Analyzing-Uber-Rides-Data-with-Python

"Analyzing Uber Rides Data with Python" dives into understanding Uber's ride data to gain insights into user behavior, peak ride hours, and geographical trends. This report uses Python libraries like Pandas, Matplotlib, and Seaborn to clean, analyze, and visualize data, helping in data-driven decisions for enhancing operational efficiency and customer satisfaction.

Importing Libraries

The following libraries will be used for the analysis:

- Pandas: With its numerous functions, this library supports loading data frames in 2D array format and allows you to complete analysis tasks all at once.
- Numpy: Large calculations can be completed quickly with numpy arrays because of their extreme speed.
- Matplotlib / Seaborn: This library is used to draw visualizations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing Dataset

Once all the libraries have been imported, you can use the pandas library to import the dataset.

```
dataset = pd.read_csv("UberDataset.csv")
dataset.head()
```

Output:

```
<bound method NDFrame.head of             START_DATE          END_DATE  CATEGORY            START  \
0     01-01-2016 21:11  01-01-2016 21:17  Business       Fort Pierce
1     01-02-2016 01:25  01-02-2016 01:37  Business       Fort Pierce
2     01-02-2016 20:25  01-02-2016 20:38  Business       Fort Pierce
3     01-05-2016 17:31  01-05-2016 17:45  Business       Fort Pierce
4     01-06-2016 14:42  01-06-2016 15:49  Business       Fort Pierce
...                ...               ...       ...               ...
1151  12/31/2016 13:24  12/31/2016 13:42  Business           Kar?chi
1152  12/31/2016 15:03  12/31/2016 15:38  Business  Unknown Location
1153  12/31/2016 21:32  12/31/2016 21:50  Business        Katunayake
1154  12/31/2016 22:08  12/31/2016 23:51  Business           Gampaha
1155            Totals               NaN       NaN               NaN

                  STOP    MILES          PURPOSE
0          Fort Pierce      5.1   Meal/Entertain
1          Fort Pierce      5.0              NaN
2          Fort Pierce      4.8  Errand/Supplies
3          Fort Pierce      4.7          Meeting
4       West Palm Beach     63.7   Customer Visit
...                ...      ...              ...
1151  Unknown Location      3.9   Temporary Site
1152  Unknown Location     16.2          Meeting
1153           Gampaha      6.4   Temporary Site
1154         Ilukwatta     48.2   Temporary Site
1155               NaN  12204.7              NaN

[1156 rows x 7 columns]>
```

To find the shape of the dataset, we can use dataset.shape

```
dataset.shape
```

Output
(1156, 7)

Knowing the datatype, null values count, and other details is necessary to gain a deeper understanding of the data. Therefore, the code below will be used for that.

```
dataset.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1156 entries, 0 to 1155
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   START_DATE  1156 non-null   object
 1   END_DATE    1155 non-null   object
 2   CATEGORY    1155 non-null   object
 3   START       1155 non-null   object
 4   STOP        1155 non-null   object
 5   MILES       1156 non-null   float64
 6   PURPOSE     1156 non-null   object
dtypes: float64(1), object(6)
memory usage: 63.3+ KB
```

Data Preprocessing:
Since we are aware that the PURPOSE column contains a large number of null values, we will fill in the null values with a NOT keyword.

```
dataset['PURPOSE'].fillna("NOT", inplace=True)
```

To enable further analysis, the START_DATE and END_DATE fields should be changed to the date_time format.

```
dataset['START_DATE'] = pd.to_datetime(dataset['START_DATE'], errors='coerce')
dataset['END_DATE'] = pd.to_datetime(dataset['END_DATE'], errors='coerce')
```

The START_DATE is split into a date and time column, and the time is then divided into four categories: morning, afternoon, evening, and night.

```
from datetime import datetime

dataset['date'] = pd.DatetimeIndex(dataset['START_DATE']).date
dataset['time'] = pd.DatetimeIndex(dataset['START_DATE']).hour

#changing into categories of day and night
dataset['day-night'] = pd.cut(x=dataset['time'],
                              bins = [0,10,15,19,24],
                              labels = ['Morning','Afternoon','Evening','Night'])
```

We can now remove rows with null values after finishing the creation of new columns.

```
dataset.dropna(inplace=True)
```

Eliminating duplicate rows from the dataset is also crucial. Use the code below to accomplish that.

```
dataset.drop_duplicates(inplace=True)
```

Data Visualization

Let's begin by examining the dataset's unique values for the object datatype columns.

```
obj = (dataset.dtypes=='object')
object_cols = list(obj[obj].index)

unique_values = {}
for col in object_cols:
    unique_values[col] = dataset[col].unique().size
unique_values
```

Output:

```
{'CATEGORY': 2, 'START': 108, 'STOP': 112, 'PURPOSE': 7, 'date': 113}
```

The CATEGORY and PURPOSE columns will now be countploted using the matplotlib and seaborn libraries.

