PROJECT REPORT

Zomato Delivery Time Prediction

Submitted By:

Arpan Das
Glen Gonsalves
Pratik Karnik
Vishwajeet Gosavi

Course / Batch: DSP 10

`

Acknowledgement

We are using this opportunity to express our gratitude to everyone who supported us throughout the course. We are thankful for their guidance, advice and constructive criticism.

We wish to thank all the faculties for their time and effort for training us for the duration of this course.

We certify that the work done for completing this project is original and authentic.

Date: 22nd Dec, 2019.

Place: Mumbai

Group Members: Arpan Das

Glen Gonsalves

Pratik Karnik

Vishwajeet Gosavi

Abstract

This Project is to predict food delivery time. In order to find the best food delivery time for online orders across different locations, we have analysed the data accordingly and evaluated various classification models to derive the best possible model which has highest recall and precision.

These data include the data visualization as well as the aspects covered for data analysis.

Introduction

Businesses are becoming more and more competitive day after day. The food industry is no better. Getting our foods delivered at the doorstep anytime anywhere is easier than ever. Thanks to the modern incredible food delivery apps. All we need to do is just download the app, choose our favourite food, place an order and make the payment right from our mobile phones.

Now it's time for the delivery team to deliver the piping hot food as quickly as possible. Late or lukewarm food leads to bad order. Even small misconceptions of 5 to 10 minutes delay can make a big difference. So it's imperative to predict the accurate delivery time to stay ahead of the competition.

When was the last time you ordered food online? And how long did it take to reach you? Our goal is to predict the online order delivery time based on the given factors. Food Delivery giant Zomato on Dec 13 2019 launched a scheme "On Time or Free" whereby it refunds the delivery amount in case, the food is not delivered as per estimated time.

Under this scheme they are charging the customers to opt for the scheme and also if the expected delivery time is not met they are refunding the customers certain value of the order. In order to sustain and make a business sensibility for such scheme it is imperative they should get their predicted delivery time as accurate as possible.

zomato
ON-TIME
OR FREE
₹20
Opt-in fee
Get up to ₹200 if food doesn't reach
50 meters of your location on time

Table of Contents

Chapter 1: Dataset Information

1. Dataset Information:

Dataset contains data from thousands of restaurants in India regarding the time they take to deliver food for online order. As data scientists, our goal is to predict the online order delivery time based on the given factors.

Size of training set: 11,094 records

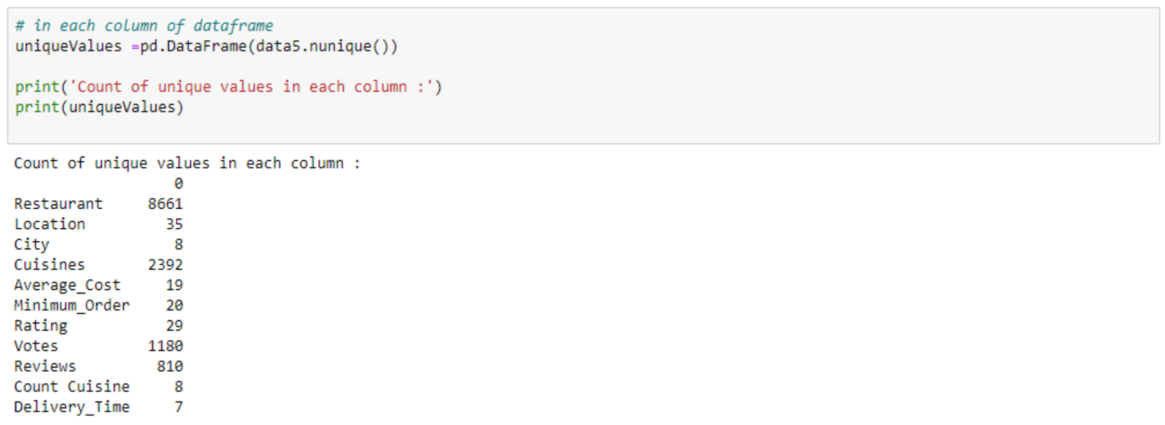
Size of test set: 2,774 records

FEATURES:

- Restaurant: A unique ID that represents a restaurant.
- Location: The location of the restaurant.
- Cuisines: The cuisines offered by the restaurant.
- Average_Cost: The average cost for one person/order.
- Minimum_Order: The minimum order amount.
- Rating: Customer rating for the restaurant.
- Votes: The total number of customer votes for the restaurant.
- Reviews: The number of customer reviews for the restaurant.

LABEL/TARGET:

- Delivery_Time: The order delivery time of the restaurant. (Target Classes)

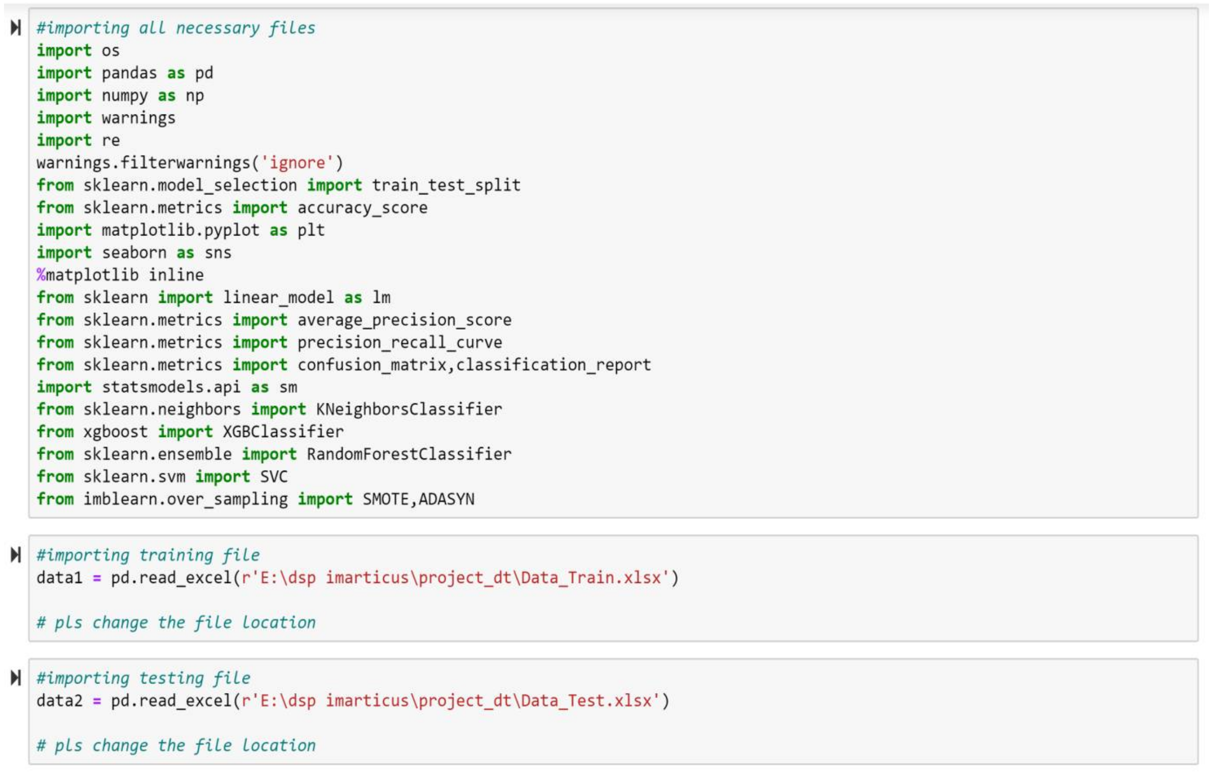


```
# in each column of dataframe
uniqueValues =pd.DataFrame(data5.nunique())

print('Count of unique values in each column :')
print(uniqueValues)
```

```
Count of unique values in each column :
                  0
Restaurant     8661
Location         35
City              8
Cuisines       2392
Average_Cost     19
Minimum_Order    20
Rating           29
Votes          1180
Reviews         810
Count Cuisine     8
Delivery_Time     7
```

Chapter 2: Data Cleaning

2.1: Import files

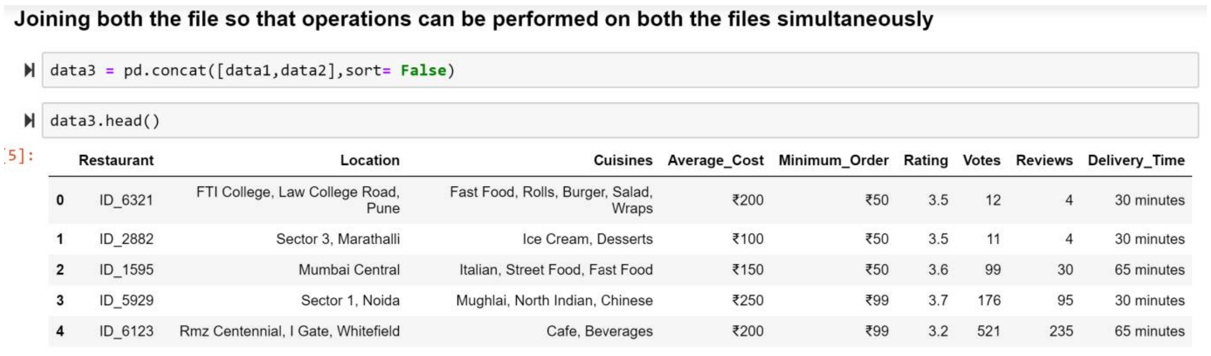


```
#importing all necessary files
import os
import pandas as pd
import numpy as np
import warnings
import re
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn import linear_model as lm
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix,classification_report
import statsmodels.api as sm
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE,ADASYN
```

```
#importing training file
data1 = pd.read_excel(r'E:\dsp imarticus\project_dt\Data_Train.xlsx')

# pls change the file location
```

```
#importing testing file
data2 = pd.read_excel(r'E:\dsp imarticus\project_dt\Data_Test.xlsx')

# pls change the file location
```

2.2: Read the data



Joining both the file so that operations can be performed on both the files simultaneously

```
data3 = pd.concat([data1,data2],sort= False)
```

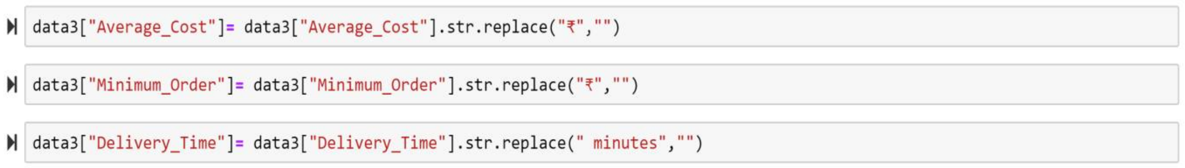
```
data3.head()
```

5]:

	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time
0	ID_6321	FTI College, Law College Road, Pune	Fast Food, Rolls, Burger, Salad, Wraps	₹200	₹50	3.5	12	4	30 minutes
1	ID_2882	Sector 3, Marathalli	Ice Cream, Desserts	₹100	₹50	3.5	11	4	30 minutes
2	ID_1595	Mumbai Central	Italian, Street Food, Fast Food	₹150	₹50	3.6	99	30	65 minutes
3	ID_5929	Sector 1, Noida	Mughlai, North Indian, Chinese	₹250	₹99	3.7	176	95	30 minutes
4	ID_6123	Rmz Centennial, I Gate, Whitefield	Cafe, Beverages	₹200	₹99	3.2	521	235	65 minutes

2.3: Remove rupee symbol and extracting city form address

To eliminate ₹ sign

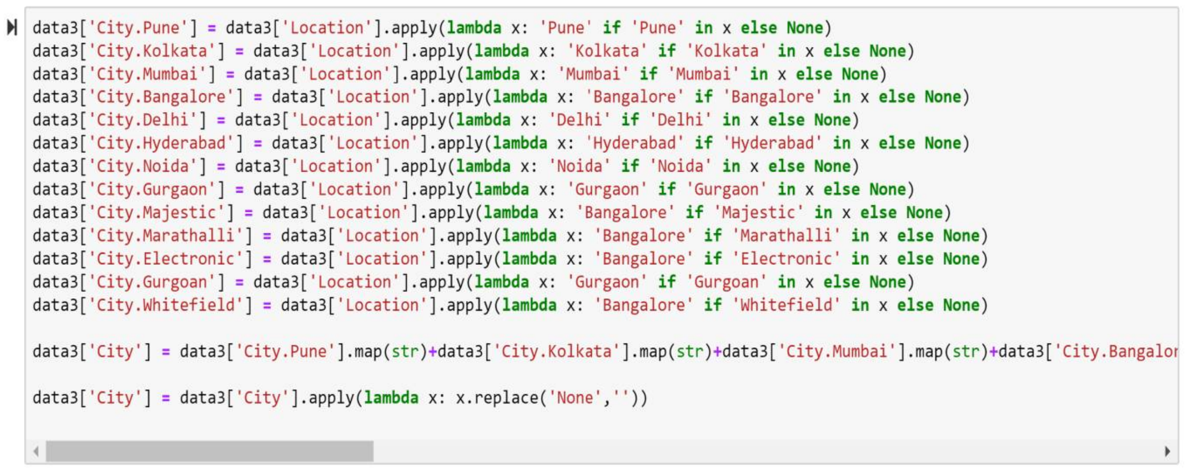


```
data3["Average_Cost"]= data3["Average_Cost"].str.replace("₹","")
```

```
data3["Minimum_Order"]= data3["Minimum_Order"].str.replace("₹","")
```

