PROJECT: PUBLIC TRANSPORTATION EFFICIENCY ANALYSIS
DEVELOPMENT PART 2
EXPLORATORY ANALYSIS AND VISUALIZATION:

- • **Exploratory Data Analysis i**s crucial for understanding your dataset, revealing hidden patterns, and guiding further analysis, modeling, or decision-making processes.

- It helps you identify potential issues and formulate questions for more in-depth analysis.

- Use more advanced visualization techniques for specific needs, such as time series plots for time-related data, geospatial maps for geographic data, or interactive visualizations for dynamic exploration.

- **Data visualization** is the graphical representation of data and information.

- It's a powerful tool for interpreting and presenting complex data in a more understandable and insightful way

- Bar Charts: Used for comparing categories of data.

- Line Charts: Ideal for showing trends over time.

- Scatter Plots: Show the relationship between two numeric variables.

- Pie Charts: Display parts of a whole.

- Histograms: Visualize the distribution of a single variable.

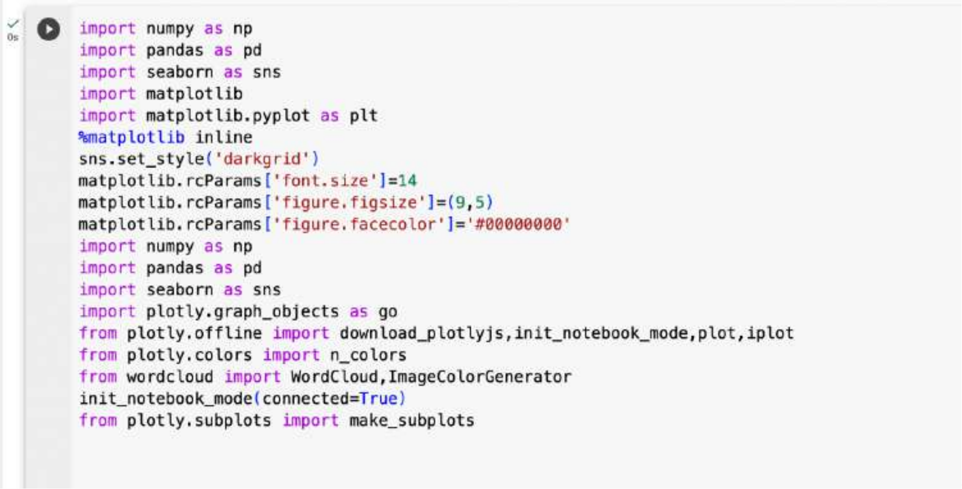
- Heatmaps: Reveal patterns and correlations in large datasets.

- Geospatial Maps: Display data on geographical maps.

- Box Plots: Show statistical summaries of data distributions.

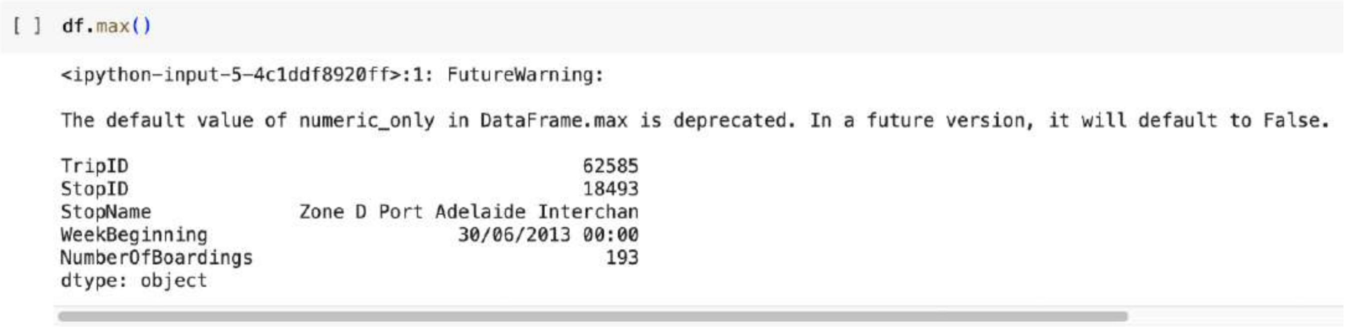
- Interactive Visualizations: Allow users to explore data dynamically.
  And many more.

- Beginning the exploratory analysis and data visualization by importing the required python libraries:



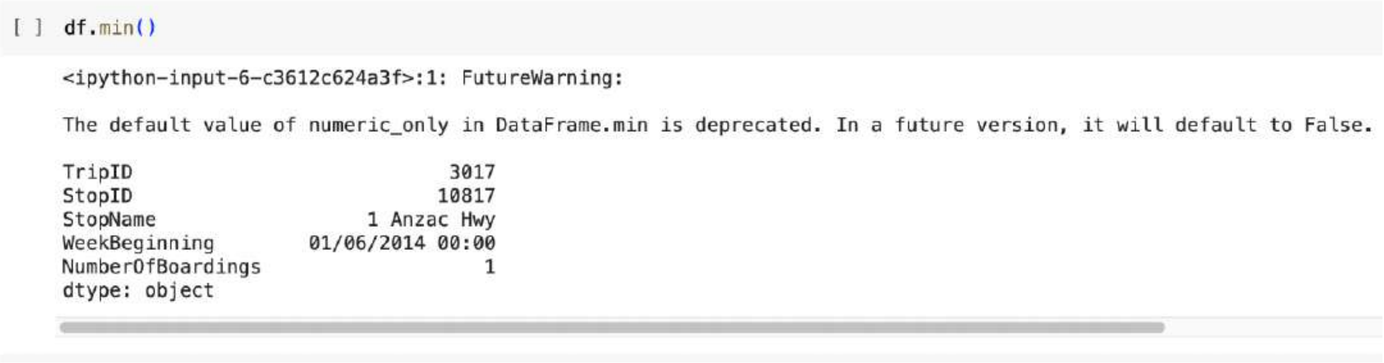
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_style('darkgrid')
matplotlib.rcParams['font.size']=14
matplotlib.rcParams['figure.figsize']=(9,5)
matplotlib.rcParams['figure.facecolor']='#00000000'
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.graph_objects as go
from plotly.offline import download_plotlyjs,init_notebook_mode,plot,iplot
from plotly.colors import n_colors
from wordcloud import WordCloud,ImageColorGenerator
init_notebook_mode(connected=True)
from plotly.subplots import make_subplots
```

1.)Calculate the maximum:



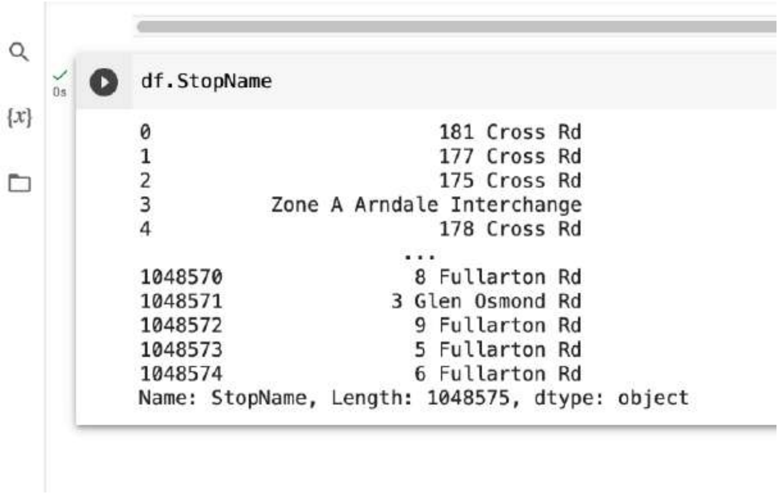
```
[ ]  df.max()