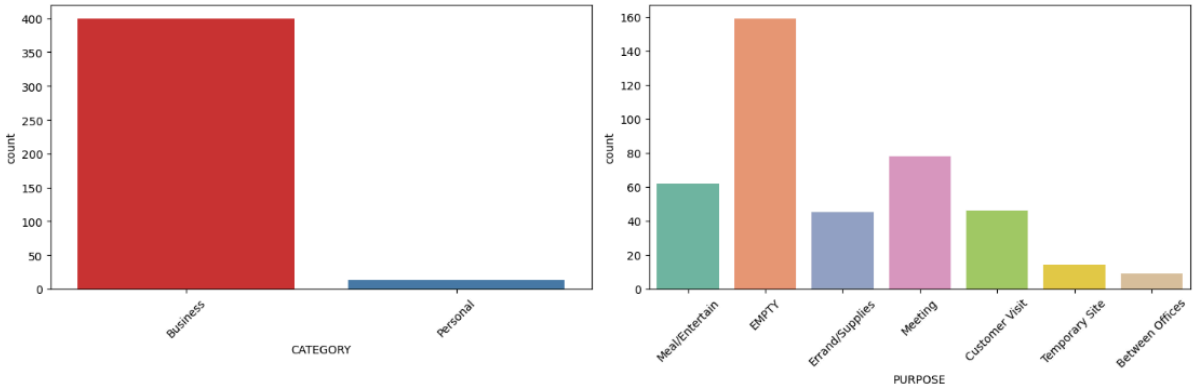
```
plt.figure(figsize=(15, 5))  # Adjust the figure size as needed

# First subplot for 'CATEGORY'
plt.subplot(1, 2, 1)
sns.countplot(x='CATEGORY', data=dataset, hue='CATEGORY', palette='Set1', dodge=False, legend=False)
plt.xticks(rotation=45)  # Change rotation angle to 45 degrees

# Second subplot for 'PURPOSE'
plt.subplot(1, 2, 2)
sns.countplot(x='PURPOSE', data=dataset, hue='PURPOSE', palette='Set2', dodge=False, legend=False)
plt.xticks(rotation=45)  # Change rotation angle to 45 degrees


plt.tight_layout()  # Adjust the layout to prevent overlap
plt.show()
```

Output:



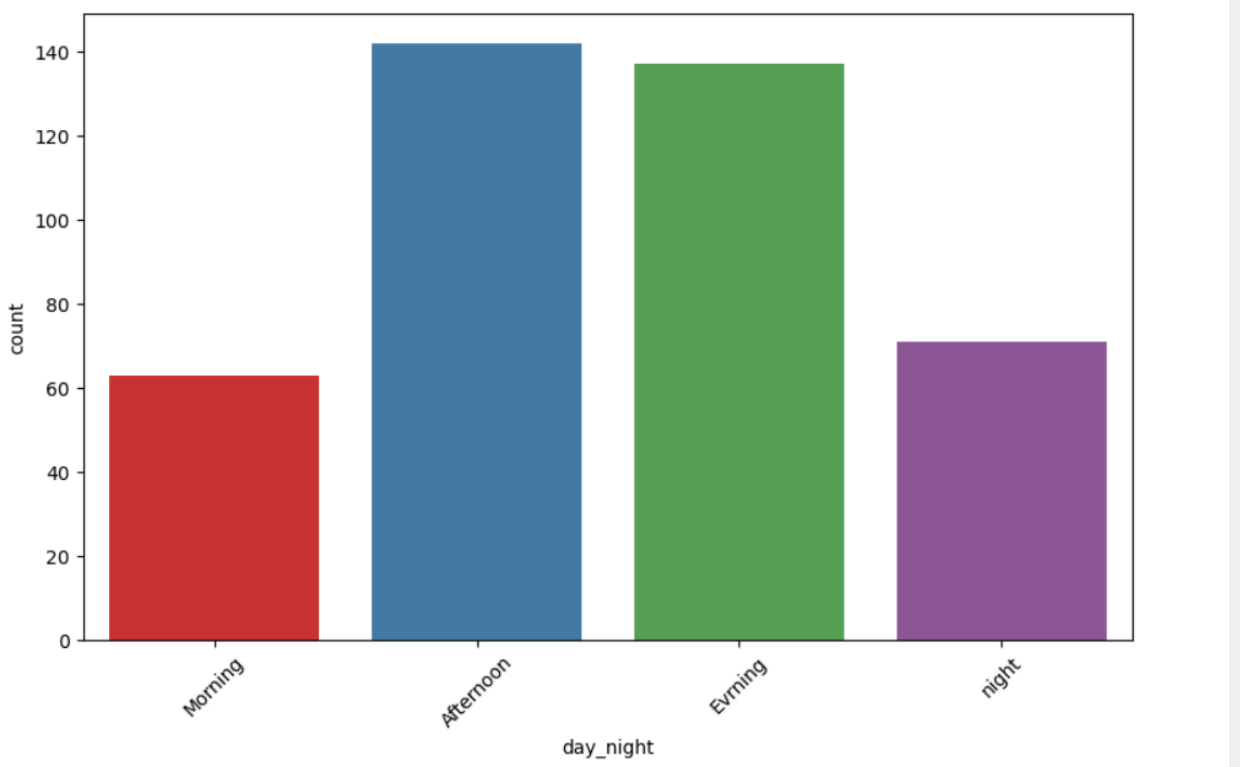
Let's repeat the process for the time column, using the time column that we previously extracted.

```
plt.figure(figsize=(10, 6))

sns.countplot(x='day_night', data=dataset, hue='day_night', palette='Set1', dodge=False, legend=False)
plt.xticks(rotation=45)

plt.show()
```

