A NEW HINT TO TRANSPORTATION-ANALYSIS OF THE NYC BIKE SHARE SYSTEM

PROJECT REPORT

Submitted by

Team ID : PNT2022TMID17752

Team Leader : LEKHASRI.K

Team Member 1 : MAHASIVAPRIYA.B

Team Member 2 : NISHAA.S

Team Member 3 : GAYATRI.M

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Institution affiliated to Anna University Chennai)

November 2022

1. INTRODUCTION

1.1 Project Overview

The goal of this analysis is to create an operating report of Citi Bike for the year 2018. let us create data visualizations to understand the total number of trips, find the most common used Customer and subscriber based on gender, find the top bike used with respect to trip duration, Calculate the number of bikes used by respective age groups, find the top 10 Start Station Names with respect to Customer age group.

1.2 Purpose

The project mainly focuses on the analyses New York City's Citi-Bike share system to understand the spatial design considerations as well as usage patterns that emerge from this analysis. The Citi-Bike trip dataset was a massive in size containing over a millon observations per month. So, Inorder to better understanding of dataset for generating useful visualizations for developing more relevant insight about the system, a set of descriptive statistics were generated.It's important to provide better data visualizations and reduce the complexity of the dataset.

2. LITERATURE SURVEY

2.1 Existing problem

[1] Ines et al.,“ Bicycle sharing systems demand” on Science Direct-Social and Behavioral Sciences 111 (2014) 518 – 527

This paper sets out a method for estimating the bike-sharing demand and it allows to geo-reference the demand, considering the characteristics of the city and of the trips

Merits:

It can be adapted to other towns and cities according its characteristics.

The method is useful in the full design ie. location of bike-sharing stations and in the dimension of the fleet, as well as in the scheduling of the investments.

Demerits:

It did not consider other socio-economic characteristics, such as population density, non-institutionalized group quarter population density.

[2]Elias et al.,”What do trip data reveal about bike-sharing system users? “ ScienceDirect Journal of Transport Geography 91 (2021) 102971

It used data from the Helsinki BSS from 2017 (~1.5 million trips) as a case to study the potential of trip data for future BSS studies.

Merits:

Based on this paper results, trip databases are well established to support spatiotemporal analyses on where and when trips are being taken in general and how the demand varies at the stations.

It can help to uncover nuanced cycling patterns or even general mobility flows in urban areas without compromising user's privacy.

Demerits:

It focuses only on urban areas. But it does not consider the rural areas for bike sharing system.

[3]FRANCESCO et al.,"Bike Sharing and Urban Mobility inaPost-Pandemic" IEEE Access 2020

They presented an analysis of the bike sharing data during the month of March 2020, observing the changes in New Yorkers' mobility patterns in response to the pandemic and the countermeasures against it

Merits:

Their analysis of mobility patterns provides evidence that bike sharing, and cycling in general, can provide a flexible and eco-friendly mode of transportation for shorter trips

Demerits:

They did not mention that the data sources could be combined with POI data for better clustering the station category and understanding the spatial variation of bikeshare ridership.

[4]"A long-term perspective on the COVID-19: The bike sharing system resilience under the epidemic environment"Journal of Transport & Health ,2021

This study applied a series of statistical techniques including spatial-temporal approach, complex network-motivated methodology and cyclist behavior analysis to capture the influence of the COVID-19 pandemic on bike sharing mobility patterns.

Merits:

It has illustrated the importance of a bike sharing system on people's daily life during the outbreak. Results reveal that a bike sharing system could potentially reduce the load on the urban transport network and improve the resilience of the transportation systems.

Demerits:

It did not do the work that the clearer picture of the role of a bike sharing program in these emergency situations can be refined and confirmed as more relevant studies are conducted in other cities with bike sharing systems

[5] Nguyen Thi Hoai Thu, Chu Thi Phuong Dung,"Multi-source DataAnalysis for Bike Sharing Systems"Vietnam 2017 International Conference on Advanced Technologies for Communications .

They developed a multi-source data analysis approach for addressing the rebalancing problem of BSSs by using BSS historical trip records, meteorological data andtaxi usage data.

Merits:

The exploration of multiple sources of data affecting BSSs is highly beneficial to improve bike demand prediction accuracy Use of ANN provides better performance.

Demerits:

It did not make use of the housing and demographic data for station clustering algorithm, bus data, and subway data to predict the bike demand and to expand the system.

2.2 References

[1] Ines et al.," Bicycle sharing systems demand" on Science Direct-Social and Behavioral Sciences 111 (2014) 518 – 527

[2]Elias et al.,"What do trip data reveal about bike-sharing system users? " ScienceDirect Journal of Transport Geography 91 (2021) 102971

[3] FRANCESCO et al.,"Bike Sharing and Urban Mobility inaPost-Pandemic" IEEE Access 2020

[4] "A long-term perspective on the COVID-19: The bike sharing system resilience under the epidemic environment"Journal of Transport & Health ,2021

[5] Nguyen Thi Hoai Thu, Chu Thi Phuong Dung,"Multi-source DataAnalysis for Bike Sharing Systems"Vietnam 2017 International Conference on Advanced Technologies for Communications .

2.3 Problem Statement Definition

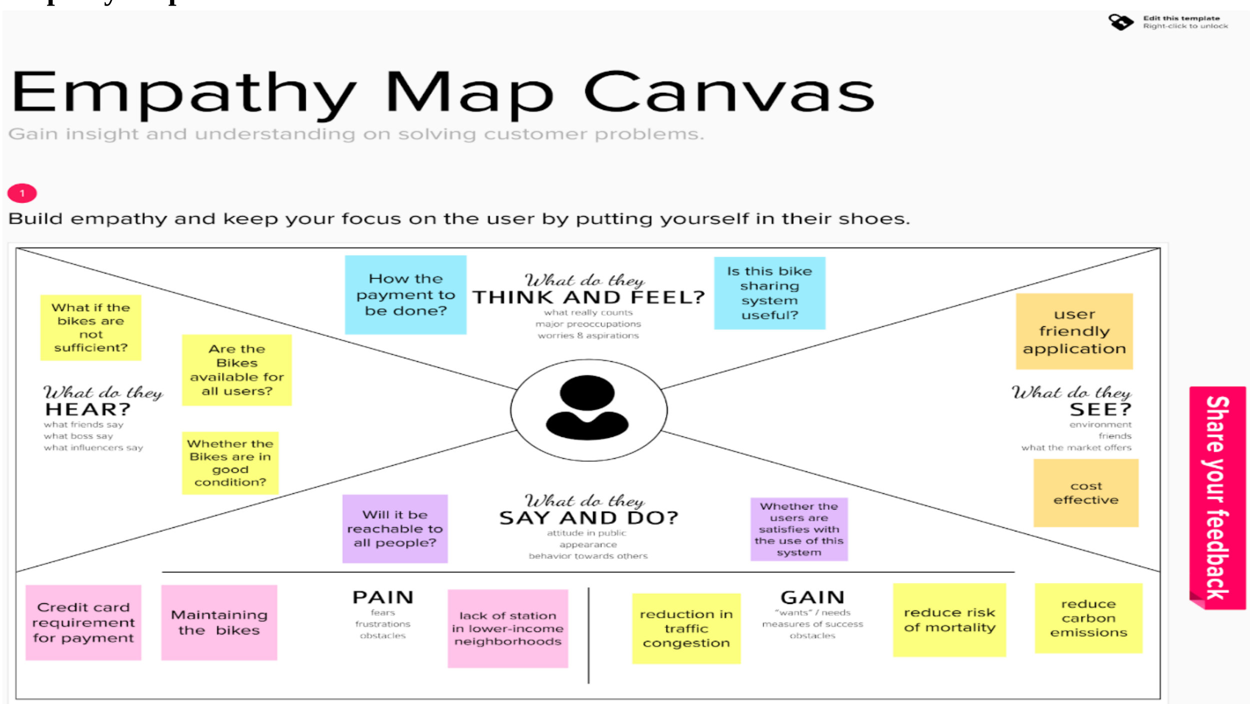
In recent years, U.S cities have increasingly adopted bike-sharing system to reduce carbon emissions and increase active travel. Bike sharing system has become increasingly popular in many cities. Bike sharing has been considered a suitable mode to support the first and last-mile connectivity problems of fixed-route transit services. These services allow users to rent bikes for utilitarian an recreational trips in the urban area. Our objective is to make analysis on this NYC bike sharing system. The main problem is when a customer wants to rent a bicycle, if the bike is not available there, it makes the customer feel bad right? analyzing the increasing bike demand in different locations is more important.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel

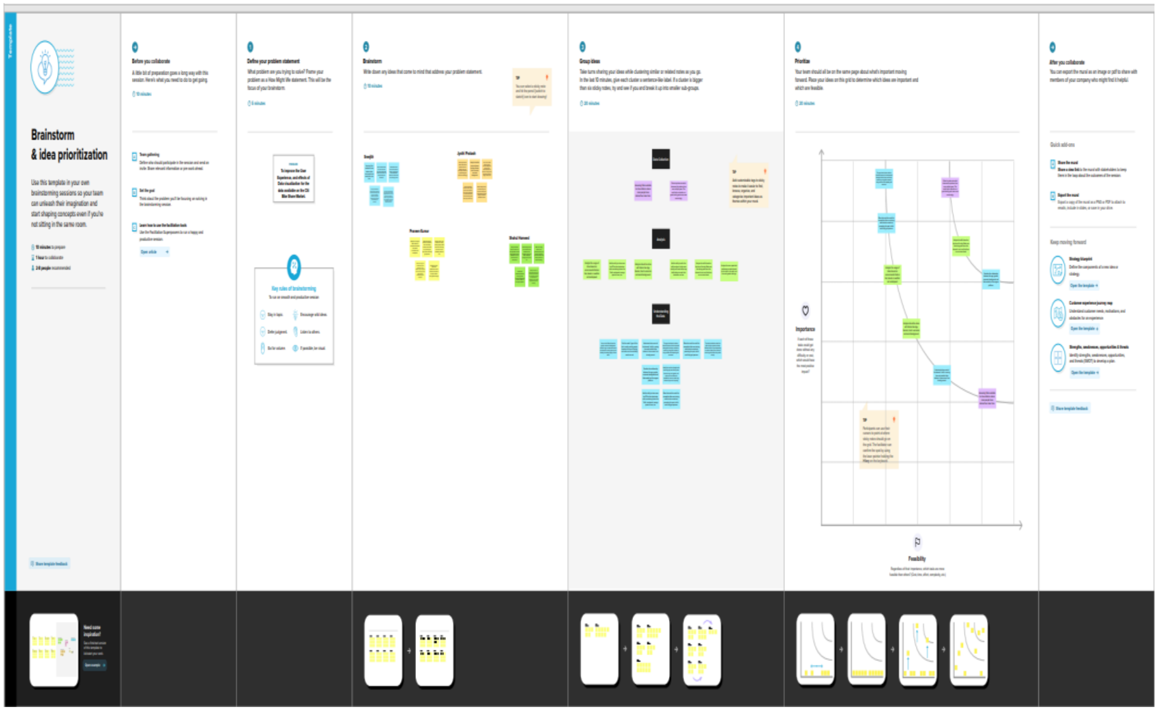
PS-1	Customer	Rent a bicycle	It is not available at that time	Initially, there are only few people on that area, but now population get expanded mainly in peak hours	Bad

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

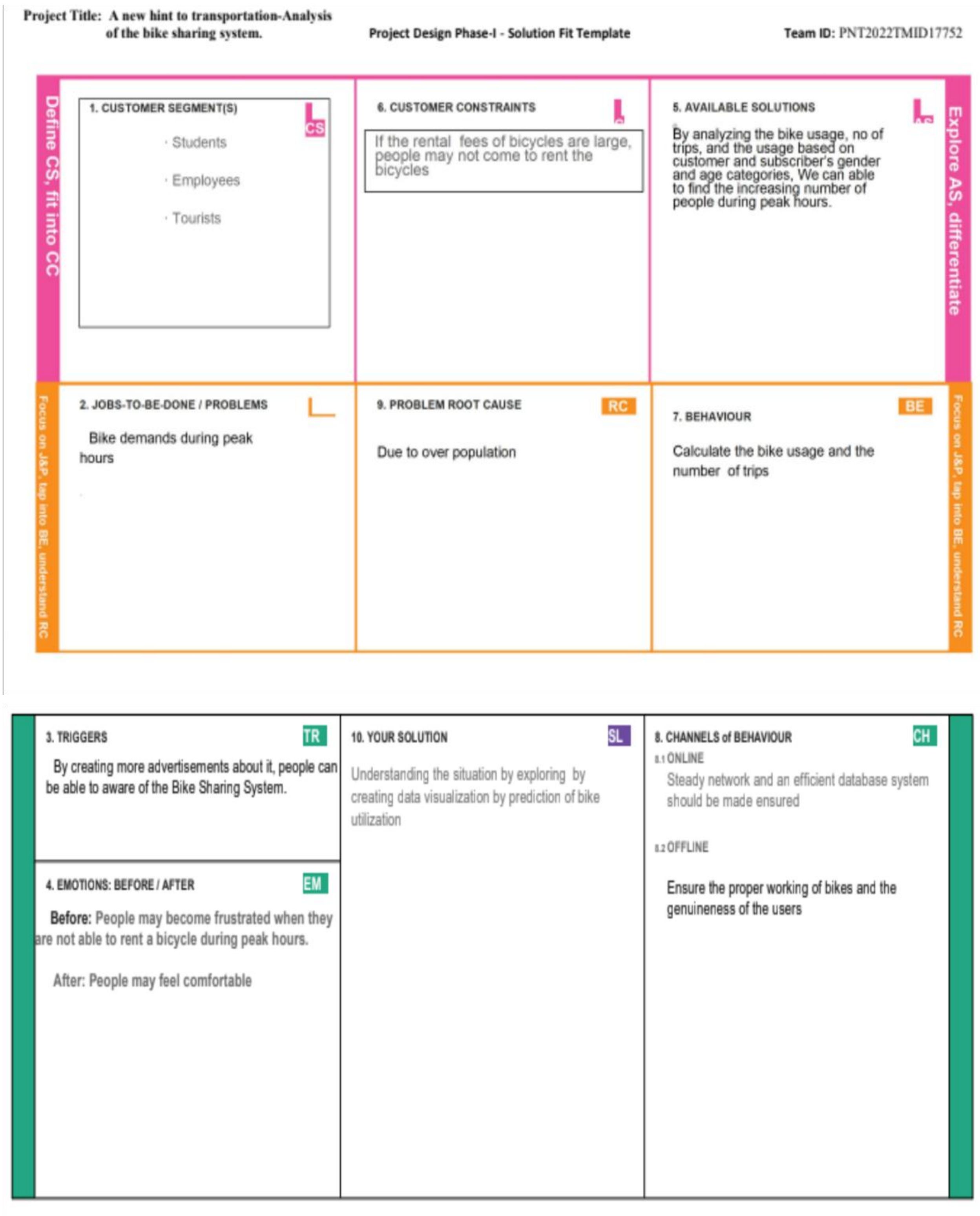


3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	People are aware of the system and effects during bad weather conditions, Bike demands during peak hours.
2.	Idea / Solution description	Analysing the bike usage, no of trips, and the usage based on customer and subscriber's gender and age categories
3.	Novelty / Uniqueness	Understanding ,Exploring by creating data visualization by prediction of bike utilization,demand.

4.	Social Impact / Customer Satisfaction	1. Reduced congestion and fuel consumption 2.Transport flexibility 3. Reductions to vehicle emissions 4. Health benefits 5. Financial savings for individuals.
5.	Business Model (Revenue Model)	Having an membership active pass makes the customer can rent bikes with amount packages based on time-constraints per weeks/days. Subscriber can rent bikes with amount packages based on time-constraints per month/year.
6.	Scalability of the Solution	can improve the productivity of citi-bike system, widespread of utilization according to the customer's demands.

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Bike usage analysis	Collect information of the number of bikes rented from the dataset
FR-2	Find number of trips taken	Find the number of trips taken by each customer.
FR-3	Find the popular times of travel	Find the popular times of travel by most common month, most common day of week, most common hour of the day
FR-4	Find the popular stations and trips	Find the popular stations and trips by most common start station, most common end station and most common trip from start to end.
FR-5	Visualize the data	Plotting the graphs and visualize the data
FR-6	Analyse the model	Use machine learning algorithms to analyse the system

4.2 Non-Functional requirements

Following are the non-functional requirements of the proposed solution.

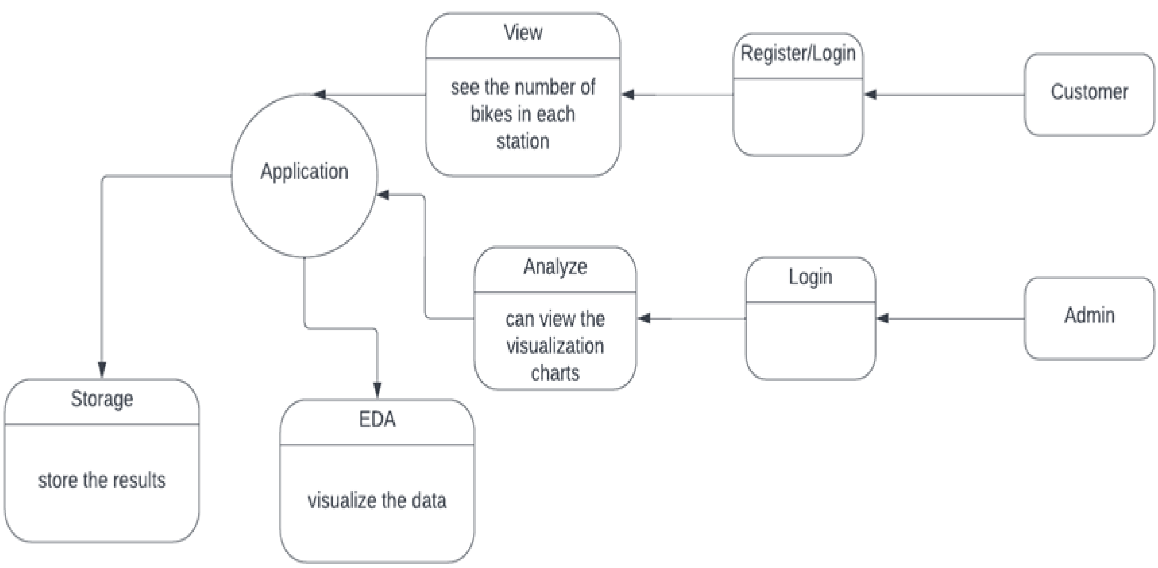
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Customer should be able to use the system at any time if he wants.
NFR-2	**Security**	The customer’s data should be kept in a secure manner.
NFR-3	**Reliability**	The system shall be completely operational for the full time.

NFR-4	**Performance**	The system should be able to support many simultaneous users.
NFR-5	**Availability**	The system should be available for 24/7 for customers without any interruption.
NFR-6	**Scalability**	The system can withstand the increase in the number of customers.

