A PROJECT REPORT ON

Restaurant Revenue Prediction

A project report submitted in fulfillment for the Diploma Degree in AI & ML Under Applied Roots with University of Hyderabad

Project submitted by

Vidvath Krishna

Under the Guidance of

Mentor: Harsh

Approved by: Mentor:

University of Hyderabad

Declaration of Authorship

We hereby declare that this thesis titled" Restaurant Revenue Prediction" and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Vidvath Krishna

Thesis Title: Restaurant Revenue Prediction

CERTIFICATE OF RECOMMENDATION

ACKNOWLEDGEMENT

I would like to acknowledge and give my warmest thanks to our project mentor Mr. Harsha who made this work possible. His advice and guidance carried me through all the stages of building up the project.

I would also like to give special thanks to the whole Applied Roots team and University of Hyderabad for giving out an opportunity to work in a project which ultimately helped to observe, learn and research on the concepts thoroughly and enhance my knowledge and understanding.

It’s been an immense pleasure and proud to work with a team like this, and I sincerely appreciate the team members for participating in discussions and sharing ideas with each other. I believe the successful completion of this project was made possible by the support of our Mentor, Applied roots team and friends.

Name: Vidvath Krishna

CONTENTS

1. **Abstract**

Food/Restaurant Industry is one of the evergreen Business fields and it has showed more of its potential in the recent days. Nowadays, Customers has got a wide range of options for Food based on the Cuisine type, Food quality, Taste and Varieties and they always looks for something new and better in every aspect. So, this industry can expect a tremendous growth in the future also. Large Capital investment, Effort and Time are required for setting up and run a restaurant smoothly. Therefore, it is crucial to consider a lot of factors while launching a restaurant anywhere. Our goal is to make use of the power of Data through various Machine Learning, Deep Learning techniques to increase the effectiveness of investments in new restaurant sites and to gain an upper hand in the competitive business.

2. Introduction

Tab Food Investments (TFI) is the company behind some of the world's most well-known brands such as Burger King, Sbarro, Popeyes, Usta Donerci and Arby's, with over 1200 quick service restaurants across the globe. They employ over 20,000 people in Europe and Asia, and they are the Leading Quick Service Restaurant (QSR) operator in Turkey and China.

3.Problem Statement

Right now, new restaurants are launched in a particular location is commonly a subjective process based on the personal judgement and experience of development teams. This subjective data is hard to meaningfully extrapolate across different geographies and cultures. It takes large investments of time and capital to set up new restaurants and running. The site closes within 18 months by incurring operating losses, when the wrong location for a restaurant brand is chosen. We are expected to find a mathematical model to increase the effectiveness of investments in new restaurant sites which would allow TFI to invest more in other important business areas, like innovation, sustainability, and training for new employees. We would try to predict the Annual Restaurant sales of 100,000 regional locations based on the given Commercial, Demographic and Real Estate data.

4.Data Description:

I. Source of the Dataset:

The TFI Dataset for predicting Restaurant Revenue prediction is available on Kaggle. Both the train and test dataset are provided here.

The dataset can be downloaded from the following link:

Link: https://www.kaggle.com/competitions/restaurant-revenue-prediction/data

II. Overview

The TFI Dataset consists the Train and Test Dataset files. There are 137 entries of Restaurants in the Train dataset which will used for training and fitting the model. The Test dataset consists of 100,000 samples of data which will be used for the evaluation of the model. We can observe that the train dataset provided is very small compared to the test dataset.

The Test Dataset has similar features of the Train Dataset except that the Revenue column can be only found in the train data. The features of the dataset are described below:

- ID – Restaurant Id
- Open Date- Date of Launch of the Restaurant
- City- The city name where the Restaurant situates
- City Group- The category of the city which the restaurant belongs.
- Type- The type of the Restaurant with categories as Food court, Drive-Thru, Mobile, Inline.
- P variables (P1-P37)- These variables represent the obfuscate data of three categories:

1.Demographic data - Population, Age, Gender etc.

