark i.e. PySpark. PySpark combines Python's learnability and ease of use with the power of Apache Spark to enable the processing and analysis of data at any size for everyone familiar with Python [3]. We ran our analysis locally inside a Docker [4] container created using the Jupyter PySpark Image. As shown in *Figure 6*, the command launches a Docker container based on the Jupyter/PySpark-notebook image, which is an image that includes Jupyter Notebook, Apache Spark, and Python libraries commonly used for data science tasks. We mapped port 8889 on the host machine to port 8888 inside the container. The main purpose of using a Docker container was to make sure our code that runs locally also runs in the same way when the container is used on an AWS cluster [5].

```
docker run -p 8889:8888 -v {path}:/home/jovyan/
exercises jupyter/pyspark-notebook
```

Figure 6: The Docker command used to launch a new container based on the Jupyter image.

The code testing was also done locally before uploading our code and data on the AWS cluster. Running our analysis in the Jupyter environment was crucial since packages like Folium work best in that environment and helped us create interactive maps. Folium was used to map top bike routes ranked by popularity and to visualize

bike stations with high and low usage based on the number of rides.

To visualize the results of the remaining analysis, our preferred library was Matplotlib. We used Matplotlib to create pie charts, line charts, and bar charts.

3.1. PySpark

PySpark is the Python API for Apache Spark, which is an open-source distributed computing system designed for processing large datasets [3]. PySpark provides a simple and intuitive interface for working with Spark that enables developers to write scalable, fault-tolerant, and distributed data processing applications using Python.

PySpark exposes the core Spark API to Python, which includes several libraries for data processing, machine learning, graph processing, and stream processing. Some of the key libraries that are included with PySpark are: Spark SQL, Spark Streaming, MLlib, GraphX, and PySpark SQL.

PySpark allows developers to work with Spark using Python, a language that is popular among data scientists and machine learning practitioners. PySpark code can be run locally on a single machine or scaled out to a cluster of machines, depending on the size of the data and the complexity of the processing.

3.2. Folium

Folium is a Python package that provides a way to create interactive maps using the Leaflet.js library [6]. With Folium, we can create a map of any location, add various types of data to the map such as markers, lines, and polygons, and customize the map. The package is built on top of the popular data visualization library Matplotlib and is designed to work seamlessly with Pandas data frames. Folium is a powerful tool for data visualization and exploration and is widely used, including in our analysis for geospatial representation.

3.3. Matplotlib

Matplotlib is a Python library for creating static, animated, and interactive visualizations in Python [7]. It is one of the most widely used data visualization libraries in the Python ecosystem and provides a wide range of customizable visualizations for scientific and engineering applications.

Matplotlib provides several APIs for creating different types of plots, such as line plots, scatter plots, bar plots, histograms, and many more. You can use Matplotlib to create highly customized plots by changing various properties such as the colors, labels, titles, axes, and legends of the plots.

Matplotlib is a powerful and versatile data visualization library that can be used for a wide range of applications, from simple line plots to complex 3D visualizations. Its flexibility and customization options make it an essential tool for us to communicate complex data insights effectively.

4. Analysis

Before beginning with the objectives, we worked on some basic analysis, to get to know the data better. The very first analysis was done by grouping the entire data using the column rideable_type i.e. the kind of bike used to make that trip. This column had two categories namely classic_bike and electric_bike. Out of the 919 thousand rides that were taken throughout the year, close to 600 thousand rides were taken on a classic bike while just above 200 thousand were taken on an electric bike, as depicted in *Figure* 7.

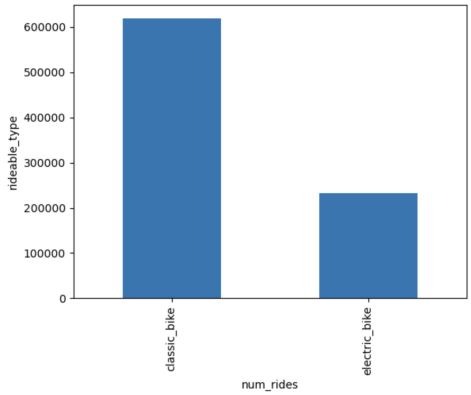

Figure 7: Number of rides calculated for column rideable_type to show usage of classic bikes versus electric bikes.

We also calculated the overall number of rides taken by members as well as non-members or casual riders. We observed that out of 919 thousand, the number of rides taken by members was 588 thousand and the number of non-member rides was just over 250 thousand.

The calculation of the total amount of time spent on bikes was done using the trip_duration column. All users combined spent over 7 million minutes riding Citi Bikes. Members spent 4.44 million minutes while casual riders spent 2.7 million minutes on Citi Bikes. This number justifies the previous calculation which shows that members took more rides than non-members.

After the basic analysis, the first objective implemented was to calculate the most popular routes by using the start and end station data and visualizing the most popular routes by mapping them using the Folium Python package. To calculate the most popular routes, we first group the data by ‘start station name’, ‘end station name’, ‘start latitude’, ‘start longitude’, ‘end latitude’, and ‘end longitude’. Then the number of rows is aggregated using count as shown in *Figure 8*.

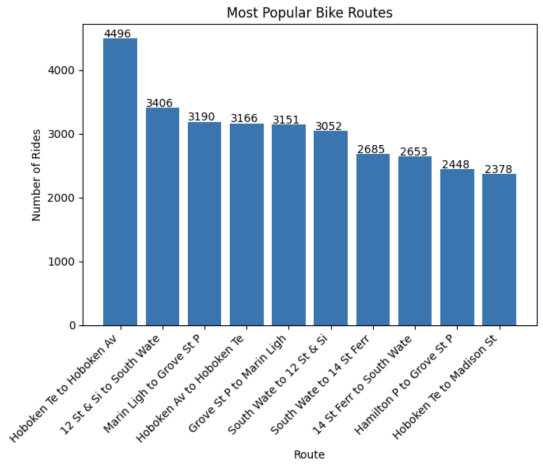

Figure 8: Number of rides over the most popular bike routes.

We also calculated the popular bike routes broken down by members and non-members, as shown in *figure 9* and *10*. This calculation was done using the previous aggregation, filtered for membership status.

