Project title: Uber demand supply gap analysis for Bengaluru, India for the year 2016.

Description:

- Identify the root cause of problem i.e., time slot for which maximum supply demand gap exist.
- Identify the type of requests i.e., Airport to City or City to Airport which is more problematic in case of driver cancellations and non availability of cars.
- Recommend possible ways to improve the situation.

Project work:

- Exploratory data analysis on a masked data set using standard python libraries such as, NumPy, Pandas.
- Optimization of different data points like request status, drop and pickup timestamps.
- Visualization using Matplotlib based graphs to assess the gap between demand and supply of cabs.

Tools:

- Anaconda Navigator-JupyterLab
- Python
- NumPy
- Pandas
- Matplotlib

AutoSave Off | Uber Request Data.csv

File Home Insert Draw Page Layout Formulas Data Review View H

K15

	A	B	C	D	E	F
1	**Request id**	**Pickup point**	**Driver id**	**Status**	**Request timestamp**	**Drop timestamp**
2	619	Airport	1	Trip Completed	11/7/2016 11:51	11/7/2016 13:00
3	867	Airport	1	Trip Completed	11/7/2016 17:57	11/7/2016 18:47
4	1807	City	1	Trip Completed	12/7/2016 9:17	12/7/2016 9:58
5	2532	Airport	1	Trip Completed	12/7/2016 21:08	12/7/2016 22:03
6	3112	City	1	Trip Completed	13-07-2016 08:33:16	13-07-2016 09:25:47
7	3879	Airport	1	Trip Completed	13-07-2016 21:57:28	13-07-2016 22:28:59

Data Extraction

- Standard python libraries used are:
  - NumPy 1.19.2,
  - Pandas 1.2.3,
  - Matplotlib 3.3.4
  - Seaborn 0.11.1

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

- Loading the dataset from csv to pandas data frame.

```
uberdata = pd.read_csv("Uber Request Data.csv")
```

Request id	Pickup point	Driver id	Status	Request timestamp	Drop timestamp
619	Airport	1.0	Trip Completed	11/7/2016 11:51	11/7/2016 13:00
867	Airport	1.0	Trip Completed	11/7/2016 17:57	11/7/2016 18:47
1807	City	1.0	Trip Completed	12/7/2016 9:17	12/7/2016 9:58
2532	Airport	1.0	Trip Completed	12/7/2016 21:08	12/7/2016 22:03
3112	City	1.0	Trip Completed	13-07-2016 08:33:16	13-07-2016 09:25:47

- Checking different columns data types.

```
uberdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6745 entries, 0 to 6744
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Request id         6745 non-null   int64
 1   Pickup point       6745 non-null   object
 2   Driver id          4095 non-null   float64
 3   Status             6745 non-null   object
 4   Request timestamp  6745 non-null   object
 5   Drop timestamp     2831 non-null   object
dtypes: float64(1), int64(1), object(4)
memory usage: 316.3+ KB
```

Data Cleaning

- Checking Null column values:
  - 'Drop timestamp' has 58 percent of null values.
  - 'Driver ID' has 39 percent of null values.
  - These entries are the rides where trip was never assigned to a driver.

```
pd.DataFrame(round((100*(uberdata.isnull().sum()/len(uberdata.index))),2))
```

Request id	0.00
Pickup point	0.00
Driver id	39.29
Status	0.00
Request timestamp	0.00
Drop timestamp	58.03

- Correcting data types of datetime columns:
  - Converting format of 'Request timestamp' and 'Drop timestamp' columns to datetime object.

```
uberdata['Request timestamp'] = pd.to_datetime(uberdata['Request timestamp'])
uberdata['Drop timestamp'] = pd.to_datetime(uberdata['Drop timestamp'])
uberdata.head(10)
```

Request id	Request timestamp	Drop timestamp
619	2016-11-07 11:51:00	2016-11-07 13:00:00
867	2016-11-07 17:57:00	2016-11-07 18:47:00
1807	2016-12-07 09:17:00	2016-12-07 09:58:00

Feature Engineering

- Addition of new columns:
  - 'Request Hours': By extracting hours from 'Request Timestamp' object column.
  - 'Drop Hours': By extracting hours from 'Drop Timestamp' object column.

```
uberdata['Request Hours'] = uberdata['Request timestamp'].apply(lambda x:x.hour)
uberdata['Drop Hours'] = uberdata['Drop timestamp'].apply(lambda x: x.hour)
uberdata.head(5)
```

