Uber Trip Analysis

Importing Libraries

The analysis will be done using the following libraries :

Pandas: This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go. **Numpy:** Numpy arrays are very fast and can perform large computations in a very short time.

Matplotlib / Seaborn: This library is used to draw visualizations.

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

Importing Dataset After importing all the libraries, download the data using the link[https://github.com/Piyush20002/Data-Analysis-Uber-Trips-using-Python/blob/main/UberDataset.csv]

Once downloaded, you can import the dataset using the pandas library.

```
[2] dataset = pd.read_csv("/content/UberDataset.csv")
    dataset.head()
```

[2]

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
0	01-01-2016 21:11	01-01-2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	01-02-2016 01:25	01-02-2016 01:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	01-02-2016 20:25	01-02-2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	01-05-2016 17:31	01-05-2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	01-06-2016 14:42	01-06-2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit

Next steps: Generate code with dataset | View recommended plots | New interactive sheet

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

```
[3] dataset.shape
```

```
(1156, 7)
```

To understand the data more deeply, we need to know about the null values count, datatype, etc. So for that we will use the below code.

```
[4] dataset.info()
```

```
<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     653 non-null    object 
dtypes: float64(1), object(6)
memory usage: 63.3+ KB
```

Data Preprocessing

As we understood that there are a lot of null values in PURPOSE column, so for that we will me filling the null values with a NOT keyword. You can try something else too.

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

Changing the START_DATE and END_DATE to the date_time format so that further it can be use to do analysis.

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

Splitting the START_DATE to date and time column and then converting the time into four different categories i.e. Morning, Afternoon, Evening, Night

Generate a slider using jupyter widgets Close

```
[8] 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'])
```

Once we are done with creating new columns, we can now drop rows with null values.

```
[9] dataset.dropna(inplace=True)
```

It is also important to drop the duplicates rows from the dataset. To do that, refer the code below.

```
[10] dataset.drop_duplicates(inplace=True)
```

```
[13] # Identifying object type columns
     obj = (dataset.dtypes == 'object')
     object_cols = list(obj[obj].index)

     # Initializing a dictionary to store unique value counts
     unique_values = {}

     # Iterating through object columns to find the number of unique values
     for col in object_cols:
         unique_values[col] = dataset[col].unique().size

     # Display the dictionary with unique value counts
     unique_values
```

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

Data Visualization

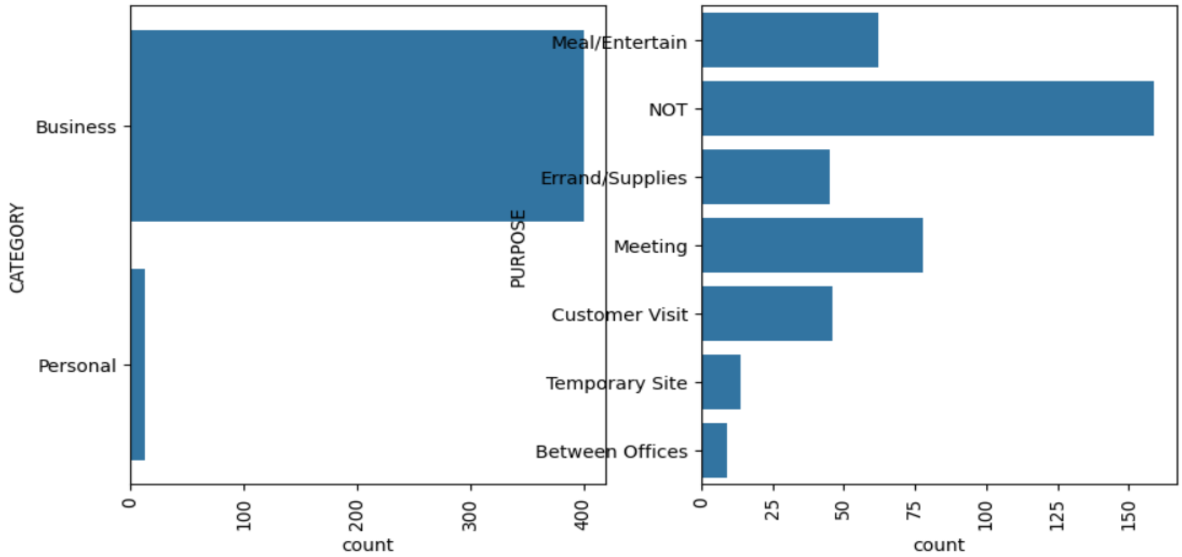
In this section, we will try to understand and compare all columns.