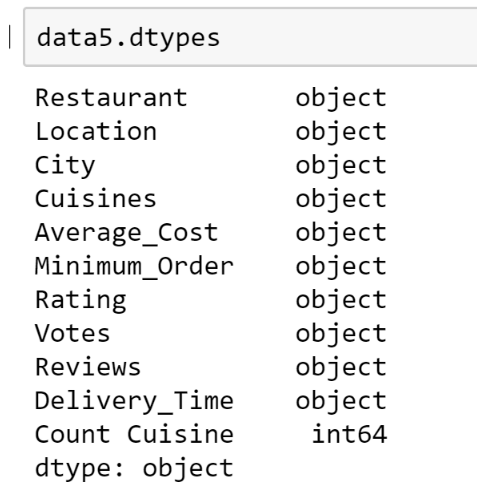
```
data3["Delivery_Time"]= data3["Delivery_Time"].str.replace(" minutes","")
```

Extracting city from address



```
data3['City.Pune'] = data3['Location'].apply(lambda x: 'Pune' if 'Pune' in x else None)
data3['City.Kolkata'] = data3['Location'].apply(lambda x: 'Kolkata' if 'Kolkata' in x else None)
data3['City.Mumbai'] = data3['Location'].apply(lambda x: 'Mumbai' if 'Mumbai' in x else None)
data3['City.Bangalore'] = data3['Location'].apply(lambda x: 'Bangalore' if 'Bangalore' in x else None)
data3['City.Delhi'] = data3['Location'].apply(lambda x: 'Delhi' if 'Delhi' in x else None)
data3['City.Hyderabad'] = data3['Location'].apply(lambda x: 'Hyderabad' if 'Hyderabad' in x else None)
data3['City.Noida'] = data3['Location'].apply(lambda x: 'Noida' if 'Noida' in x else None)
data3['City.Gurgaon'] = data3['Location'].apply(lambda x: 'Gurgaon' if 'Gurgaon' in x else None)
data3['City.Majestic'] = data3['Location'].apply(lambda x: 'Bangalore' if 'Majestic' in x else None)
data3['City.Marathalli'] = data3['Location'].apply(lambda x: 'Bangalore' if 'Marathalli' in x else None)
data3['City.Electronic'] = data3['Location'].apply(lambda x: 'Bangalore' if 'Electronic' in x else None)
data3['City.Gurgoan'] = data3['Location'].apply(lambda x: 'Gurgaon' if 'Gurgoan' in x else None)
data3['City.Whitefield'] = data3['Location'].apply(lambda x: 'Bangalore' if 'Whitefield' in x else None)

data3['City'] = data3['City.Pune'].map(str)+data3['City.Kolkata'].map(str)+data3['City.Mumbai'].map(str)+data3['City.Bangalor

data3['City'] = data3['City'].apply(lambda x: x.replace('None',''))
```

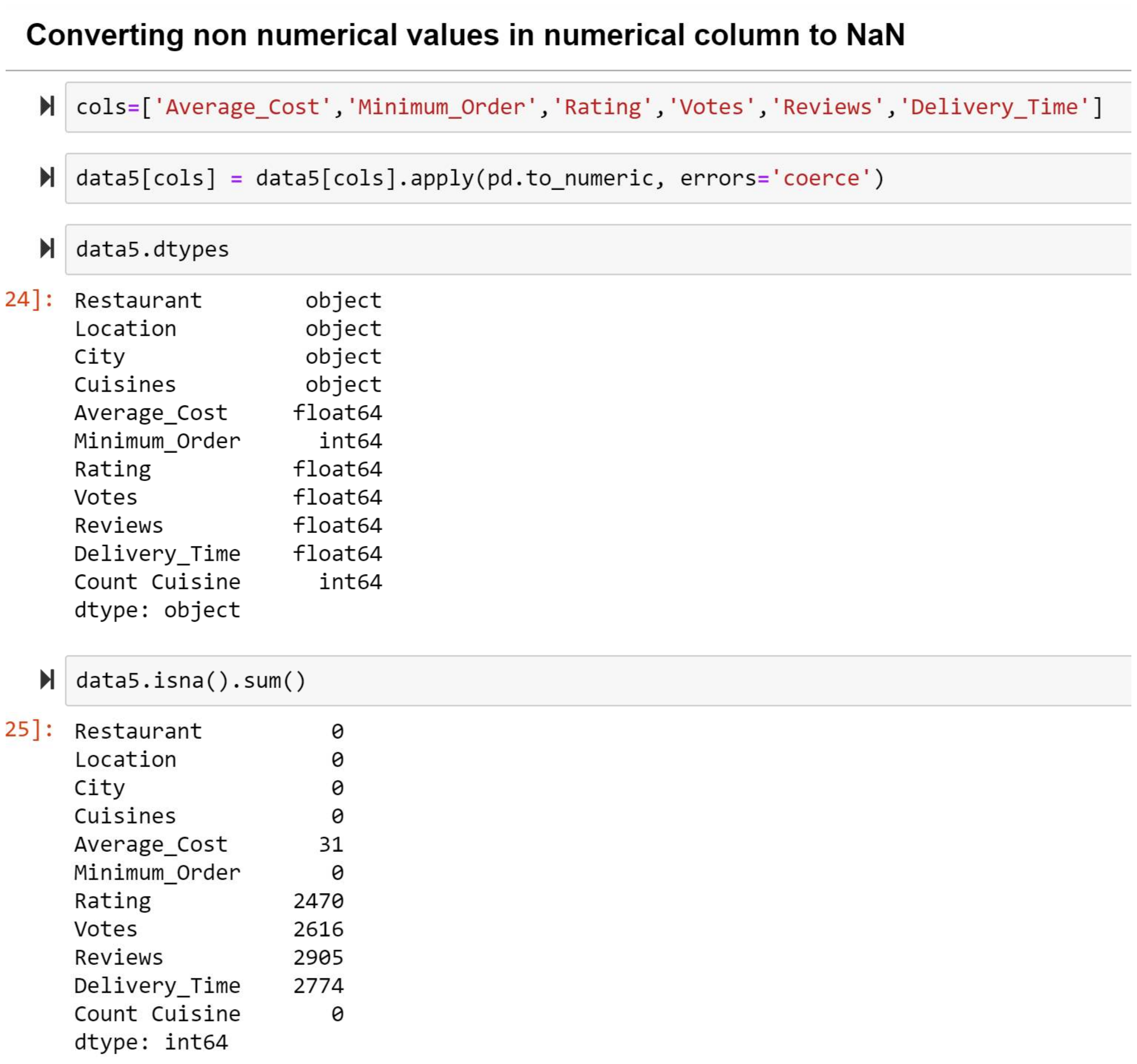
2.4: Check Data Type



```
data5.dtypes
```

```
Restaurant       object
Location         object
City             object
Cuisines         object
Average_Cost     object
Minimum_Order    object
Rating           object
Votes            object
Reviews          object
Delivery_Time    object
Count Cuisine     int64
dtype: object
```

***2.5**: Converting non numerical values in numerical column to NaN*



Converting non numerical values in numerical column to NaN

```
cols=['Average_Cost','Minimum_Order','Rating','Votes','Reviews','Delivery_Time']
```

```
data5[cols] = data5[cols].apply(pd.to_numeric, errors='coerce')
```

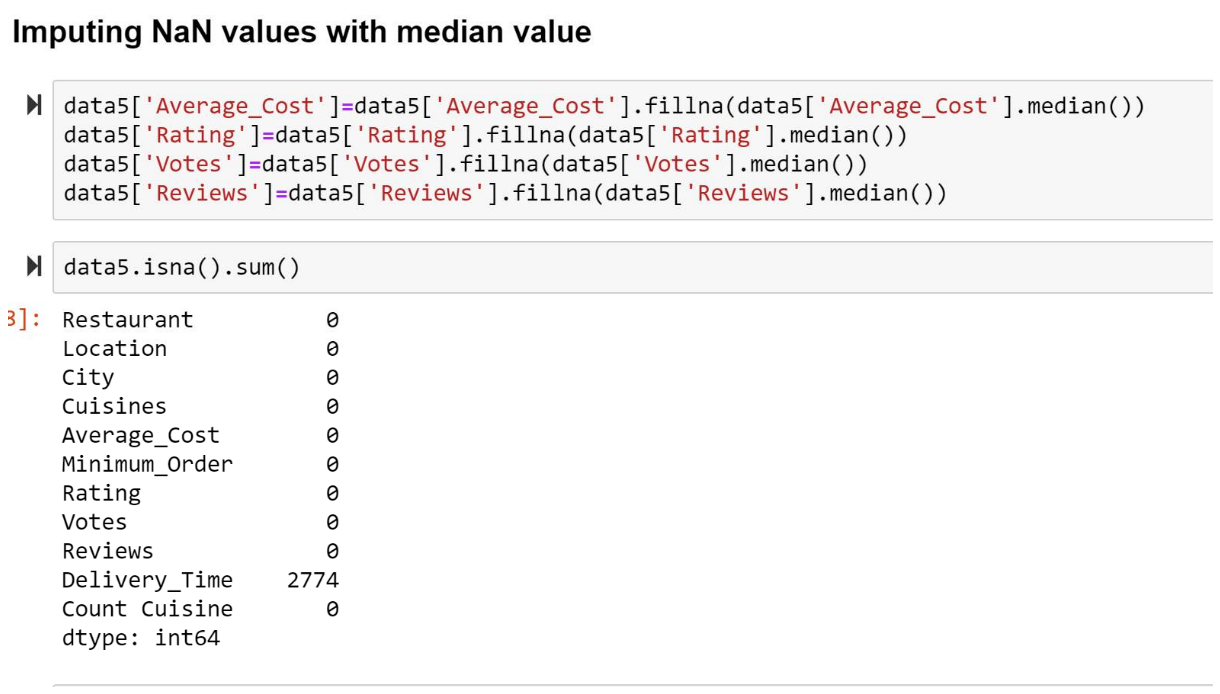
```
data5.dtypes
```

```
24]: Restaurant        object
     Location          object
     City              object
     Cuisines          object
     Average_Cost     float64
     Minimum_Order      int64
     Rating           float64
     Votes            float64
     Reviews          float64
     Delivery_Time    float64
     Count Cuisine      int64
     dtype: object
```

```
data5.isna().sum()
```

```
25]: Restaurant          0
     Location            0
     City                0
     Cuisines            0
     Average_Cost       31
     Minimum_Order       0
     Rating           2470
     Votes            2616
     Reviews          2905
     Delivery_Time    2774
     Count Cuisine       0
     dtype: int64
```

2.6: Importing NaN values with median values



Imputing NaN values with median value

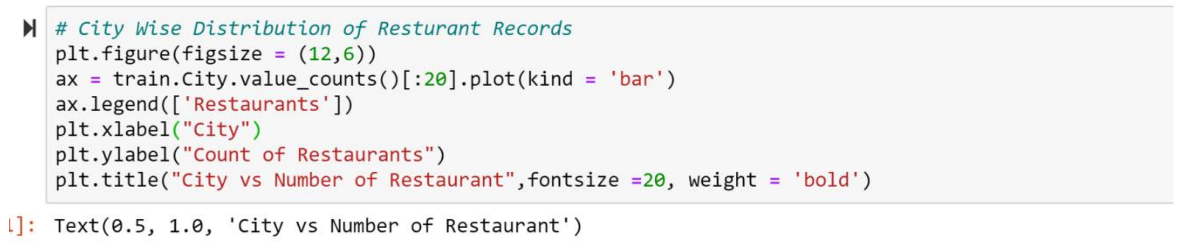
```python
data5['Average_Cost']=data5['Average_Cost'].fillna(data5['Average_Cost'].median())
data5['Rating']=data5['Rating'].fillna(data5['Rating'].median())
data5['Votes']=data5['Votes'].fillna(data5['Votes'].median())
data5['Reviews']=data5['Reviews'].fillna(data5['Reviews'].median())
```

```python
data5.isna().sum()
```

```
3]: Restaurant          0
    Location            0
    City                0
    Cuisines            0
    Average_Cost        0
    Minimum_Order       0
    Rating              0
    Votes               0
    Reviews             0
    Delivery_Time    2774
    Count Cuisine       0
    dtype: int64
```