5. PROJECT DESIGN

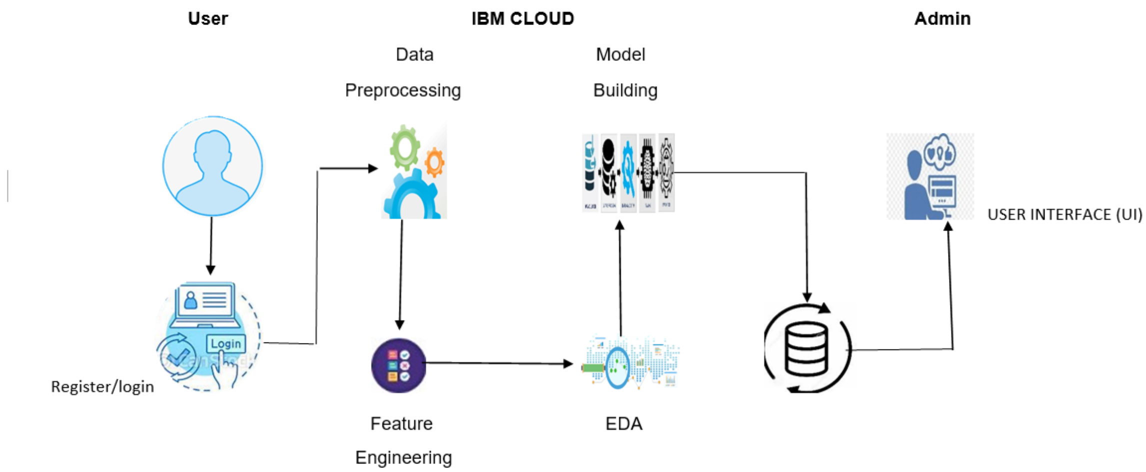
5.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



5.2 Solution & Technical Architecture

Technical Architecture:



5.3 User Stories

User Type	Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority
Administrator	Sprint-1	Data preparation	USN-1	As an analyst.I can extract the Citi-bike dataset for the year 2018	High
	Sprint-1		USN-2	As an analyst,I upload the dataset into cognos platform.	High
	Sprint-1	Data Cleaning	USN-3	As an analyst, I remove the null and duplicate values	High
	Sprint-1		USN-4	As an analyst, I identify patterns and relationships between the various attributes	High
Administrator	Sprint-2	Feature Engineering	USN-5	I made computations on the different attribute to find the new attribute value.	Medium

	Sprint-2		USN-6	I have dropped few attributes from the data set which are not needed.	Medium
	Sprint-2	Visualization	USN-7	As an analyst, I visualize the data and infer the knowledge in cognos platform.	High
Administrator	Sprint -3		USN-8	As an analyst, I made visualization charts of the data using python	Medium
	Sprint -3	Dashboard	USN-9	As an analyst, I create a dashboard with the created visualizations to supplement business insights during the decision-making process at Citi dataset.	High
	Sprint-4	Prediction	USN-10	To predict the most common user type ie customers and subscribers using various machine learning algorithms.	High
Customer	Sprint-4	Registration	USN-11	As a user, I can register and login in the application	High

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Data preparation	USN-1	As an analyst.I can extract the Citi-bike dataset for the year 2018	5	High	B.Mahasivapriya K.Lekhasri
Sprint-1		USN-2	As an analyst,I upload the dataset into cognos platform.	5	High	M.Gayathri S.Nishaa
Sprint-1	Data Cleaning	USN-3	As an analyst, I remove the null and duplicate values	4	High	B.Mahasivapriya K.Lekhasri
Sprint-1		USN-4	As an analyst, I identify patterns and relationships between the various attributes	5	High	M.Gayathri S.Nishaa
Sprint-2	Feature Engineering	USN-5	I made computations on the different attribute to find the new attribute value.	6	Medium	B.Mahasivapriya K.Lekhasri
Sprint-2		USN-6	I have dropped few attributes from the data set which are not needed.	6	Medium	M.Gayathri S.Nishaa
Sprint-2	Visualization	USN-7	As an analyst, I visualize the data and infer the knowledge in cognos platform.	8	High	B.Mahasivapriya K.Lekhasri
Sprint -3		USN-8	As an analyst, I made visualization charts of the data using python	8	Medium	M.Gayathri S.Nishaa

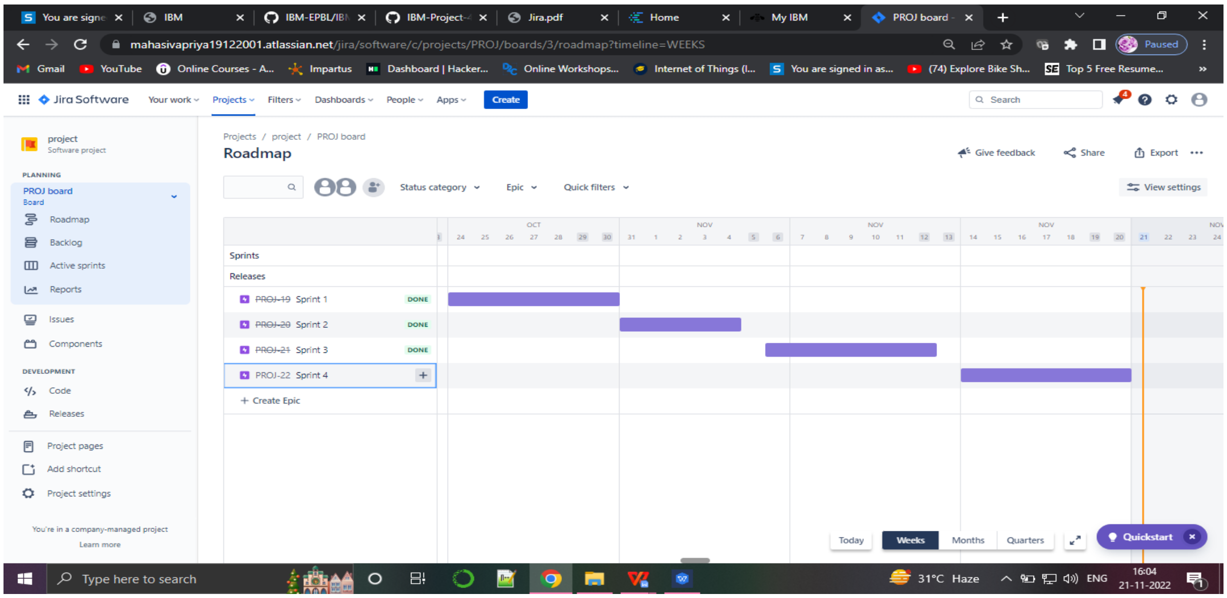
Sprint -3	Dashboard	USN-9	As an analyst, I create a dashboard with the created visualizations to supplement business insights during the decision-making process at Citi dataset.	7	High	B.Mahasivapriya K.Lekhasri
Sprint-4	Prediction	USN-10	To predict the most common user type ie customers and subscribers using various machine learning algorithms.	8	High	M.Gayathri S.Nishaa
Sprint-4	Registration	USN-11	As a user, I can register and login in the application	4	High	B.Mahasivapriya K.Lekhasri M.Gayathri S.Nishaa

6.2 Sprint Delivery Schedule

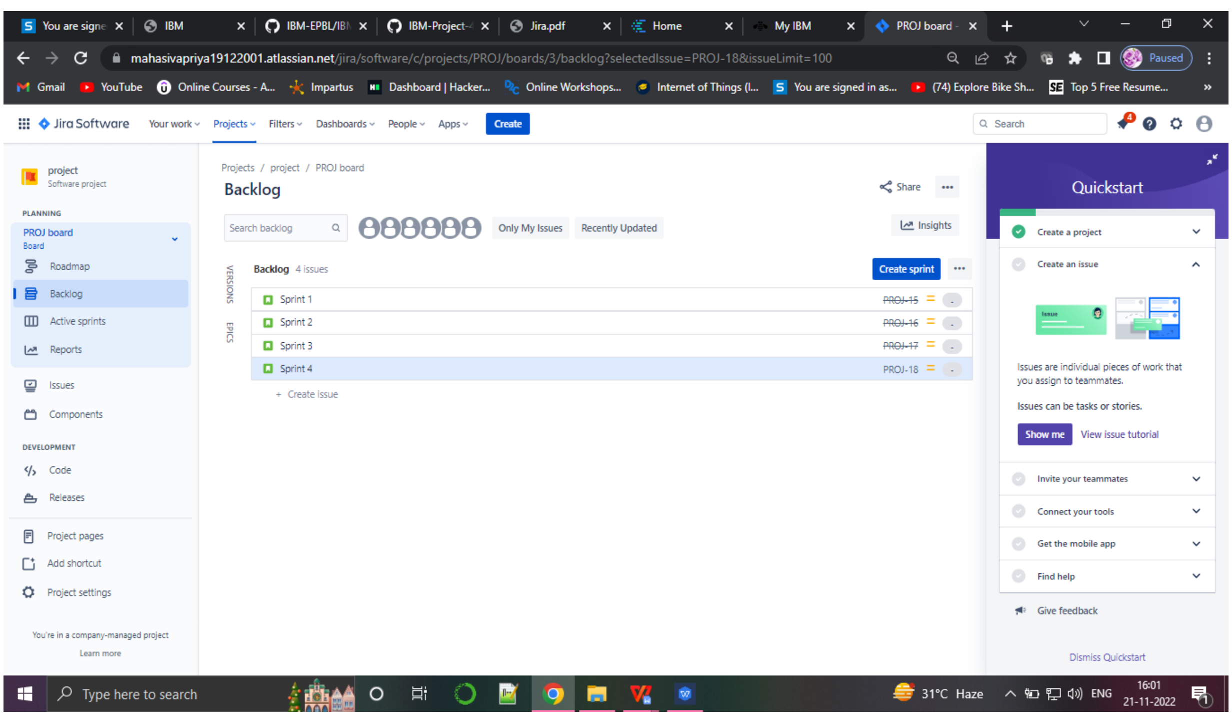
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	19	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	15	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	12	19 Nov 2022