Let's start with checking the unique values in dataset of the columns with object datatype.

Now, we will be using matplotlib and seaborn library for countplot the CATEGORY and PURPOSE columns.

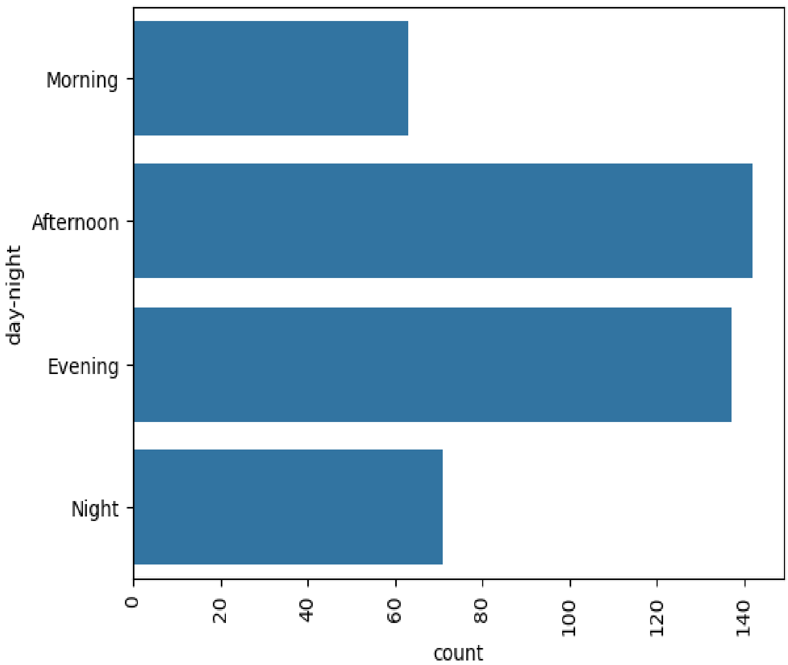
```
[14] plt.figure(figsize=(10,5))

     plt.subplot(1,2,1)
     sns.countplot(dataset['CATEGORY'])
     plt.xticks(rotation=90)

     plt.subplot(1,2,2)
     sns.countplot(dataset['PURPOSE'])
     plt.xticks(rotation=90)
```

```
(array([  0.,  25.,  50.,  75., 100., 125., 150., 175.]),
 [Text(0.0, 0, '0'),
  Text(25.0, 0, '25'),
  Text(50.0, 0, '50'),
  Text(75.0, 0, '75'),
  Text(100.0, 0, '100'),
  Text(125.0, 0, '125'),
  Text(150.0, 0, '150'),
  Text(175.0, 0, '175')])
```