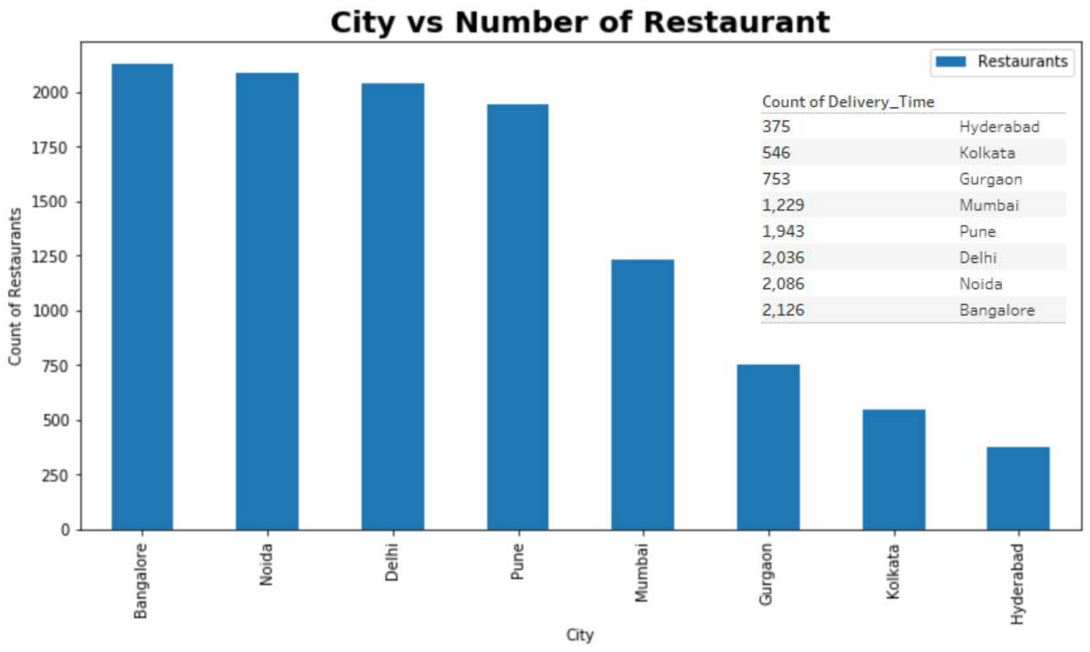
Chapter 3: EDA

3.1: City Wise Distribution of Restaurant Records



```
# City Wise Distribution of Resturant Records
plt.figure(figsize = (12,6))
ax = train.City.value_counts()[:20].plot(kind = 'bar')
ax.legend(['Restaurants'])
plt.xlabel("City")
plt.ylabel("Count of Restaurants")
plt.title("City vs Number of Restaurant",fontsize =20, weight = 'bold')
```

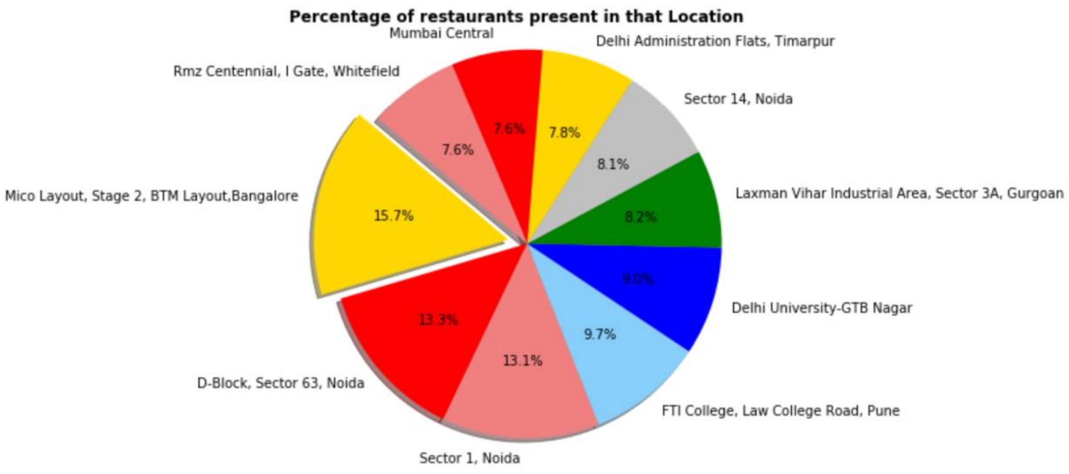
```
]: Text(0.5, 1.0, 'City vs Number of Restaurant')
```



Count of Delivery_Time	
375	Hyderabad
546	Kolkata
753	Gurgaon
1,229	Mumbai
1,943	Pune
2,036	Delhi
2,086	Noida
2,126	Bangalore

3.2: Location Wise Distribution of Resturant Records

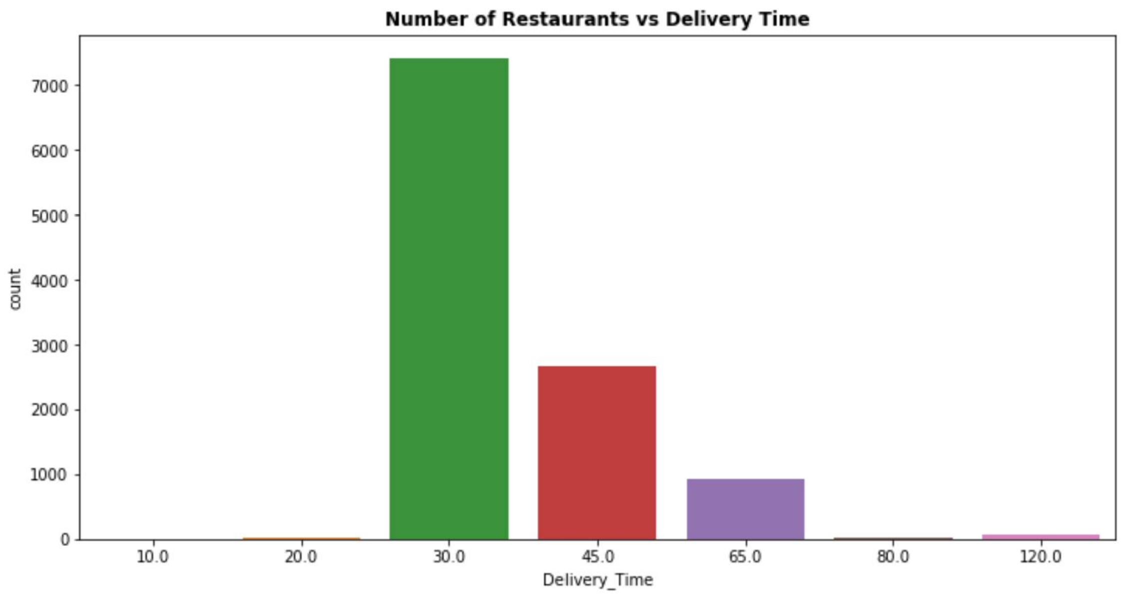
```
# Location Wise Distribution of Resturant Records
plt.figure(figsize = (12,6))
names = train['Location'].value_counts()[:10].index
values = train['Location'].value_counts()[:10].values
colors = ['gold', 'red', 'lightcoral', 'lightskyblue','blue','green','silver']
explode = (0.1, 0, 0, 0,0,0,0,0,0,0)

plt.pie(values, explode=explode, labels=names, colors= colors,autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.title("Percentage of restaurants present in that Location", weight = 'bold')
plt.show()
```



3.3: Number of Restaurants vs Delivery Time

```
plt.figure(figsize=(12,6))
ax1= sns.countplot(train['Delivery_Time'])
plt.title('Number of Restaurants vs Delivery Time', weight='bold')
plt.xlabel('Delivery_Time')
```

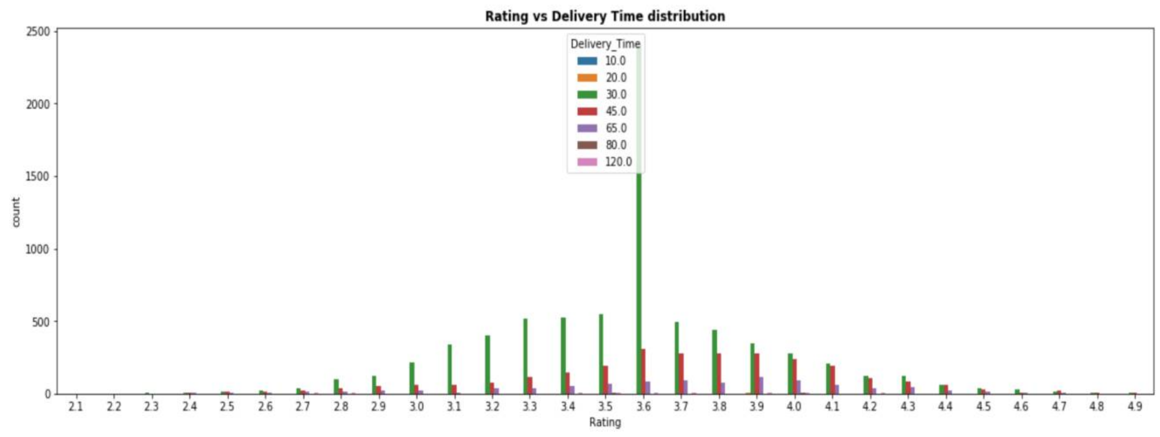
```
]: Text(0.5, 0, 'Delivery_Time')
```



3.4: Rating vs Delivery Time

```
plt.figure(figsize=(20,6))
ax2 = sns.countplot(x="Rating", hue="Delivery_Time", data=train)
plt.title('Rating vs Delivery Time distribution', weight='bold')
```

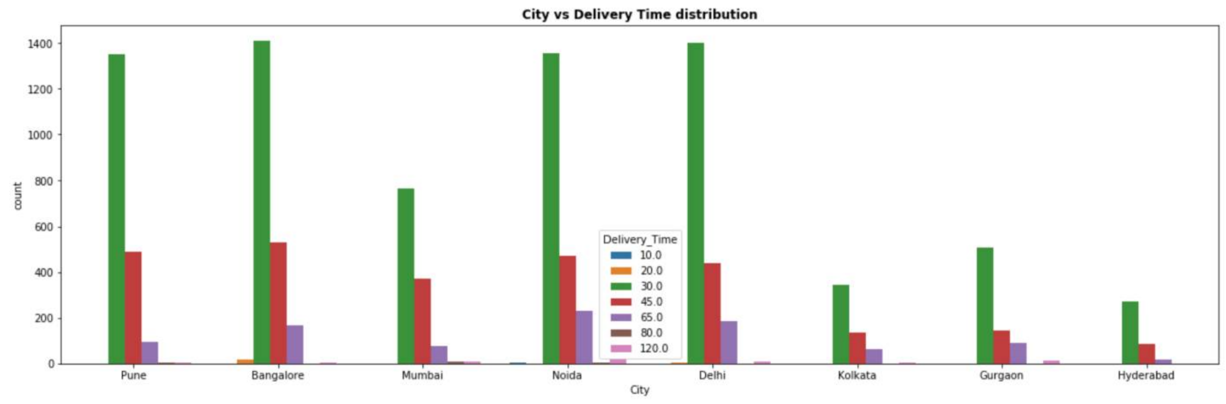
```
]: Text(0.5, 1.0, 'Rating vs Delivery Time distribution')
```

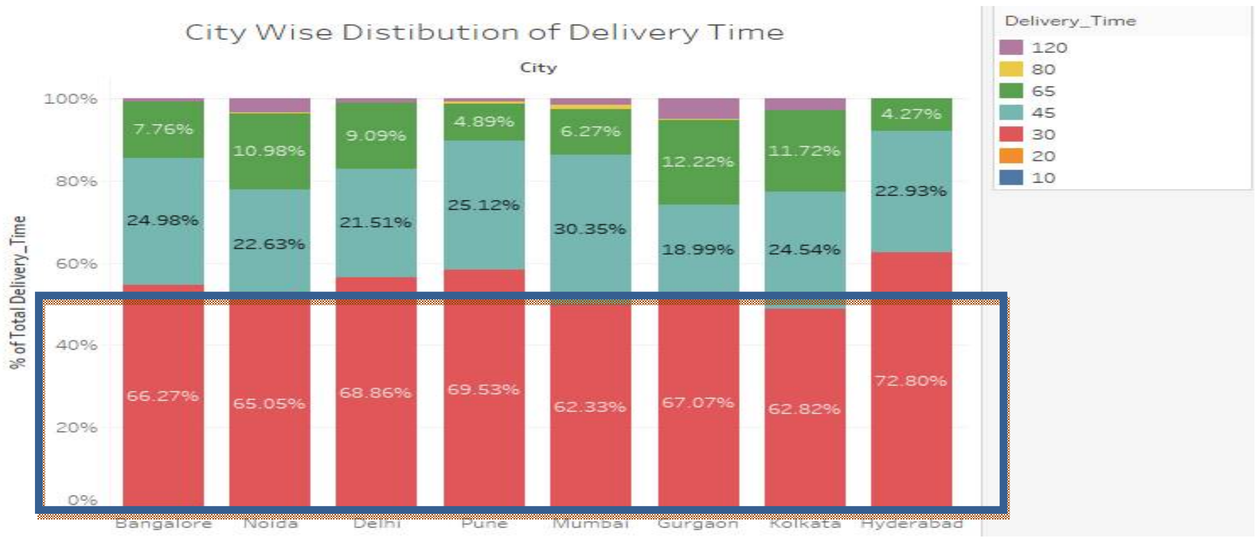


3.5: City vs Delivery Time

```
plt.figure(figsize=(20,6))
ax4 = sns.countplot(x="City", hue="Delivery_Time", data=train)
plt.title('City vs Delivery Time distribution', weight='bold')
```

```
]: Text(0.5, 1.0, 'City vs Delivery Time distribution')
```