Request id	Request timestamp	Drop timestamp	Request Hours	Drop Hours
619	2016-11-07 11:51:00	2016-11-07 13:00:00	11	13.0
867	2016-11-07 17:57:00	2016-11-07 18:47:00	17	18.0
1807	2016-12-07 09:17:00	2016-12-07 09:58:00	9	9.0
2532	2016-12-07 21:08:00	2016-12-07 22:03:00	21	22.0
3112	2016-07-13 08:33:16	2016-07-13 09:25:47	8	9.0

- Dividing all requests into different time slots:

```
def determine_time_slot(x):
    if (x >=0 and x < 8):
        return "Early morning hours"     #12am-7am
    elif (x >= 8 and x < 12):
        return "Peak morning hours"      #8am-11am
    elif (x >= 12 and x < 17):
        return "Noon hours"              #12pm-4pm
    elif (x >= 17 and x < 21):
        return "Evening hours"           #5pm-8pm
    elif (x >= 21):
        return "Night hours"             #9pm onwards


uberdata['Request Time Slot'] = uberdata['Request Hours'].apply(determine_time_slot)
uberdata[['Request id','Pickup point','Request Time Slot']].head(5)
```

Request id	Pickup point	Request Time Slot
619	Airport	Peak morning hours
867	Airport	Evening hours
1807	City	Peak morning hours
2532	Airport	Night hours
3112	City	Peak morning hours

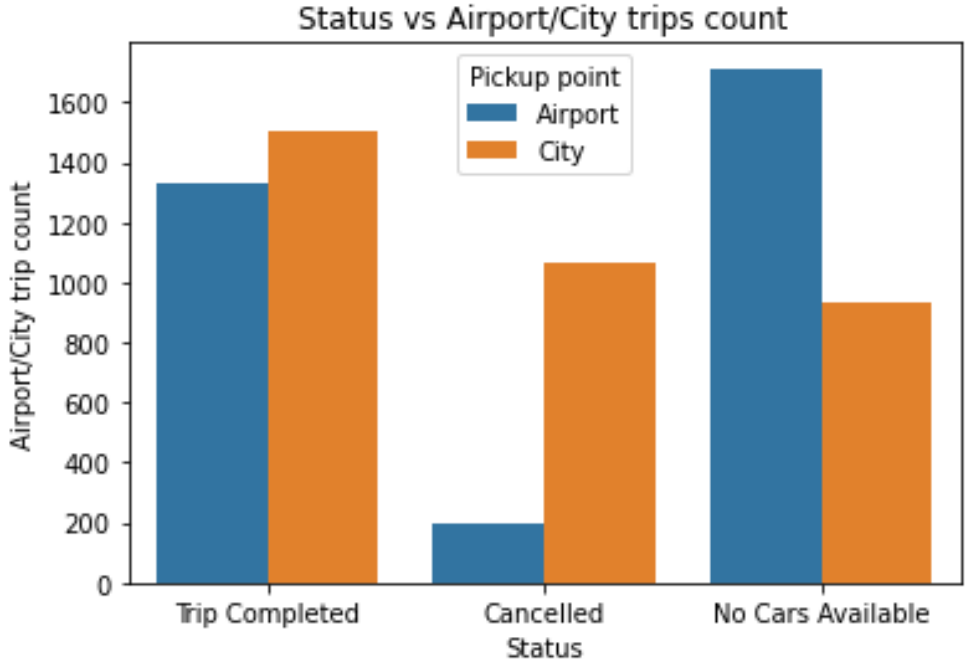
Plots and Observations

- Most problematic pickup point status wise:

```
sns.countplot(x="Status",hue="Pickup point",data = uberdata)
plt.title('Status vs Airport/City trips count')
plt.ylabel('Airport/City trip count')
```

