  
 Projexino Web Technologies .LLP

WWW.Projexino.com

**Analysis of Uber Data and Status Prediction**

Project report

CH. Vamsi Krishna

Intern

**CONTENTS**

**S.NO TITLE PAGE NO**

**0**  **ABSTRACT 3**

**1 INTRODUCTION 4**

**2 DETAILS OF DATASET 4**

**3 Business Objectives 5**

**4 Importing modules 6**

**5 Importing Data 7**

**6 methods used in Data cleaning 9**

**7 Data Cleaning 10**

**8 VISUVALIZATIONS 11-15**

**9 Machine Learning model 16-23**

**10 CONCLUSION 24**



Analysis of uber data

**ABSTRACT**

This project’s main theme is to analysis the data to provide useful l information to reduce waiting time,

Project consists of a machine learning model to predict the status if requested trip based on previous

Trips.

Project will be built using python and Data Science modules like Pandas ,Numpy ,Seaborn, matplotlib, Seaborn, Skleatn, iplot , ETC..

This project is useful for people who uses uber services and wants to know the status of their trip.

**Intern name**

**CH.Vamsi Krishna (16g21a0522)**

**Introduction**

* This directory contains data on over 6 thousand Uber pickups in New York City 11th to 15th July of 2016, as well as aggregated data for 329 FHV companies, is also included. All the files are as they were received on on the same month
* obtained the data from the [NYC Taxi & Limousine Commission (TLC)](http://www.nyc.gov/html/tlc/html/home/home.shtml) by submitting a Freedom of Information Law request on July 15, 2016. The TLC has sent us the data in batches as it continues to review trip data Uber and other HFV companies have submitted to it. The TLC's
* Simple Exploratory Data Analysis
* Data preprocessing
* Machine Learning Model

#### Data Preprocessing:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in the certain behaviors or trends and is likely to contain many errors. Data preprocessing is the proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

#### Details of Dataset:

Uber dataset consists the data of 6000 customers .

Dataset has 5 columns and 6000 rows

Columns

Request id

Pickup point

Driver id

Status

Request timestamp

Drop timestamp

**Business Objective**

**Analyzing and identifying the root cause of the problem (i.e. cancellation and non-availability of cars)**

Recommend ways to improve the situation.

Possible hypotheses of the problem

Problems faced by customers- Impact on Uber Business

If drivers cancel the request of riders or if cars are unavailable, Uber loses out on its revenue.

### importing modules:

**In [1]:**

*"""Pandas is a Python package designed to do work with “labeled” and “relational” data simple and intuitive.*

*Pandas is a perfect tool for data wrangling.*

*It designed for quick and easy data manipulation, aggregation, and visualization.*

*"""*

**import pandas as pd**

*"""The most fundamental package, around which the scientific computation stack is built, is NumPy*

*(stands for Numerical Python).*

*It provides an abundance of useful features for operations on n-arrays and matrices in Python.*

*"""*

**import numpy as np**

*"""*

*Python Library that is tailored for the generation of simple and powerful visualizations with ease is Matplotlib.*

*"""*

**import matplotlib.pyplot as plt**

*"""*

*Seaborn is mostly focused on the visualization of statistical models;*

*such visualizations include heat maps,*

*those that summarize the data but still depict the overall distributions.*

*"""*

**import seaborn as sns**

**%matplotlib inline**

**from datetime import datetime**

### importing data

In [2]:

data = pd.read\_csv("Uber\_Request\_Data.csv")

data.head(9)