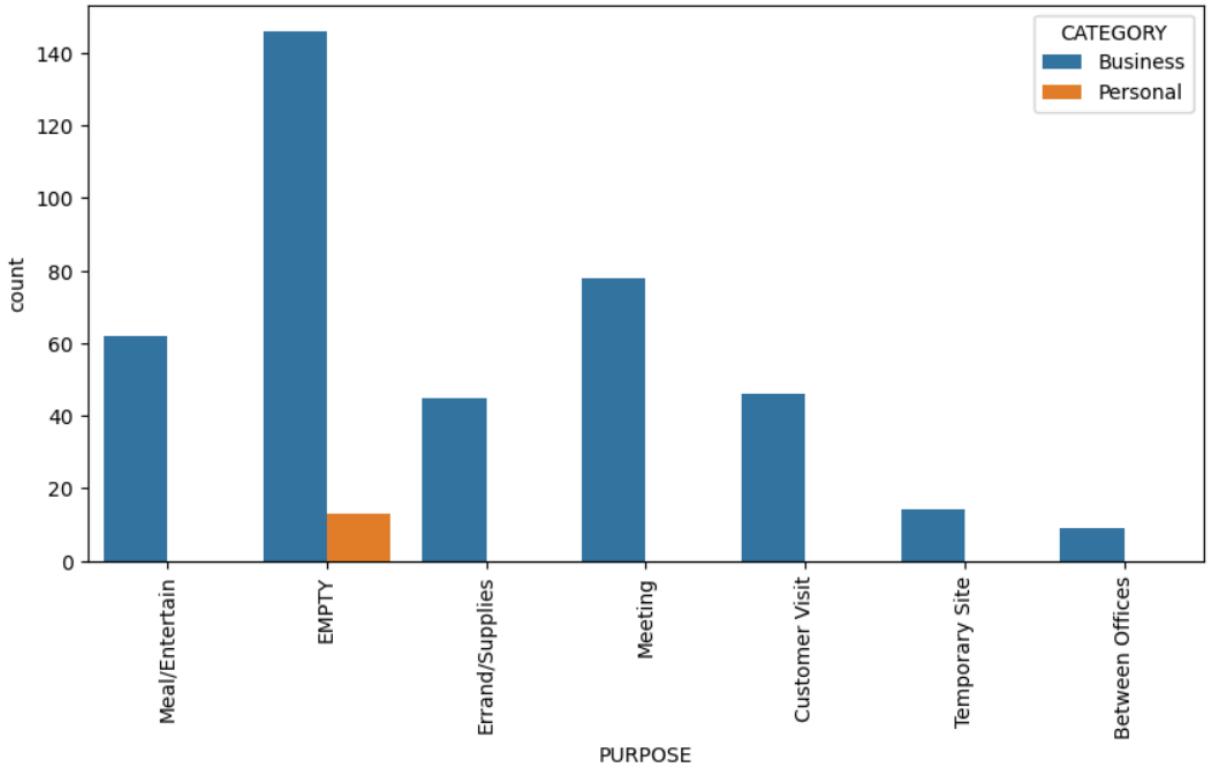
Output:



The user's PURPOSE will now be compared with the two distinct categories.

```
plt.figure(figsize=(10, 5))
sns.countplot(data=dataset, x='PURPOSE', hue='CATEGORY')
plt.xticks(rotation=90)
plt.show()
```

Output:



Conclusions drawn from the count-plots above:

- The majority of the rides are reserved for business travel.
- The majority of people reserve taxis for meetings, meals, and entertainment.
- The majority of taxis are reserved between 10 a.m. and 5 p.m. (afternoon).