     <ipython-input-5-4c1ddf8920ff>:1: FutureWarning:

     The default value of numeric_only in DataFrame.max is deprecated. In a future version, it will default to False.

     TripID                                    62585
     StopID                                    18493
     StopName             Zone D Port Adelaide Interchan
     WeekBeginning                    30/06/2013 00:00
     NumberOfBoardings                               193
     dtype: object
```

2.)Calculate the minimum:



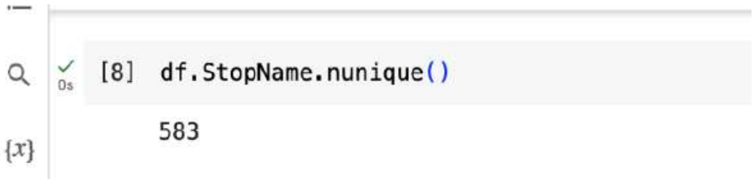
```
[ ]  df.min()

     <ipython-input-6-c3612c624a3f>:1: FutureWarning:

     The default value of numeric_only in DataFrame.min is deprecated. In a future version, it will default to False.

     TripID                           3017
     StopID                          10817
     StopName                  1 Anzac Hwy
     WeekBeginning        01/06/2014 00:00
     NumberOfBoardings                   1
     dtype: object
```

3.)Accessing column from a given dataset:



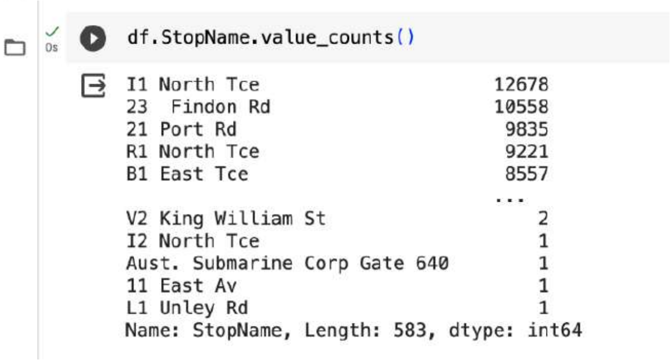
```
df.StopName

0                            181 Cross Rd
1                            177 Cross Rd
2                            175 Cross Rd
3              Zone A Arndale Interchange
4                            178 Cross Rd
                               ...
1048570                    8 Fullarton Rd
1048571                  3 Glen Osmond Rd
1048572                    9 Fullarton Rd
1048573                    5 Fullarton Rd
1048574                    6 Fullarton Rd
Name: StopName, Length: 1048575, dtype: object
```

4.) The nunique() method returns the number of unique values for each column. By specifying the column axis (axis='columns'), the nunique() method searches column-wise and returns the number of unique values for each row.



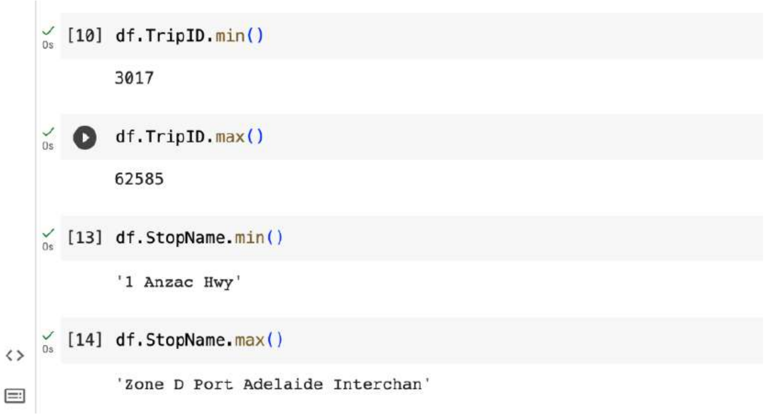
```
[8]  df.StopName.nunique()

     583
```

5.) EXPLORING A COLUMN by computing the number of Stop names:



```
df.StopName.value_counts()

I1 North Tce                      12678
23  Findon Rd                     10558
21 Port Rd                         9835
R1 North Tce                       9221
B1 East Tce                        8557
                                    ...
V2 King William St                    2
I2 North Tce                          1
Aust. Submarine Corp Gate 640         1
11 East Av                            1
L1 Unley Rd                           1
Name: StopName, Length: 583, dtype: int64
```

6.)**ANALYSE THE MINIMUM AND MAXIMUM OF** Stop Names and TRIP ID's mentioned in the dataset("Calculates through the lexicographic order")



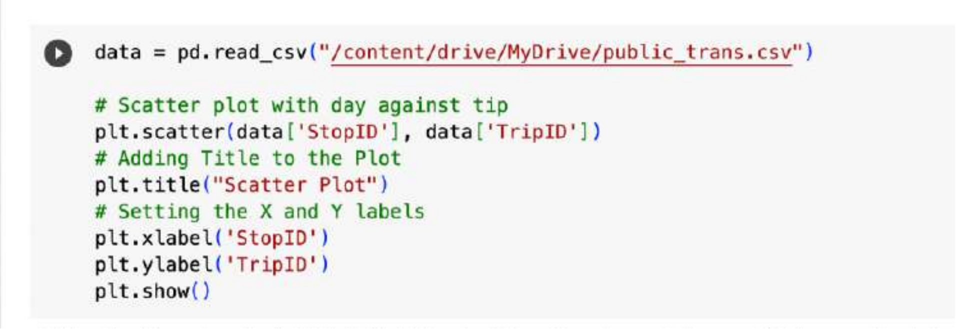
```
[10] df.TripID.min()

     3017

df.TripID.max()

     62585

[13] df.StopName.min()

     '1 Anzac Hwy'

[14] df.StopName.max()