Out[2]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Request id Pickup point** | | **Driver id** | | **Status** | **Request timestamp** | **Drop timestamp** |
|  |  |  |  |  |  |  |  |  |
| **0** | | 619 | Airport |  | 1.0 | Trip Completed | 11/7/2016 11:51 | 11/7/2016 13:00 |
| **1** | | 867 | Airport |  | 1.0 | Trip Completed | 11/7/2016 17:57 | 11/7/2016 18:47 |
| **2** | | 1807 | City |  | 1.0 | Trip Completed | 12/7/2016 9:17 | 12/7/2016 9:58 |
| **3** | | 2532 | Airport |  | 1.0 | Trip Completed | 12/7/2016 21:08 | 12/7/2016 22:03 |
| **4** | | 3112 | City |  | 1.0 | Trip Completed | 13-07-2016 08:33:16 | 13-07-2016 09:25:47 |
| **5** | | 3879 | Airport |  | 1.0 | Trip Completed | 13-07-2016 21:57:28 | 13-07-2016 22:28:59 |
| **6** | | 4270 | Airport |  | 1.0 | Trip Completed | 14-07-2016 06:15:32 | 14-07-2016 07:13:15 |
| **7** | | 5510 | Airport |  | 1.0 | Trip Completed | 15-07-2016 05:11:52 | 15-07-2016 06:07:52 |
| **8** | | 6248 | City |  | 1.0 | Trip Completed | 15-07-2016 17:57:27 | 15-07-2016 18:50:51 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In [3]: data.info() |  |  |  |  |  |  |
| <class 'pandas.core.frame.DataFrame'> | | | | |  |  |
| RangeIndex: 6745 entries, | | | 0 to 6744 | |  |  |
| Data columns (total 6 columns): | | | |  |  |  |
| Request id | 6745 | | non-null int64 | |  |  |
| Pickup point | 6745 | | non-null object | |  |  |
| Driver id | 4095 | | non-null float64 | |  |  |
| Status | 6745 | | non-null object | |  |  |
| Request timestamp | 6745 | | non-null object | |  |  |
| Drop timestamp | 2831 | | non-null object | |  |  |

dtypes: float64(1), int64(1), object(4)

memory usage: 316.2+ KB

In [4]: data1 **=**data[["Pickup point","Status",

"Request timestamp","Drop timestamp"]]

data1.head(9)

Out[4]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Pickup point** | **Status** | **Request timestamp** | **Drop timestamp** |
|  |  |  |  |  |
| **0** | Airport | Trip Completed | 11/7/2016 11:51 | 11/7/2016 13:00 |
| **1** | Airport | Trip Completed | 11/7/2016 17:57 | 11/7/2016 18:47 |
| **2** | City | Trip Completed | 12/7/2016 9:17 | 12/7/2016 9:58 |
| **3** | Airport | Trip Completed | 12/7/2016 21:08 | 12/7/2016 22:03 |
| **4** | City | Trip Completed | 13-07-2016 08:33:16 | 13-07-2016 09:25:47 |
| **5** | Airport | Trip Completed | 13-07-2016 21:57:28 | 13-07-2016 22:28:59 |
| **6** | Airport | Trip Completed | 14-07-2016 06:15:32 | 14-07-2016 07:13:15 |
| **7** | Airport | Trip Completed | 15-07-2016 05:11:52 | 15-07-2016 06:07:52 |

**8** City Trip Completed  15-07-2016 17:57:27  15-07-2016 18:50:51

**methods used in Data cleaning and Data preprocessing:**

**Method 1**

**def**to\_date(dat): dat **=**str(dat) **if**len(dat)**>**3:

**if "-" in dat:**

**try:**

datetime.strptime(dat,"%d-%m-%Y %H:%M:%S")

**except**:

**return**datetime.strptime(dat,"%d-%m-%Y %H:%M")

 **elif**"/" **in**dat:

**try:**

**return**datetime.strptime(dat,"%d/%m/%Y %H:%M:%S")

**except**:

**return**datetime.strptime(dat,"%d/%m/%Y %H:%M")

**else:**

**return np.NaN**

**Method 2**

**def**tday(per):

**return**per.strftime("%A")

**Method 3**

|  |
| --- |
| timestamp1 **=**"04:00:00" |
|  | timestamp2 **=**"10:00:00" |
|  | timestamp3 **=**"16:00:00" |

timestamp4 **=**"22:00:00"

t1 **=**datetime.strptime(timestamp1, "%H:%M:%S")

t2 **=**datetime.strptime(timestamp2, "%H:%M:%S")

t3 **=**datetime.strptime(timestamp3, "%H:%M:%S")

t4 **=**datetime.strptime(timestamp4, "%H:%M:%S")