Both the CATEGORY and PURPOSE columns are crucial, as we have seen. We will now categorize them using OneHotEncoder.

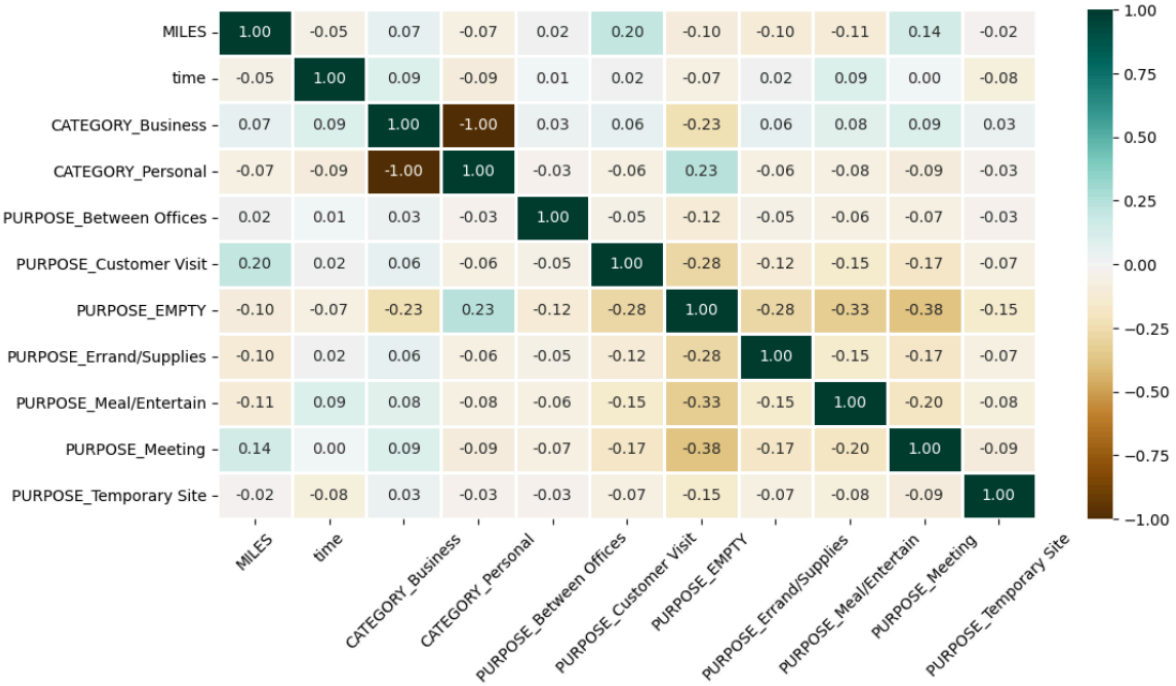
```
from sklearn.preprocessing import OneHotEncoder
object_cols = ['CATEGORY', 'PURPOSE']
OH_encoder = OneHotEncoder(sparse_output=False)
OH_cols = pd.DataFrame(OH_encoder.fit_transform(dataset[object_cols]))
OH_cols.index = dataset.index
OH_cols.columns = OH_encoder.get_feature_names_out()
df_final = dataset.drop(object_cols, axis=1)
dataset = pd.concat([df_final, OH_cols], axis=1)
```

Next, we can use a heatmap to determine the correlation between the columns.

```
numeric_data = dataset.select_dtypes(include=['number'])

# Plot the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(numeric_data.corr(), cmap='BrBG', fmt='.2f', linewidths=2, annot=True)
plt.xticks(rotation=45)
plt.show()
```

Output



Insights from the heatmap:

- As previously demonstrated, there is a strong negative correlation between the business and personal categories. Thus, this plot supports the conclusions mentioned above.
- The features don't really correlate with one another.

We must now visualize the month's data. This can be completed in the same way as before (for hours).