2.Real Estate data – Car parking availability, Front Facade etc.

3.Commercial data - Points of Interest such as Colleges, Market etc.

- Revenue- Annual Revenue of a Restaurant in a given year.

So, there are 43 features in the Train dataset including the Restaurant id. Since the revenue column is excluded in the test data it has 42 attributes in total.

III. Challenges:

The difference in the size between the Train and Test dataset is huge. The fact that the train dataset has only a small number of data points would be a challenging part to deal with, from the Data modelling perspective. There is big probability that the models can overfit easily, especially the complex ones, which can negatively impact in the prediction of the revenue. To prevent this, we would be needed to use regularization techniques.

Also, the dataset contains some missing values, categorical features. These data have to be preprocessed before going ahead with the Modelling. We would be using the different libraries like pandas, scikit-learn for Data Preprocessing.

5.Approach, Methodology and Results

1) Requirement Analysis

a) Functional Requirements,

- **Purpose**

  To find a Mathematical model to predict the Annual restaurant sales of 100,000 regional locations based on the given Commercial, Demographic and Real Estate data.
- **Inputs**

  The Commercial, Demographic and Real Estate data regarding the restaurant.
- **Outputs**

  The model predicts the annual revenue generated by the given restaurant based on the input data.

b) Software Requirements,

- Anaconda Navigator/ Jupyter notebook
- Python 3

- Streamlit
- MS Excel/ Notepad
- Frameworks/Libraries- Numpy, scikit-learn, Pandas
- Web browser
- Heroku Cloud Platform

2) Data Overview and Initial Preprocessing

- Importing necessary Libraries,

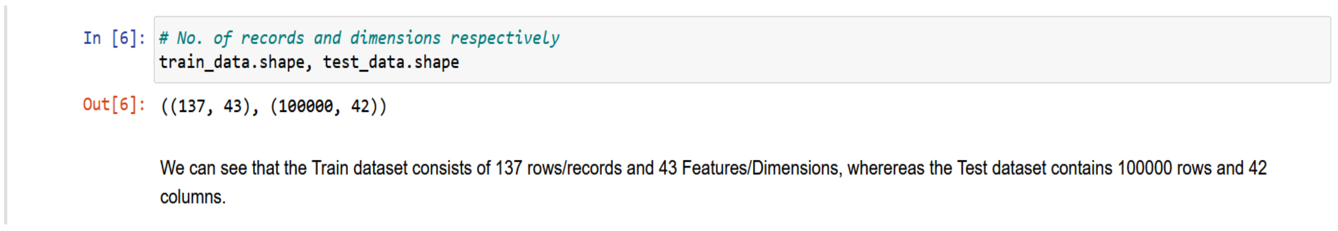
  The first step before doing anything on the data is importing the necessary libraries required for our project. Some of the basic and important libraries that we have used are listed below:

  - Pandas
  - NumPy
  - Matplotlib
  - Seaborn
  - SciPy
  - Scikit-learn

- Loading the dataset,

  We have two datasets to load - Train Dataset(train.csv) & Test Dataset (test.csv). These data files are loaded into Python pandas dataframes using pandas library.