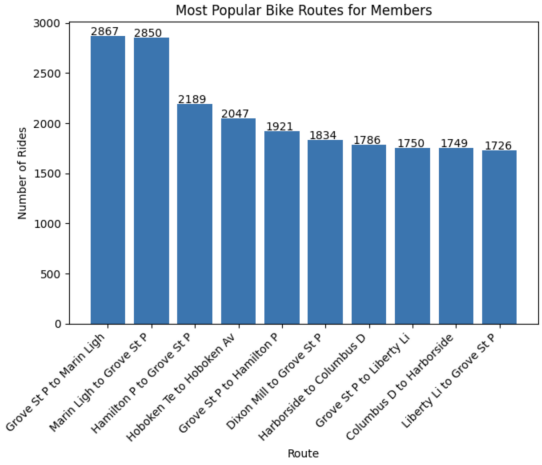

Figure 9: Number of most popular bike routes for members.

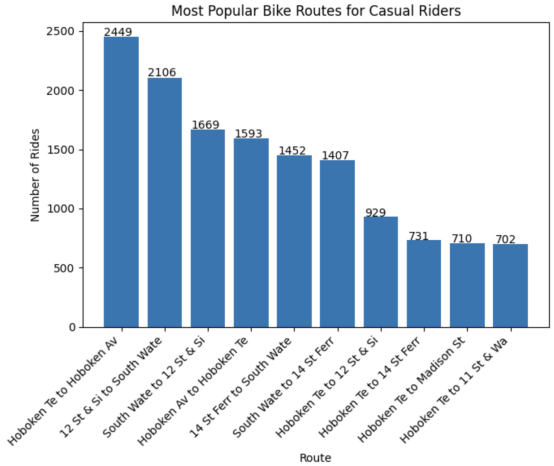

Figure 10: Number of most popular bike routes for casual users.

After we calculated the most popular routes, we mapped the most popular routes by using the Folium package in Python. The first step was to define the latitude and longitude of New York City and create a Folium map centered on those coordinates. In order to map each route, we extracted the start station names and end station names along with the coordinates for the first 20 rows of the data frame based on popularity. Then we marked the start and end points by providing the Folium package with the longitudes and latitudes of these stations. These stations were marked using a red Folium maker and connected using a blue line as shown in *Figure 11*.

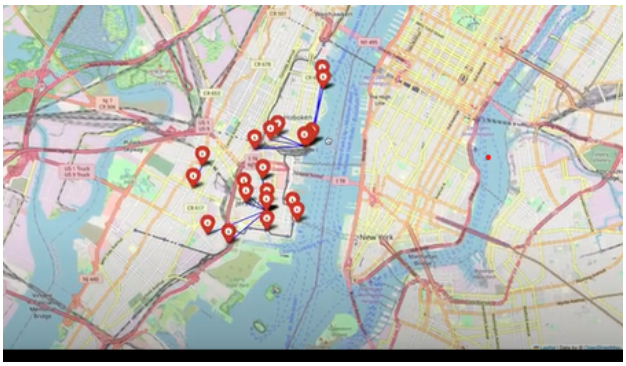

Figure 11: Image of Folium map showcasing red markers and blue lines.

Our second objective was to calculate the high and low usage stations to optimize the placement of bike-share stations and bikes. For this section, all rides greater than one minute were considered. The first step in the analysis was to calculate the total rides for both the start and end stations by using the 'groupby' method to group the rows by 'start_station_id', 'start_station_name', 'end_station_id', and 'end_station_name'. Then, we calculated the total rides by using the aggregate method with the count function. *Figure 12* shows the top 20 stations where rides began, along with the number of rides that started at those stations, the station ids, and the station names.

start_station_id	start_station_name	total_rides
JC005	Grove St PATH	35385
HB102	Hoboken Terminal ...	35165
HB103	South Waterfront ...	34854
HB101	Hoboken Terminal ...	29777
HB105	City Hall - Washi...	24546
JC008	Newport Pkwy	21963
JC009	Hamilton Park	20920
JC066	Newport PATH	20572
JC105	Hoboken Ave at Mo...	19694
JC013	Marin Light Rail	19047
HB202	14 St Ferry - 14 ...	17631
HB201	12 St & Sinatra Dr N	17136
HB607	Hudson St & 4 St	16605
HB502	11 St & Washingto...	16188
JC104	Harborside	15654
JC052	Liberty Light Rail	15135
JC098	Washington St	15104
HB603	8 St & Washington St	15053
JC106	Columbus Dr at Ex...	14828
HB402	Madison St & 1 St	14359

Figure 12: Top 20 stations to start a ride, showing number of rides, station id, and station names.

end_station_id	end_station_name	total_rides
JC005	Grove St PATH	36914
HB103	South Waterfront ...	35573
HB102	Hoboken Terminal ...	33286
HB101	Hoboken Terminal ...	30980
HB105	City Hall - Washi...	24970
JC008	Newport Pkwy	22132
JC009	Hamilton Park	21129
JC066	Newport PATH	20348
JC105	Hoboken Ave at Mo...	20319
JC013	Marin Light Rail	18952
HB202	14 St Ferry - 14 ...	18283
HB201	12 St & Sinatra Dr N	17657
HB607	Hudson St & 4 St	16539
JC104	Harborside	16188
HB502	11 St & Washingto...	16188
JC052	Liberty Light Rail	15488
JC098	Washington St	15338
JC106	Columbus Dr at Ex...	14963
HB603	8 St & Washington St	14669
JC003	City Hall	14588

Figure 13: Top 20 stations to end a ride with station ids and station names.

Similarly, *Figure 13* depicts the top 20 stations where rides ended along with the number of rides that ended at those stations. It also shows the station ids and station names.

After calculating the total number of rides for the start and end stations, we calculated the average number of rides at each station. This was done by calculating the number of rides per station and days of usage per station. Then, we divided the number of rides by rides per station, and the number of days by days per station. *Figure 14* shows this calculation of average rides taken for the top 20 stations.

start_station_name	num_rides	num_days	avg_rides_per_day
Grove St PATH	43735	364	120.1510989010989
Hoboken Terminal ...	35165	314	111.99044585987261
South Waterfront ...	34854	365	95.4904109589041
Hoboken Terminal ...	29777	365	81.58082191780822
City Hall - Washi...	24546	364	67.43406593406593
Sip Ave	5476	83	65.97590361445783
Newport Pkwy	21963	361	60.8393351800554
Newport PATH	20572	344	59.80232558139535
Hamilton Park	20920	364	57.472527472527474
Bergen Ave & Sip Ave	14215	259	54.884169884169886
Hoboken Ave at Mo...	19694	365	53.95616438356164
Marin Light Rail	19047	365	52.18356164383562
Columbus Dr at Ex...	14828	295	50.264406779661016
14 St Ferry - 14 ...	17631	364	48.43681318681319
12 St & Sinatra Dr N	17136	363	47.20661157024794
11 St & Washingto...	16188	351	46.11965811965812
Hudson St & 4 St	16605	365	45.49315068493151
8 St & Washington St	15053	332	45.34036144578313
Liberty Light Rail	15135	349	43.36676217765043
Harborside	15654	365	42.88767123287671

Figure 14: Average rides per station per day.