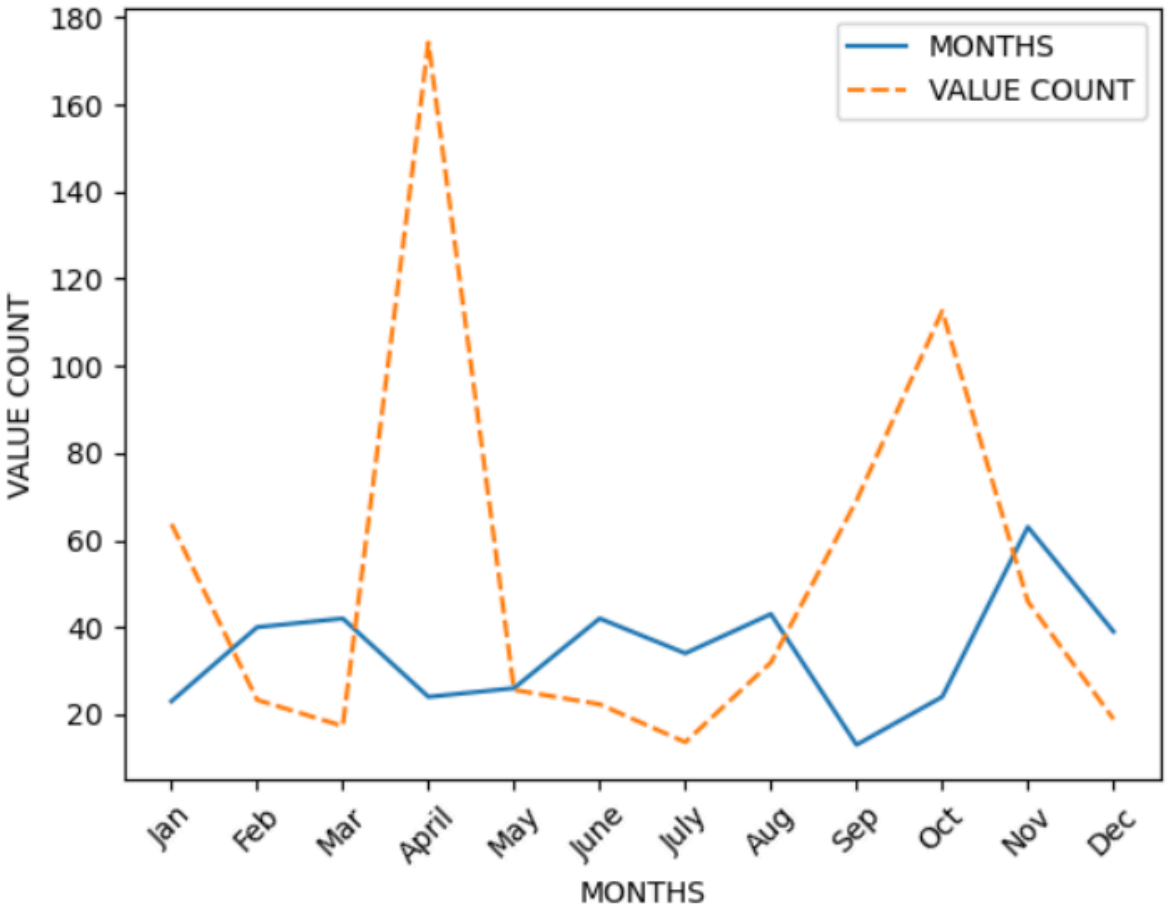
```
dataset['MONTH'] = pd.DatetimeIndex(dataset['START_DATE']).month
month_label = {1.0: 'Jan', 2.0: 'Feb', 3.0: 'Mar', 4.0: 'April',
               5.0: 'May', 6.0: 'June', 7.0: 'July', 8.0: 'Aug',
               9.0: 'Sep', 10.0: 'Oct', 11.0: 'Nov', 12.0: 'Dec'}
dataset["MONTH"] = dataset.MONTH.map(month_label)

mon = dataset.MONTH.value_counts(sort=False)

# Month total rides count vs Month ride max count
df = pd.DataFrame({"MONTHS": mon.values,
                   "VALUE COUNT": dataset.groupby('MONTH',
                                                  sort=False)['MILES'].max()})

p = sns.lineplot(data=df)
p.set(xlabel="MONTHS", ylabel="VALUE COUNT")
plt.xticks(rotation=45)
plt.show()
```

Output:



Insights from the above plot :

- The counts are very irregular.
- Still its very clear that the counts are very less during Nov, Dec, Jan, which justifies the fact that time winters are there in Florida, US.