6.3 Reports from JIRA

Roadmap



Backlog



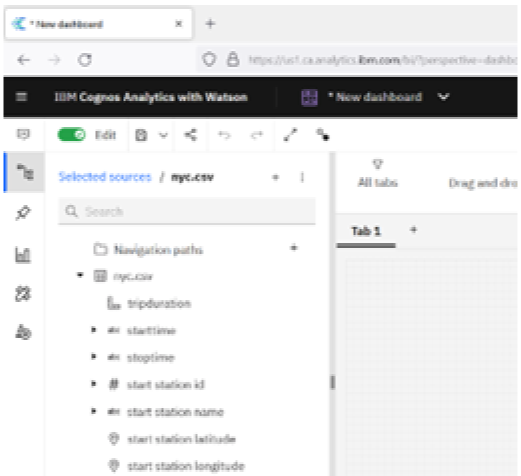
7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1

Data Preparation:

Dataset link : https://s3.amazonaws.com/tripdata/index.html

Uploading the dataset



Data Cleaning:

Finding the duplicates:

df = data[data.birthyear.notnull()]

df.duplicated().sum()

0 # There are no null values present in birthyear

```
birthyear
1980.0
1969.0
1975.0
1984.0
1994.0
```

Checking null values:

d=df.isna().sum()

d.head()

```
tripduration          0
starttime             0
stoptime              0
start station id     47
start station name   47
dtype: int64
```

Replace null values:

df.replace(np.nan,'-',inplace = True)

df.isnull().sum()

```
tripduration               0
starttime                  0
stoptime                   0
start station id           0
start station name         0
start station latitude     0
start station longitude    0
end station id             0
end station name           0
end station latitude       0
end station longitude      0
bikeid                     0
usertype                   0
birthyear                  0
gender                     0
tripduration_bins          0
Age                        0
dtype: int64
```

7.2 Feature 2

Feature Engineering:

calculating Age from birth year

from datetime import datetime, date

age=2018-df['birth_year']

df['Age']=age

df.head()

	tripduration	starttime	stoptime	start station id	start station name	start station latitude	start station longitude	end station id	end station name	end station latitude	end station longitude	bikeid	usertype	birth_year	gender	tripduration_bins	Age
0	11.583333	2013-06-01 00:00:01	2013-06-01 00:11:36	444	Broadway & W 24 St	40.742354	-73.989151	434.0	9 Ave & W 18 St	40.743174	-74.003664	19678	Subscriber	1983.0	1	(0.0, 30.0]	35.0
1	11.550000	2013-06-01 00:00:08	2013-06-01 00:11:41	444	Broadway & W 24 St	40.742354	-73.989151	434.0	9 Ave & W 18 St	40.743174	-74.003664	16649	Subscriber	1984.0	1	(0.0, 30.0]	34.0
3	2.050000	2013-06-01 00:01:04	2013-06-01 00:03:07	475	E 15 St & Irving Pl	40.735243	-73.987586	262.0	Washington Park	40.691782	-73.973730	16352	Subscriber	1960.0	1	(0.0, 30.0]	58.0
4	25.350000	2013-06-01 00:01:22	2013-06-01 00:26:43	2008	Little West St & 1 Pl	40.705693	-74.016777	310.0	State St & Smith St	40.689269	-73.989129	15567	Subscriber	1983.0	1	(0.0, 30.0]	35.0
6	34.283333	2013-06-01 00:02:33	2013-06-01 00:36:50	285	Broadway & E 14 St	40.734546	-73.990741	532.0	S 5 Pl & S 5 St	40.710451	-73.960876	15693	Subscriber	1991.0	1	(30.0, 60.0]	27.0

calculating age group from age

max_limit = df['Age'].max()

max_limit

bins = [0,20,40,60,max_limit]

agegroup = pd.cut(df['Age'], bins=bins).value_counts()

Agegroup

```
(20.0, 40.0]     161563
(40.0, 60.0]     148805
(60.0, 119.0]     27014
(0.0, 20.0]           0
Name: Age, dtype: int64
```

calculating hour

peak_hour['Start Date'] = pd.to_datetime(df['starttime'])

peak_hour['Stop Date'] = pd.to_datetime(df['stoptime'])

peak_hour['year'] =peak_hour["Start Date"].dt.year

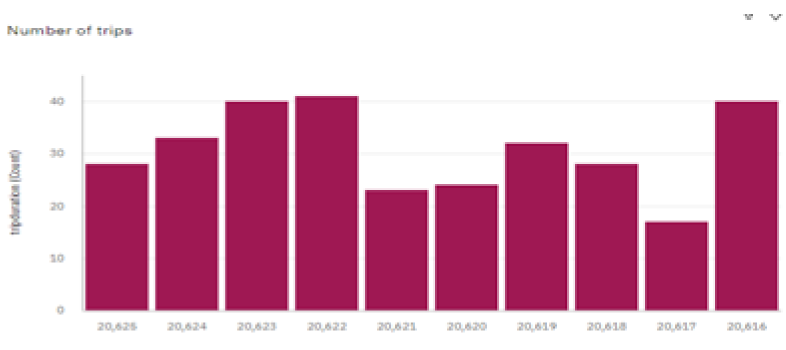
peak_hour["Hour"] = peak_hour["Start Date"].dt.hour

	Start Date	Stop Date	year	Hour	bikeid
0	2013-06-01 00:00:01	2013-06-01 00:11:36	2013	0	19678
1	2013-06-01 00:00:08	2013-06-01 00:11:41	2013	0	16649
3	2013-06-01 00:01:04	2013-06-01 00:03:07	2013	0	16352
4	2013-06-01 00:01:22	2013-06-01 00:26:43	2013	0	15567
6	2013-06-01 00:02:33	2013-06-01 00:36:50	2013	0	15693
...	...	...	...	...	...
577687	2013-06-30 23:58:09	2013-07-01 00:05:25	2013	23	19454
577689	2013-06-30 23:57:52	2013-07-01 00:00:57	2013	23	16746
577690	2013-06-30 23:58:39	2013-07-01 00:08:34	2013	23	19290
577698	2013-06-30 23:59:27	2013-07-01 00:14:52	2013	23	15250
577700	2013-06-30 23:59:33	2013-07-01 00:02:14	2013	23	18910

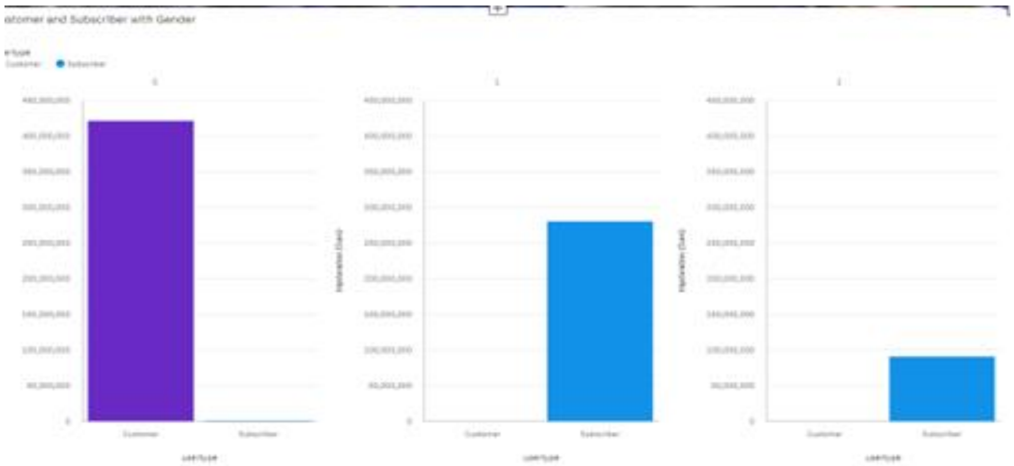
337382 rows × 5 columns

Visualization of the dataset in COGNOS Platform:

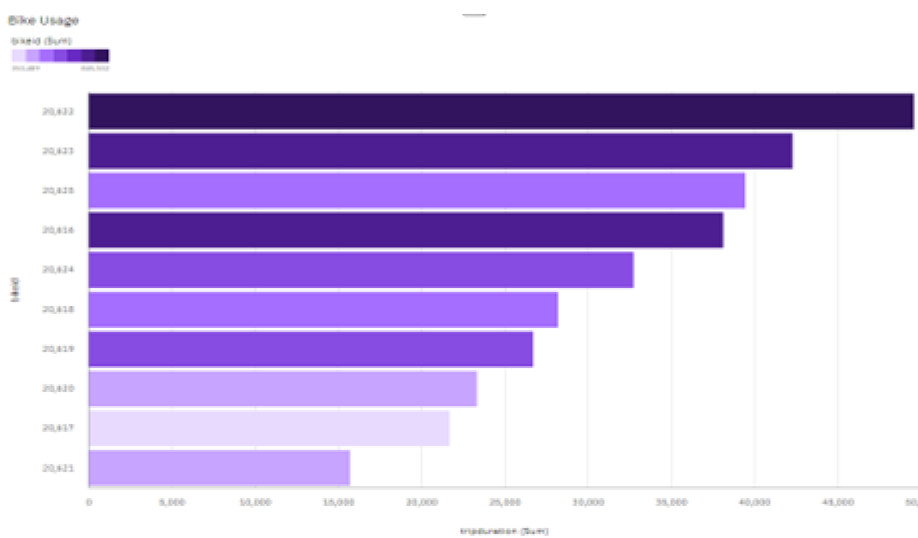
Finding the number of trips per each bike:



Finding the percentage of customers and subscribers



Bike Usage - Bike Id Vs Trip Duration:



Age Group Differentiation by BikeId:

<u>Calculation:</u>

if(age<=20) then

('<20')

else if(age>=21 and age<=30) then

('21-30')

else if(age>=31 and age<=40) then

('31-40')

else if(age>=41 and age<=55) then

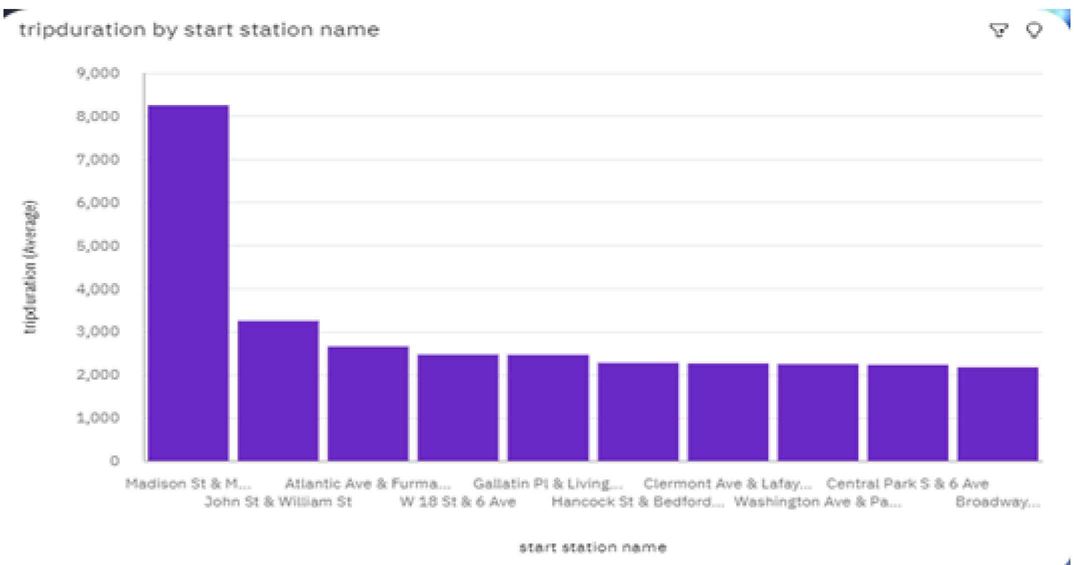
('41-55')

else('>55')

bikeid and Age_Group

Age_Group	bikeid
21-30	5,721
31-40	5,749
41=55	5,741
<20	1,625
>55	5,781
Summary	5,794

Finding the top 10 start stations with customer age group:



7.3 Sprint 3

Creating a dashboard including all the visualizations created in the cognos platform:

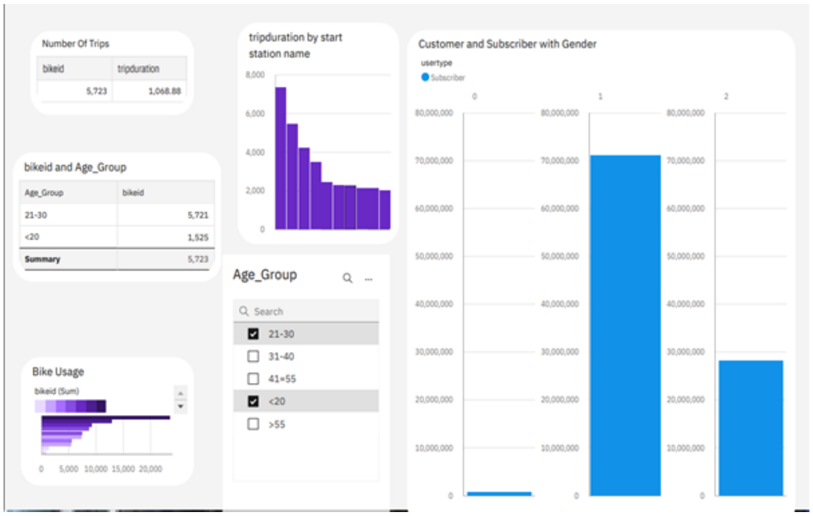
This dashboard has the charts including

- **i)** **Number of trips**
- **ii)** **Customer and Subscriber percentage with gender**

iii) **Bike Usage**

iv) **BikeId and Age Group**

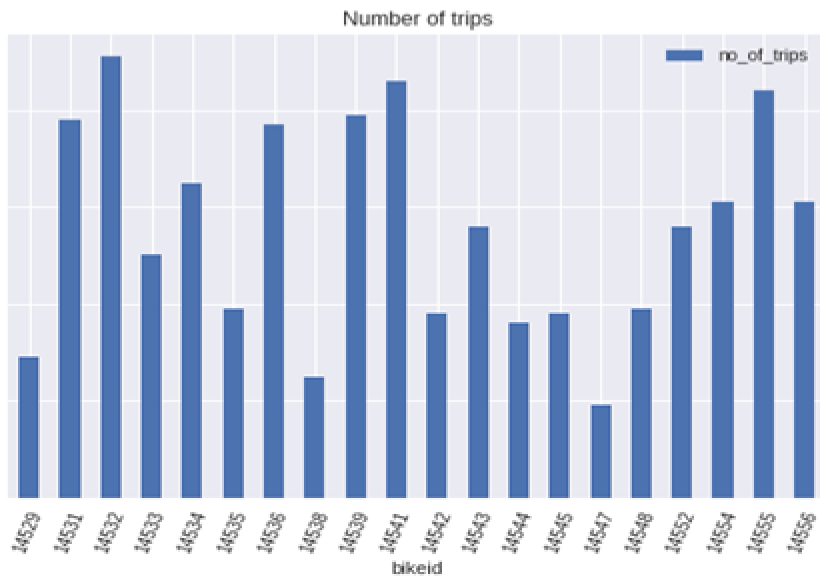
v) **Trip duration by start station name**



Visualization Charts using Python:

Finding the number of trips per bike:

```
trips = pd.DataFrame() #creating a dataframe

trips['no_of_trips'] = df.groupby("bikeid")["bikeid"].count() #finding the number of trips by each bike

trips['avg_duration'] = df.groupby("bikeid")["tripduration"].mean() #avg duration of the trips

trips_graph=trips.head(20)

trips_graph.plot.bar(x="bikeid", y="no_of_trips", rot=70, title="Number of trips")
```

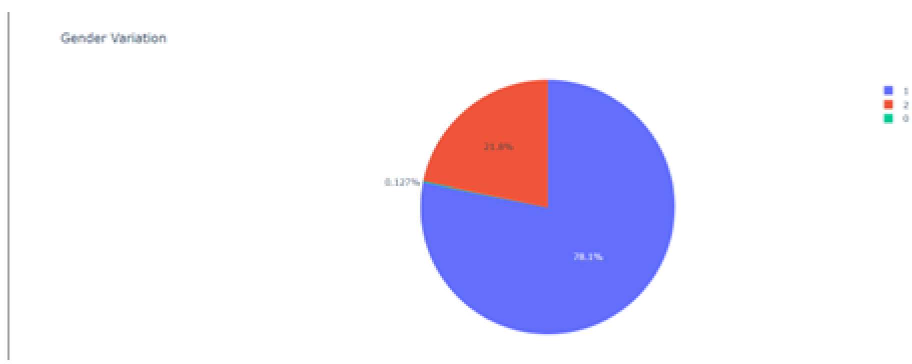


Gender Variation:

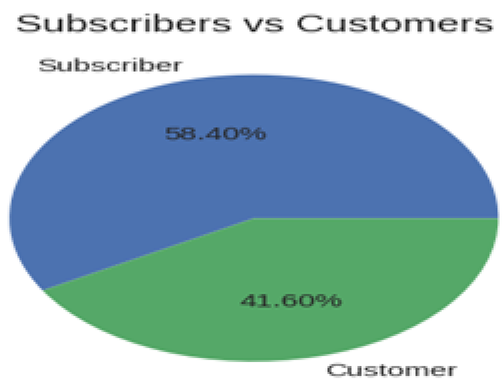
plt.pie(values = df_bike['Gender'].value_counts(),

names =df_bike['Gender'].value_counts().index,

title ="Gender Variation")

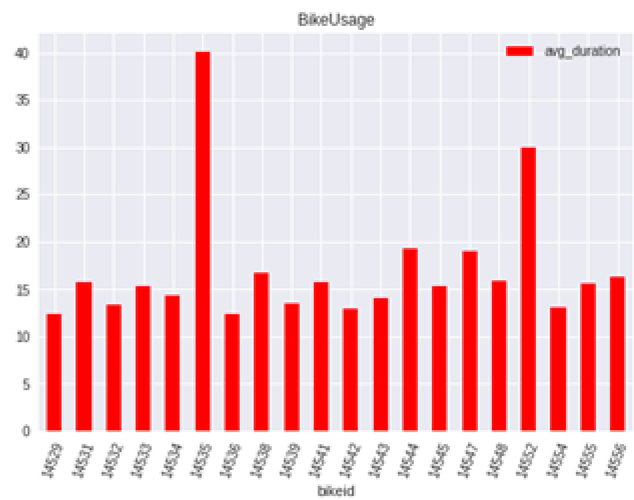


Percentage of Subscribers and Customers:



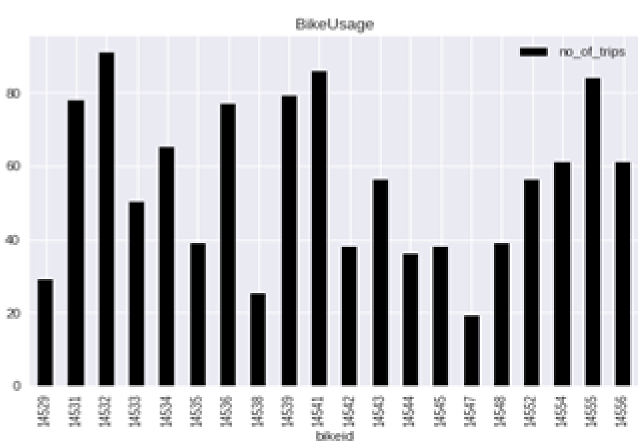
Bike Usage Based on Average Duration:

trips_graph.plot.bar(x="bikeid", y="avg_duration", rot=70, title="BikeUsage",color="red")