For further visualization, we plotted the top ten and the bottom ten stations based on the average number of rides taken from those stations per day using the Matplotlib package as shown in *Figures 15* and *16*.

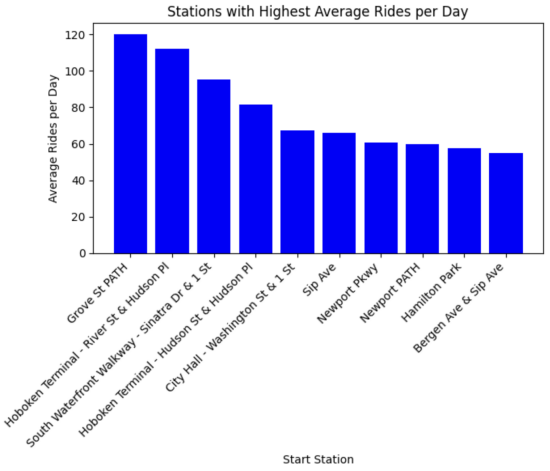

Figure 15: Top 10 stations with highest average rides per day.

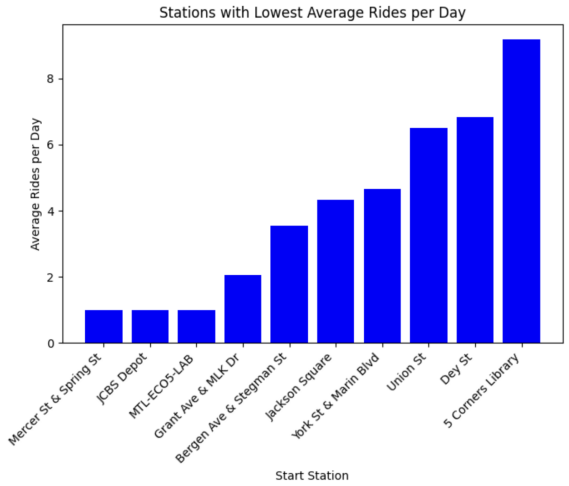

Figure 16: 10 stations with the lowest average rides per day.

Using the Folium package we also mapped the stations with the highest and lowest average rides. We mapped the higher ride stations and lower ride stations by using the average rides per day overall as a baseline. The calculated average number of rides per day is ~2518 rides. In the figure below, the green markers represent stations that are higher usage stations and the yellow markers represent stations that are lower than the average rides.

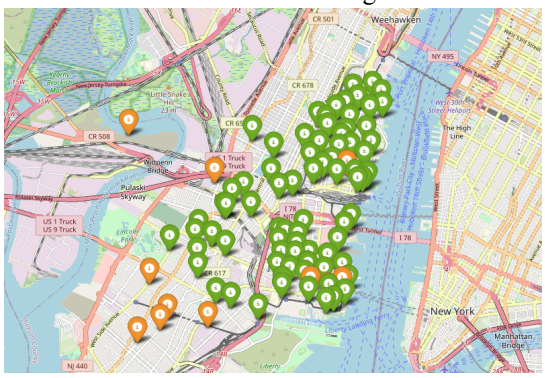

Figure 17: Highest and lowest usage stations based on average rides taken per day.

The third objective was to calculate the busiest times when users used Citi Bikes as a method of transportation. The visualization was done using Seaborn and Matplotlib packages in Python. The preprocessing step for this objective was slightly different, as the rides weren't filtered to be less than thirty minutes. All rides greater than one minute are considered in this section. The objective is divided into three sub-categories for better understanding namely, busiest time of the week, busiest time of the day to start a ride, and busiest time of the day with most active rides. The difference between the busiest time to start a ride and the most active rides is that most active rides do not depend on when the rides started. It is the hour of the day when the Citi Bike was being used to its maximum, irrespective of the ride starting time.

The busiest day of the week was calculated by creating a new data frame with the ride_id and the day_of_week column. The day_of_week column was created by converting the column started_at into the day of the week using the date_format function in PySpark. Then we grouped all the rides by the new column consisting of the day when the rides began. To calculate the busiest days for members and non-members, we added the member_casual column to this data frame and filtered the outcome by either 'member' or 'casual' to get the final output.

As shown in *Figure 18*, the busiest day of the week for all users combined is Friday, with 137 thousand rides, followed by Wednesday. The least busy day of the week is Monday, with close to 120 thousand rides.

```
+-----------+------+
|day_of_week| count|
+-----------+------+
|        Fri|137733|
|        Wed|137436|
|        Sat|137307|
|        Thu|135390|
|        Tue|128156|
|        Sun|123292|
|        Mon|119793|
+-----------+------+
```

Figure 18: Busiest day of the week sorted by number of rides.

This information is also visualized using Seaborn's heatmap shown in *Figure 19*. The darker and lighter colors represent the highest to lowest number of rides, respectively.

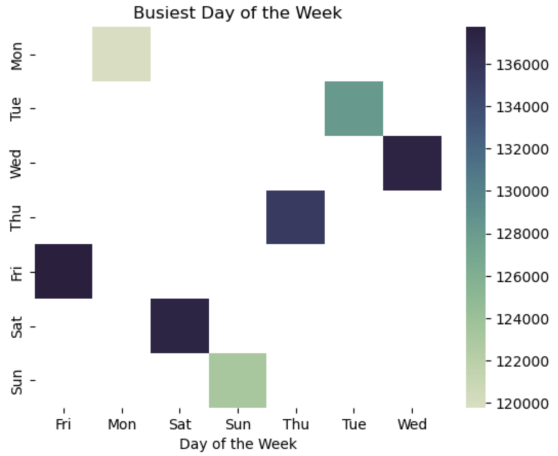

Figure 19: Heatmap showing the busiest day of the week for Citi Bike users.