Table 1. Distribution of airport/City trips status wise

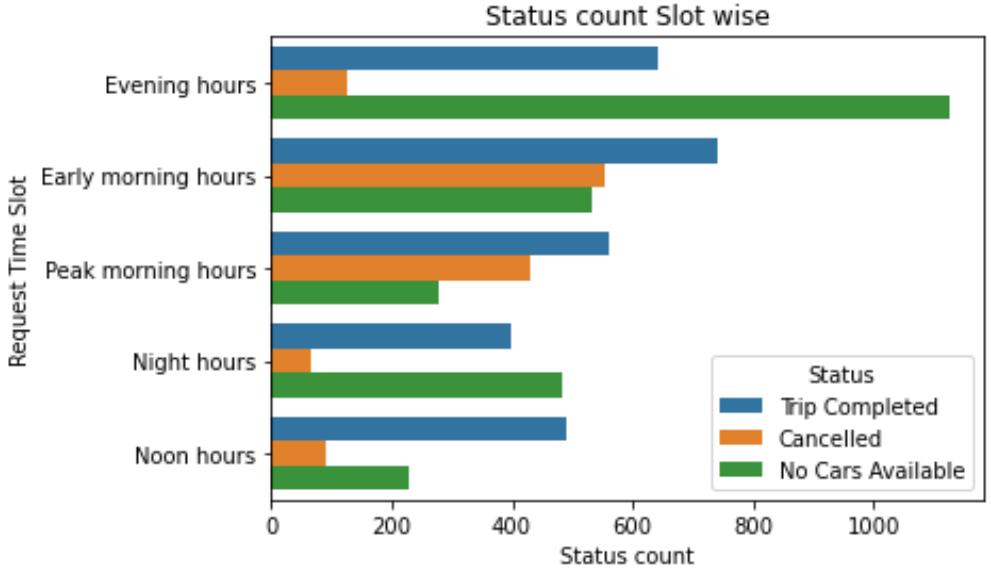
Status	Pickup Point	
	Airport	City
No Cars Available	1600-1700	900
Cancelled	175-200	1400-1500
Trip Completed	1300-1350	1500



- Most problematic time slots where rides were unsuccessful:

Table 1. Problematic time slots

Status	Request Time Slot	
	Evening	Early Morning
No Cars Available	1100-1200	500-600
Cancelled	100	550
Trip Completed	700	750-775



Code Execution

- Execution of jupyter notebook

1. Data Extraction: Load the data set and inspect different features.
2. Data Cleaning: Correcting datetime formats and filling in missing values.
3. Feature Engineering: Determine new features.
4. Plotting: Create plots to see correlation between data points and draw important insights out of data.

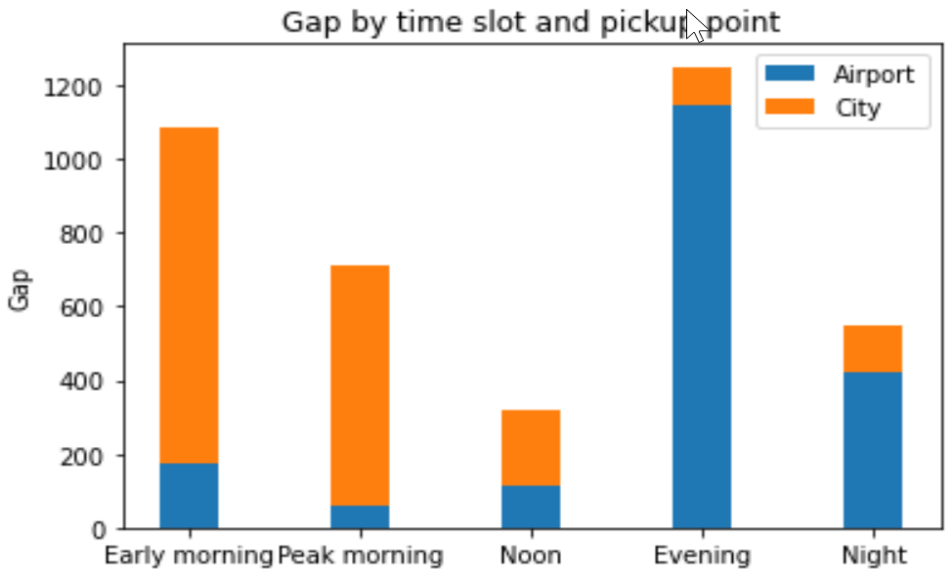
Summary of Results

– Gap for airport pickup point is maximum in evening.