Bike Usage Based on No of Trips:

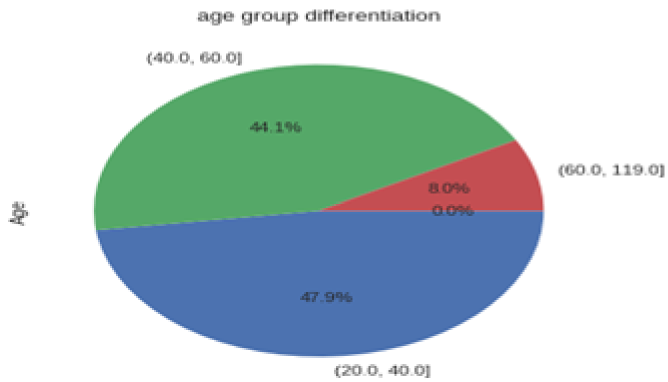
trips_graph.plot.bar(x="bikeid", y="no_of_trips", rot=90, title="BikeUsage",color="black")



Age Group Differentiation:

agegroup = pd.cut(df['Age'], bins=bins).value_counts()

agegroup.plot.pie(autopct="%.1f%%",title='age group differentiation',counterclock=False);



Top 10 Start Station:

most=pd.DataFrame()

most_graph=pd.DataFrame()

most['name']=df["start station name"].value_counts().index

most['count']=df["start station name"].value_counts().values

most_graph=most.head(15)

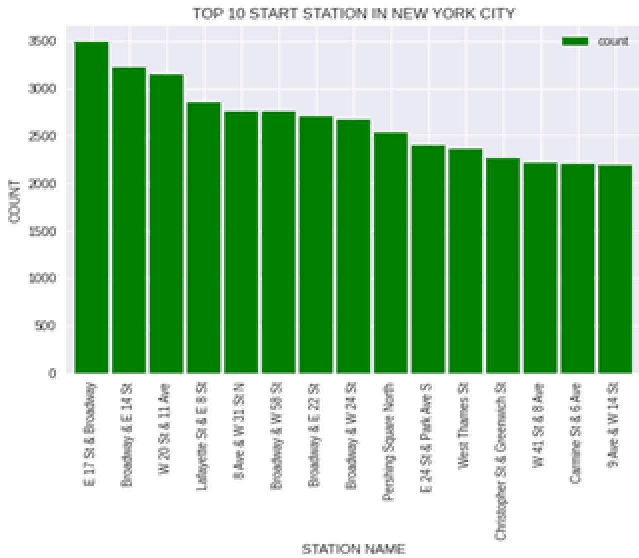
most_graph.plot.bar(x="name", y="count", width=0.9,rot=90, title="BikeUsage",color="green")

plt.xlabel("STATION NAME")

plt.ylabel("COUNT")

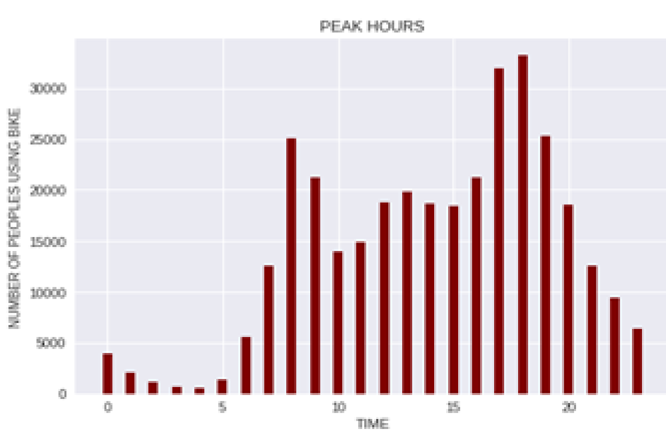
plt.title("TOP 10 START STATION IN NEW YORK CITY")

plt.show()



Finding the Peak Hours of Travel:

```
ind=peak_hour["Hour"].value_counts().index

y=peak_hour["Hour"].value_counts().values

plt.bar(ind, y, color ='maroon', width = 0.4)

plt.xlabel("TIME")

plt.ylabel("NUMBER OF PEOPLES USING BIKE")

plt.title("PEAK HOURS")

plt.show()
```



Bike Trend for the month June:

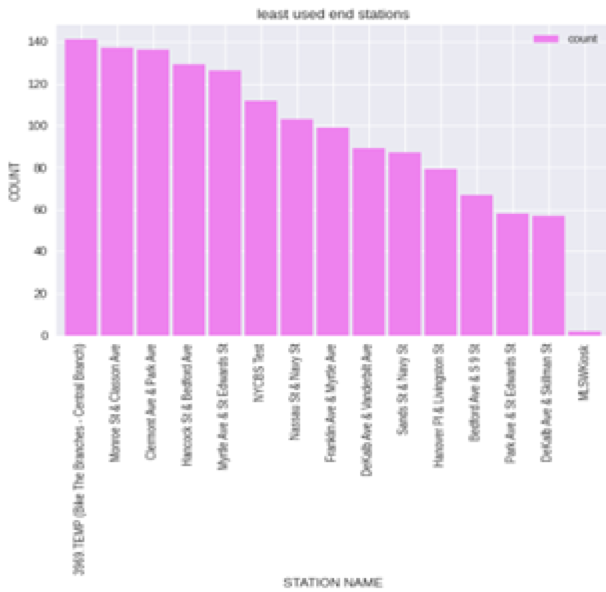
```
#converting string to datetime object

df['starttime']= pd.to_datetime(df['starttime'])

 #since we are dealing with single month, we grouping by days

#using count aggregation to get number of occurances i.e, total trips per day

start_time_count = df.set_index('starttime').groupby(pd.Grouper(freq='D')).count()

 #we have data from July month for only one day which is at last row, lets drop it

start_time_count.drop(start_time_count.tail(1).index, axis=0, inplace=True)

 #again grouping by day and aggregating with sum to get total trip duration per day

#which will used while plotting

trip_duration_count = df.set_index('starttime').groupby(pd.Grouper(freq='D')).sum()

 #again dropping the last row for same reason

trip_duration_count.drop(trip_duration_count.tail(1).index, axis=0, inplace=True)


#plotting total rides per day

#using start station id to get the count

fig,ax=plt.subplots(figsize=(25,10))

ax.bar(start_time_count.index, 'start station id', data=start_time_count, label='Total riders')

#bbox_to_anchor is to position the legend box

ax.legend(loc ="lower left", bbox_to_anchor=(0.01, 0.89), fontsize='20')

ax.set_xlabel('Days of the month June 2018', fontsize=30)

ax.set_ylabel('Riders',  fontsize=40)

ax.set_title('Bikers trend for the month June', fontsize=50)

 #creating twin x axis to plot line chart is same figure

ax2=ax.twinx()

#plotting total trip duration of all user per day

ax2.plot('tripduration', data=trip_duration_count, color='y', label='Total trip duration', marker='o',
linewidth=5, markersize=12)
```

```
ax2.set_ylabel('Time duration',  fontsize=40)
ax2.legend(loc ="upper left", bbox_to_anchor=(0.01, 0.9), fontsize='20')
 ax.set_xticks(trip_duration_count.index)
ax.set_xticklabels([i for i in range(1,31)])
 #tweeking x and y ticks labels of axes1
ax.tick_params(labelsize=30, labelcolor='#eb4034')
#tweeking x and y ticks labels of axes2
ax2.tick_params(labelsize=30, labelcolor='#eb4034')
 plt.show()
```

Least Used End Stations:

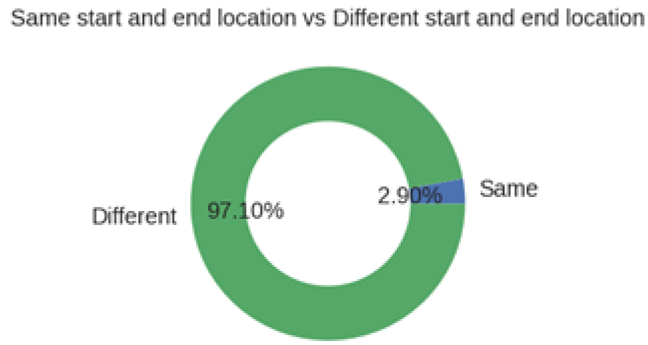
```
least=pd.DataFrame()
least_graph=pd.DataFrame()
least['name']=df["end station name"].value_counts().index
least['count']=df["end station name"].value_counts().values
least_graph=most.tail(15)
least_graph
least_graph.plot.bar(x="name", y="count", width=0.9,rot=90, title="BikeUsage",color="violet")
plt.xlabel("STATION NAME")
plt.ylabel("COUNT")
```

```
plt.title("least used end stations")

plt.show()
```



Same start and end location Vs Different start and end location:

```
#number of trips that started and ended at same station

start_end_same = df[df['start station name'] == df['end station name']].shape[0]

#number of trips that started and ended at different station

start_end_diff = df.shape[0]-start_end_same

fig,ax=plt.subplots()

ax.pie([start_end_same,start_end_diff], labels=['Same', 'Different'], autopct='%1.2f%%', textprops={'fontsize': 20})

ax.set_title('Same start and end location vs Different start and end location', fontsize=20)

circle = Circle((0,0), 0.6, facecolor='white')

ax.add_artist(circle)

plt.show()
```



8. TESTING

8.1 Test Cases

- ❖ Verify that design and dimension of the application are as per the specifications.
- ❖ Verify that the different colors used in the bike are of the correct shades as per the specifications.
- ❖ Verify that the weight of the bike is as per the specifications.
- ❖ Check the material used in different parts of the bike – outer body, tires, seat, etc.