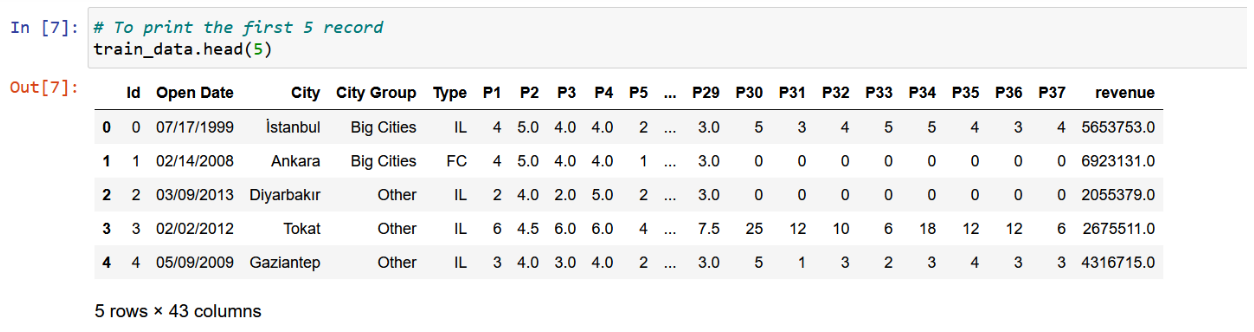
- Exploring the Dataset,
    - Viewing no. of record and features in Dataset,



```
In [6]: # No. of records and dimensions respectively
        train_data.shape, test_data.shape
Out[6]: ((137, 43), (100000, 42))
```

We can see that the Train dataset consists of 137 rows/records and 43 Features/Dimensions, wherereas the Test dataset contains 100000 rows and 42 columns.

- Viewing first 5 records of Train and Test Dataset,
    - Train Dataset,

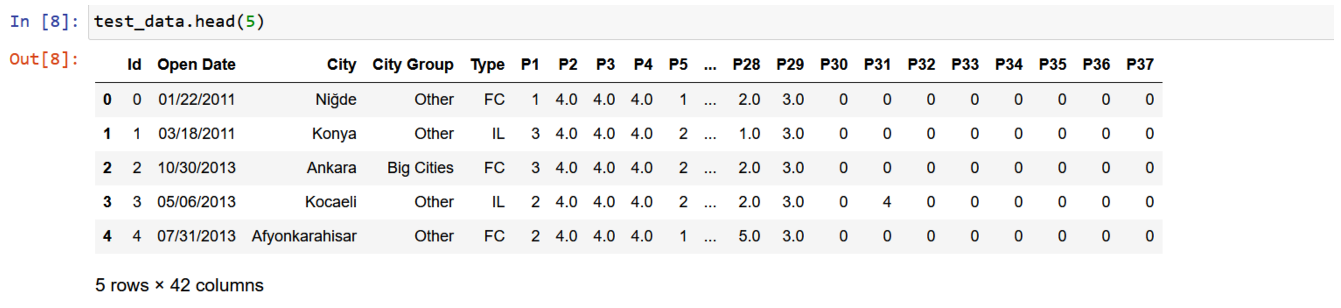


```
In [7]: # To print the first 5 record
        train_data.head(5)
Out[7]:
```

	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	...	P29	P30	P31	P32	P33	P34	P35	P36	P37	revenue
0	0	07/17/1999	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	...	3.0	5	3	4	5	5	4	3	4	5653753.0
1	1	02/14/2008	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	...	3.0	0	0	0	0	0	0	0	0	6923131.0
2	2	03/09/2013	Diyarbakır	Other	IL	2	4.0	2.0	5.0	2	...	3.0	0	0	0	0	0	0	0	0	2055379.0
3	3	02/02/2012	Tokat	Other	IL	6	4.5	6.0	6.0	4	...	7.5	25	12	10	6	18	12	12	6	2675511.0
4	4	05/09/2009	Gaziantep	Other	IL	3	4.0	3.0	4.0	2	...	3.0	5	1	3	2	3	4	3	3	4316715.0

5 rows × 43 columns

- Test Dataset,



```
In [8]: test_data.head(5)
Out[8]:
```

	Id	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	...	P28	P29	P30	P31	P32	P33	P34	P35	P36	P37
0	0	01/22/2011	Niğde	Other	FC	1	4.0	4.0	4.0	1	...	2.0	3.0	0	0	0	0	0	0	0	0
1	1	03/18/2011	Konya	Other	IL	3	4.0	4.0	4.0	2	...	1.0	3.0	0	0	0	0	0	0	0	0
2	2	10/30/2013	Ankara	Big Cities	FC	3	4.0	4.0	4.0	2	...	2.0	3.0	0	0	0	0	0	0	0	0
3	3	05/06/2013	Kocaeli	Other	IL	2	4.0	4.0	4.0	2	...	2.0	3.0	0	4	0	0	0	0	0	0
4	4	07/31/2013	Afyonkarahisar	Other	FC	2	4.0	4.0	4.0	1	...	5.0	3.0	0	0	0	0	0	0	0	0