     'Zone D Port Adelaide Interchan'
```

CREATIING VISUALIZATIONS OF DATA USING THE GIVEN DATASET:

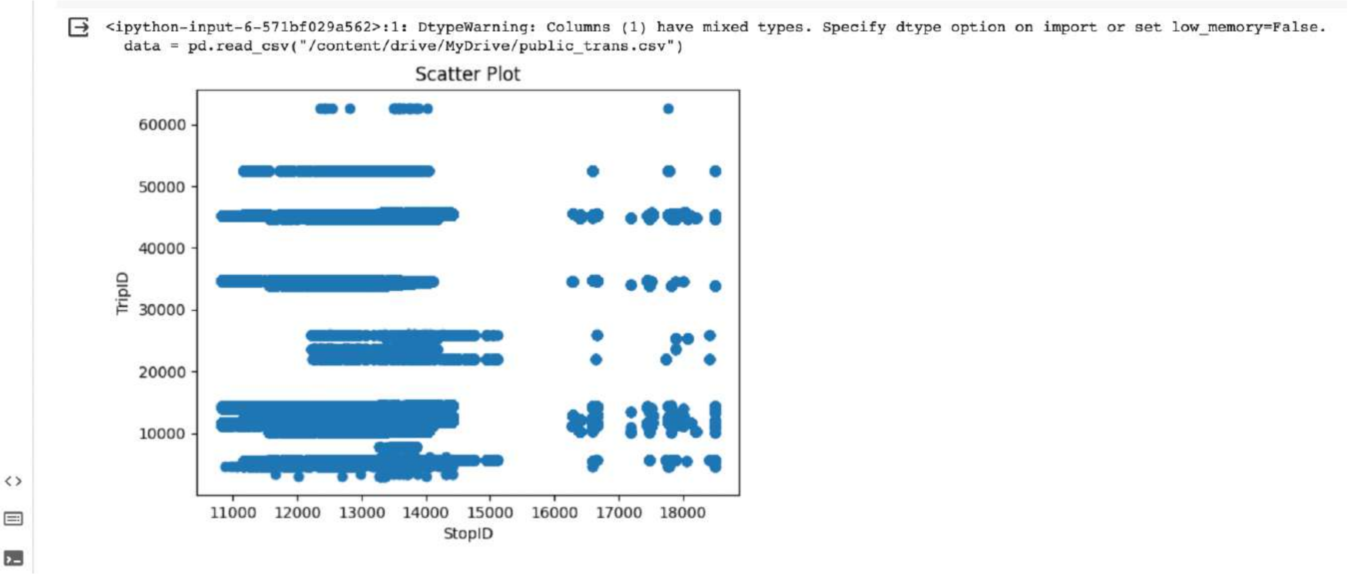
Creating a scatter plot is a simple way to visualize data, and it can be explained in just two steps:

- Prepare Your Data: First, you need to have a set of data that includes two variables, typically referred to as X and Y.
- Each data point should have a pair of values (x, y). Make sure your data is organized and ready for plotting.
- Create the Scatter Plot: Use a data visualization tool like Excel, Python's Matplotlib, or any other plotting software.
- Label your axes for clarity.

7.)SCATTER PLOT TO VISUALIZE STOP ID AND TRIP ID

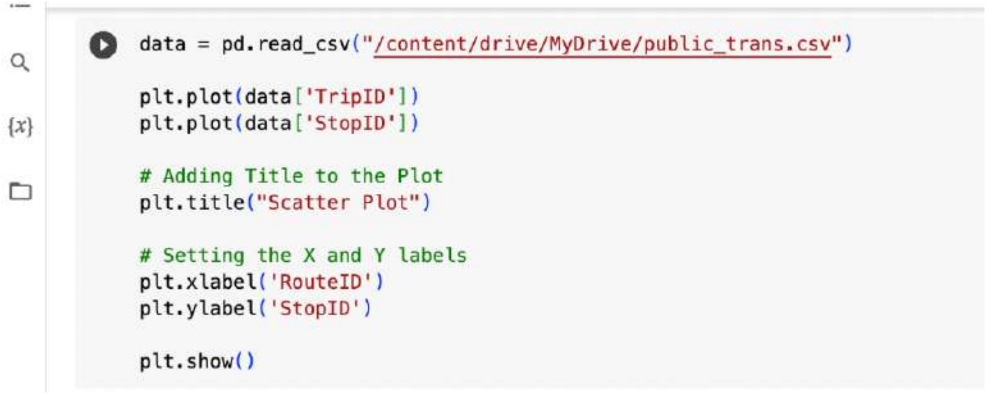


```
data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")

# Scatter plot with day against tip
plt.scatter(data['StopID'], data['TripID'])
# Adding Title to the Plot
plt.title("Scatter Plot")
# Setting the X and Y labels
plt.xlabel('StopID')
plt.ylabel('TripID')
plt.show()
```

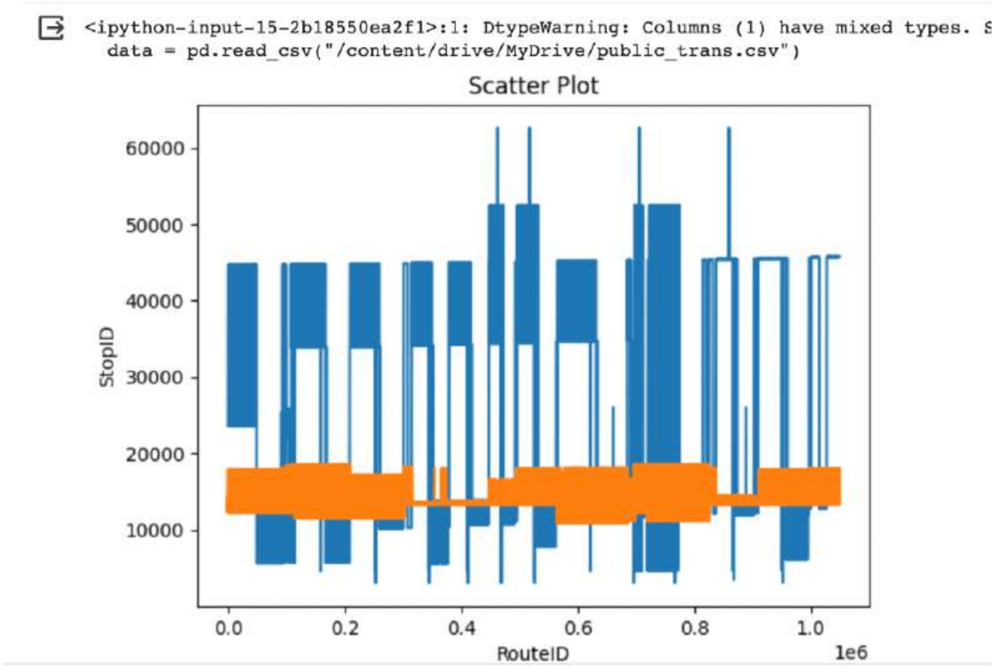