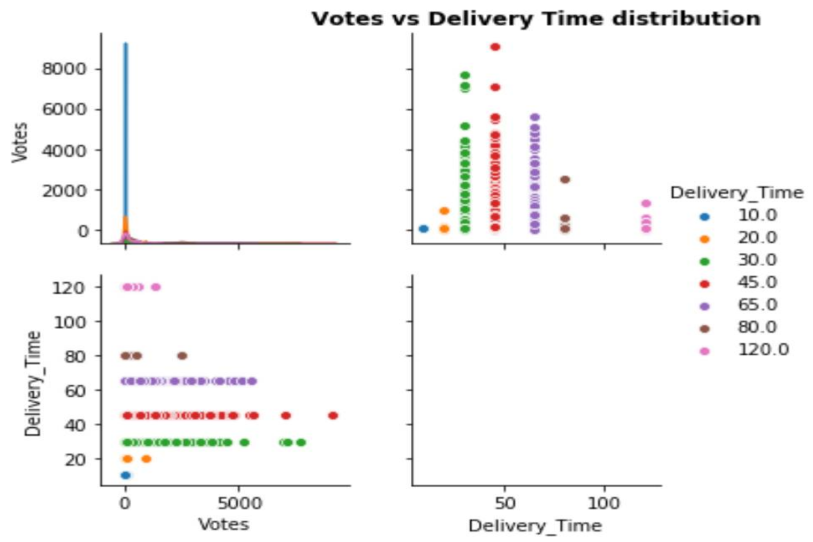




3.6: Votes vs Delivery Time

```
plt.figure(figsize=(20,6))
ax5 = sns.pairplot(train, vars=["Votes", "Delivery_Time"], hue = "Delivery_Time" )
plt.title('Votes vs Delivery Time distribution', weight='bold')
```

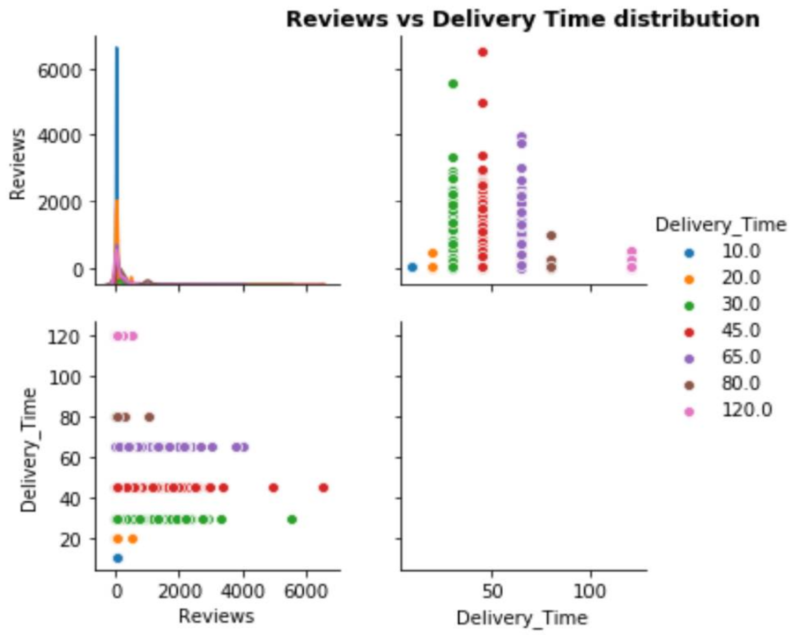
```
]: Text(0.5, 1, 'Votes vs Delivery Time distribution')

   <Figure size 1440x432 with 0 Axes>
```



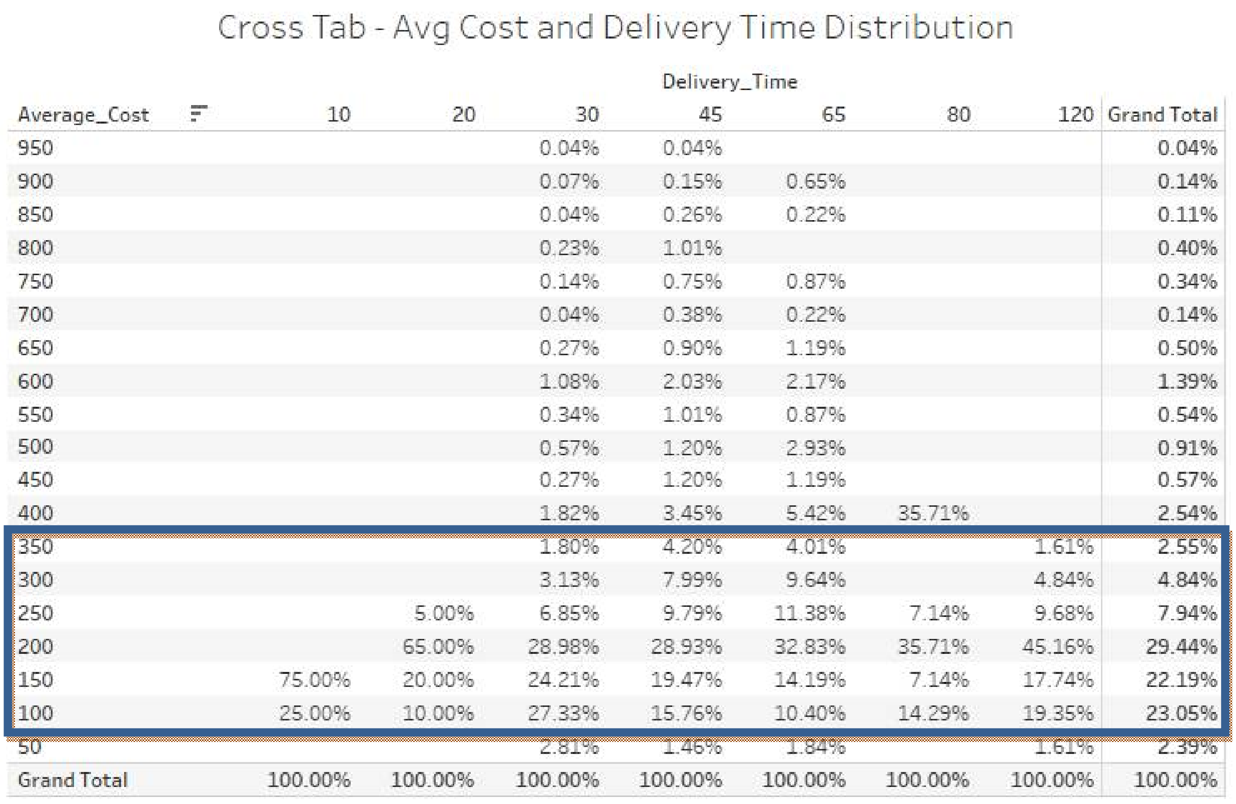
3.7: Reviews vs Delivery Time

```
plt.figure(figsize=(20,6))
ax6 = sns.pairplot(train, vars=["Reviews", "Delivery_Time"], hue = "Delivery_Time" )
plt.title('Reviews vs Delivery Time distribution', weight='bold')
```

```
]: Text(0.5, 1, 'Reviews vs Delivery Time distribution')

   <Figure size 1440x432 with 0 Axes>
```



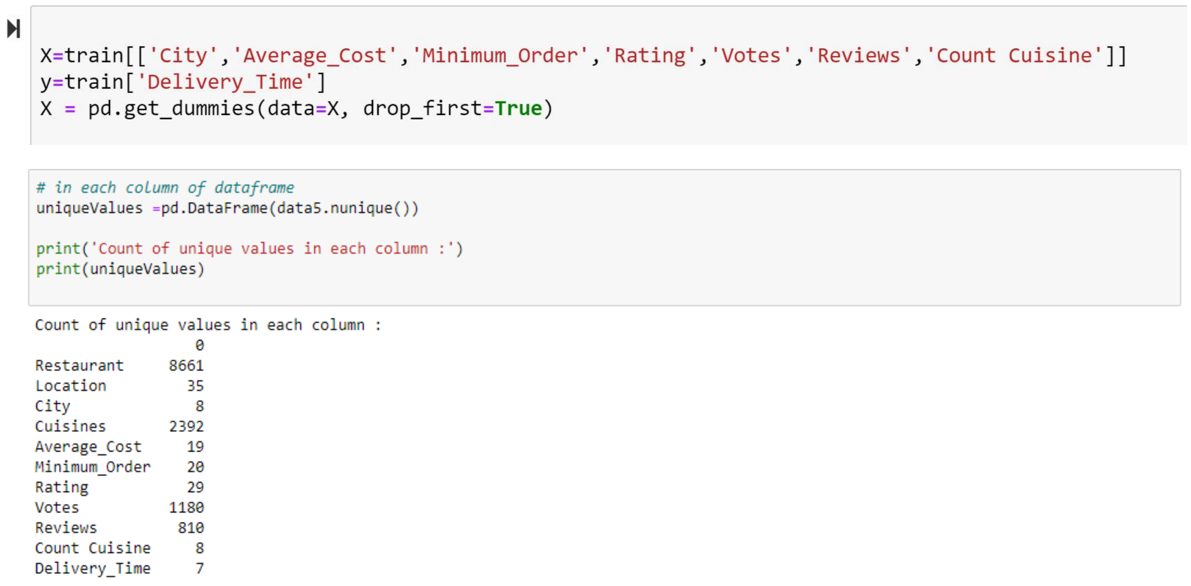
3.8: Avg Cost vs Delivery Time


Cross Tab - Avg Cost and Delivery Time Distribution

Average_Cost	Delivery_Time 10	20	30	45	65	80	120	Grand Total
950			0.04%	0.04%				0.04%
900			0.07%	0.15%	0.65%			0.14%
850			0.04%	0.26%	0.22%			0.11%
800			0.23%	1.01%				0.40%
750			0.14%	0.75%	0.87%			0.34%
700			0.04%	0.38%	0.22%			0.14%
650			0.27%	0.90%	1.19%			0.50%
600			1.08%	2.03%	2.17%			1.39%
550			0.34%	1.01%	0.87%			0.54%
500			0.57%	1.20%	2.93%			0.91%
450			0.27%	1.20%	1.19%			0.57%
400			1.82%	3.45%	5.42%	35.71%		2.54%
350			1.80%	4.20%	4.01%		1.61%	2.55%
300			3.13%	7.99%	9.64%		4.84%	4.84%
250		5.00%	6.85%	9.79%	11.38%	7.14%	9.68%	7.94%
200		65.00%	28.98%	28.93%	32.83%	35.71%	45.16%	29.44%
150	75.00%	20.00%	24.21%	19.47%	14.19%	7.14%	17.74%	22.19%
100	25.00%	10.00%	27.33%	15.76%	10.40%	14.29%	19.35%	23.05%
50			2.81%	1.46%	1.84%		1.61%	2.39%
Grand Total	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Chapter 4: One Hot Coding

Generating dummy variables for one hot encoding



```
X=train[['City','Average_Cost','Minimum_Order','Rating','Votes','Reviews','Count Cuisine']]
y=train['Delivery_Time']
X = pd.get_dummies(data=X, drop_first=True)
```

```
# in each column of dataframe
uniqueValues =pd.DataFrame(data5.nunique())

print('Count of unique values in each column :')
print(uniqueValues)
```

```
Count of unique values in each column :
                  0
Restaurant     8661
Location         35
City              8
Cuisines       2392
Average_Cost     19
Minimum_Order    20
Rating           29
Votes          1180
Reviews         810
Count Cuisine     8
Delivery_Time     7
```

The dataset contains high number of unique values for categorical features like Restaurant, Location, and Cuisines.