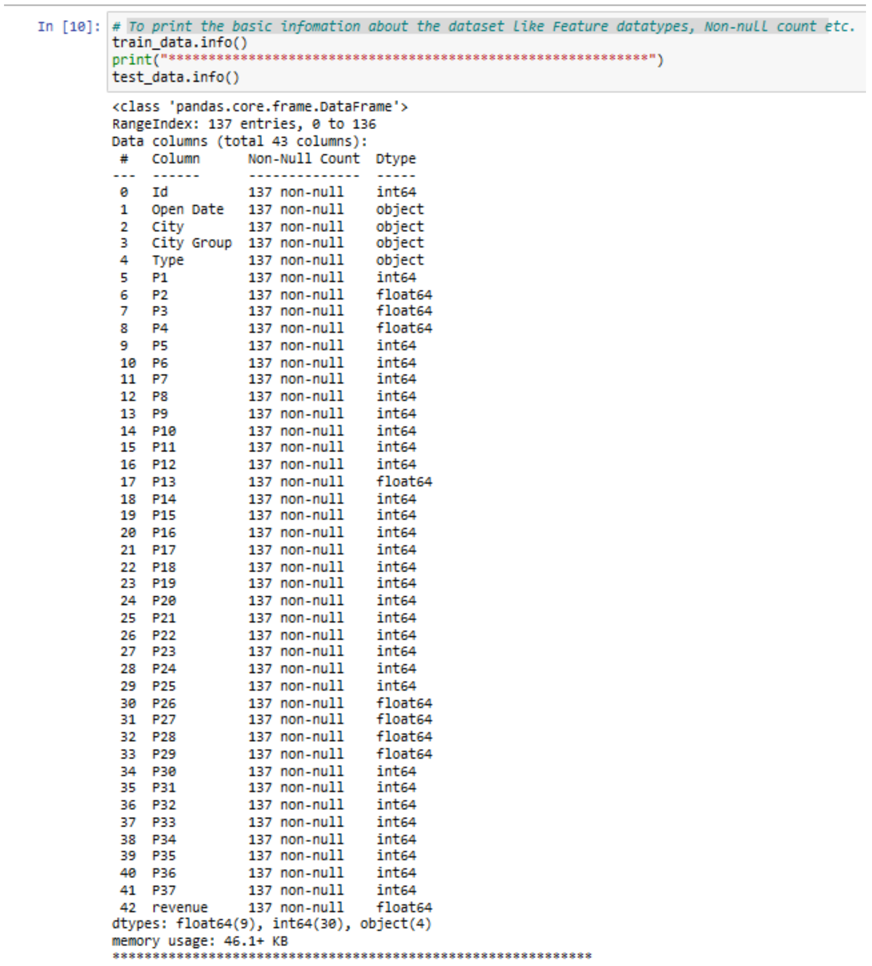
5 rows × 42 columns

From the above tables we could find the Test dataset doesn't consist of the 'revenue' column.

So, the target variable is only present in the Train Dataset. We must predict the Revenue variable using the features given in the test dataset.

- To view the dataset information,

  The info() method **prints information about the DataFrame**. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column.