```
<ipython-input-6-571bf029a562>:1: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")
```

8.)SCATTER PLOT TO VISUALIZE ROUTE ID AND STOP ID:



```
data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")

plt.plot(data['TripID'])
plt.plot(data['StopID'])

# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('RouteID')
plt.ylabel('StopID')

plt.show()
```



```
<ipython-input-15-2b18550ea2f1>:1: DtypeWarning: Columns (1) have mixed types. S
  data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")
```

9. HISTOGRAM:

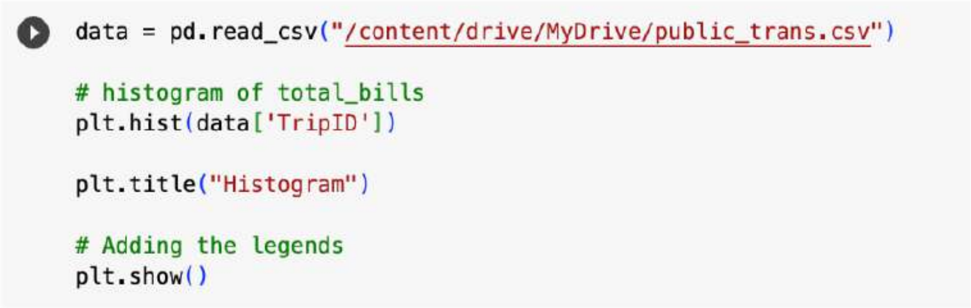
Creating a histogram can be done in two steps:

Prepare Data: Gather the dataset that should be visualized.

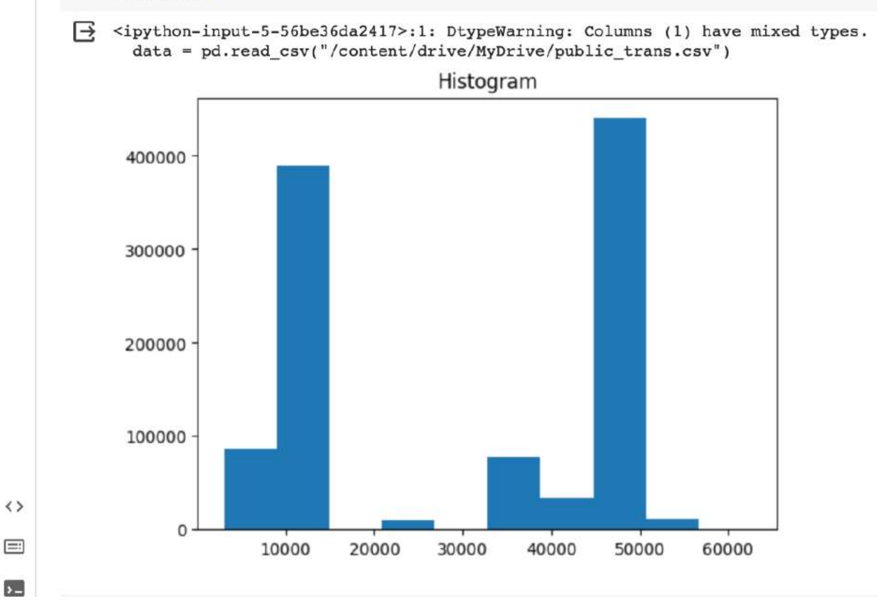
- This data should consist of a single variable for which you want to create a histogram.
- Ensure your data is organized and ready for plotting.

Create the Histogram: Utilize data visualization tools like Excel, Python's Matplotlib, or other software.

- Input your data and instruct the software to generate a histogram.
- Specify the variable you want to plot, the number of bins (intervals) to divide the data.
- Label the axes for clarity.



```
data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")

# histogram of total_bills
plt.hist(data['TripID'])

plt.title("Histogram")

# Adding the legends
plt.show()
```



```
<ipython-input-5-56be36da2417>:1: DtypeWarning: Columns (1) have mixed types.
  data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")
```

10.) BAR CHART TO VISUALIZE TRIP ID AND STOP ID:

Creating a bar chart can be explained in two steps:

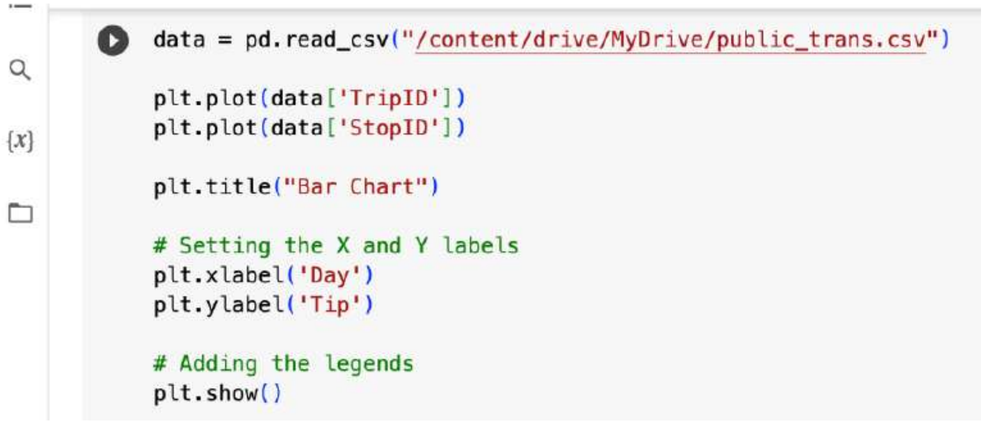
1. Prepare Your Data:

- Collect the dataset you want to represent in a bar chart.
- This data should typically consist of categories or groups and their corresponding values.
- Ensure your data is organized and ready for visualization.

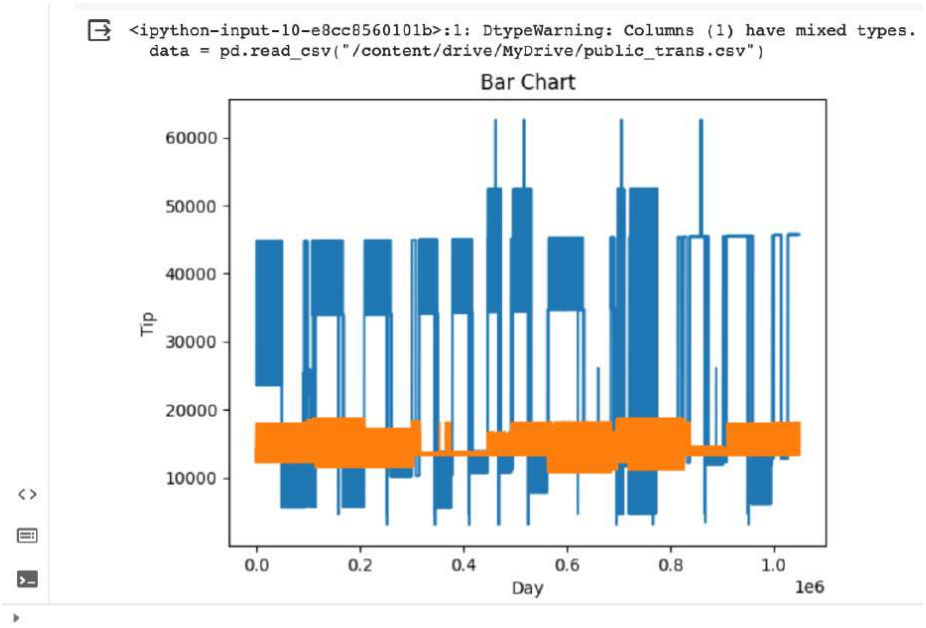
2. Create the Bar Chart:

- Use data visualization software like Excel, Python's Matplotlib, or other charting tools.

- Input your data and instruct the software to generate a bar chart.
- Assign the categories to the horizontal axis (X-axis) and the values to the vertical axis (Y-axis).
- Label the axes and format the chart.

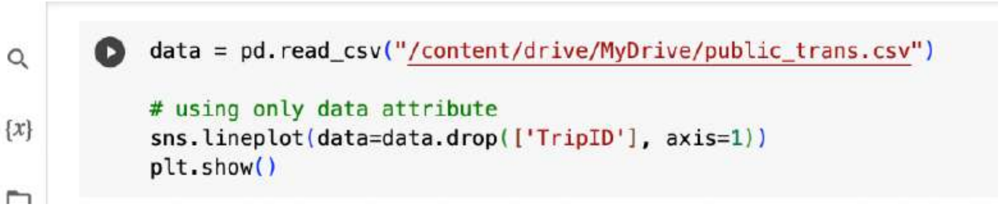