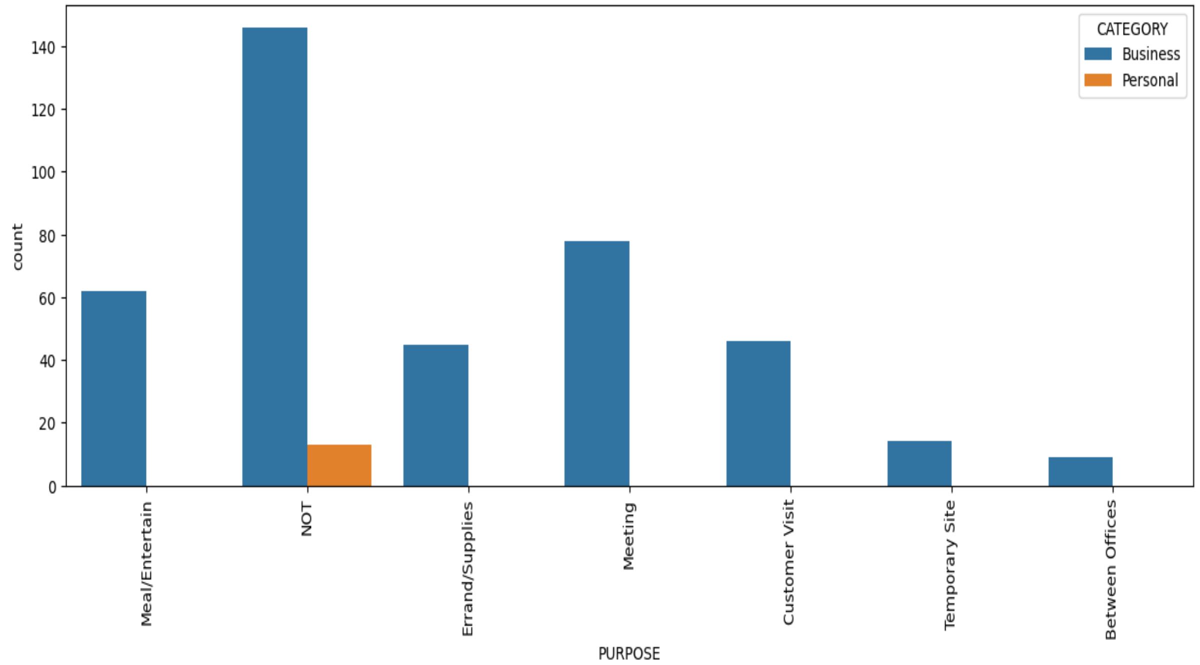

```python
sns.countplot(dataset['day-night'])
plt.xticks(rotation=90)
```

```
(array([  0.,  20.,  40.,  60.,  80., 100., 120., 140., 160.]),
 [Text(0.0, 0, '0'),
  Text(20.0, 0, '20'),
  Text(40.0, 0, '40'),
  Text(60.0, 0, '60'),
  Text(80.0, 0, '80'),
  Text(100.0, 0, '100'),
  Text(120.0, 0, '120'),
  Text(140.0, 0, '140'),
  Text(160.0, 0, '160')])
```



Now, we will be comparing the two different categories along with the PURPOSE of the user.

```
plt.figure(figsize=(15, 5))
sns.countplot(data=dataset, x='PURPOSE', hue='CATEGORY')
plt.xticks(rotation=90)
plt.show()
```



Insights from the above count-plots :

Most of the rides are booked for business purpose. Most of the people book cabs for Meetings and Meal / Entertain purpose. Most of the cabs are booked in the time duration of 10am-5pm (Afternoon).

As we have seen that CATEGORY and PURPOSE columns are two very important columns. So now we will be using OneHotEncoder to categories them.

```
from sklearn.preprocessing import OneHotEncoder

object_cols = ['CATEGORY', 'PURPOSE']

# Creating OneHotEncoder with the updated 'sparse_output' argument
OH_encoder = OneHotEncoder(sparse_output=False)

# Applying OneHotEncoder to the dataset's categorical columns
OH_cols = pd.DataFrame(OH_encoder.fit_transform(dataset[object_cols]))

# Maintaining original dataset index
OH_cols.index = dataset.index

# Using the updated method 'get_feature_names_out'
OH_cols.columns = OH_encoder.get_feature_names_out()

# Dropping the original categorical columns
df_final = dataset.drop(object_cols, axis=1)

# Concatenating the original dataset with the newly created OneHot encoded columns
dataset = pd.concat([df_final, OH_cols], axis=1)

# Display the updated dataset
dataset.head()
```