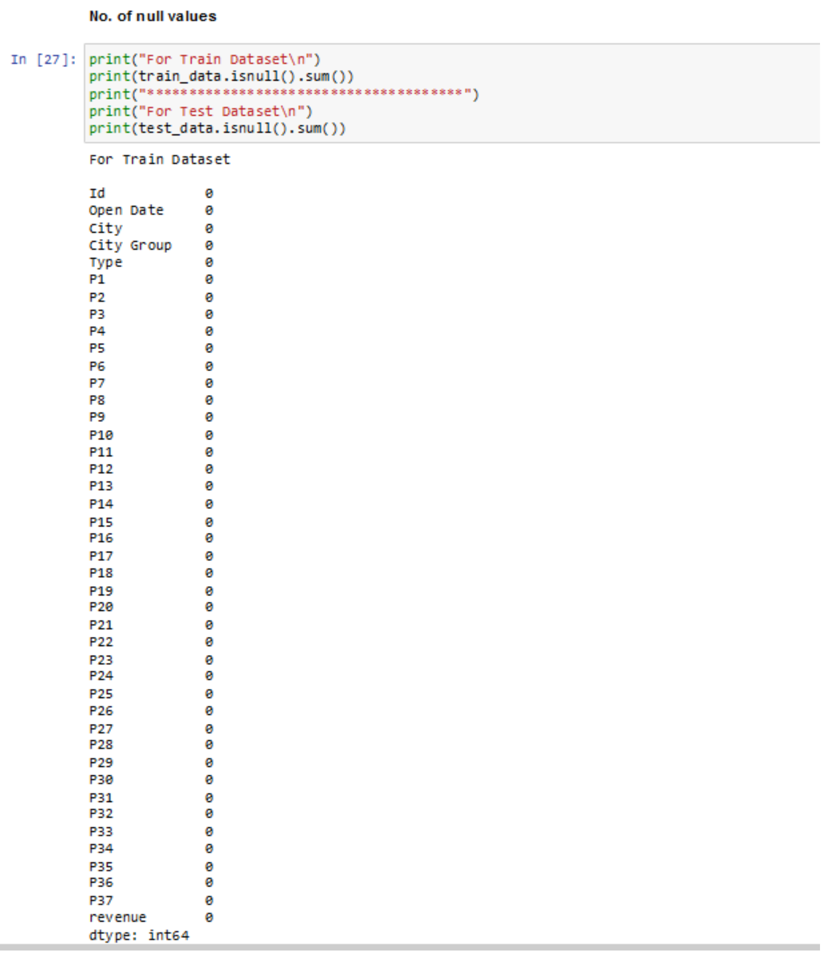
```
In [10]: # To print the basic infomation about the dataset like Feature datatypes, Non-null count etc.
         train_data.info()
         print("************************************************************")
         test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137 entries, 0 to 136
Data columns (total 43 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Id          137 non-null    int64  
 1   Open Date   137 non-null    object 
 2   City        137 non-null    object 
 3   City Group  137 non-null    object 
 4   Type        137 non-null    object 
 5   P1          137 non-null    int64  
 6   P2          137 non-null    float64
 7   P3          137 non-null    float64
 8   P4          137 non-null    float64
 9   P5          137 non-null    int64  
 10  P6          137 non-null    int64  
 11  P7          137 non-null    int64  
 12  P8          137 non-null    int64  
 13  P9          137 non-null    int64  
 14  P10         137 non-null    int64  
 15  P11         137 non-null    int64  
 16  P12         137 non-null    int64  
 17  P13         137 non-null    float64
 18  P14         137 non-null    int64  
 19  P15         137 non-null    int64  
 20  P16         137 non-null    int64  
 21  P17         137 non-null    int64  
 22  P18         137 non-null    int64  
 23  P19         137 non-null    int64  
 24  P20         137 non-null    int64  
 25  P21         137 non-null    int64  
 26  P22         137 non-null    int64  
 27  P23         137 non-null    int64  
 28  P24         137 non-null    int64  
 29  P25         137 non-null    int64  
 30  P26         137 non-null    float64
 31  P27         137 non-null    float64
 32  P28         137 non-null    float64
 33  P29         137 non-null    float64
 34  P30         137 non-null    int64  
 35  P31         137 non-null    int64  
 36  P32         137 non-null    int64  
 37  P33         137 non-null    int64  
 38  P34         137 non-null    int64  
 39  P35         137 non-null    int64  
 40  P36         137 non-null    int64  
 41  P37         137 non-null    int64  
 42  revenue     137 non-null    float64
dtypes: float64(9), int64(30), object(4)
memory usage: 46.1+ KB
************************************************************
```

- Number of Null/Missing values,

  Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA (Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected, or it never existed.