8.2 User Acceptance Testing

Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved.

Resolution	**Severity 1**	**Severity 2**	**Severity 3**	**Severity 4**	**Subtotal**
By Design	10	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6

Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2
Outsource Shipping	3	0	0	3

9. RESULTS

9.1 Performance Metrics

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Customer should be able to use the system at any time if he wants.
NFR-2	**Security**	The customer's data should be kept in a secure manner.
NFR-3	**Reliability**	The system shall be completely operational for the full time.
NFR-4	**Performance**	The system should be able to support many simultaneous users.
NFR-5	**Availability**	The system should be available for 24/7 for customers without any interruption.
NFR-6	**Scalability**	The system can withstand the increase in the number of customers.

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- ❖ Avoids Data Redunancy and Inconsistency.
- ❖ Conveys Data in Interactive Visualizations.
- ❖ It helps Organisations or Government to understand the works, usages and trends of NYC Bike's.
- ❖ Analysis of mobility patterns provides evidence that Bike Sharing, and Cycling in general can provide a flexible and eco-friendly mode of transportation for shorter trips.
- ❖ It shows the improvement also of protected infrastructure like new footpaths and temporary and permanent Bike Lanes.

DISADVANTAGES:

- ❖ Machine errors are unavoidable when occured.(Hardware failure, Network failure, Others).
- ❖ Did not mention of the combined POI data for the better clustering.

11. CONCLUSION

Our Project uses IBM Cognos Analytics to do Explonatory Data Analysis and Creating Dashboard, Story and Report. We use Google colab to do Dataset Cleaning Processing, Feature Engineering, EDA, Visualization and create a website using HTML, Flask, Python, Matplotlib, Numpy, Pandas, Seaborn and other libraries.

12. FUTURE SCOPE

- We can analyze which station needs more bikes and any area needs new station to be installed.
- Sensitivity analysis should be developed in further analysis in order to see the influence of shortening of traffic areas dimensions on the 'Bike-Sharing' Systems use proportion.
- Having detailed hourly data would help create to produce a impressive stats and analysis.
- There exists a lot of scope in this research area.

13. APPENDIX

Source Code

Bike Availability Prediction:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
df = pd.read_csv(r'C:\Users\DELL\Desktop\tripdata\merged-tripdata.csv',\
        usecols=['starttime','start station id',\
                'stoptime','end station id'],\
        parse_dates=['starttime','stoptime'])
df.info()
df1 = pd.read_csv(r'C:\Users\DELL\Desktop\tripdata\merged-tripdata.csv',\
        usecols=['starttime','start station id',\
                'stoptime','end station id','bikeid'],\
        parse_dates=['starttime','stoptime'])
df1.info()
dfbike=df1.sort_values(by=['bikeid','starttime'])
dfbike.head(10)
offset = pd.DataFrame({'starttime': pd.to_datetime('2010-09-01'),\
  'start station id':0,'stoptime': pd.to_datetime('2010-09-01'),\
  'end station id':0,'bikeid':0},index=[0])
```

```
offset
dfbike1 = pd.concat([offset,dfbike]).reset_index(drop=True)
dfbike2 = pd.concat([dfbike,offset]).reset_index(drop=True)
dfbike=pd.concat ([dfbike1[['bikeid','stoptime','end station id']]\
          ,dfbike2[['bikeid','starttime','start station id']] ],\
           axis=1 )
dfbike.head()
dfbike.columns=['bikeid1','starttime','start station id',\
             'bikeid2','stoptime','end station id']
dfrebal = dfbike[['starttime','start station id',\
               'stoptime','end station id']].\
         loc[(dfbike.bikeid1==dfbike.bikeid2) & \
        (dfbike['start station id'] != dfbike['end station id']) ]
dfrebal.reset_index(drop=True, inplace=True)
dfrebal
df = pd.concat([df,dfrebal])
df.reset_index(drop=True, inplace=True)
df
dfs=df[['starttime','start station id']].assign(act=-1)
dfe=df[['stoptime','end station id']].assign(act=1)
dfs.columns=['docktime','stationid','act']
dfe.columns=['docktime','stationid','act']
dfse=pd.concat([dfs,dfe])
dfse.sort_values(by=['docktime'], inplace=True)
dfse.reset_index(drop=True, inplace=True)
dfse.head(100)
dfstations = \
  pd.read_csv(r'C:\Users\DELL\Desktop\tripdata\merged-tripdata.csv',\
  usecols=['start station id','start station name']).\
  drop_duplicates()
dfstations.columns=['stationid','station name']
dfstations.set_index('stationid',drop=True, inplace=True)
dfstations
def availabilty (station,hr,time):
```

```
# inputs: station name, day
# requires: dfstations, dfse
sid = dfstations.loc[dfstations['station name']==station]\
    .index[0] # lookup station id
#print(sid)
dfstation = dfse.loc[(dfse.stationid==sid) ]
#print(dfstation)
dfstation.reset_index(drop=True, inplace=True)
dfstation = dfstation.assign(cnt = dfstation.act.cumsum())
dfstation.at[0, 'act'] =+ abs(dfstation.act.cumsum().min())
dfstation = dfstation.assign(cnt = dfstation.act.cumsum())
#print(dfstation)
dt=time+"."+hr
flag=0
c=4
for ind in dfstation.index:
    if dfstation['docktime'][ind]==dt:
        c=dfstation['cnt'][ind]
        flag=1
if flag==0:
    m=int(time[0:2])
    s=int(time[3:5])
    while int(s)<60:
        s=int(s)+1
        if len(str(m))<2:
            m="0"+str(m)
        if len(str(s))<2:
            s="0"+str(s)
        time=str(m)+":"+str(s)
        dt=time+"."+hr
        for ind in dfstation.index:
            if dfstation['docktime'][ind]==dt:
                c=dfstation['cnt'][ind]
if c>0:
```

```
        return 1
      else:
        return 0
  #return dfstation
 hr=input("Enter the hour\n")
 time=input("Enter the time like 00:00\n")
 s_name=input("Enter the station name\n")
 ans=availabilty(s_name,hr,time)
 if ans==1:
   print("Available")
 else:
   print("Not Available")
```

Predicting most common usertype using decision tree:

```
import pandas as pd
df=pd.read_csv("merged-data.csv",error_bad_lines=False,engine="python")
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:3326: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version.


  exec(code_obj, self.user_global_ns, self.user_ns)
Skipping line 430585: unexpected end of data
```

```
df2=pd.DataFrame()
df2['usertype']=df['usertype']
df2.head()
```

	usertype
0	Customer
1	Customer
2	Subscriber
3	Customer
4	Subscriber

```
df3=pd.DataFrame()
df3['start_time']=pd.to_datetime(df['starttime'],errors='coerce',format='%Y-%m-%d %H:%M:%S')
df3['stop_time']=pd.to_datetime(df['stoptime'],errors='coerce',format='%Y-%m-%d %H:%M:%S')
df['startdate'] = pd.to_datetime(df3['start_time'])
df['stopdate'] = pd.to_datetime(df3['stop_time'])
df.head()
```

```
df.head()
```

	tripduration	starttime	stoptime	start station id	start station name	start station latitude	start station longitude	end station id	end station name	end station latitude	end station longitude	bikeid	usertype	birth year	gender	startdate	stopda
0	634	2013-07-01 00:00:00	2013-07-01 00:10:34	164	E 47 St & 2 Ave	40.753231	-73.970325	504	1 Ave & E 15 St	40.732219	-73.981656	16950	Customer	\N	0	2013-07-01 00:00:00	2013-07- 00:10:
1	1547	2013-07-01 00:00:02	2013-07-01 00:25:49	388	W 26 St & 10 Ave	40.749718	-74.002950	459	W 20 St & 11 Ave	40.746745	-74.007756	19816	Customer	\N	0	2013-07-01 00:00:02	2013-07- 00:25:
2	178	2013-07-01 00:01:04	2013-07-01 00:04:02	293	Lafayette St & E 8 St	40.730287	-73.990765	237	E 11 St & 2 Ave	40.730473	-73.986724	14548	Subscriber	1980	2	2013-07-01 00:01:04	2013-07- 00:04:
3	1580	2013-07-01 00:01:06	2013-07-01 00:27:26	531	Forsyth St & Broome St	40.718939	-73.992663	499	Broadway & W 60 St	40.769155	-73.981918	16063	Customer	\N	0	2013-07-01 00:01:06	2013-07- 00:27:
4	757	2013-07-01 00:01:10	2013-07-01 00:13:47	382	University Pl & E 14 St	40.734927	-73.992005	410	Suffolk St & Stanton St	40.720664	-73.985180	19213	Subscriber	1986	1	2013-07-01 00:01:10	2013-07- 00:13:

```
df['startmonth'] = df['startdate'].dt.month
df['startday'] = df['startdate'].dt.day
df['startminute'] = df['startdate'].dt.minute
df['startweek'] = df['startdate'].dt.minute
df['startweekday'] = df['startdate'].dt.minute
df['startweek'] = df['startdate'].dt.week
df['startweekofyear'] = df['startdate'].dt.weekofyear
df['startweekday'] = df['startdate'].dt.weekday
df['startdayofyear'] = df['startdate'].dt.dayofyear

df['startquarter'] = df['startdate'].dt.quarter
df['stopmonth'] = df['stopdate'].dt.month
df['stopday'] = df['startdate'].dt.day
df['stopminute'] = df['stopdate'].dt.minute
df['stopweek'] = df['stopdate'].dt.minute
df['stopweekday'] = df['stopdate'].dt.minute
df['stopdayofyear'] = df['stopdate'].dt.dayofyear
df['stopquarter'] = df['stopdate'].dt.quarter
df.head()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().week instead.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().week instead.
  import sys
```