However, the busiest day differs when calculated for members and non-members of the bike-sharing app. For members, the busiest day is Wednesday with close to 100 thousand rides and the least number of rides taken by members was on Sunday with just above 69 thousand rides. The most number of rides taken by non-members

was on a Saturday, just shy of 61 thousand and the least popular day was Tuesday. *Figure 120* depicts this information using a bar chart.

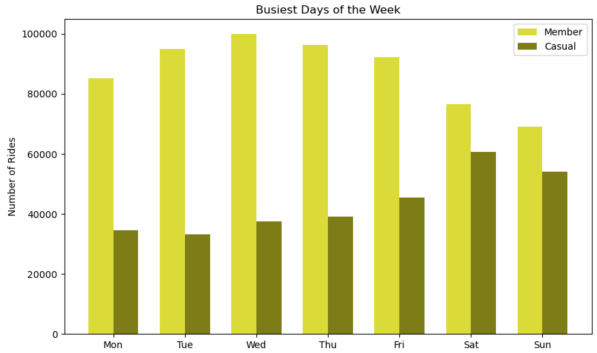

Figure 20: Busiest days of the week based on membership status.

To calculate the busiest time of the day to start a ride, we created a data frame with ride_id and hour_of_day. The column hour_of_day was calculated using the 'hour' function in PySpark. Then this data frame was grouped based on the hour_of_day column to get the busiest hour. Furthermore, to calculate the busiest time based on membership, the column member_casual was added to the data frame and it was filtered to display the number of rides for each hour for members or casual riders.

The busiest time of the day to start a ride for the general population, as well as members/non-members, was calculated by breaking down started_at into hourly buckets. There are two peaks we notice in this graph as shown in *Figures 21* and *22*. The busiest time to begin a ride was between 1800 and 1900 hours i.e. 6:00 pm to 7:00 pm, with ~89 thousand rides. However, there is a second peak which shows that 800 to 900 hours was also a very busy time for Citi Bike users.

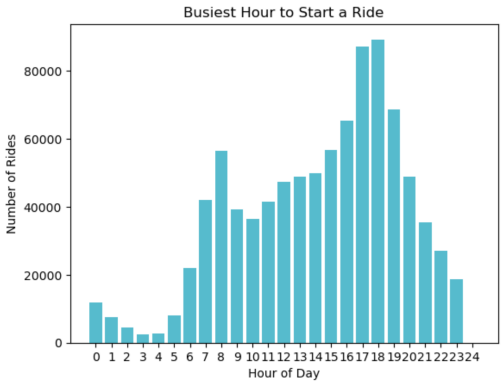

Figure 21: Busiest hour to start a ride for all users.

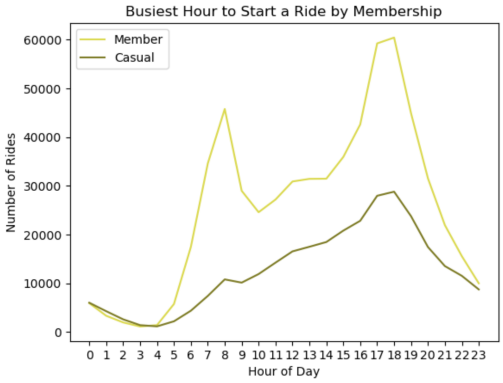

Figure 22: Busiest hour to start a ride for members and casual riders.

The most active time of the day was calculated by creating a data frame with ride_id, start_hour, end_hour, and active columns. The start_hour and end_hour were hour equivalents for the columns started_at and ended_at respectively. The active column consisted of a list for each ride, with the hours for which that ride was active based on the difference between start_hour and end_hour. This data frame can be seen in *Figure 23* below. Then we created an active_dict which contained each hour of the day, i.e. 0 to 24 as the keys and the number of times that hour appears in the column active as its values. This dictionary was converted into a Pandas data frame, sorted based on the hour of the day starting at 0, and visualized using Matplotlib line graph.

```
+----------------+----------+--------+--------------------+
|         ride_id|start_hour|end_hour|              active|
+----------------+----------+--------+--------------------+
|919C40A703A965D7|        15|      15|                [15]|
|3B40831921DAD6C1|        10|      10|                [10]|
|69C5C0766309F73D|        14|      14|                [14]|
|304B82055A1D4A24|        19|      19|                [19]|
|1CB760ECBCB08609|        11|      11|                [11]|
|E9CF8C3831A65032|        13|      15|        [13, 14, 15]|
|12B178ADBE12B8DF|        14|      19|[14, 15, 16, 17, ...|
|FAB45D41892C673B|        14|      14|                [14]|
|B4B70AD5E55683BA|         9|       9|                 [9]|
|57E29E1B4A6AB3A8|        16|      16|                [16]|
|AECC2C99837884A2|        16|      16|                [16]|
|E576379E1C00F017|        14|      14|                [14]|
|9BAD9C2EA323598D|        16|      16|                [16]|
|6DCAD4B36CA61446|        11|      12|            [11, 12]|
|040385786C00229B|        15|      15|                [15]|
|00A32EBEFEA61094|        14|      14|                [14]|
|B727CC9B1559ABBA|        21|      21|                [21]|
|310C2280DA70BF74|        20|      20|                [20]|
|F8C3513C34B876AF|        20|      20|                [20]|
|304ABE7A1614EB23|        17|      17|                [17]|
+----------------+----------+--------+--------------------+
only showing top 20 rows
```

Figure 23: Dataframe showing ride_id, start_hour, end_hour, and active hours for each ride.

The most active time of the day for Citi Bike users was not very different from the visualizations shown in *Figures 21* and *22*[2]. Similar to the busiest hour to start a ride, the most activity was between 6:00 pm and 7:00 pm for all users, with ~111 thousand rides taking place.

The fourth objective of our analysis was to compute

[2] These visualization diagrams have not been included here but are available in the code notebook.

and observe the trends before and after COVID-19. A few basic computations depict that the number of bike-sharing rides taking place after the pandemic has dramatically increased. We can observe that from *Figure 24* which shows the number of members and casual riders Citi Bike had before (July 2019) and after (July 2022) COVID-19.