  So it is very important to deal with the missing values, hence we have checked the number of missing values in each columns using the *isnull().sum()* method.



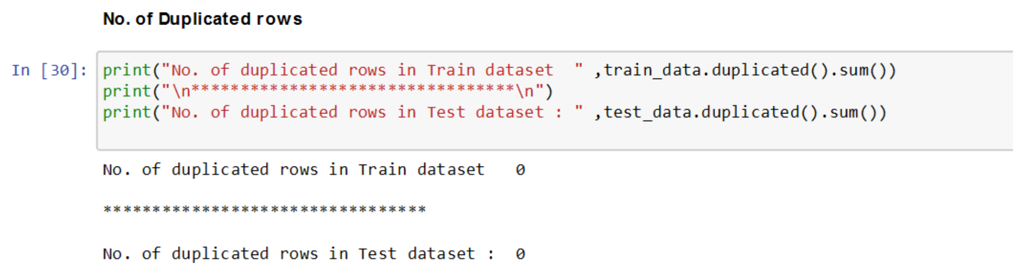
No. of null values

```
In [27]: print("For Train Dataset\n")
         print(train_data.isnull().sum())
         print("****************************************")
         print("For Test Dataset\n")
         print(test_data.isnull().sum())
```

```
For Train Dataset

Id            0
Open Date     0
City          0
City Group    0
Type          0
P1            0
P2            0
P3            0
P4            0
P5            0
P6            0
P7            0
P8            0
P9            0
P10           0
P11           0
P12           0
P13           0
P14           0
P15           0
P16           0
P17           0
P18           0
P19           0
P20           0
P21           0
P22           0
P23           0
P24           0
P25           0
P26           0
P27           0
P28           0
P29           0
P30           0
P31           0
P32           0
P33           0
P34           0
P35           0
P36           0
P37           0
revenue       0
dtype: int64
```

There are no null values in the Train and Test dataset which implies we don't have to implement the different preprocessing steps for handling the null or missing values.

- Number of Duplicated rows,



No. of Duplicated rows

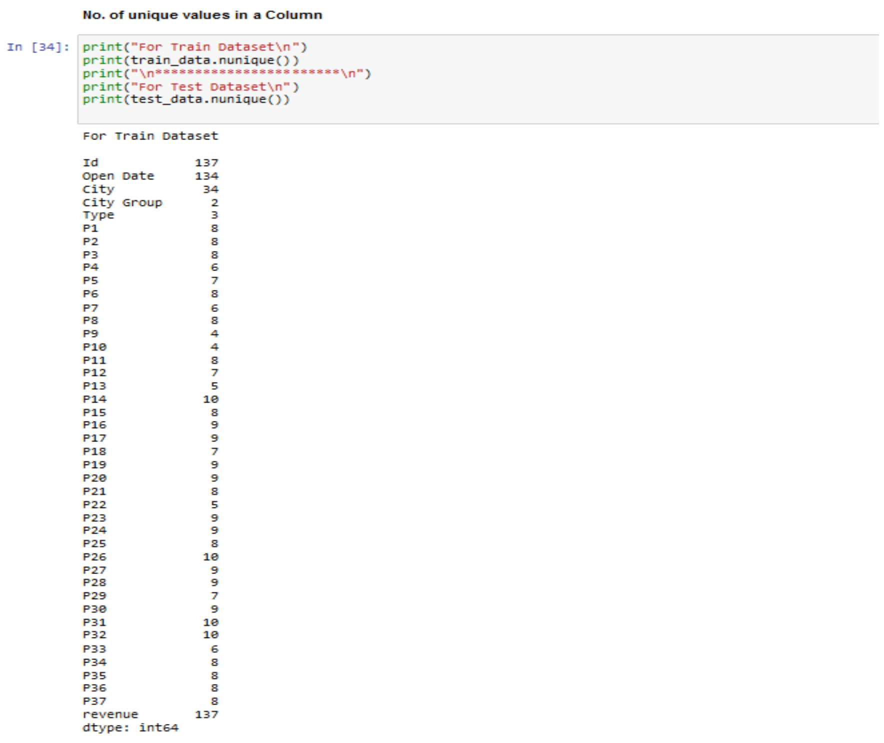
```
In [30]: print("No. of duplicated rows in Train dataset  " ,train_data.duplicated().sum())
         print("\n**********************************\n")
         print("No. of duplicated rows in Test dataset : " ,test_data.duplicated().sum())
```

```
No. of duplicated rows in Train dataset   0

**********************************

No. of duplicated rows in Test dataset :  0
```