Had Label Encoding been done it would have transformed categorical value into numerical values, but then the imposed ordinality. Still there are algorithms like decision trees and random forests that can work with categorical variables just fine and Label Encoder can be used to store values using less disk space.

One-Hot-Encoding has the advantage that the result is binary rather than ordinal and that everything sits in an orthogonal vector space. The disadvantage is that for high cardinality, the feature space can really blow up quickly and you start fighting with the curse of dimensionality. However, in our data we have extracted city column from the location so there number of new columns generated after one hot encoding is 8.

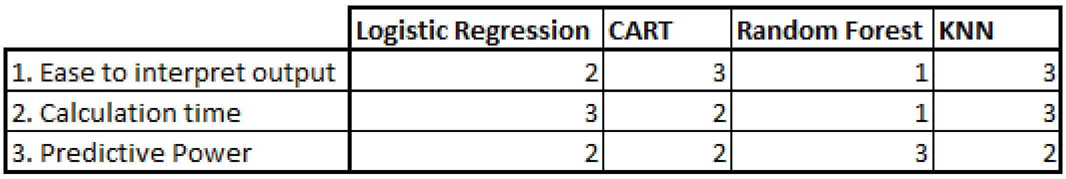
Hence, very few columns are added after one hot encoding.

Chapter 5: Classiffication Analysis

5.1: KNN Model

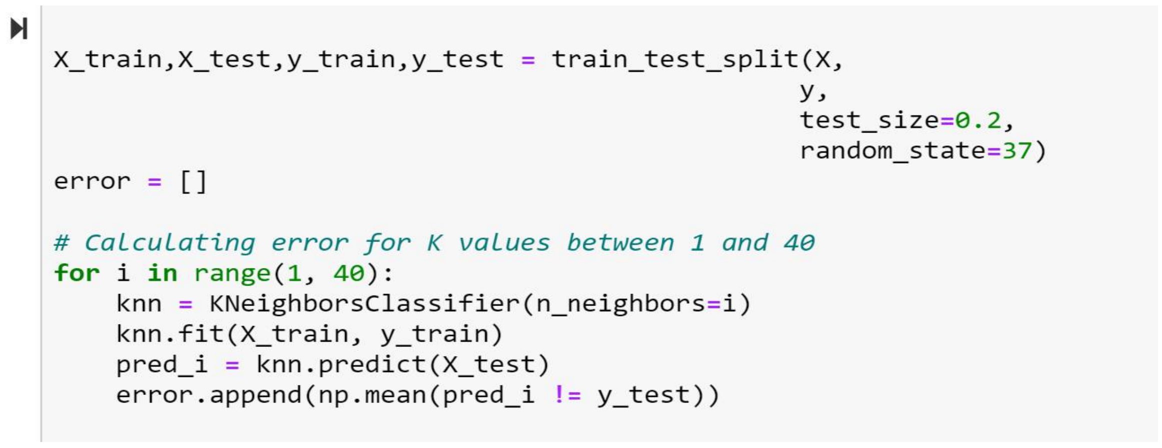
KNN is widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:
1. Ease to interpret output
2. Calculation time
3. Predictive Power

Let us take a few examples to place KNN in the scale:

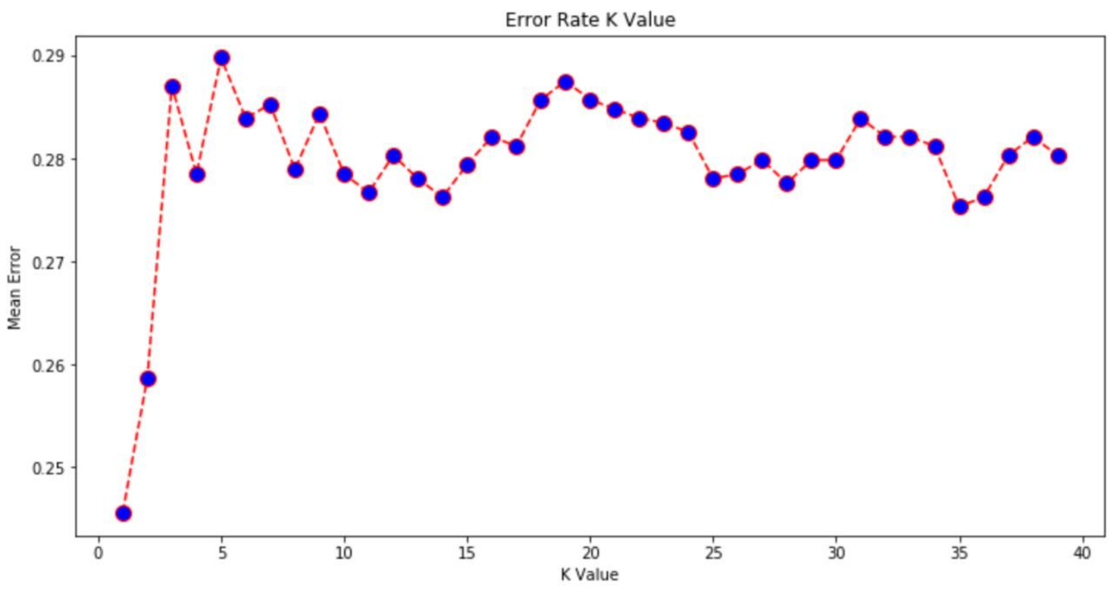


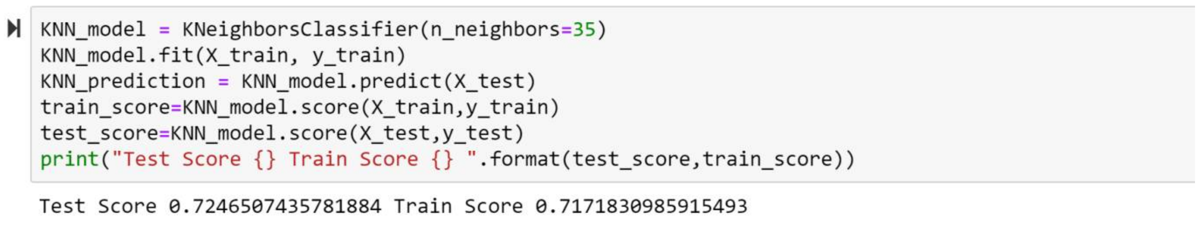
	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time.



```
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 y,
                                                 test_size=0.2,
                                                 random_state=37)
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

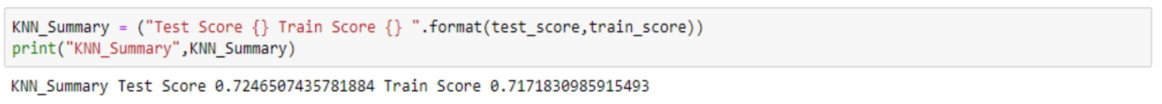




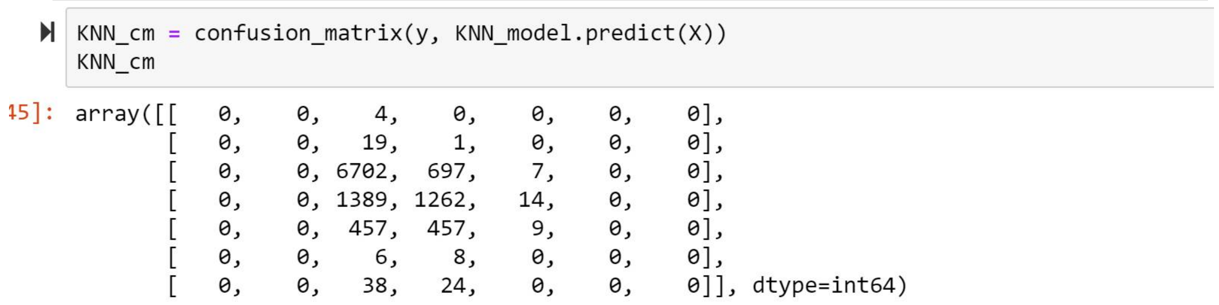
```
KNN_model = KNeighborsClassifier(n_neighbors=35)
KNN_model.fit(X_train, y_train)
KNN_prediction = KNN_model.predict(X_test)
train_score=KNN_model.score(X_train,y_train)
test_score=KNN_model.score(X_test,y_test)
print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.7246507435781884 Train Score 0.7171830985915493
```

For KNN Model, a for loop from 1 to 40 neighbours to identify the neighbours value at which error is minimum. A k value of 35 is discovered, where the error is minimum, hence 35 is selected as n_neighbors value and the model is run.

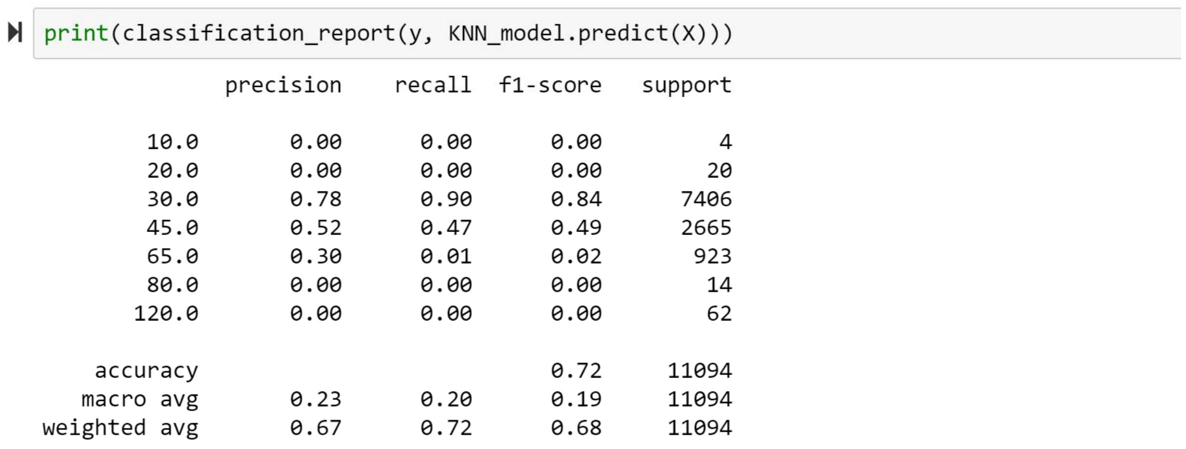


```
KNN_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("KNN_Summary",KNN_Summary)

KNN_Summary Test Score 0.7246507435781884 Train Score 0.7171830985915493
```



```
KNN_cm = confusion_matrix(y, KNN_model.predict(X))
KNN_cm
```

```
45]: array([[   0,    0,    4,    0,    0,    0,    0],
            [   0,    0,   19,    1,    0,    0,    0],
            [   0,    0, 6702,  697,    7,    0,    0],
            [   0,    0, 1389, 1262,   14,    0,    0],
            [   0,    0,  457,  457,    9,    0,    0],
            [   0,    0,    6,    8,    0,    0,    0],
            [   0,    0,   38,   24,    0,    0,    0]], dtype=int64)
```



```
print(classification_report(y, KNN_model.predict(X)))
```

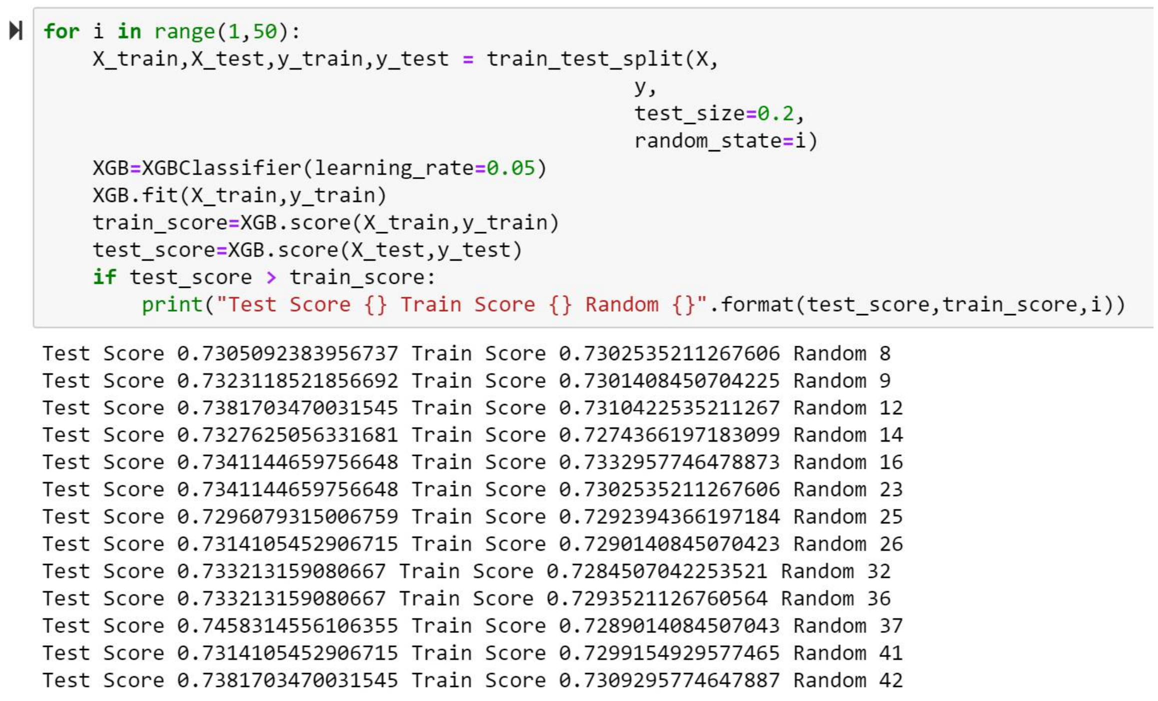
```
              precision    recall  f1-score   support