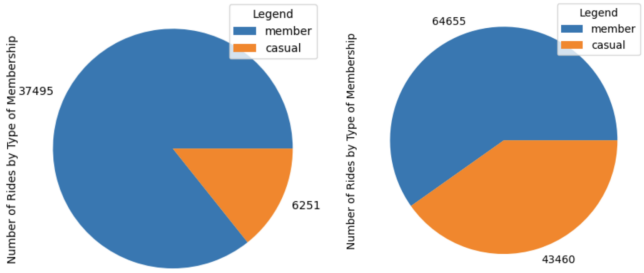

Figure 24: Pie chart showing the number of members and non-members pre-Covid (left) and post-Covid (right).

The left pie chart shows the number of members and casual riders in July 2019. We can see that the total number of rides taken was 43,746, and about 85 percent were taken by members. The pie chart on the right shows that the total number of rides increased by 2.5 times, to 108,115 in July 2022. We also observed that the share of non-members riding was now close to 40 percent as opposed to a mere 15 percent before the pandemic.

After performing basic analysis on pre- and post-Covid data, the next step was to identify popular routes. In the pre-Covid analysis, we noticed Grove St PATH is the most popular route, however, the popularity decreased for the post-Covid data. This can be attributed to people potentially riding the bikes directly to their destinations rather than to a subway station. Also, the addition of new stations can also be responsible for these changes. See *Figures 25* and *26*.

start_station_name	end_station_name	trip_dur	num_rides
Hamilton Park	Grove St PATH	5030.0	960
Grove St PATH	Hamilton Park	3271.0	609
Brunswick & 6th	Grove St PATH	3218.0	569
Marin Light Rail	Grove St PATH	2425.0	543
Grove St PATH	Marin Light Rail	2412.0	515
Jersey & 6th St	Grove St PATH	1802.0	441
Monmouth and 6th	Grove St PATH	2256.0	439
Brunswick St	Grove St PATH	1980.0	430
Dixon Mills	Grove St PATH	1355.0	428
McGinley Square	Sip Ave	1680.0	387
Grove St PATH	Dixon Mills	1315.0	350
Grove St PATH	Brunswick & 6th	2002.0	346
Newport PATH	Harborside	2060.0	340
Grove St PATH	Brunswick St	1574.0	316
Grove St PATH	Monmouth and 6th	1495.0	306
Harborside	Newport PATH	1605.0	299
Grove St PATH	Jersey & 6th St	1199.0	275
Sip Ave	McGinley Square	1198.0	268
Fairmount Ave	Sip Ave	1463.0	264
Van Vorst Park	Grove St PATH	779.0	248

Figure 25: Popular routes for pre-Covid data.

start_station_name	end_station_name	trip_dur	num_rides
Hoboken Terminal ...	Hoboken Ave at Mo...	5151.0	604
12 St & Sinatra Dr N	South Waterfront ...	5605.0	532
Grove St PATH	Marin Light Rail	2265.0	450
Marin Light Rail	Grove St PATH	2073.0	439
Hoboken Ave at Mo...	Hoboken Terminal ...	3711.0	428
South Waterfront ...	12 St & Sinatra Dr N	4578.0	422
14 St Ferry - 14 ...	South Waterfront ...	3578.0	328
Grove St PATH	Liberty Light Rail	3109.0	318
Hamilton Park	Grove St PATH	1670.0	299
Grove St PATH	Hamilton Park	1744.0	292
Hoboken Terminal ...	Madison St & 1 St	1598.0	287
Liberty Light Rail	Grove St PATH	2481.0	281
South Waterfront ...	14 St Ferry - 14 ...	3227.0	281
Hoboken Terminal ...	12 St & Sinatra Dr N	2735.0	273
Harborside	Columbus Dr at Ex...	764.0	259
Grove St PATH	Lafayette Park	2769.0	254
Lafayette Park	Grove St PATH	2367.0	253
Dixon Mills	Grove St PATH	839.0	246
Baldwin at Montgo...	Grove St PATH	1940.0	236
Columbus Dr at Ex...	Harborside	957.0	230

Figure 26: Popular routes for post-Covid data.

After we figured out which routes were most popular, we analyzed if there was a change in the usage of the stations. In order to learn about this, the next step was determining whether the average rides per day per station showed a discernible change between the pre- and post-Covid data. See *Figures 27* and *28*.

The pre- and post-Covid usage analysis shows a sizeable jump in the average number of rides per day. The most popular route, Grove St PATH, is still seen as the highest ride per day for both pre- and post-Covid, and the average number for that specific station only changes by three rides when comparing pre-Covid with post-Covid. However, there is also a huge variance between the stations, meaning the stations that we see as top ten for pre-Covid are not the same as the stations we see as top ten for post-Covid. This could be due to people directly using bike rides to their destination rather than using the subway in between, or due to new stations that are more convenient for users.

start_station_name	num_rides	num_days	avg_rides_per_day
Grove St PATH	4798	31	154.7741935483871
Hamilton Park	2670	31	86.12903225806451
Harborside	2014	31	64.96774193548387
Sip Ave	2003	31	64.61290322580645
Newport PATH	1740	31	56.12903225806452
Marin Light Rail	1636	31	52.774193548387096
Columbus Dr at Ex...	1068	21	50.857142857142854
Newport Pkwy	1226	31	39.54838709677419
Newark Ave	1215	31	39.193548387096776
Morris Canal	1187	31	38.29032258064516
Paulus Hook	1165	31	37.58064516129032
Washington St	1117	31	36.03225806451613
Brunswick & 6th	1101	31	35.516129032258064
Warren St	1095	31	35.32258064516129
City Hall	1083	31	34.935483870967744
Liberty Light Rail	1013	31	32.67741935483871
Jersey & 6th St	978	31	31.548387096774192
Van Vorst Park	917	31	29.580645161290324
Jersey & 3rd	913	31	29.451612903225808
Columbus Drive	903	31	29.129032258064516

Figure 27: Average rides per day for pre-Covid data.

start_station_name	num_rides	num_days	avg_rides_per_day
Grove St PATH	4677	31	150.8709677419355
South Waterfront ...	4354	31	140.4516129032258
Hoboken Terminal ...	3748	30	124.93333333333334
Newport Pkwy	3111	31	100.35483870967742
Hoboken Terminal ...	3047	31	98.29032258064517
Marin Light Rail	2724	31	87.87096774193549
12 St & Sinatra Dr N	2704	31	87.2258064516129
City Hall - Washi...	2695	31	86.93548387096774
Hamilton Park	2680	31	86.45161290322581
Newport PATH	1746	22	79.36363636363636
Hoboken Ave at Mo...	2306	31	74.38709677419355
Liberty Light Rail	2224	30	74.13333333333334
Washington St	2223	31	71.70967741935483
Columbus Dr at Ex...	2209	31	71.25806451612904
14 St Ferry - 14 ...	2155	31	69.51612903225806
Harborside	2113	31	68.16129032258064
Hudson St & 4 St	1922	31	62.0
8 St & Washington St	1832	31	59.096774193548384
City Hall	1717	31	55.38709677419355
Madison St & 1 St	1645	31	53.064516129032256

Figure 28: Average rides per day for post-Covid data.