```
data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")

plt.plot(data['TripID'])
plt.plot(data['StopID'])

plt.title("Bar Chart")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

# Adding the legends
plt.show()
```

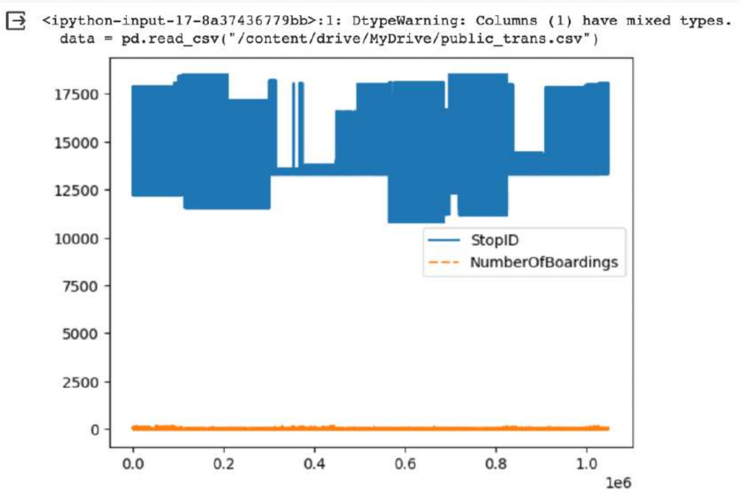


```
<ipython-input-10-e8cc8560101b>:1: DtypeWarning: Columns (1) have mixed types.
  data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")
```

11.) Line plot by using seaborn by importing it and # using only data attribute



```
data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")

# using only data attribute
sns.lineplot(data=data.drop(['TripID'], axis=1))
plt.show()
```

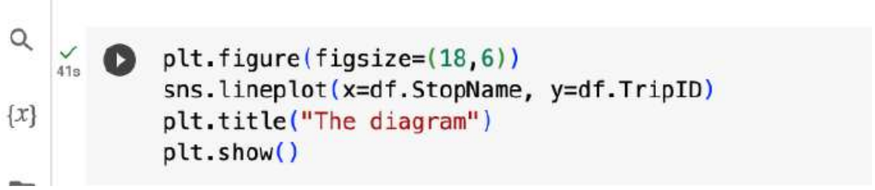