	START_DATE	END_DATE	START	STOP	MILES	date	time	day-night	CATEGORY_Business	CATEGORY_Personal	PURPOSE_Between Offices	PURPOSE_Customer Visit	PURPOSE_Errand/Supplies	PURPOSE_Meal/Entertain	PURPO
0	2016-01-01 21:11:00	2016-01-01 21:17:00	Fort Pierce	Fort Pierce	5.1	2016-01-01	21.0	Night	1.0	0.0	0.0	0.0	0.0	1.0	
1	2016-01-02 01:25:00	2016-01-02 01:37:00	Fort Pierce	Fort Pierce	5.0	2016-01-02	1.0	Morning	1.0	0.0	0.0	0.0	0.0	0.0	
2	2016-01-02 20:25:00	2016-01-02 20:38:00	Fort Pierce	Fort Pierce	4.8	2016-01-02	20.0	Night	1.0	0.0	0.0	0.0	1.0	0.0	
3	2016-01-05 17:31:00	2016-01-05 17:45:00	Fort Pierce	Fort Pierce	4.7	2016-01-05	17.0	Evening	1.0	0.0	0.0	0.0	0.0	0.0	
4	2016-01-06 14:42:00	2016-01-06 15:49:00	Fort Pierce	West Palm Beach	63.7	2016-01-06	14.0	Afternoon	1.0	0.0	0.0	1.0	0.0	0.0	

Next steps: Generate code with dataset | View recommended plots | New interactive sheet

After that, we can now find the correlation between the columns using heatmap.

```python
# Filter the dataset to include only numeric columns
numeric_cols = dataset.select_dtypes(include=['number'])

# Generate the correlation matrix
corr_matrix = numeric_cols.corr()

# Plotting the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(corr_matrix,
            cmap='BrBG',
            fmt='.2f',
            linewidths=2,
            annot=True)

# Show the plot
plt.show()
```

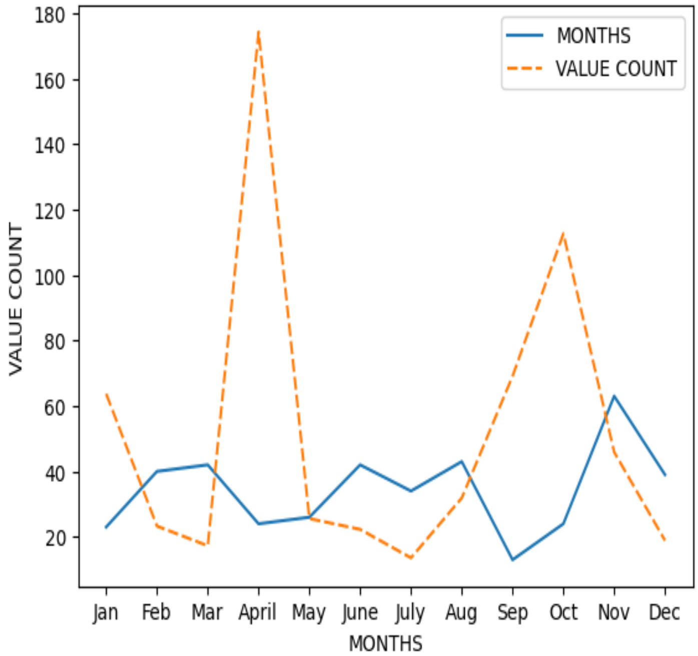
Insights from the heatmap:

Business and Personal Category are highly negatively correlated, this have already proven earlier. So this plot, justifies the above conclusions.

There is not much correlation between the features. Now, as we need to visualize the month data. This can we same as done 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")
```