**def**to\_time(per): per **=**per.time() slot **=**""

**if**per **>=**t1.time() **and**per **<=**t2.time(): slot **=**"morning"

**elif**per **>**t2.time() **and**per **<=**t3.time(): slot **=**"daytime"

**elif**per **>**t3.time() **and**per **<=**t4.time(): slot **=**"evening"

**else:**

slot **=**"midnight" **return**slot

**Method 4:**

**def**time\_in\_sec(per):

**try:**

**return per.total\_seconds() except:**

**return pd.NaT**

**Method 5:**

**def**tmonth(per):

**return**per.strftime("%B")

**DATA CLEANING :**

Above defined functions are applied to the Dataset in data cleaning phase

In [11]:

data1["Drop timestamp"] **=**data1["Drop timestamp"].apply(to\_date)

data1["Request timestamp"] **=**data1["Request timestamp"].apply(to\_date)

data1["time taken"] **=**data1["Drop timestamp"]**-**data1["Request timestamp"]

data1["total\_time\_in\_sec"]**=**data1["time taken"].apply(time\_in\_sec)

data1["day of weak"]**=**data1["Request timestamp"].apply(tday)

data1["month"] **=**data1["Request timestamp"].apply(tmonth)

data1["time\_slot"] **=**data1["Request timestamp"].apply(to\_time)

In [12]: data1.info()

<class 'pandas.core.frame.DataFrame'>

|  |  |  |
| --- | --- | --- |
| RangeIndex: 6745 entries, | | 0 to 6744 |
| Data columns (total 9 columns): | | |
| Pickup point | 6745 | non-null object |
| Status | 6745 | non-null object |
| Request timestamp | 6745 | non-null datetime64[ns] |
| Drop timestamp | 2831 | non-null datetime64[ns] |
| time taken | 2831 | non-null timedelta64[ns] |
| total\_time\_in\_sec | 2831 | non-null float64 |
| day of weak | 6745 | non-null object |
| month | 6745 | non-null object |
| time\_slot | 6745 | non-null object |

dtypes: datetime64[ns](2), float64(1), object(5), timedelta64[ns](1)

memory usage: 474.3+ KB

the above information shows the structure and data types and null values of cleaned data

In [13]: data1.head()

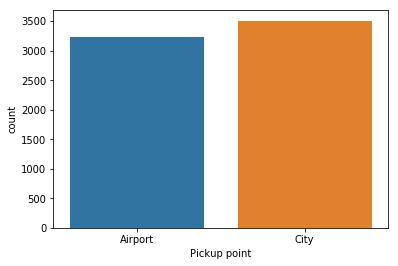
Out[13]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | **Pickup** | **Status** | **Request** | **Drop** | **time** | **total\_time\_in\_sec** | **day of** | **month** | | **time\_** |
|  |  |  |  | **point** | **timestamp** | **timestamp** | **taken** | **weak** |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **0** | | | Airport | Trip | 2016-07-11 | 2016-07-11 | 01:09:00 | 4140.0 | Monday | July | | day |
|  | Completed | 11:51:00 | 13:00:00 |
|  | **1** | | | Airport | Trip | 2016-07-11 | 2016-07-11 | 00:50:00 | 3000.0 | Monday | July | | eve |
|  | Completed | 17:57:00 | 18:47:00 |
|  | **2** | | |  | Trip | 2016-07- | 2016-07- |  |  |  |  |  |  |
|  | City | 12 | 12 | 00:41:00 | 2460.0 | Tuesday | July | | mor |
|  | Completed |
|  |  |  |  |  | 09:17:00 | 09:58:00 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | **3** | | |  | Trip | 2016-07- | 2016-07- |  |  |  |  |  |  |
|  | Airport | 12 | 12 | 00:55:00 | 3300.0 | Tuesday | July | | eve |
|  | Completed |
|  |  |  |  |  | 21:08:00 | 22:03:00 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | **4** | | |  | Trip | 2016-07- | 2016-07- |  |  |  |  |  |  |
|  | City | 13 | 13 | 00:52:31 | 3151.0 | Wednesday | July | | mor |
|  | Completed |
|  |  |  |  |  | 08:33:16 | 09:25:47 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