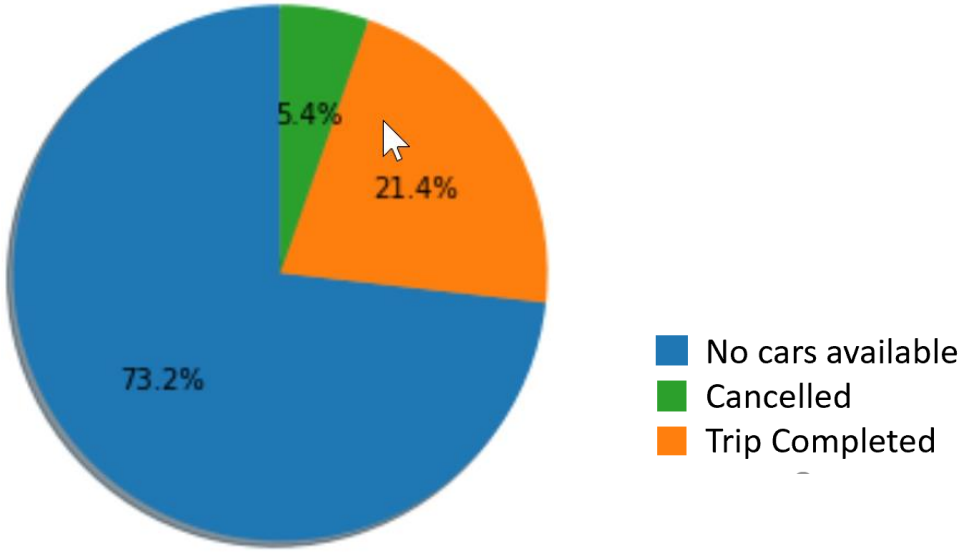
Request Time Slot	Demand_From_Airport	Supply_To_Airport	Gap_From_Airport
Evening	1457	312	1145

– Gap for city pickup point is maximum in early morning.

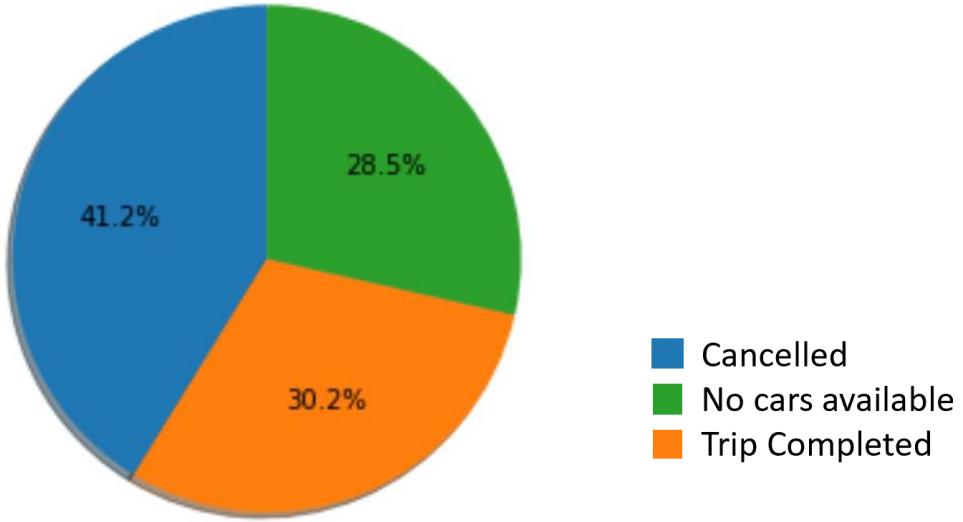
Request Time Slot	Demand_From_City	Supply_To_City	Gap_From_City
Early morning	1310	396	914



– For 73 % of total airport requests cars were not available.



– 41% of city requests were cancelled by the drivers.



Conclusion

- Recommendations to solve the problem:
  - For the trips in the early morning, drivers can be incentivized to make those trips.
    1. Uber can pay for the gas mileage of drivers to come back to the city without a rider.
    2. Uber can increase the demand at the airport to reduce idle time – by increased marketing and price cuts for the passengers.
    3. Uber can request a feedback from drivers to understand reasons behind ride cancellation.
  - For the trips in evening, some of the ways are:
    1. Uber can also pay drivers to come without a passenger to the airport.
    2. Another innovative way can be to encourage passengers to pool the ride with others so that lesser number of cars can serve more passengers.
- Learnings:
  - Use of python libraries like Matplotlib and Seaborn for plotting graphs.
  - Series of basic steps to be followed to prepare data for training prediction models.
- Future work:
  - With the addition of uber requests data for additional regions, a more generalized analysis can be done to identify the problems resulting in revenue loss for Uber globally.
- Challenges encountered:
  - Understanding the correct use of different plots in matplotlib and seaborn libraries.