```
<ipython-input-17-8a37436779bb>:1: DtypeWarning: Columns (1) have mixed types.
  data = pd.read_csv("/content/drive/MyDrive/public_trans.csv")
```

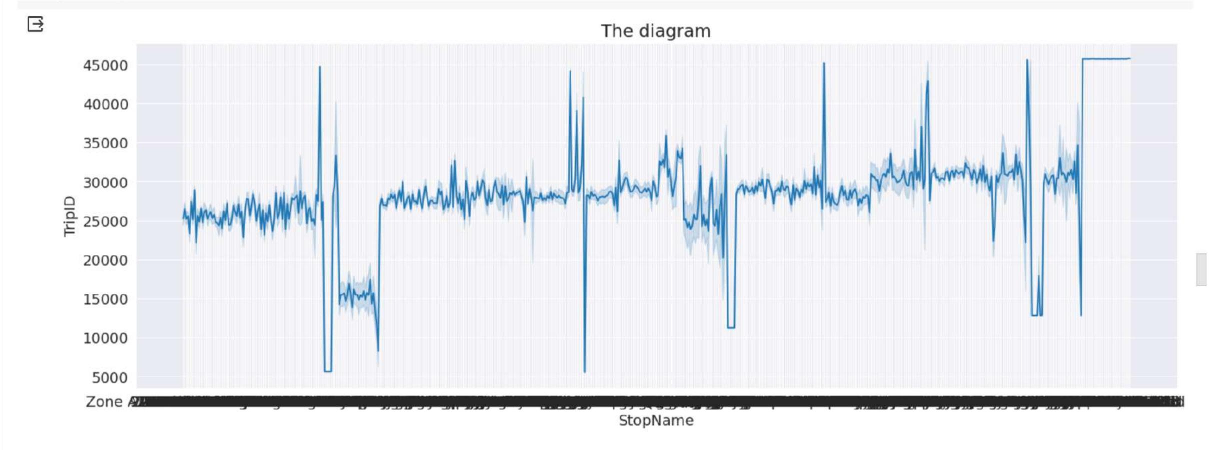
12.) CREATION AND VISUALIZATION OF ECG-LIKE PLOTS USING SEABORN LIBRARIES:

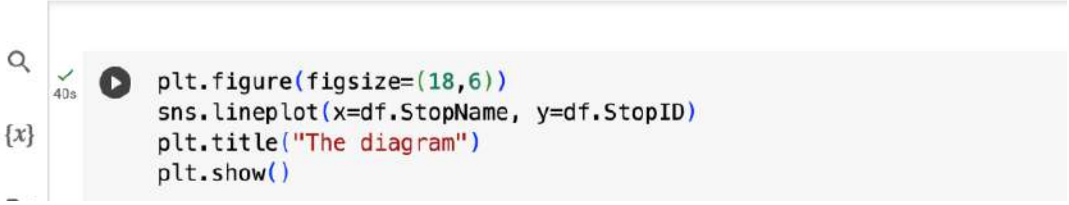
Creating line plots similar to ECG (electrocardiogram) graphs :

- **Data Collection:** Collect the data you want to represent in your ECG-like line plot.
- **Plotting the Line Graph:** Use software or libraries suited for time-series data visualization, such as Python's Matplotlib, R, or specialized medical visualization tools. Input your time-series data and create a line plot. Ensure the x-axis represents time, and the y-axis represents voltage or amplitude.
- Creating ECG-like line plots can be more complex than standard line plots due to the specific requirements for representing cardiac electrical activity accurately.
- Therefore, it's often done using dedicated ECG analysis software to ensure the necessary level of detail and precision.

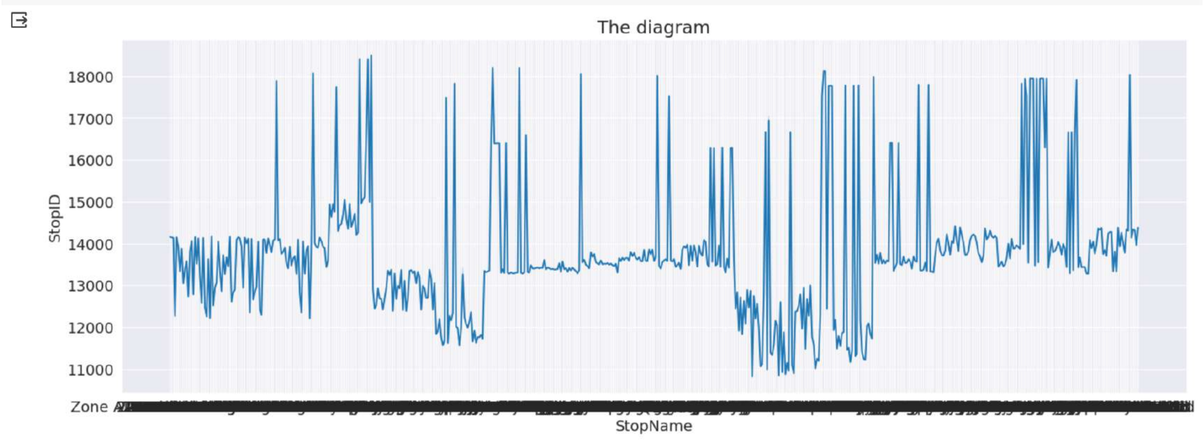


```
plt.figure(figsize=(18,6))
sns.lineplot(x=df.StopName, y=df.TripID)
plt.title("The diagram")
plt.show()
```

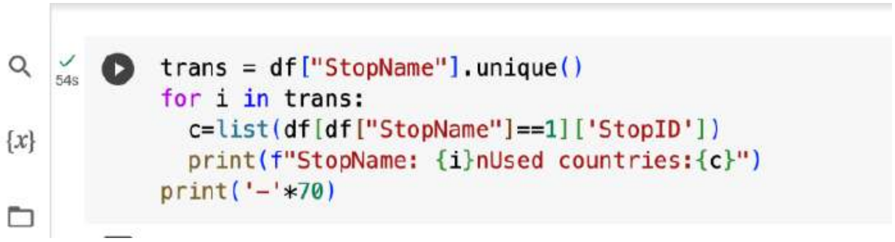




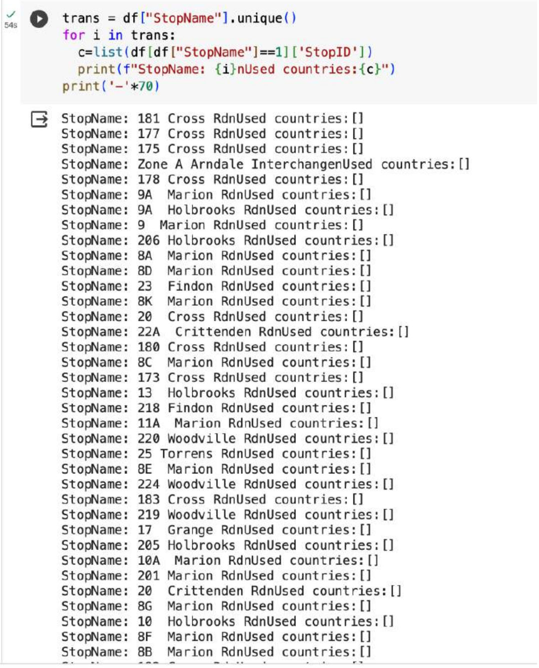
```
plt.figure(figsize=(18,6))
sns.lineplot(x=df.StopName, y=df.StopID)
plt.title("The diagram")
plt.show()
```



13.)ACCESSING THE USED STOP NAMES:



```
trans = df["StopName"].unique()
for i in trans:
  c=list(df[df["StopName"]==1]['StopID'])
  print(f"StopName: {i}nUsed countries:{c}")
print('-'*70)
```



```
trans = df["StopName"].unique()
for i in trans:
  c=list(df[df["StopName"]==1]['StopID'])
  print(f"StopName: {i}nUsed countries:{c}")
print('-'*70)
```