**Data Visuvalisation:**

In [14]: sns.countplot(x **=**"Pickup point",data **=** data1)

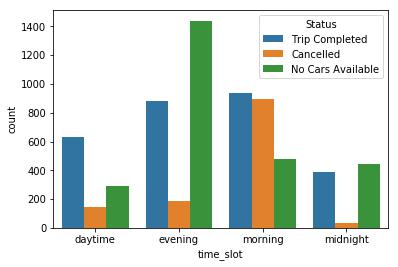
Out[14]: <matplotlib.axes.\_subplots.AxesSubplot at 0x24d11efe898>



From the above visuvalisation the no of customers at Airport is almost equal to no on customers in the city

In [15]: sns.countplot(x **=** "time\_slot",data**=** data1,hue **=** "Status")

Out[15]: <matplotlib.axes.\_subplots.AxesSubplot at 0x24d11e12d68>



Data has been divided into time slots:

Early Morning(04:00-09:59 A.M)

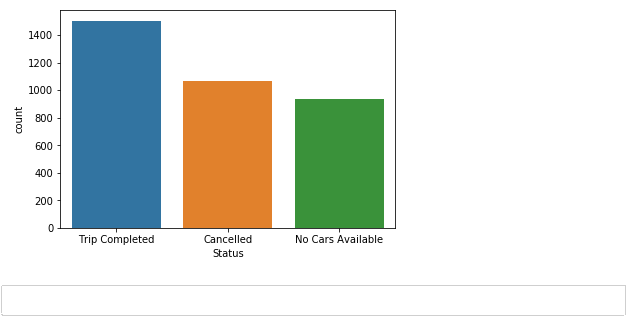
Day Time (10:00 A.M-04:50 P.M)

Late Evening (05:00 P.M-11:59 P.M)

Late Night(12 Midnight-03:59 A.M)

In [16]: sns.countplot(x **=** "Status",data**=** data1[data1["Pickup point"]**==**"City"])

Out[16]: <matplotlib.axes.\_subplots.AxesSubplot at 0x24d13175cc0>



❖Number of cars cancelled - 1264

❖ Number of No Cars Available- 2650

❖ Number of Trip Completed -2831

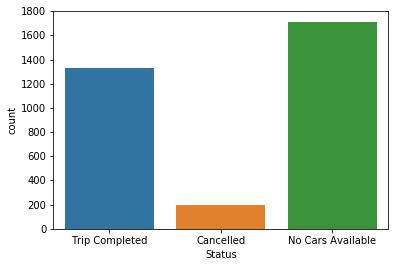
As 68% of requests are not being addressed, it is majorly impacting the UBER business

In [17]:

sns.countplot(x **=** "Status",data**=** data1[data1["Pickup point"]**!=**"City"])

Out[17]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x24d131ca6d8>



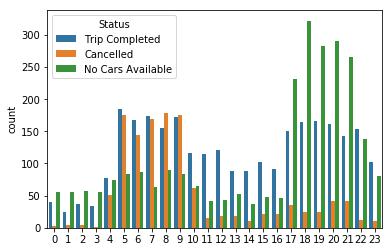
In [18]: a**=** []

**for** i **in** range(len(data1["Request timestamp"])):

a.append(data1["Request timestamp"][i].hour)

sns.countplot(x **=**a ,hue**=**data1["Status"])

Out[18]: <matplotlib.axes.\_subplots.AxesSubplot at 0x24d13247b70>



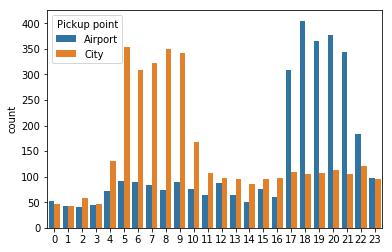
❖Only 42% of the requests are getting completed

❖Unavailability of cars is maximum at around 7 P.M.