Next, for further visualization, we plotted the top ten and the bottom ten stations based on the average number of rides taken from those stations per day using the Matplotlib package as shown in *Figures 29* and *30*.

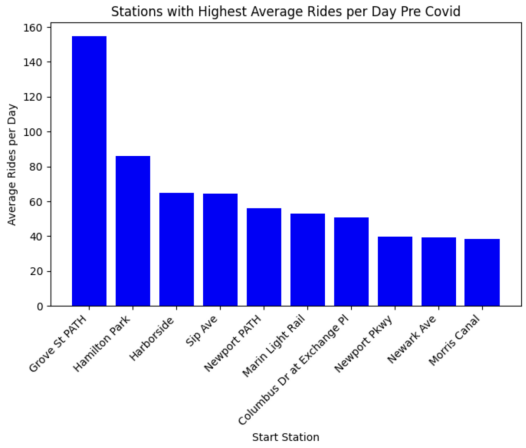

Figure 29: Top ten stations with the highest average rides per day for pre-Covid data.

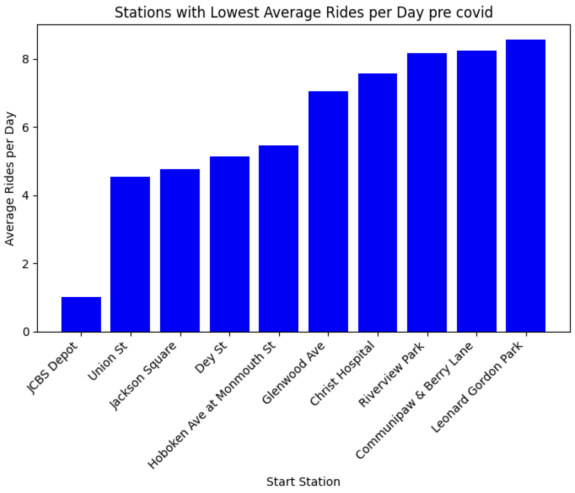

Figure 30: Bottom ten stations with the lowest average rides per day for pre-Covid data.

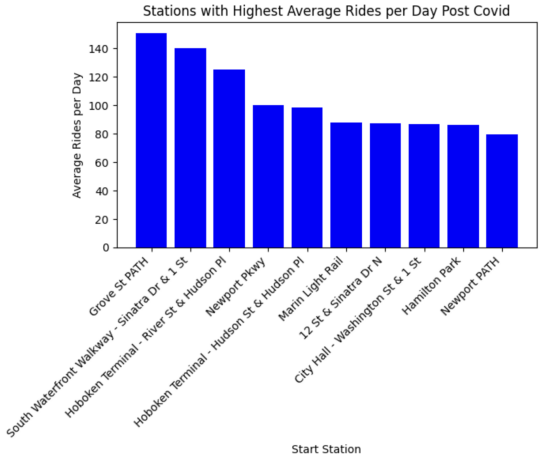

Figure 31: Top ten stations with the highest average rides per day for post-Covid data.

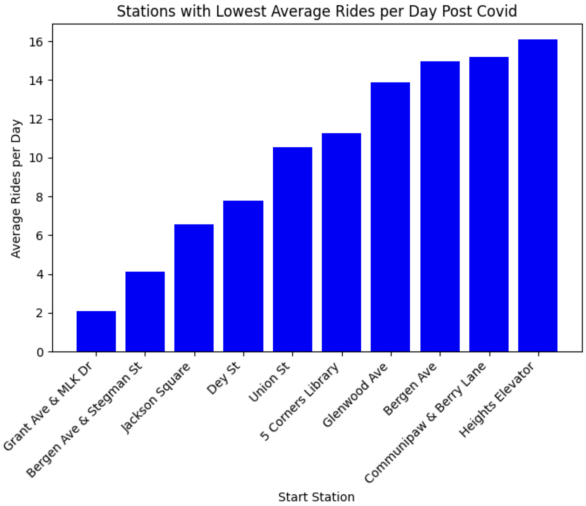

Figure 32: Bottom ten stations with the highest average rides per day for post-Covid data.

Using the Folium package we also mapped the stations with the highest and lowest average rides per day. We mapped the higher and lower usage stations by using the average rides per day overall as a baseline. The calculated average number of rides per day is ~2,400 rides. As shown in the map below, pre-Covid has fewer stations that have high amounts of rides compared to the average number of rides. In *Figure 33*, you can see that only a couple are marked as green. Additionally, if you look at *Figure 34*, you will see that the map indicates more stations on the post-Covid analysis, which supports our analysis of more stations being recognized.

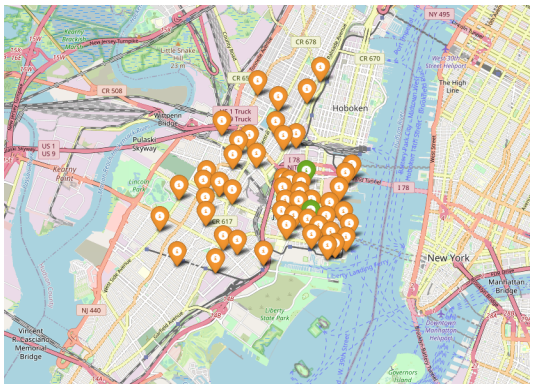

Figure 33: Highest and lowest usage stations based on average rides taken per day for pre-Covid data.