```
StopName: 181 Cross RdnUsed countries:[]
StopName: 177 Cross RdnUsed countries:[]
StopName: 175 Cross RdnUsed countries:[]
StopName: Zone A Arndale InterchangenUsed countries:[]
StopName: 178 Cross RdnUsed countries:[]
StopName: 9A  Marion RdnUsed countries:[]
StopName: 9A  Holbrooks RdnUsed countries:[]
StopName: 9  Marion RdnUsed countries:[]
StopName: 206 Holbrooks RdnUsed countries:[]
StopName: 8A  Marion RdnUsed countries:[]
StopName: 8D  Marion RdnUsed countries:[]
StopName: 23  Findon RdnUsed countries:[]
StopName: 8K  Marion RdnUsed countries:[]
StopName: 20  Cross RdnUsed countries:[]
StopName: 22A  Crittenden RdnUsed countries:[]
StopName: 180 Cross RdnUsed countries:[]
StopName: 8C  Marion RdnUsed countries:[]
StopName: 173 Cross RdnUsed countries:[]
StopName: 13  Holbrooks RdnUsed countries:[]
StopName: 218 Findon RdnUsed countries:[]
StopName: 11A  Marion RdnUsed countries:[]
StopName: 220 Woodville RdnUsed countries:[]
StopName: 25 Torrens RdnUsed countries:[]
StopName: 8E  Marion RdnUsed countries:[]
StopName: 224 Woodville RdnUsed countries:[]
StopName: 183 Cross RdnUsed countries:[]
StopName: 219 Woodville RdnUsed countries:[]
StopName: 17  Grange RdnUsed countries:[]
StopName: 205 Holbrooks RdnUsed countries:[]
StopName: 10A  Marion RdnUsed countries:[]
StopName: 201 Marion RdnUsed countries:[]
StopName: 20  Crittenden RdnUsed countries:[]
StopName: 8G  Marion RdnUsed countries:[]
StopName: 10  Holbrooks RdnUsed countries:[]
StopName: 8F  Marion RdnUsed countries:[]
StopName: 8B  Marion RdnUsed countries:[]
```

14.) USING GROUPBY METHOD:

A **groupby operation** involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

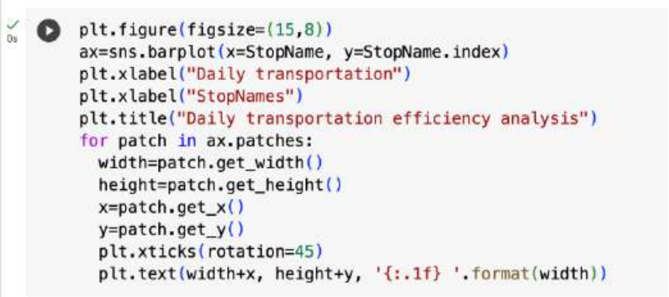
15.) PATCHES TO ANALYSE THE STOPNAMES AND STOP ID:

1. Import a Graphics Library:

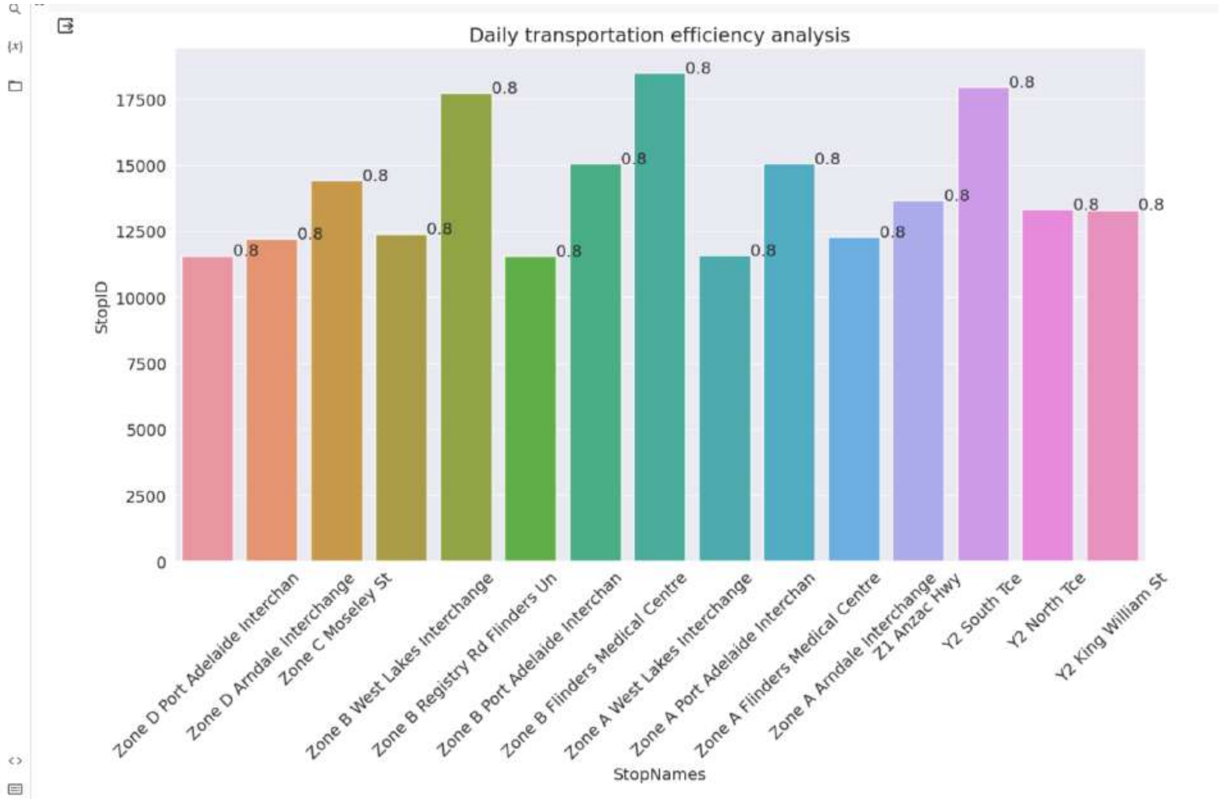
- To create patches, you need to use a graphics library or software such as Matplotlib in Python.
- These libraries provide functions and classes for drawing and manipulating graphical objects, including patches.

2. Defining and Drawing the Patch:

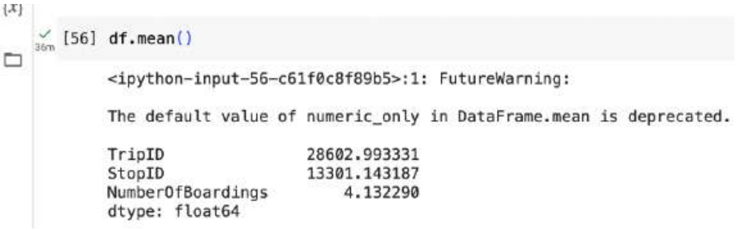
- Use the library's functions or methods to define the characteristics of your patch, such as its shape (e.g., rectangle, circle, polygon), size, position, and style (e.g., color, fill, outline).
- Then, instruct the library to draw the patch on your canvas or graphical display.
- This typically involves specifying the coordinates and attributes of the patch within the graphical context.