There are no duplicate rows as well in the Train and Test Dataset.

- Number of unique values,

Understanding the unique values in a feature can give more clarity about our dataset.



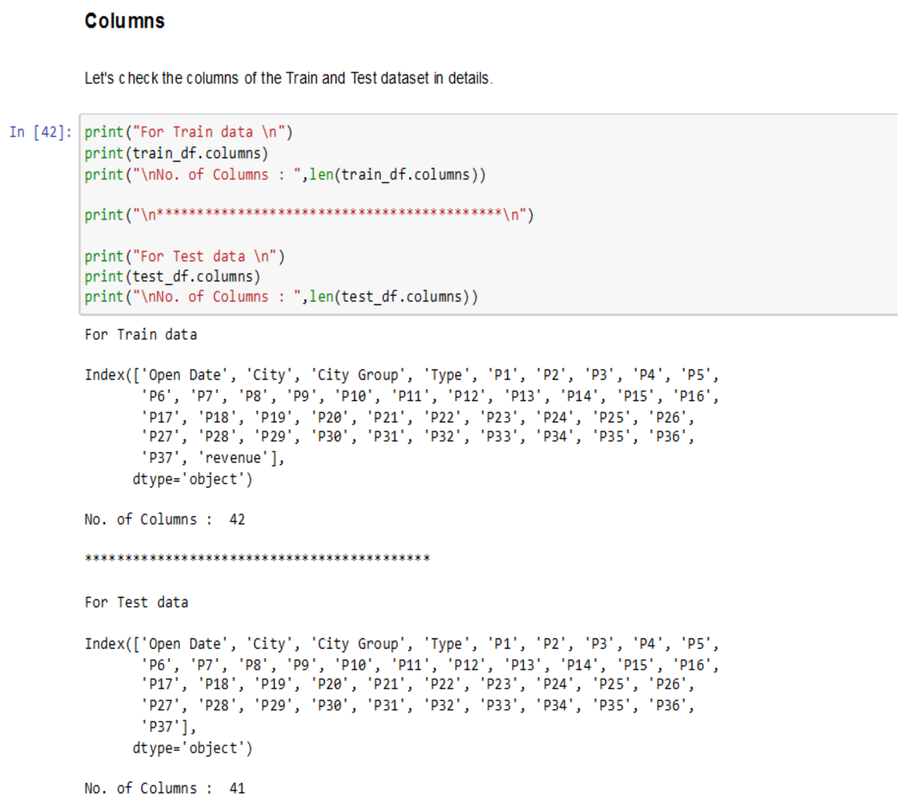
No. of unique values in a Column

```
In [34]: print("For Train Dataset\n")
         print(train_data.nunique())
         print("\n*************************\n")
         print("For Test Dataset\n")
         print(test_data.nunique())
```

```
For Train Dataset

Id            137
Open Date     134
City           34
City Group      2
Type            3
P1              8
P2              8
P3              8
P4              6
P5              7
P6              8
P7              6
P8              8
P9              4
P10             4
P11             8
P12             7
P13             5
P14            10
P15             8
P16             9
P17             9
P18             7
P19             9
P20             9
P21             8
P22             5
P23             9
P24             9
P25             8
P26            10
P27             9
P28             9
P29             7
P30             9
P31            10
P32            10
P33             6
P34             8
P35             8
P36             8
P37             8
revenue       137
dtype: int64
```