Visualization for days data:

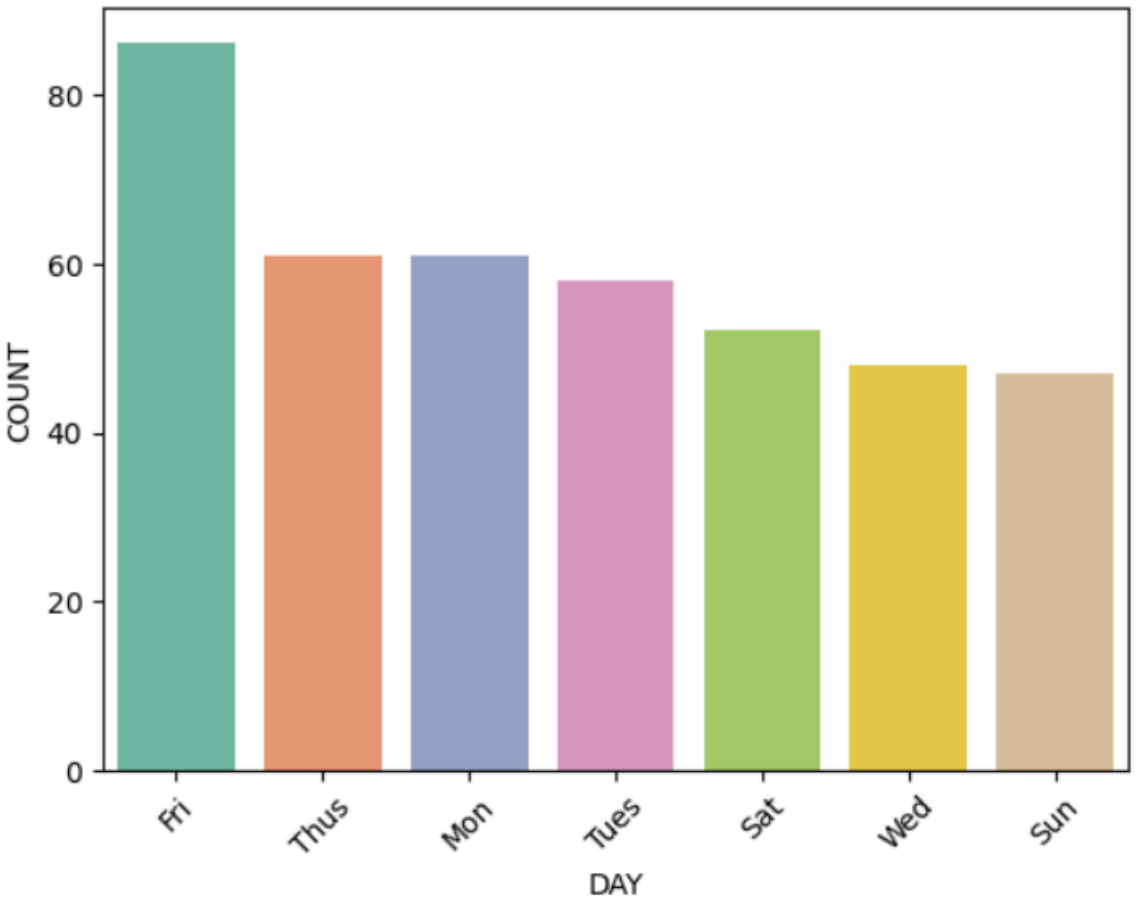
```
dataset['DAY'] = dataset.START_DATE.dt.weekday
day_label = {
    0: 'Mon', 1: 'Tues', 2: 'Wed', 3: 'Thus', 4: 'Fri', 5: 'Sat', 6: 'Sun'
}
dataset['DAY'] = dataset['DAY'].map(day_label)
```

```
day_label = dataset.DAY.value_counts().reset_index()
day_label.columns = ['DAY', 'COUNT']

# Create a color palette with different colors
colors = sns.color_palette("Set2", len(day_label))

# Plot with different colors
sns.barplot(data=day_label, x='DAY', y='COUNT', palette=colors, hue='DAY', dodge=False, legend=False)
plt.xlabel('DAY')
plt.ylabel('COUNT')
plt.xticks(rotation=45)
plt.show()
```

Output:

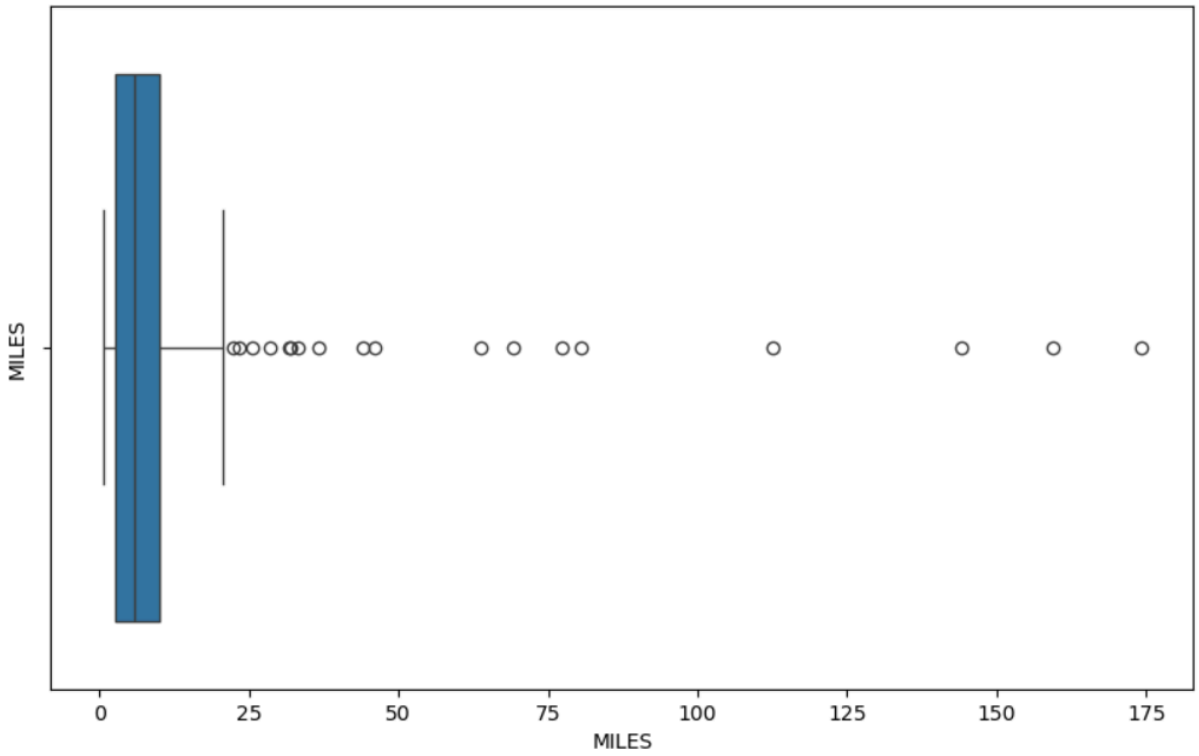


Let's now investigate the MILES Column.

To examine the column's distribution, we can utilize a boxplot.