```
plt.figure(figsize=(15,8))
ax=sns.barplot(x=StopName, y=StopName.index)
plt.xlabel("Daily transportation")
plt.xlabel("StopNames")
plt.title("Daily transportation efficiency analysis")
for patch in ax.patches:
  width=patch.get_width()
  height=patch.get_height()
  x=patch.get_x()
  y=patch.get_y()
  plt.xticks(rotation=45)
  plt.text(width+x, height+y, '{:.1f} '.format(width))
```

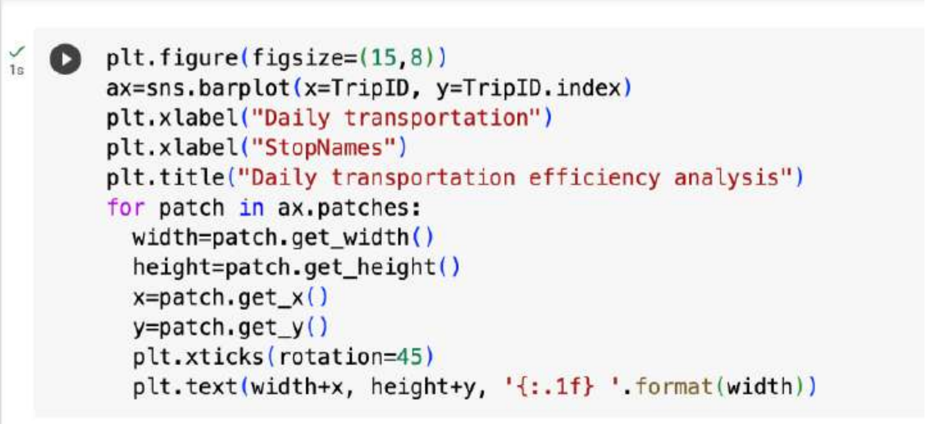


17.COMPUTING MEAN FROM THE GIVEN DATASET:

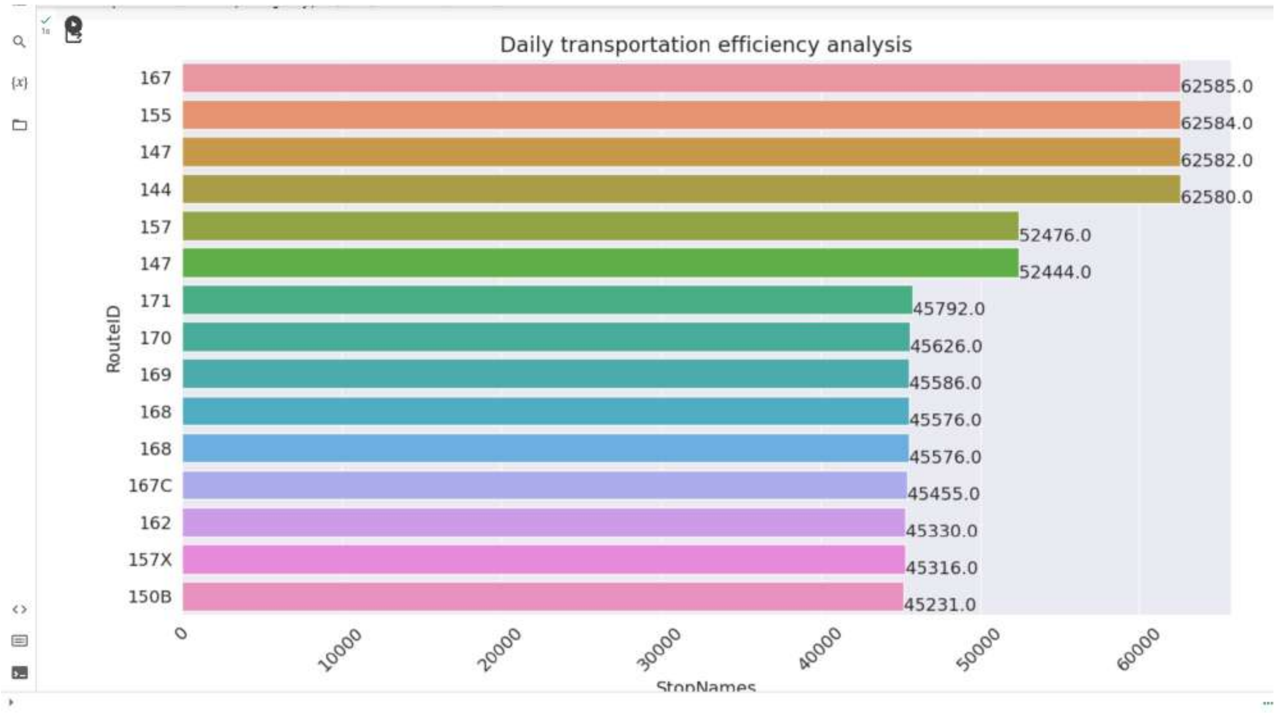


```
[56] df.mean()

<ipython-input-56-c61f0c8f89b5>:1: FutureWarning:

The default value of numeric_only in DataFrame.mean is deprecated.

TripID                 28602.993331
StopID                 13301.143187
NumberOfBoardings          4.132290
dtype: float64
```

18.BARPLOT TO VISUALIZE THE TRIP ID:



```
plt.figure(figsize=(15,8))
ax=sns.barplot(x=TripID, y=TripID.index)
plt.xlabel("Daily transportation")
plt.xlabel("StopNames")
plt.title("Daily transportation efficiency analysis")
for patch in ax.patches:
  width=patch.get_width()
  height=patch.get_height()
  x=patch.get_x()
  y=patch.get_y()
  plt.xticks(rotation=45)
  plt.text(width+x, height+y, '{:.1f} '.format(width))
```



IBM Cognos analytics:

1. **Access Cognos Analytics:**
   - Log in to your IBM Cognos Analytics environment.

2. **Data Source Connection:**
   - Connect to your data source (e.g., a database, spreadsheet, or data file).

3. **Create a Report:**

- Start a new report or open an existing one.

4. **Select Data:**
- Choose the data you want to visualize by adding data items from your data source to the report.

5. **Choose Visualization Type:**
- Select the type of visualization you want to create (e.g., bar chart, line chart, pie chart, etc.).

6. **Customize Visualization:**
- Customize the visualization by specifying axes, labels, colors, and other properties.

7. **Apply Filters:**
- Add filters to your visualization to refine the data displayed.

8. **Group and Aggregate Data:**
- Group and aggregate data as needed to provide meaningful insights.

9. **Add Interactivity:**
- Enhance your visualization by adding interactivity elements

INSIGHTS:

- We were able to witness the visualizations physically.
- It made us more comfortable to see the data in different figures.
- We chose python as it had many libraries associated with it.
- We noted the plotting of the given columns in the dataset.
- The patches were used to create bars in bar charts or custom shapes to highlight specific data points.
- Differences,mean,maximum and minimum were easily computed for a particular dataset.
- Implementing these methods really helped in every way to understand the dataset thoroughly.

CONCLUSION: In summary, the exploratory analysis demonstrates that the public transportation system is a vital component of urban mobility. There are opportunities for enhancing efficiency, inclusivity, and sustainability, which can be achieved through route optimization, technology integration, and a strong commitment to passenger experience to make some application devices and so on.