```
df.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please us
e Series.dt.isocalendar().week instead.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please us
e Series.dt.isocalendar().week instead.
  import sys
```

Out[62]:

start station longitude	end station id	end station name	end station latitude	...	startweekofyear	startdayofyear	startquarter	stopmonth	stopday	stopminute	stopweek	stopweekday	stopdayofyear	stopquarter
-73.970325	504	1 Ave & E 15 St	40.732219	...	27	182	3	7	1	10	10	10	182	3
-74.002950	459	W 20 St & 11 Ave	40.746745	...	27	182	3	7	1	25	25	25	182	3
-73.990765	237	E 11 St & 2 Ave	40.730473	...	27	182	3	7	1	4	4	4	182	3
-73.992663	499	Broadway & W 60 St	40.769155	...	27	182	3	7	1	27	27	27	182	3
-73.992005	410	Suffolk St & Stanton St	40.720664	...	27	182	3	7	1	13	13	13	182	3

```
df['stopweek'] = df['stopdate'].dt.isocalendar().week
df['stopweekofyear'] = df['stopdate'].dt.weekofyear
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Series.dt.isocalendar().week instead.
  """Entry point for launching an IPython kernel.

data=df.dropna()
data.isna().sum()
tripduration               0
starttime                  0
stoptime                   0
start station id           0
start station name         0
start station latitude     0
start station longitude    0
end station id             0
end station name           0
end station latitude       0
end station longitude      0
bikeid                     0
usertype                   0
birth year                 0
gender                     0
startdate                  0
stopdate                   0
startmonth                 0
startday                   0
startminute                0
startweek                  0
startweekday               0
startweekofyear            0
startdayofyear             0
startquarter               0
stopmonth                  0
stopday                    0
```

```
stopminute           0
stopweek             0
stopweekday          0
stopdayofyear        0
stopquarter          0
stopweekofyear       0
dtype: int64
data1=data.drop(['start station name'],axis=1)
data1.head()
data1=data1.drop(['end station name'],axis=1)
data1.head()
data1=data1.drop(['starttime'],axis=1)
data1=data1.drop(['stoptime'],axis=1)
data1['startmonth'] = data1['startmonth'].astype(int)
data1['startday'] = data1['startday'].astype(int)
data1['startminute'] = data1['startminute'].astype(int)
data1['startweek'] = data1['startweek'].astype(int)
data1['startweekday'] = data1['startweekday'].astype(int)
data1['startweekofyear'] = data1['startweekofyear'].astype(int)
data1['startdayofyear'] = data1['startdayofyear'].astype(int)
data1['startquarter'] = data1['startquarter'].astype(int)
data1['stopmonth'] = data1['stopmonth'].astype(int)
data1['stopday'] = data1['stopday'].astype(int)
data1['stopminute'] = data1['stopminute'].astype(int)
data1['stopweek'] = data1['stopweek'].astype(int)
data1['stopweekday'] = data1['stopweekday'].astype(int)
data1['stopweekofyear'] = data1['stopweekofyear'].astype(int)
data1['stopdayofyear'] = data1['stopdayofyear'].astype(int)
data1['stopquarter'] = data1['stopquarter'].astype(int)
display(data1.dtypes)
tripduration                int64
start station id            int64
start station latitude      float64
start station longitude     float64
end station id              int64
end station latitude        float64
end station longitude       float64
bikeid                      int64
usertype                    object
birth year                  object
gender                      int64
startdate          datetime64[ns]
stopdate           datetime64[ns]
startmonth                  int64
startday                    int64
startminute                 int64
startweek                   int64
startweekday                int64
startweekofyear             int64
startdayofyear              int64
startquarter                int64
```

```
stopmonth                      int64
stopday                       int64
stopminute                     int64
stopweek                      int64
stopweekday                     int64
stopdayofyear                   int64
stopquarter                    int64
stopweekofyear                   int64
dtype: object


display(data1.dtypes)
tripduration                   int64
start station id               int64
start station latitude         float64
start station longitude          float64
end station id                  int64
end station latitude            float64
end station longitude            float64
bikeid                        int64
usertype                      object
birth year                    object
gender                        int64
startdate              datetime64[ns]
stopdate               datetime64[ns]
startmonth                     int64
startday                      int64
startminute                    int64
startweek                      int64
startweekday                    int64
startweekofyear                  int64
startdayofyear                  int64
startquarter                   int64
stopmonth                      int64
stopday                       int64
stopminute                     int64
stopweek                      int64
stopweekday                     int64
stopdayofyear                   int64
stopquarter                    int64
stopweekofyear                   int64
dtype: object
df['startdate'] = df['startdate'].astype(str)
data1=data1.drop(['startdate'],axis=1)
data1=data1.drop(['stopdate'],axis=1)
data1.dtypes
tripduration             int64
start station id         int64
start station latitude     float64
start station longitude    float64
end station id           int64
```

```
end station latitude       float64
end station longitude      float64
bikeid                     int64
usertype                   object
birth year                 object
gender                     int64
startmonth                 int64
startday                   int64
startminute                int64
startweek                  int64
startweekday               int64
startweekofyear            int64
startdayofyear             int64
startquarter               int64
stopmonth                  int64
stopday                    int64
stopminute                 int64
stopweek                   int64
stopweekday                int64
stopdayofyear              int64
stopquarter                int64
stopweekofyear             int64
dtype: object
df3=df2.replace('Subscriber',1)
df3=df3.replace('Customer',0)
df3.head()
```

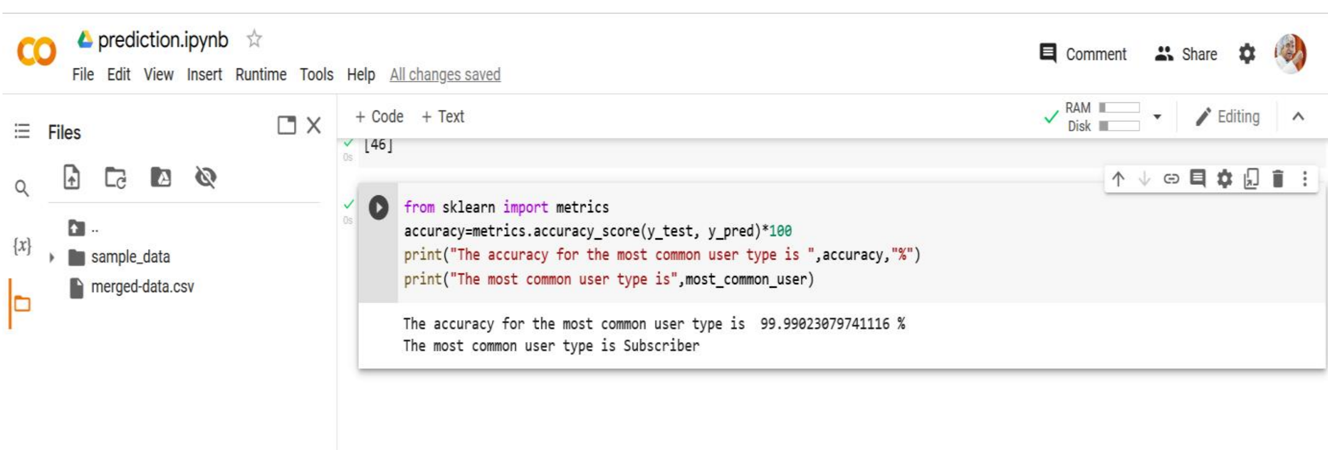
	usertype
0	0
1	0
2	1
3	0
4	1

```
data1=data1.drop(['birth year'],axis=1)
df3['usertype'] = df3['usertype'].astype(int)
data1=data1.drop(['usertype'],axis=1)
data1.dtypes
tripduration               int64
start station id           int64
start station latitude     float64
start station longitude    float64
end station id             int64
end station latitude       float64
end station longitude      float64
bikeid                     int64
gender                     int64
startmonth                 int64
startday                   int64
startminute                int64
```

```
startweek                int64
startweekday             int64
startweekofyear          int64
startdayofyear           int64
startquarter             int64
stopmonth                int64
stopday                  int64
stopminute               int64
stopweek                 int64
stopweekday              int64
stopdayofyear            int64
stopquarter              int64
stopweekofyear           int64
dtype: object
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data1, df3, test_size=0.20)
X_train.dtypes
tripduration             int64
start station id         int64
start station latitude     float64
start station longitude    float64
end station id           int64
end station latitude       float64
end station longitude      float64
bikeid                   int64
gender                   int64
startmonth               int64
startday                 int64
startminute              int64
startweek                int64
startweekday             int64
startweekofyear          int64
startdayofyear           int64
startquarter             int64
stopmonth                int64
stopday                  int64
stopminute               int64
stopweek                 int64
stopweekday              int64
stopdayofyear            int64
stopquarter              int64
stopweekofyear           int64
dtype: object
y_train.dtypes
usertype    int64
dtype: object
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
DecisionTreeClassifier()
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
[[19720    11]
 [    9 66377]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     19731
           1       1.00      1.00      1.00     66386

    accuracy                           1.00     86117
   macro avg       1.00      1.00      1.00     86117
weighted avg       1.00      1.00      1.00     86117

from numpy import average
avg=average(y_pred)
if((avg*100)>50):
  most_common_user='Subscriber'
else:
  most_common_user='Customer'
print(df3['usertype'].value_counts()[0])
98436
print(df3['usertype'].value_counts()[1])
332147
df3['usertype'].count()
430583
from sklearn import metrics
accuracy=metrics.accuracy_score(y_test, y_pred)*100
print("The accuracy for the most common user type is ",accuracy,"%")
print("The most common user type is",most_common_user)
The accuracy for the most common user type is  99.99023079741116 %
The most common user type is Subscriber
```

Output:



GitHub Link:

https://github.com/IBM-EPBL/IBM-Project-2294-1658469370/tree/main/Final%20Deliverables