❖Cancellation is maximum at around 9 A.M

In [19]: sns.countplot(x **=**a ,hue**=**data1["Pickup point"])

Out[19]: <matplotlib.axes.\_subplots.AxesSubplot at 0x24d13348fd0>



In [20]: sns.countplot(x **=**data1["time\_slot"] ,hue**=**data1["Pickup point"])

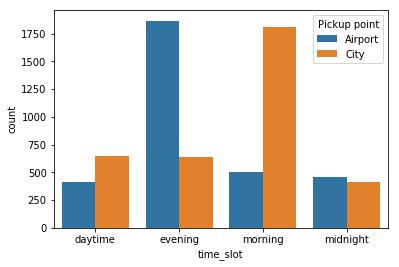
"""more evening requestes at airport

more morning requeses at city

"""

Out[20]:

'\n\n more evening requestes at airport\n more morning requeses at city\n





**Machine Learning:**

In [21]: ml\_data **=** data1[["Pickup point","Status","total\_time\_in\_sec","day of weak","time ml\_data.head()

Out[21]:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Pickup point** | **Status** | **total\_time\_in\_sec** | **day of weak** | **time\_slot** |
|  |  |  |  |  |  |
| **0** | Airport | Trip Completed | 4140.0 | Monday | daytime |
| **1** | Airport | Trip Completed | 3000.0 | Monday | evening |
| **2** | City | Trip Completed | 2460.0 | Tuesday | morning |
| **3** | Airport | Trip Completed | 3300.0 | Tuesday | evening |
| **4** | City | Trip Completed | 3151.0 | Wednesday | morning |

Above ml\_data is estracted from data1 and it will be used to implement a machine learning model

In [22]:

X**=** ml\_data.iloc[:,[0,3,4]]

X.head()

Out[22]:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Pickup point** | **day of weak** | **time\_slot** |
|  |  |  |  |
| **0** | Airport | Monday | daytime |
| **1** | Airport | Monday | evening |
| **2** | City | Tuesday | morning |
| **3** | Airport | Tuesday | evening |
| **4** | City | Wednesday | morning |

Ml\_data is divided into two datasets ie: Dependent Data and independent Data

Where X is Indipendent and y is dependent

We train the model with independent dsts to predict the dependent vlues

**In [23]:**

In [23]: X **=** ml\_data.iloc[:,[0,3,4]].values

y **=** ml\_data.iloc[:, [1]].values

**from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoderlabelencoder\_0 **=** LabelEncoder()

X[:, 0] **=** labelencoder\_0.fit\_transform(X[:, 0])

X[:, 1] **=** labelencoder\_0.fit\_transform(X[:, 1])

X[:, 2] **=** labelencoder\_0.fit\_transform(X[:, 2])

onehotencoder **=** OneHotEncoder(categorical\_features **=** [0,1,2])

X **=** onehotencoder.fit\_transform(X).toarray()

In [24]: pd.DataFrame(X).head()

Out[24]:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **0** | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| **1** | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **2** | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **3** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **4** | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 |

The ml\_data is encoded inorder to get accurate results, encoding is done as follows

airport = 1

city = 0

monday = 0100

tuesday = 0001

wednesday = 0000

thursday = 0010

friday = 1000

daytime = 100

evening = 010

morning = 000

midnight = 001

In [25]: X**=**pd.DataFrame(X[:,[0,2,3,4,5,7,8,9]]).values

pd.DataFrame(X).head()