        10.0       0.00      0.00      0.00         4
        20.0       0.00      0.00      0.00        20
        30.0       0.78      0.90      0.84      7406
        45.0       0.52      0.47      0.49      2665
        65.0       0.30      0.01      0.02       923
        80.0       0.00      0.00      0.00        14
       120.0       0.00      0.00      0.00        62

    accuracy                           0.72     11094
   macro avg       0.23      0.20      0.19     11094
weighted avg       0.67      0.72      0.68     11094
```

Model is generalized with 72% accuracy. However, Precision and Recall for some classes are 0.

5.2: XGBoost Classifier Model

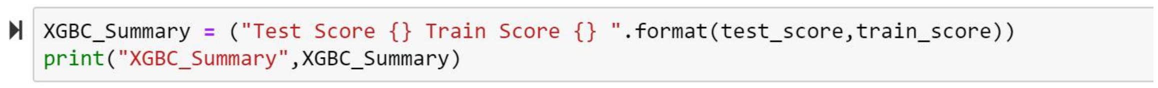
The beauty of this powerful algorithm lies in its scalability, which drives fast learning through parallel and distributed computing and offers efficient memory usage. XGBoost emerged as the most useful, straightforward and robust solution. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost is an ensemble learning method. Sometimes, it may not be sufficient to rely upon the results of just one machine learning model. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

In our notebook we have set learning rate= 0.05 and ran a for loop for 1 to 50 random states to find best value for random state and selected that random state which is 37.

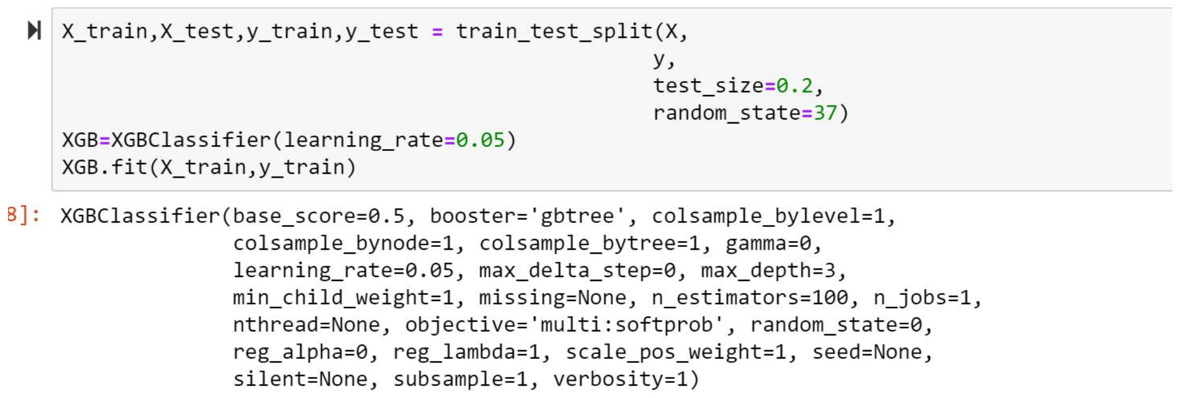


```
for i in range(1,50):
    X_train,X_test,y_train,y_test = train_test_split(X,
                                                      y,
                                                      test_size=0.2,
                                                      random_state=i)
    XGB=XGBClassifier(learning_rate=0.05)
    XGB.fit(X_train,y_train)
    train_score=XGB.score(X_train,y_train)
    test_score=XGB.score(X_test,y_test)
    if test_score > train_score:
        print("Test Score {} Train Score {} Random {}".format(test_score,train_score,i))
```

```
Test Score 0.7305092383956737 Train Score 0.7302535211267606 Random 8
Test Score 0.7323118521856692 Train Score 0.7301408450704225 Random 9
Test Score 0.7381703470031545 Train Score 0.7310422535211267 Random 12
Test Score 0.7327625056331681 Train Score 0.7274366197183099 Random 14
Test Score 0.7341144659756648 Train Score 0.7332957746478873 Random 16
Test Score 0.7341144659756648 Train Score 0.7302535211267606 Random 23
Test Score 0.7296079315006759 Train Score 0.7292394366197184 Random 25
Test Score 0.7314105452906715 Train Score 0.7290140845070423 Random 26
Test Score 0.733213159080667 Train Score 0.7284507042253521 Random 32
Test Score 0.733213159080667 Train Score 0.7293521126760564 Random 36
Test Score 0.7458314556106355 Train Score 0.7289014084507043 Random 37
Test Score 0.7314105452906715 Train Score 0.7299154929577465 Random 41
Test Score 0.7381703470031545 Train Score 0.7309295774647887 Random 42
```



```
XGBC_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("XGBC_Summary",XGBC_Summary)
```



```
X_train,X_test,y_train,y_test = train_test_split(X,
                                                  y,
                                                  test_size=0.2,
                                                  random_state=37)
XGB=XGBClassifier(learning_rate=0.05)
XGB.fit(X_train,y_train)
```

```
8]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=1, gamma=0,
                  learning_rate=0.05, max_delta_step=0, max_depth=3,
                  min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                  nthread=None, objective='multi:softprob', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=1, verbosity=1)
```

```
XGB_cm = confusion_matrix(y, XGB.predict(X))
XGB_cm
```

```
49]: array([[   0,    0,    4,    0,    0,    0,    0],
       [   0,    0,   19,    1,    0,    0,    0],
       [   0,    0, 6661,  743,    2,    0,    0],
       [   0,    0, 1211, 1450,    4,    0,    0],
       [   0,    0,  379,  537,    7,    0,    0],
       [   0,    0,    7,    2,    0,    5,    0],
       [   0,    0,   39,   22,    0,    0,    1]], dtype=int64)
```

```
print(classification_report(y, XGB.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       0.00      0.00      0.00         4
        20.0       0.00      0.00      0.00        20
        30.0       0.80      0.90      0.85      7406
        45.0       0.53      0.54      0.54      2665
        65.0       0.54      0.01      0.01       923
        80.0       1.00      0.36      0.53        14
       120.0       1.00      0.02      0.03        62

    accuracy                           0.73     11094
   macro avg       0.55      0.26      0.28     11094
weighted avg       0.71      0.73      0.70     11094
```

Generalized model with 74% accuracy, However precision and recall for only few classes are high for some it is 0.

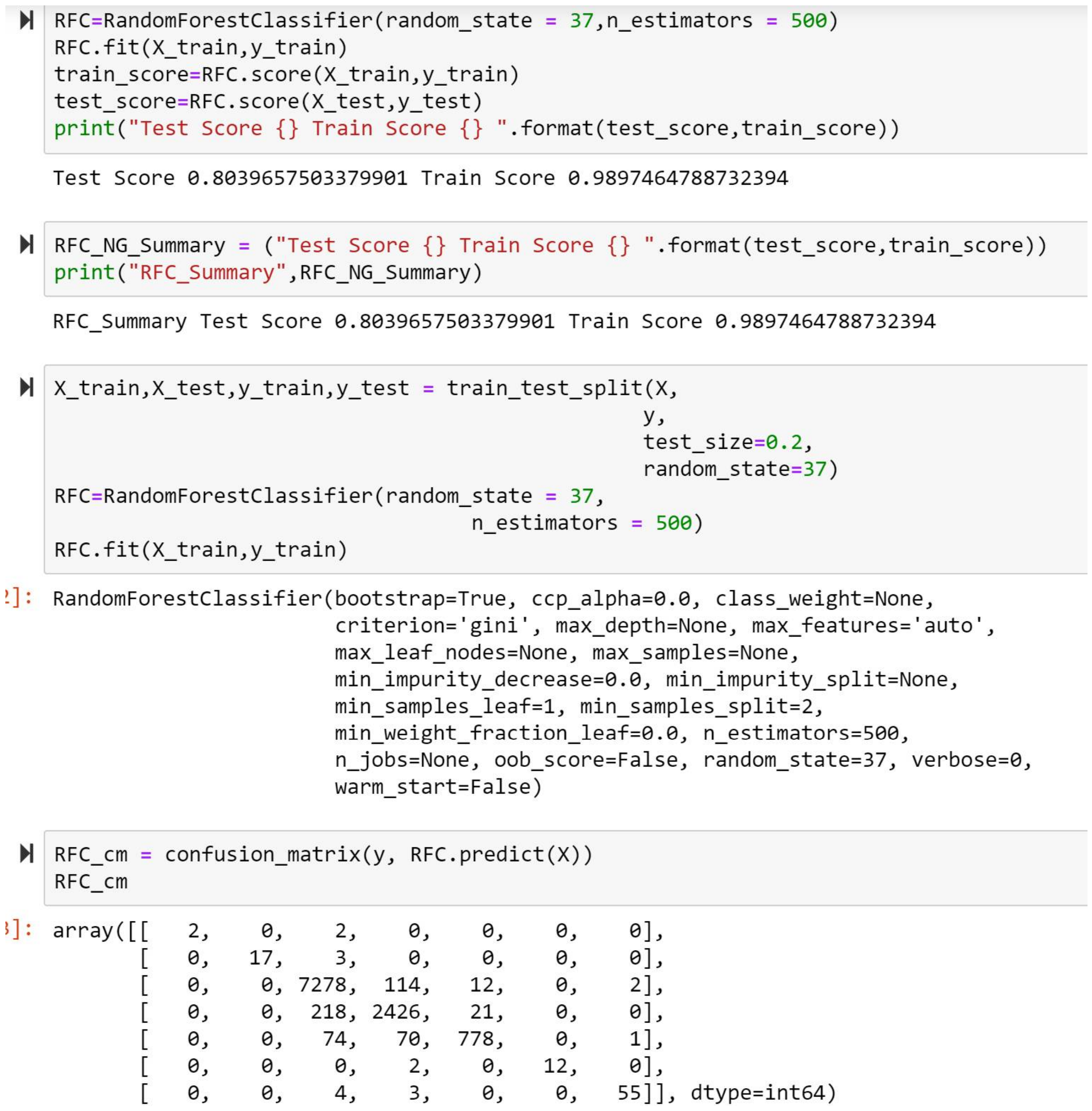
5.3: Random Forest Classifier

Random Forest operates by constructing a multitude of trees at training time and outputting the class that is mode of the classes or mean prediction of trees.

The concept of how a Random Forest model works from scratch will be discussed in detail in the later sections of the course, but here is a brief introduction in Jeremy Howard's words:

- Random forest is a kind of universal machine learning technique
- It can be used for both regression (target is a continuous variable) or classification (target is a categorical variable) problems
- It also works with columns of any kinds, like pixel values, zip codes, revenue, etc.
- In general, random forest does not overfit (it's very easy to stop it from overfitting)
- You do not need a separate validation set in general. It can tell you how well it generalizes even if you only have one dataset
- It has few (if any) statistical assumptions (it doesn't assume that data is normally distributed, data is linear, or that you need to specify the interactions)
- Requires very few feature engineering tactics, so it's a great place to start.

In the model, number of estimators =500 and ran a for loop for 1 to 50 random states to find best value for random state and selected that random state which is 37.