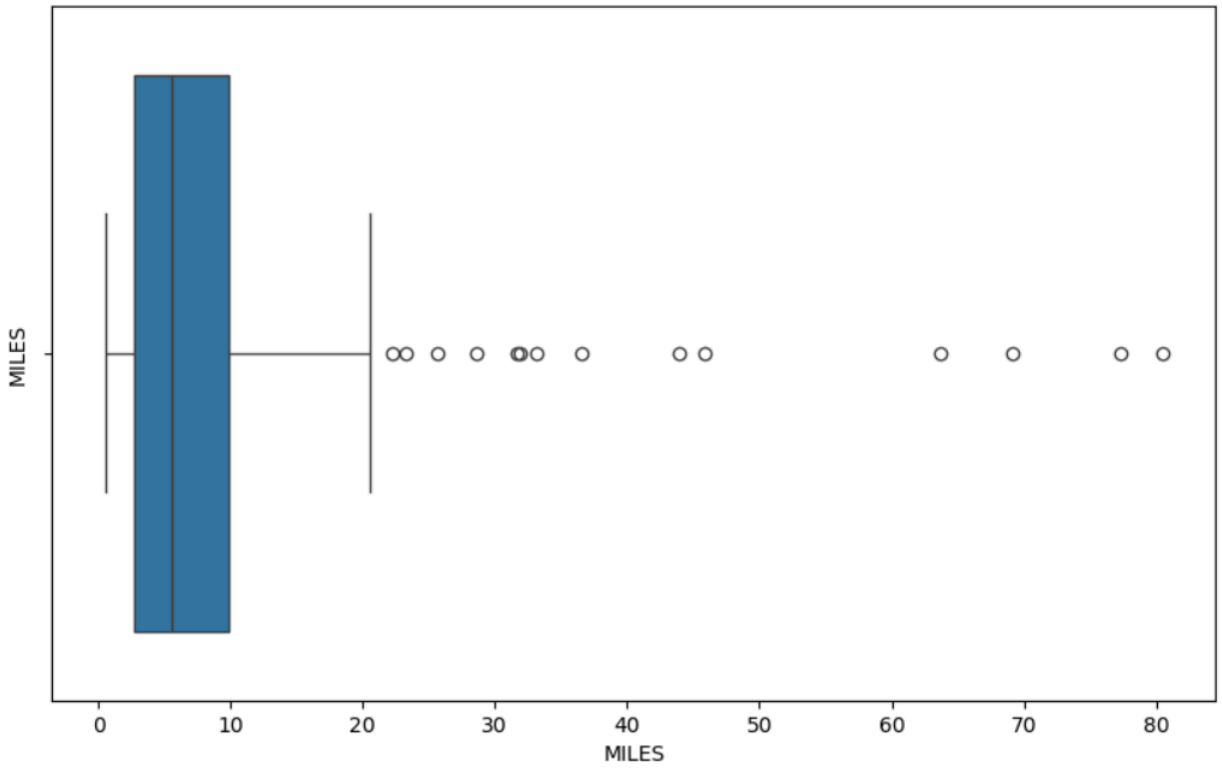
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=dataset['MILES'])
plt.ylabel('MILES')
plt.show()
```

Output:



because it is difficult to understand the graph. For values less than 100, let's zoom in.

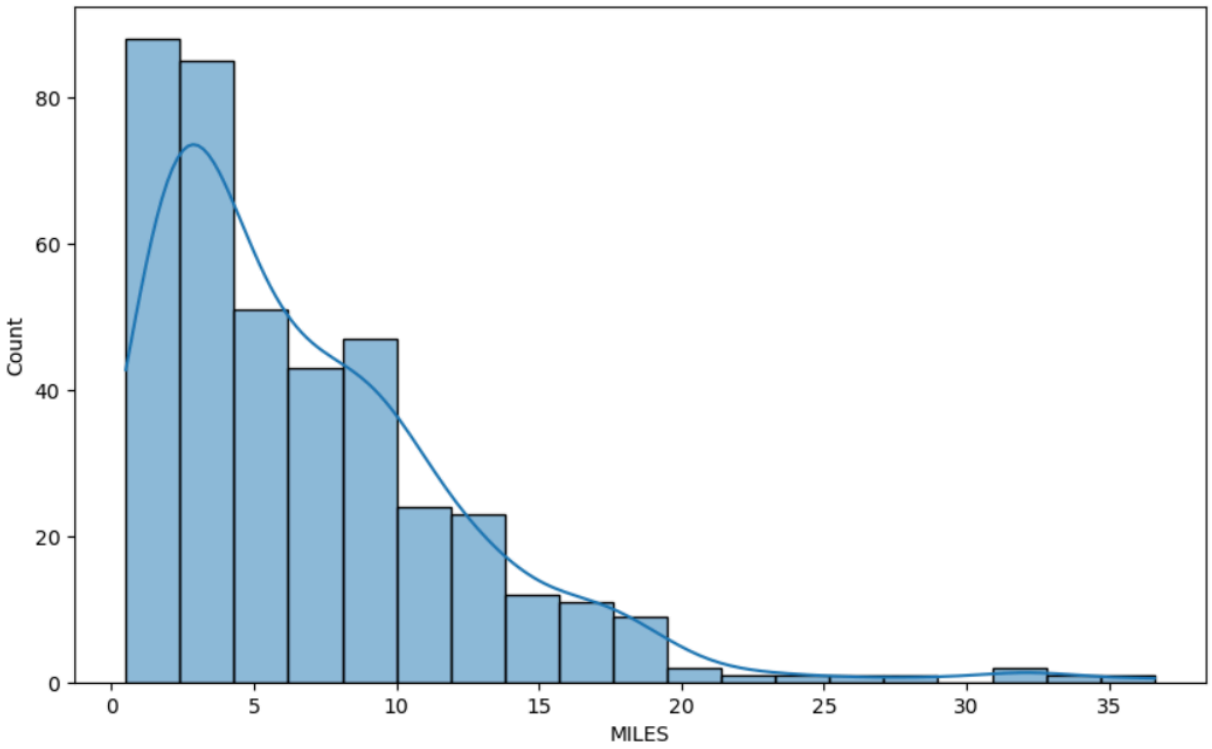
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=dataset[dataset['MILES']<100]['MILES'])
plt.ylabel('MILES')
plt.show()
```



It is somewhat visible. However, for values under 40, we can use a distplot to gain more clarity.

```
plt.figure(figsize=(10, 6))
sns.histplot(data=dataset[dataset['MILES'] < 40], x='MILES', kde=True)
plt.xlabel('MILES')
plt.show()
```

Output:



Insights from the above plots :

- Most of the cabs booked for the distance of 4-5 miles.
- Majorly people chooses cabs for the distance of 0-20 miles.
- For distance more than 20 miles cab counts is nearly negligible.

In the final section, we'd like to extend our heartfelt thanks to all GitHub visitors who took the time to view, star, or contribute to the "Analyzing Uber Rides Data with Python" project. Your interest and support motivate us to continue enhancing the report, sharing insights, and refining our skills in data analysis. ♥ Thank you for being part of our journey!