Out[25]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|  |  |  |  |  |  |  |  |  |
| **0** | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **1** | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| **3** | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

From the encoded data dummy valus are removed in order to prevent dummy veriable trap

In [26]: y

Out[26]: array([['Trip Completed'],

['Trip Completed'],

['Trip Completed'],...,

['No Cars Available'],

['No Cars Available'],

['No Cars Available']], dtype=object)

In [27]: **from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoderlabelencoder\_y **=** LabelEncoder()

y **=** labelencoder\_y.fit\_transform(y)

|  |  |
| --- | --- |
| In [28]: | y **=** ml\_data.iloc[:, [1,0]].values |
|  | **from** sklearn.preprocessing **import** LabelEncoder, OneHotEncoder |
|  | labelencoder\_y **=** LabelEncoder() |
|  | y[:,0] **=** labelencoder\_y.fit\_transform(y[:,0]) |
|  | y[:,1] **=** labelencoder\_y.fit\_transform(y[:,1]) |
|  | onehotencoder **=** OneHotEncoder(categorical\_features **=** [0]) |
|  | y **=** onehotencoder.fit\_transform(y).toarray() |

|  |  |  |
| --- | --- | --- |
| In [30]: | y**=**y[:,:**-**1] |  |
| In [31]: | y **=** y[:,:**-**1] | |
| In [32]: | y |  |
| Out[32]: array([[0., | | 0.], |
|  | [0., | 0.], |
|  | [0., | 0.], |
|  | ..., |  |
|  | [0., | 1.], |
|  | [0., | 1.], |
|  | [0., | 1.]]) |

Y value is encodesd in the same way as X

**Random Forest Regrassion Model:**

In [33]: **from** sklearn.ensemble **import** RandomForestRegressor

regressor **=** RandomForestRegressor(n\_estimators **=** 10, random\_state **=** 0)

regressor.fit(X, y)

In [34]: **def** ml\_array(per1,per2):

day\_dic**=**{"Monday" : "0100","Tuesday" : "0001","Wednesday" : "0000","Thursday

timeslot\_dict **=** {"daytime" : "100","evening" : "010","morning" : "000","midn

predict\_dict **=** {"00" : "trip will be completed","01":"no cars available","10

arr **=** []

pre\_arr**=**""

arr.append(int(per2))

**for** i **in** day\_dic[tday(per1)]:

arr.append(int(i))

**for** i **in** timeslot\_dict[to\_time(per1)]:

arr.append(int(i))

temp **=** regressor.predict([arr])

*#predict\_dict = {"00" : "trip will be completed","01":"no cars available","1* **try**:

**return** predict\_dict[str(int(temp[0][0]))**+**str(int(temp[0][1]))] **except**:

**return** "enter proper date"

the above function is used to fit and transform the training data and predict the result

In [35]: d **=** to\_date(input("enter date\n format day-month-year hour:min:sec/\n\n"))

p **=** input("airport = 1\ncity = 0\n\n")

ml\_array(d,p)

enter date

format day-month-year hour:min:sec/

10-02-2020 12:20:00

airport = 1

city = 0

1

Out[35]: 'trip will be completed'

**Conclusion :**

1. the number of requests are slightly higher in the city, the problem is most severe at the airport.
2. During Evening slot (05:00 P.M-11:59 P.M),at the airport the gap is around 53% due to unavailability of the cars which is severely impacting business
3. The second major impact on business is in the city due to cancellation of cars by drivers in the Early Morning(04:00-09:59 A.M) slot. Around 1066 requests get cancelled

**Ways to resolve Supply-Demand Gap**:

1. Deploy extra vehicles at the airport from 5P.M to 11P.M
2. Deploy extra vehicles at the cityt from Morning(04:00-09:59 A.M)

Machine learning model can be used by the customers to predict status of their trip by providing date and place as input

The End

Thank You