```
RFC=RandomForestClassifier(random_state = 37,n_estimators = 500)
RFC.fit(X_train,y_train)
train_score=RFC.score(X_train,y_train)
test_score=RFC.score(X_test,y_test)
print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.8039657503379901 Train Score 0.9897464788732394
```

```
RFC_NG_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("RFC_Summary",RFC_NG_Summary)
```

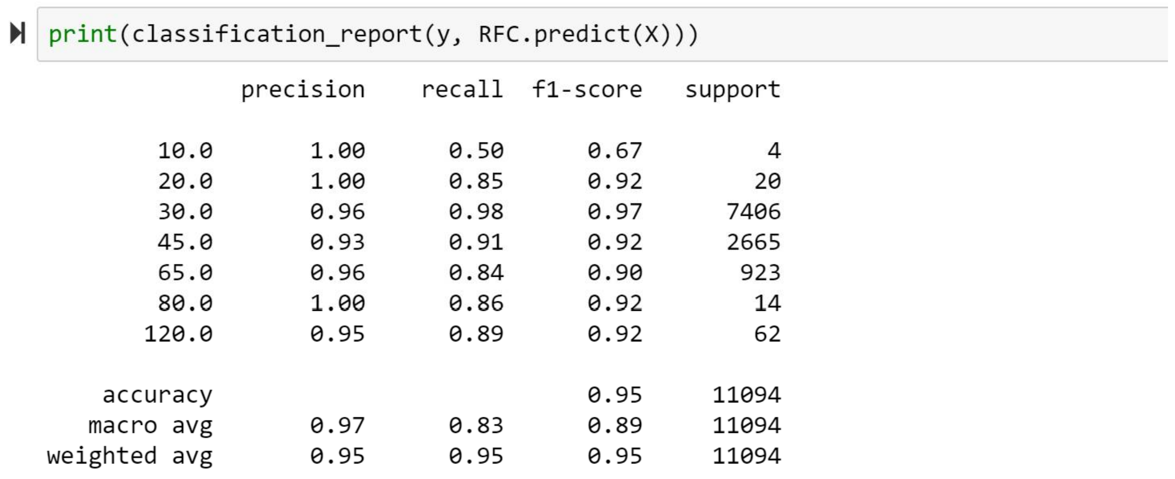
```
RFC_Summary Test Score 0.8039657503379901 Train Score 0.9897464788732394
```

```
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 y,
                                                 test_size=0.2,
                                                 random_state=37)
RFC=RandomForestClassifier(random_state = 37,
                                  n_estimators = 500)
RFC.fit(X_train,y_train)
```

```
]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                          criterion='gini', max_depth=None, max_features='auto',
                          max_leaf_nodes=None, max_samples=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=500,
                          n_jobs=None, oob_score=False, random_state=37, verbose=0,
                          warm_start=False)
```

```
RFC_cm = confusion_matrix(y, RFC.predict(X))
RFC_cm
```

```
]: array([[   2,    0,    2,    0,    0,    0,    0],
          [   0,   17,    3,    0,    0,    0,    0],
          [   0,    0, 7278,  114,   12,    0,    2],
          [   0,    0,  218, 2426,   21,    0,    0],
          [   0,    0,   74,   70,  778,    0,    1],
          [   0,    0,    0,    2,    0,   12,    0],
          [   0,    0,    4,    3,    0,    0,   55]], dtype=int64)
```



```
print(classification_report(y, RFC.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       1.00      0.50      0.67         4
        20.0       1.00      0.85      0.92        20
        30.0       0.96      0.98      0.97      7406
        45.0       0.93      0.91      0.92      2665
        65.0       0.96      0.84      0.90       923
        80.0       1.00      0.86      0.92        14
       120.0       0.95      0.89      0.92        62

    accuracy                           0.95     11094
   macro avg       0.97      0.83      0.89     11094
weighted avg       0.95      0.95      0.95     11094
```

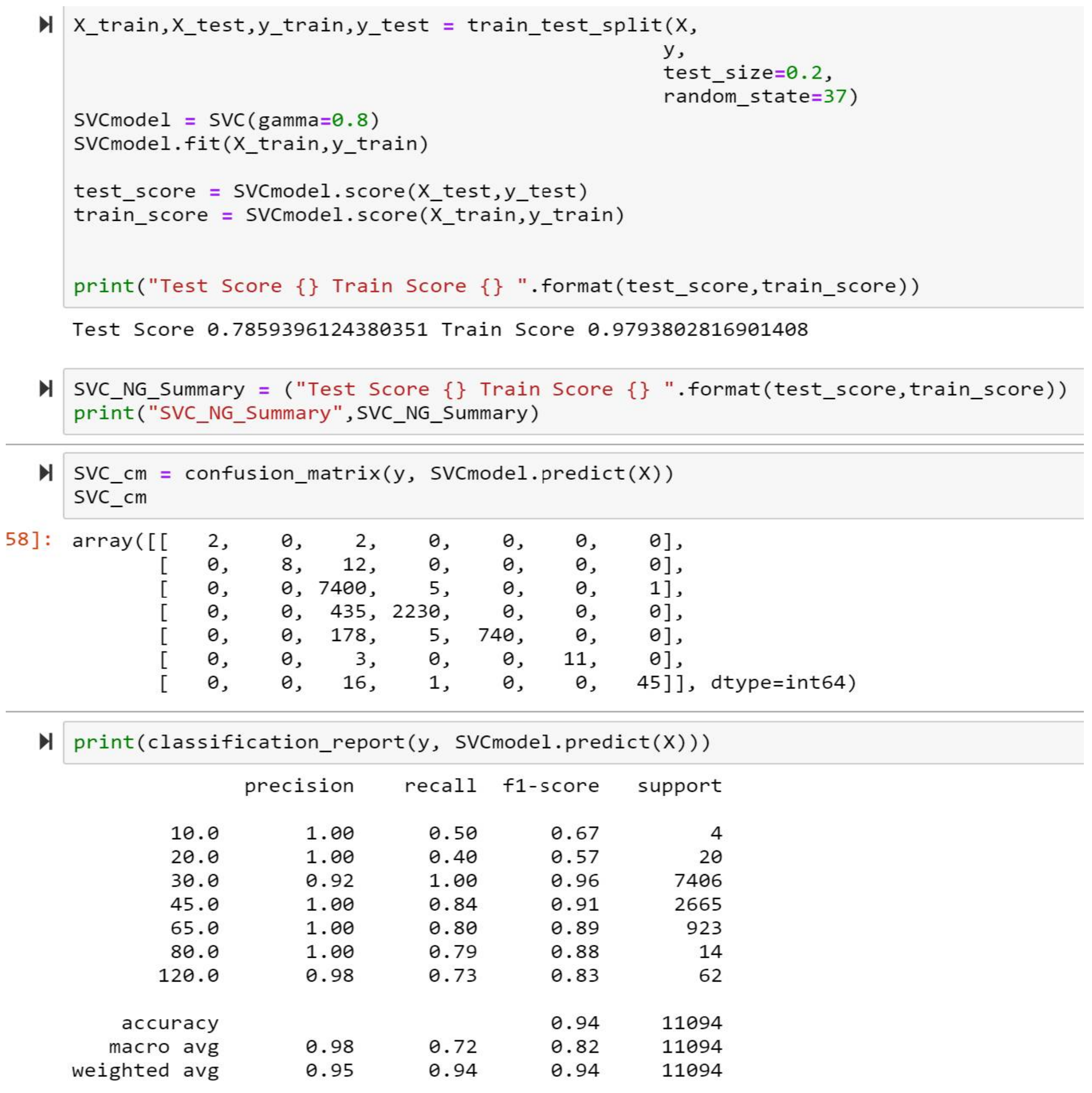
Non Generalized model with 80% accuracy, High precision and recall for all classes is predicted..

5.4: Support Vector Classifier

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

Gamma is the parameter to specify how wide or narrow the gap should be.

In the model, selected random state=37 and gamma=0.8 (from trial and error based on accuracy) and run the model.



```
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 y,
                                                 test_size=0.2,
                                                 random_state=37)
SVCmodel = SVC(gamma=0.8)
SVCmodel.fit(X_train,y_train)

test_score = SVCmodel.score(X_test,y_test)
train_score = SVCmodel.score(X_train,y_train)


print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.7859396124380351 Train Score 0.9793802816901408
```

```
SVC_NG_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("SVC_NG_Summary",SVC_NG_Summary)
```

```
SVC_cm = confusion_matrix(y, SVCmodel.predict(X))
SVC_cm
```

```
58]: array([[    2,     0,     2,     0,     0,     0,     0],
              [    0,     8,    12,     0,     0,     0,     0],
              [    0,     0,  7400,     5,     0,     0,     1],
              [    0,     0,   435,  2230,     0,     0,     0],
              [    0,     0,   178,     5,   740,     0,     0],
              [    0,     0,     3,     0,     0,    11,     0],
              [    0,     0,    16,     1,     0,     0,    45]], dtype=int64)
```

```
print(classification_report(y, SVCmodel.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       1.00      0.50      0.67         4
        20.0       1.00      0.40      0.57        20
        30.0       0.92      1.00      0.96      7406
        45.0       1.00      0.84      0.91      2665
        65.0       1.00      0.80      0.89       923
        80.0       1.00      0.79      0.88        14
       120.0       0.98      0.73      0.83        62

    accuracy                           0.94     11094
   macro avg       0.98      0.72      0.82     11094
weighted avg       0.95      0.94      0.94     11094
```

Non generalized model with 94% accuracy. Giving much high precision and recall for every class.

5.5: SMOTE

Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. For any imbalanced data set, if the event to be predicted belongs to the minority class and the event rate is less than 5%, it is usually referred to as a rare event.

The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets.

Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

While working in an imbalanced domain accuracy is not an appropriate measure to evaluate model performance**.** For eg: A classifier which achieves an accuracy of 98 % with an event rate of 2 % is not accurate, if it classifies all instances as the majority class and eliminates the 2 % minority class observations as noise.

Synthetic Minority Over-sampling Technique

This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

Advantages

Mitigates the problem of overfitting caused by random oversampling as synthetic examples are- generated rather than replication of instances.

No loss of useful information

Disadvantages

While generating synthetic examples SMOTE does not take into consideration neighbouring examples from other classes. This can result in increase in overlapping of classes and can introduce additional noise

SMOTE is not very effective for high dimensional data.

Since the dataset is highly imbalanced using SMOTE - SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them.

```
sm = SMOTE(k_neighbors=2)
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X, y, test_size=0.2, random_state=37)
X_train2, Y_train2 = sm.fit_resample(X_train2, Y_train2)
np.unique(Y_train2, return_counts=True)
```

```
0]: (array([ 10.,  20.,  30.,  45.,  65.,  80., 120.]),
     array([5864, 5864, 5864, 5864, 5864, 5864, 5864], dtype=int64))
```

```
SVC_sm = SVC(gamma=0.8)
SVC_sm.fit(X_train2, Y_train2)
test_score = SVC_sm.score(X_test2,Y_test2)
train_score = SVC_sm.score(X_train2,Y_train2)


print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.7030193780982424 Train Score 0.989207756772559
```

```
SVC_NG_SM_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("SVC_NG_SM_Summary",SVC_NG_SM_Summary)
```

```
SVC_sm_cm = confusion_matrix(y, SVC_sm.predict(X))
SVC_sm_cm
```

```
5]: array([[   2,    0,    2,    0,    0,    0,    0],
           [   0,   17,    2,    1,    0,    0,    0],
           [   0,    6, 6869,  433,   86,    1,   11],
           [   0,    2,  128, 2518,   15,    0,    2],
           [   0,    1,   31,  121,  768,    0,    2],
           [   0,    0,    0,    2,    0,   12,    0],
           [   0,    0,    4,    4,    0,    0,   54]], dtype=int64)
```

```
print(classification_report(y, SVC_sm.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       1.00      0.50      0.67         4
        20.0       0.65      0.85      0.74        20
        30.0       0.98      0.93      0.95      7406
        45.0       0.82      0.94      0.88      2665
        65.0       0.88      0.83      0.86       923
        80.0       0.92      0.86      0.89        14
       120.0       0.78      0.87      0.82        62

    accuracy                           0.92     11094
   macro avg       0.86      0.83      0.83     11094
weighted avg       0.93      0.92      0.92     11094
```

Non-generalized model with 70% accuracy. Precision and Recall over 0.5 for each class

5.6 ADASYN

The other oversampling technique implemented in imlearn is adaptive synthetic sampling, or ADASYN. ADASYN is similar to SMOTE, and derived from it, featuring just one important difference. It will bias the sample space (that is, the likelihood that any particular point will be chosen for duping) towards points which are located not in homogenous neighbourhoods.