```
[Text(0.5, 0, 'MONTHS'), Text(0, 0.5, 'VALUE COUNT')]
```



Insights from the above plot :

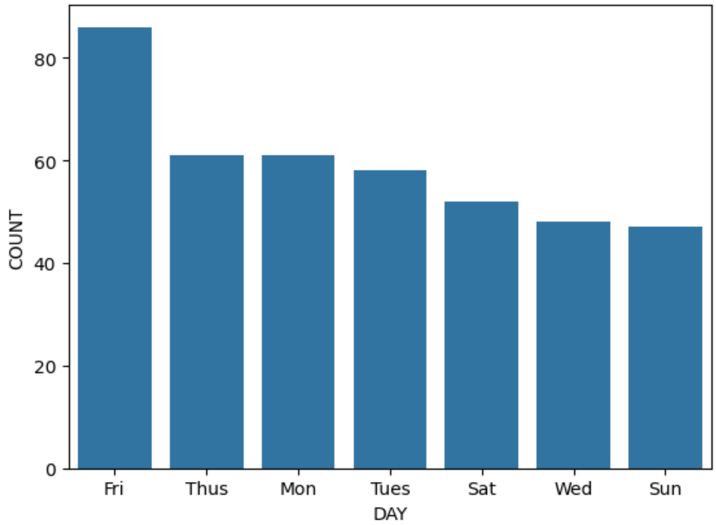
The counts are very irregular. Still its very clear that the counts are very less during Nov, Dec, Jan, which justifies the fact that time winters are there in Florida, US.

Visualization for days data.

```python
[22] 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)
```

```python
[23] day_label = dataset.DAY.value_counts()
     sns.barplot(x=day_label.index, y=day_label);
     plt.xlabel('DAY')
     plt.ylabel('COUNT')
```

```
Text(0, 0.5, 'COUNT')
```

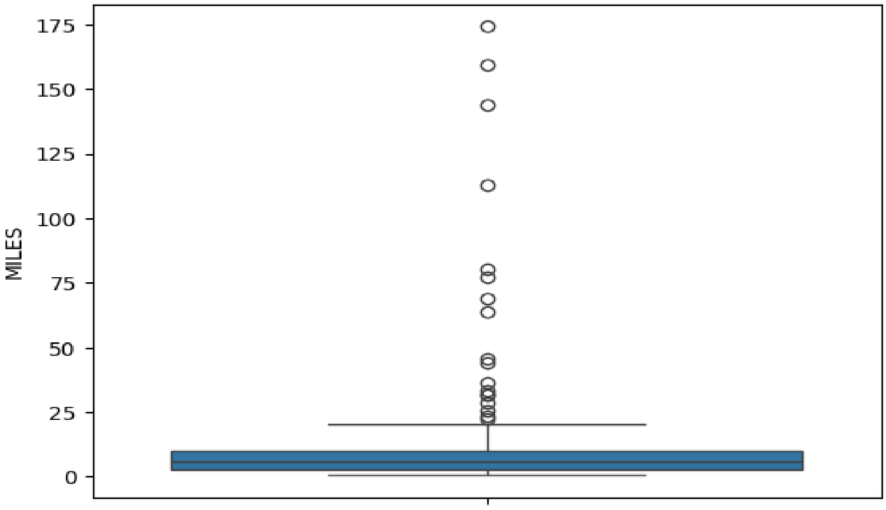


Now, let's explore the MILES Column .

We can use boxplot to check the distribution of the column.

```python
sns.boxplot(dataset['MILES'])
```

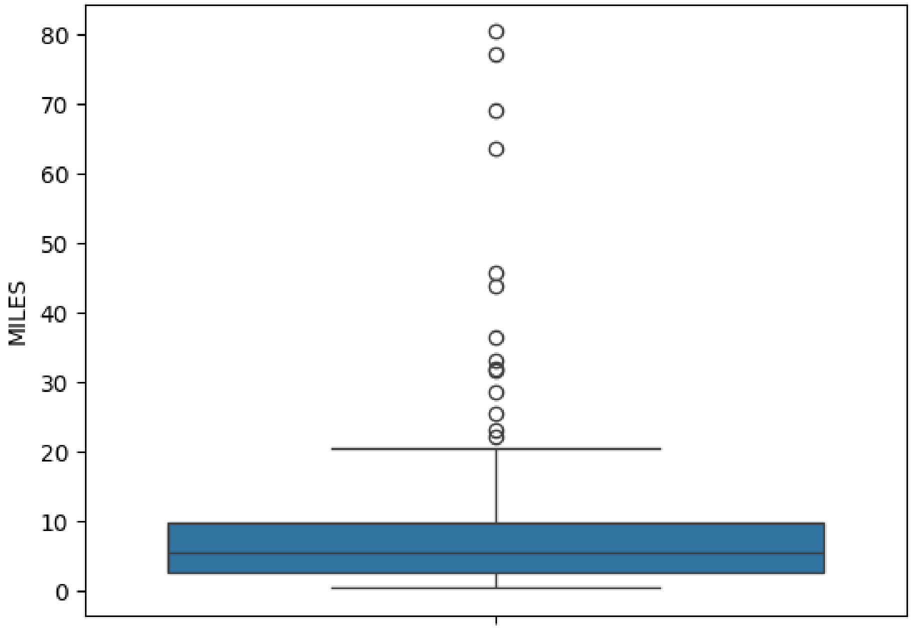
```
<Axes: ylabel='MILES'>
```



As the graph is not clearly understandable. Let's zoom in it for values lees than 100.