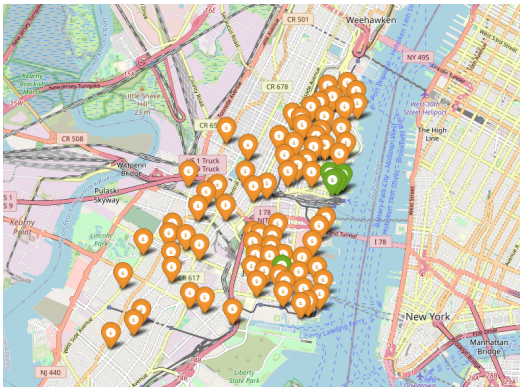

Figure 34: Highest and lowest usage stations based on average rides taken per day for post-Covid data

The busiest day of the week observed for all users before COVID-19 was Tuesday, with just above 7,500 rides. The busiest day for all users after COVID-19 was Saturday, with ~19 thousand rides taking place. See *Figure 35*.

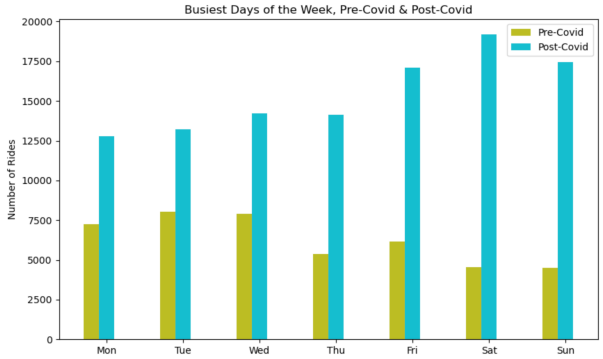

Figure 35: Bar graph showing the busiest day of the week pre-Covid and post-Covid.

For members, Tuesday and Wednesday were the most popular days before the pandemic, while Friday was the most popular day after the pandemic. As for non-members, the busiest day of the week to use Citi Bike was Saturday, both before and after COVID-19. See *Figure 36*.

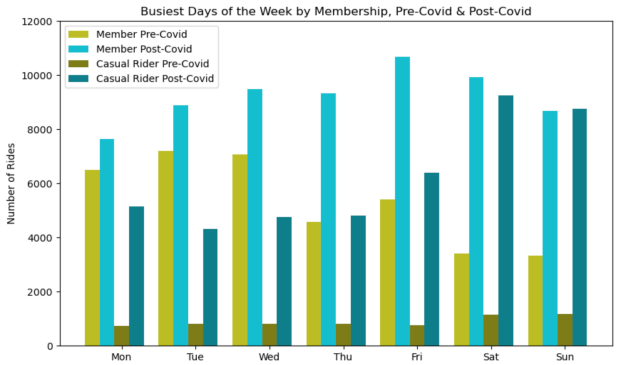

Figure 36: Busiest day of the week by membership status, showing pre-Covid and post-Covid trends.

The busiest hour of the day to start a ride remained similar for both scenarios, with 1800 to 1900 hours being the most popular hour. Similar to our latest analysis of 2022-2023, we can observe two peaks, the second being 800 to 900 hours as shown in *Figure 37*.

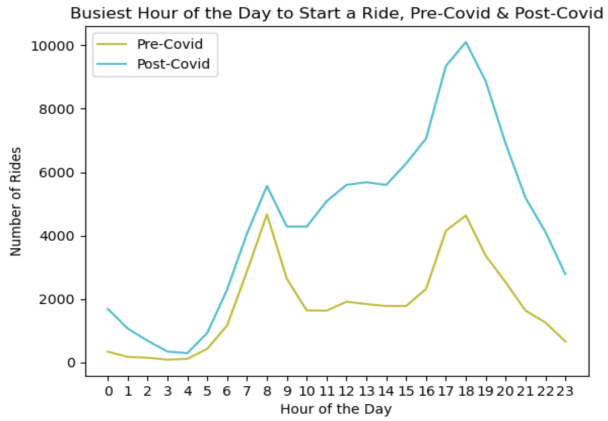

Figure 37: Busiest hour of the day to start a ride pre-Covid and post-Covid.

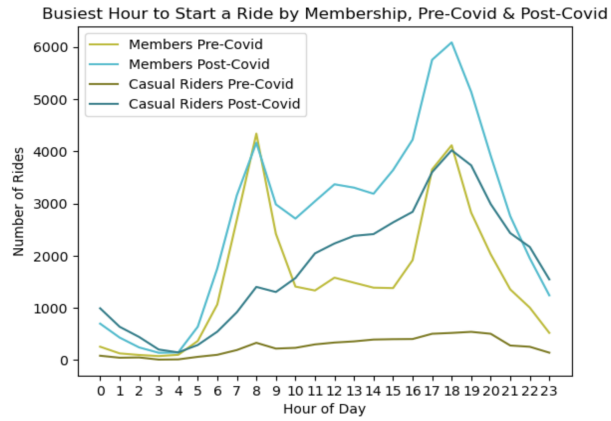

Figure 38: Busiest hour of the day to start a ride based on membership status, pre-Covid and post-Covid.

Figure 38 depicts the busiest hour of the day to start rides for members and casual riders. This analysis also reflects the common trends seen in *Figure 22*, done on April 2022 to March 2023 datasets.

As per *Figures 39* and *40*, the most active hour of the day for rides is 6:00 pm to 7:00 pm, followed closely by 5:00 pm to 6:00 pm. This is true for both July 2019 as well as July 2022. A similar pattern is also followed by pre- and post-Covid analysis done for members and casual riders using Citi Bikes.

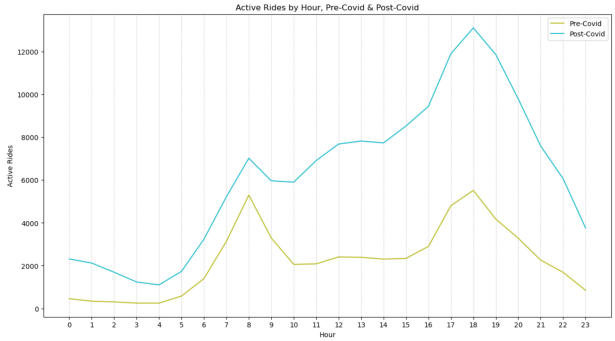

Figure 39: Most active rides by the hour for pre- and post-Covid analysis.