```
ada = ADASYN(n_neighbors=2)
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X, y, test_size=0.2, random_state=37)
X_train2, Y_train2 = ada.fit_resample(X_train2, Y_train2)
np.unique(Y_train2, return_counts=True)
```

```
7]: (array([ 10.,  20.,  30.,  45.,  65.,  80., 120.]),
     array([5863, 5860, 5864, 5817, 5628, 5865, 5869], dtype=int64))
```

```
ada_SVC = SVC(gamma=0.8)
ada_SVC.fit(X_train2, Y_train2)
test_score = ada_SVC.score(X_test2,Y_test2)
train_score = ada_SVC.score(X_train2,Y_train2)


print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.6926543488057684 Train Score 0.9866800765343668
```

```
SVC_NG_ADA_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("SVC_NG_ADA_Summary",SVC_NG_ADA_Summary)
```

```
ada_SVC_cm = confusion_matrix(y, ada_SVC.predict(X))
ada_SVC_cm
```

```
9]: array([[   2,    0,    2,    0,    0,    0,    0],
           [   0,   17,    2,    1,    0,    0,    0],
           [   0,    6, 6841,  466,   80,    1,   12],
           [   0,    2,  120, 2528,   14,    0,    1],
           [   0,    1,   32,  123,  765,    0,    2],
           [   0,    0,    0,    2,    0,   12,    0],
           [   0,    0,    4,    4,    0,    0,   54]], dtype=int64)
```

```
print(classification_report(y, ada_SVC.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       1.00      0.50      0.67         4
        20.0       0.65      0.85      0.74        20
        30.0       0.98      0.92      0.95      7406
        45.0       0.81      0.95      0.87      2665
        65.0       0.89      0.83      0.86       923
        80.0       0.92      0.86      0.89        14
       120.0       0.78      0.87      0.82        62

    accuracy                           0.92     11094
   macro avg       0.86      0.83      0.83     11094
weighted avg       0.93      0.92      0.92     11094
```

Non-generalized model with 69% accuracy. Precision and Recall over 0.5 for each class

```
ada_RCF = RandomForestClassifier(random_state = 37,
                                 n_estimators = 500)
ada_RCF.fit(X_train2, Y_train2)
test_score = ada_RCF.score(X_test2,Y_test2)
train_score = ada_RCF.score(X_train2,Y_train2)


print("Test Score {} Train Score {} ".format(test_score,train_score))
```

```
Test Score 0.7832356917530419 Train Score 0.9927145169994603
```

```
RCF_NG_ADA_Summary = ("Test Score {} Train Score {} ".format(test_score,train_score))
print("RCF_NG_ADA_Summary",RCF_NG_ADA_Summary)
```

```
ada_RCF_cm = confusion_matrix(y, ada_RCF.predict(X))
ada_RCF_cm
```

```
2]: array([[   2,    0,    2,    0,    0,    0,    0],
           [   0,   17,    3,    0,    0,    0,    0],
           [   0,    1, 7145,  174,   83,    0,    3],
           [   0,    0,  171, 2455,   37,    0,    2],
           [   0,    0,   61,   76,  785,    0,    1],
           [   0,    0,    0,    2,    0,   12,    0],
           [   0,    0,    4,    1,    2,    0,   55]], dtype=int64)
```

```
print(classification_report(y, ada_RCF.predict(X)))
```

```
              precision    recall  f1-score   support

        10.0       1.00      0.50      0.67         4
        20.0       0.94      0.85      0.89        20
        30.0       0.97      0.96      0.97      7406
        45.0       0.91      0.92      0.91      2665
        65.0       0.87      0.85      0.86       923
        80.0       1.00      0.86      0.92        14
       120.0       0.90      0.89      0.89        62

    accuracy                           0.94     11094
   macro avg       0.94      0.83      0.87     11094
weighted avg       0.94      0.94      0.94     11094
```

```
#### Non-generalized model with 78.3% accuracy. Precision and Recall over 0.5 for each class
```

Comparison of Accuracies of all the models -

Model	Test Score	Train Accuracy	Remarks
KNN Classifier	0.72	0.71	Generalized Model
XGBoost Classifier	0.745	0.728	Generalized Model
Random Forest Classifier	**0.80**	**0.989**	**Non Generalized Model**
SVC	0.785	0.979	Non Generalized Model
SMOTE_SVC	0.707	0.989	Non Generalized Model
ADASYN_SVC	0.696	0.986	Non Generalized Model
ADASYN_RCF	0.78	0.992	Non Generalized Model

Comparion of Precision(P) and Recall (R) of all the class for each of the models -

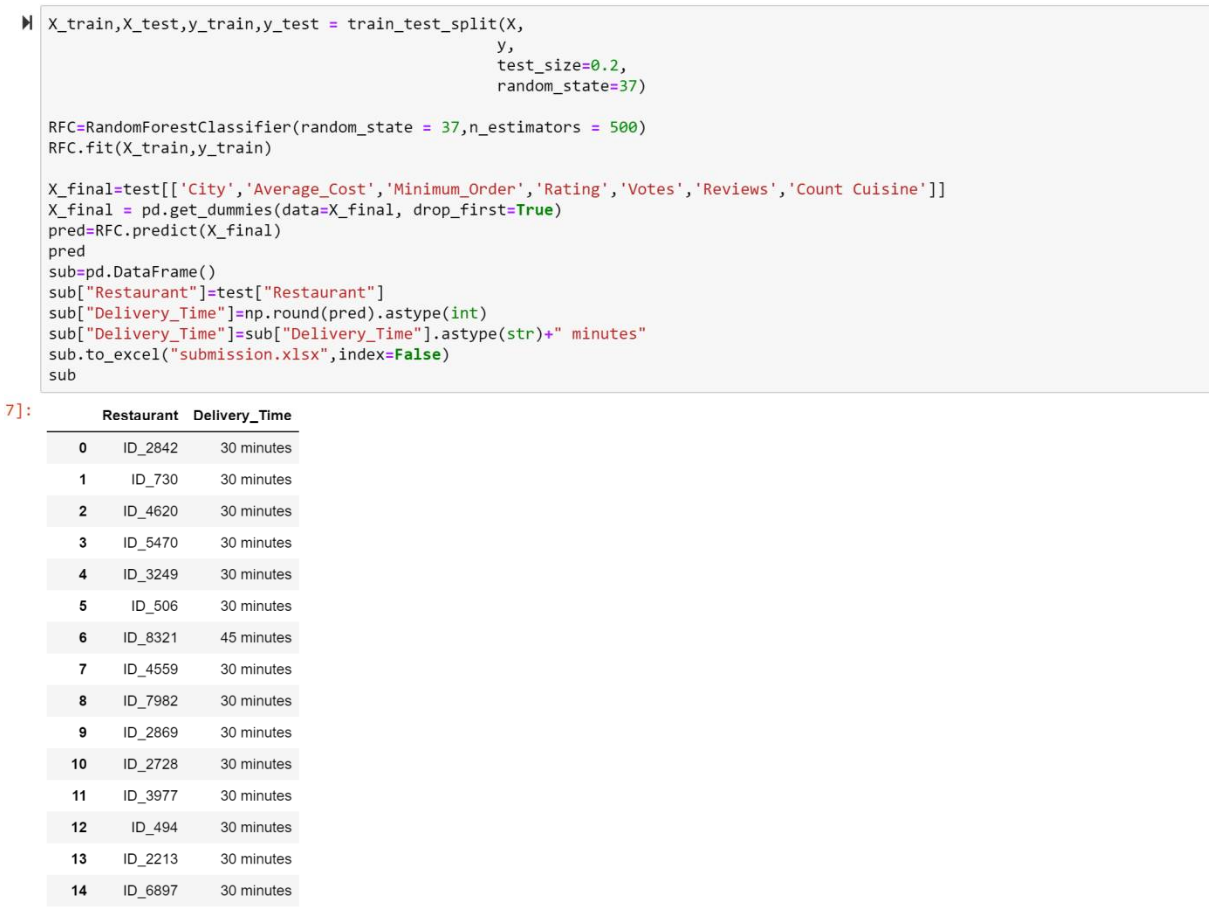
	KNN		XGBC		RFC		SVC		SMOTE_SVC		ADASYN_SVC		ADASYN_RCF	
	P	R	P	R	P	R	P	R	P	R	P	R	P	R
10	0	0	0	0	1	0.50	1	0.50	1	0.50	1	0.50	1	0.50
20	0	0	0	0	1	0.85	1	0.40	0.65	0.85	0.65	0.85	0.94	0.85
30	0.78	0.9	0.8	0.9	0.96	0.98	0.92	1	0.98	0.93	0.98	0.92	0.97	0.96
45	0.52	0.47	0.53	0.54	0.93	0.91	1	0.84	0.82	0.95	0.81	0.95	0.90	0.92
65	0.3	0.01	0.53	0.01	0.96	0.84	1	0.80	0.91	0.83	0.90	0.83	0.87	0.85
80	0	0	1	0.36	1	0.86	1	0.79	0.92	0.86	0.92	0.86	1	0.86
120	0	0	1	0.02	0.95	0.89	0.98	0.73	0.78	0.87	0.77	0.87	0.90	0.89

Precision: It is implied as the measure of the correctly identified positive cases from all the predicted positive cases. Thus, it is useful when the costs of False Positives is high.

Recall: It is the measure of the correctly identified positive cases from all the actual positive cases. It is important when the cost of False Negatives is high.

Based on the Accuracy Score and Precision, Recall Value, Random Forest Classifier is chosen as the model to predict the target classes for the actual test data.

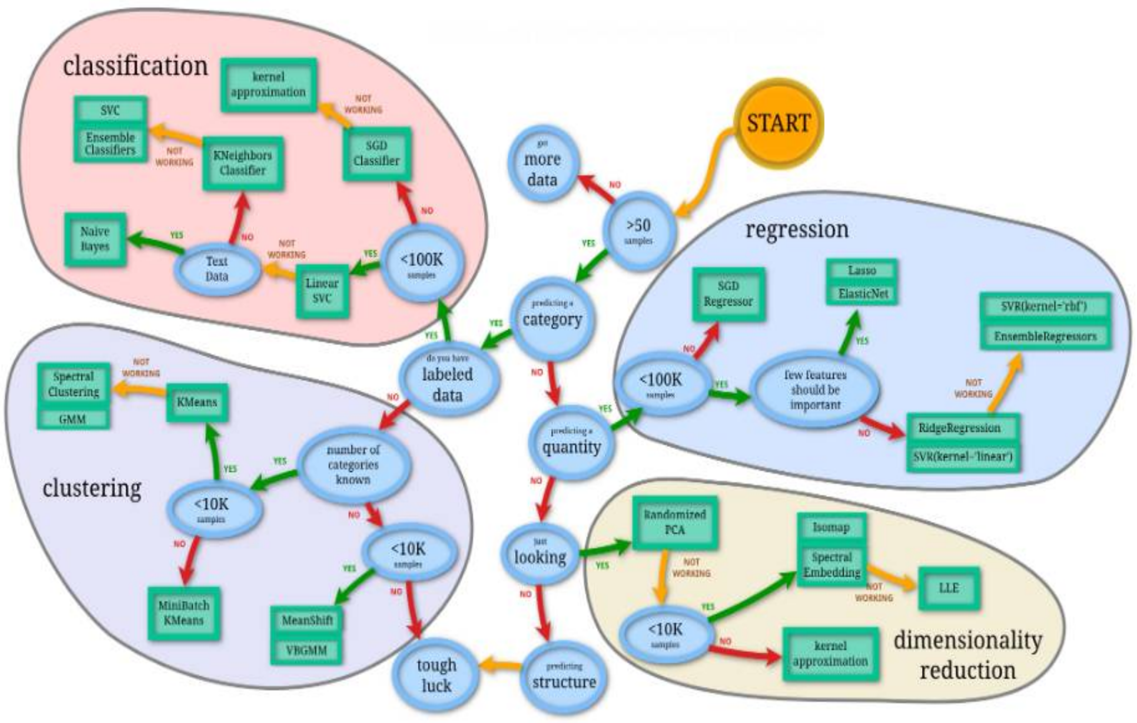
Running model on test data and generating predictions with RFC Non Generalized Model as it is giving the better accuracy precision and recall on test features



```
X_train,X_test,y_train,y_test = train_test_split(X,
                                                 y,
                                                 test_size=0.2,
                                                 random_state=37)

RFC=RandomForestClassifier(random_state = 37,n_estimators = 500)
RFC.fit(X_train,y_train)

X_final=test[['City','Average_Cost','Minimum_Order','Rating','Votes','Reviews','Count Cuisine']]
X_final = pd.get_dummies(data=X_final, drop_first=True)
pred=RFC.predict(X_final)
pred
sub=pd.DataFrame()
sub["Restaurant"]=test["Restaurant"]
sub["Delivery_Time"]=np.round(pred).astype(int)
sub["Delivery_Time"]=sub["Delivery_Time"].astype(str)+" minutes"
sub.to_excel("submission.xlsx",index=False)
sub
```

7]:

	Restaurant	Delivery_Time
0	ID_2842	30 minutes
1	ID_730	30 minutes
2	ID_4620	30 minutes
3	ID_5470	30 minutes
4	ID_3249	30 minutes
5	ID_506	30 minutes
6	ID_8321	45 minutes
7	ID_4559	30 minutes
8	ID_7982	30 minutes
9	ID_2869	30 minutes
10	ID_2728	30 minutes
11	ID_3977	30 minutes
12	ID_494	30 minutes
13	ID_2213	30 minutes
14	ID_6897	30 minutes



Chapter 6: Reference

https://www.machinehack.com/

https://www.analyticsvidhya.com/