```
sns.boxplot(dataset[dataset['MILES']<100]['MILES'])
```

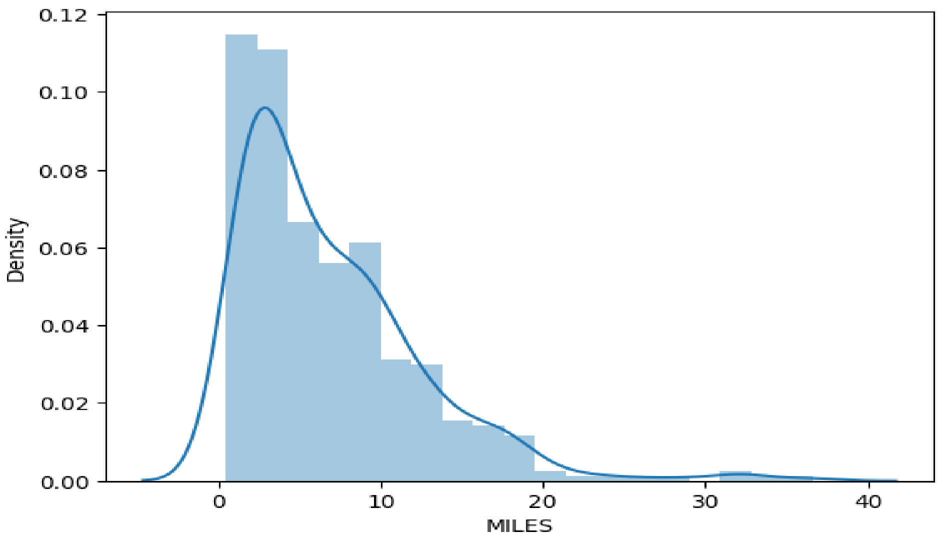
```
<Axes: ylabel='MILES'>
```



It's bit visible. But to get more clarity we can use distplot for values less than 40.

```
[26] sns.distplot(dataset[dataset['MILES']<40]['MILES'])
```

```
sns.distplot(dataset[dataset['MILES']<40]['MILES'])
<Axes: xlabel='MILES', ylabel='Density'>
```



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.

Conclusion

The Uber trip analysis provided valuable insights into the patterns and behaviors related to Uber rides over the analyzed period. We identified key trends in ride frequency, peak times, locations, and fare structures. Specifically:

- **Peak Hours**: The analysis revealed a clear surge in demand during specific time frames, notably during morning and evening rush hours. This reflects the high dependency of users on Uber for commuting purposes.
- **Popular Pickup and Drop-off Locations**: The geographical distribution of rides showed that certain areas experienced higher traffic, indicating potential economic or social hubs.
- **Fare Distribution**: Pricing patterns demonstrated variability based on distance, demand, and time of day, correlating with expected dynamic pricing models.

This analysis helps Uber and its stakeholders understand rider behavior, optimize driver availability, and enhance user experience through better demand prediction. However, limitations such as data gaps and external factors (e.g., weather conditions, special events) may affect the accuracy of these findings.

For future analyses, incorporating additional data sources like customer satisfaction ratings and weather information could offer even more precise insights.