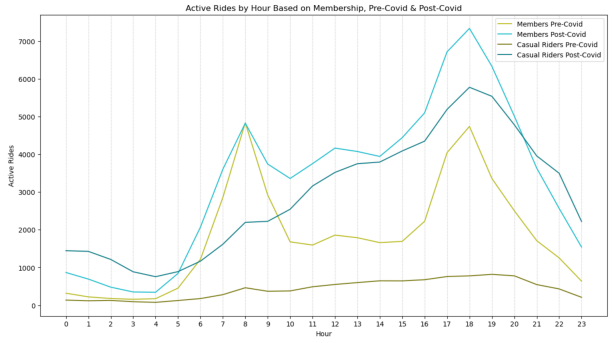

Figure 40: Most active rides by the hour based on membership, for pre- and post-Covid analysis.

5. Results and Insights

The findings presented in *Section 4* show bike usage patterns, user behavior, and system performance over a period of time. Beginning with our basic analysis (*Figure 7)*, there is a huge difference in the number of rides taken using classic versus electric bikes. This abnormal difference was expected because Citi Bike introduced electric bikes in May 2022 only for its members, and for all the riders the electric bikes were available only starting in Fall 2022 [8]. As for trends in membership, the number of rides taken by members is far more than the number of rides taken by casual riders. This is true for the latest data, as well as pre- and post-Covid data. We believe that this may be due to the fact that becoming a member of Citi Bike has its benefits, such as increased ride time, reduced e-bike prices, priority support, etc [1].

Our next finding was on the most popular bike routes for Citi Bike riders as shown in *Figure 8*. Upon examining the data, we found that the most popular bike route in the general analysis is from Hoboken Terminal to Hoboken Avenue, which is not surprising given these stations centrality in the city.

After conducting the highest and lowest station usage analysis, we observed that Grove St. Path has the highest total rides as shown in *Figure 15*. This shows that this station has more users and is more convenient to start or end rides, likely due to it being located in the rapid transit system in northeastern New Jersey as well as Lower and Midtown Manhattan in New York City [9]. On the other hand, the lowest usage station, with an average ride of one per day, is Mercer St. and Spring St. stations (shown in *Figure 16*) which may be due to the station being a local station resulting in less recognition or need for it.

As for analysis on the busiest day of the week for Citi Bike users, we observe that members use the bike-sharing service during weekdays, while casual riders prefer to use it over the weekend. The reason for this could perhaps be that members use Citi Bike to commute to and from work

during the weekdays, while non-members use it to take rides for leisure and/or recreational purposes on the weekend. Talking about the busiest hour to start a ride and most active time of use, there are two peaks in the graphs as seen in *Figures 21* and *22*. The first peak is at 1800 hours while the second highest peak is at 800 hours. This information confirms our explanation that Citi Bikes are used by office goers since 8:00 am and 6:00 pm are rush hours to travel to and from workplaces, respectively.

Our pre- and post-Covid analysis suggests that the number of rides taken in July 2022 had dramatically increased from that in July 2019. We can associate this with the fact that people were skeptical of using crowded public transport like subways after the pandemic. They preferred to use bikes that could be easily sanitized and reduced the risk of getting infected with COVID-19.

To reiterate, our analysis of the bike-sharing app Citi Bike provided us with some useful insights into the user behavior, patterns, and performance of a ride-sharing app. Analyzing pre- and post-Covid data revealed how people have changed their ways and habits in order to adapt to the changed environment in response to the Covid-19 pandemic.

6. Implications and Conclusion

One of the main implications of our study is that bike-sharing systems can provide a cost-effective and sustainable mode of transportation in urban areas. Our analysis revealed that bike-sharing is most frequently used for short-term trips, such as commuting to work or running errands. This suggests that bike-sharing can be a viable alternative to traditional forms of transportation, such as private cars or public transit, for short trips.

Another implication of our study is that bike-sharing usage patterns vary significantly depending on the time of day and day of the week. We found that bike-sharing usage peaks during the morning and evening rush hours, suggesting that the system should be optimized to meet the demand during these periods. Our analysis also revealed that bike-sharing is used more frequently on weekdays than on weekends, indicating that bike-sharing systems may be more effective in urban areas with a high proportion of commuters.

In conclusion, our study provides in-depth insights into bike-sharing usage. Our recommendations for optimizing bike-sharing systems include improving the distribution of bikes at stations with a high popularity, increasing the number of stations in areas with high demand, and providing incentives for users to use bikes over the weekend and at non-peak times. By implementing these recommendations, bike-sharing operators can promote the use of sustainable transportation, increase their revenue and provide better service to users*.

References

[1] *Citi Bike: NYC's Official Bike Sharing System: Citi Bike NYC*. Citi Bike: NYC's Official Bike Sharing System | Citi Bike NYC | Citi Bike NYC. (n.d.). Retrieved March 2023, from https://citibikenyc.com/homepage

[2] *What is unix time?* Narrative Knowledge Base. (n.d.). Retrieved April 2023, from https://kb.narrative.io/what-is-unix-time

[3] *PySpark overview*. PySpark Overview - PySpark 3.4.0 documentation. (n.d.). Retrieved April 2023, from https://spark.apache.org/docs/latest/api/python/

[4] *Accelerated, containerized application development*. Docker. (2023, April 4). Retrieved May 3, 2023, from https://www.docker.com/

[5] *Cluster - Amazon Elastic Container Service*. (n.d.). Retrieved April 2023, from https://docs.aws.amazon.com/AmazonECS/latest/APIReference/API_Cluster.html

[6] *Folium*. Folium - Folium 0.14.0 documentation. (n.d.). Retrieved March 2023, from https://python-visualization.github.io/folium/

[7] *Visualization with python*. Matplotlib. (n.d.). Retrieved April 2023, from https://matplotlib.org/

[8] David Meyer, Lyft introducing hundreds of new electric bicycles for NYC Citi Bike members, New York Post, April 29, 2022 https://nypost.com/2022/04/29/lyft-introducing-hundreds-of-new-e-bikes-for-nyc-citi-bike-members/

[9] Wikimedia Foundation. (2023, April 1). *Grove Street Station (PATH)*. Wikipedia. Retrieved April 2023, from https://en.wikipedia.org/wiki/Grove_Street_station_(PATH)

[10] *Chat.openai.com*. (n.d.). Retrieved from https://chat.openai.com/

* Parts of this section were written with the help of ChatGPT [10].