Since the Id column/feature is just an index it will not add any importance to our model. So we have dropped this column from the dataframe.

- Columns/ Features,


Columns

Let's check the columns of the Train and Test dataset in details.

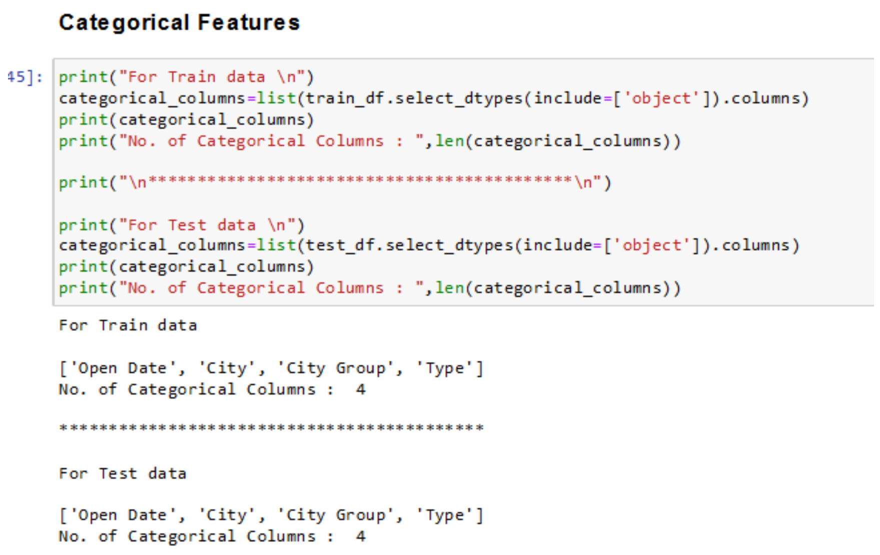
```
In [42]: print("For Train data \n")
         print(train_df.columns)
         print("\nNo. of Columns : ",len(train_df.columns))

         print("\n*******************************************\n")

         print("For Test data \n")
         print(test_df.columns)
         print("\nNo. of Columns : ",len(test_df.columns))
```

```
For Train data

Index(['Open Date', 'City', 'City Group', 'Type', 'P1', 'P2', 'P3', 'P4', 'P5',
       'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16',
       'P17', 'P18', 'P19', 'P20', 'P21', 'P22', 'P23', 'P24', 'P25', 'P26',
       'P27', 'P28', 'P29', 'P30', 'P31', 'P32', 'P33', 'P34', 'P35', 'P36',
       'P37', 'revenue'],
      dtype='object')

No. of Columns :  42

*******************************************

For Test data

Index(['Open Date', 'City', 'City Group', 'Type', 'P1', 'P2', 'P3', 'P4', 'P5',
       'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16',
       'P17', 'P18', 'P19', 'P20', 'P21', 'P22', 'P23', 'P24', 'P25', 'P26',
       'P27', 'P28', 'P29', 'P30', 'P31', 'P32', 'P33', 'P34', 'P35', 'P36',
       'P37'],
      dtype='object')

No. of Columns :  41
```

There are 42 features in the train dataset which consist of 41 independent variables and 1 dependent/target/response variable (after removing the id column).

The test dataset consists of the same independent variables whereas it doesn't have the response variable. We will use Kaggle scores for validating the score of our model on the test dataset.

- Categorical features,



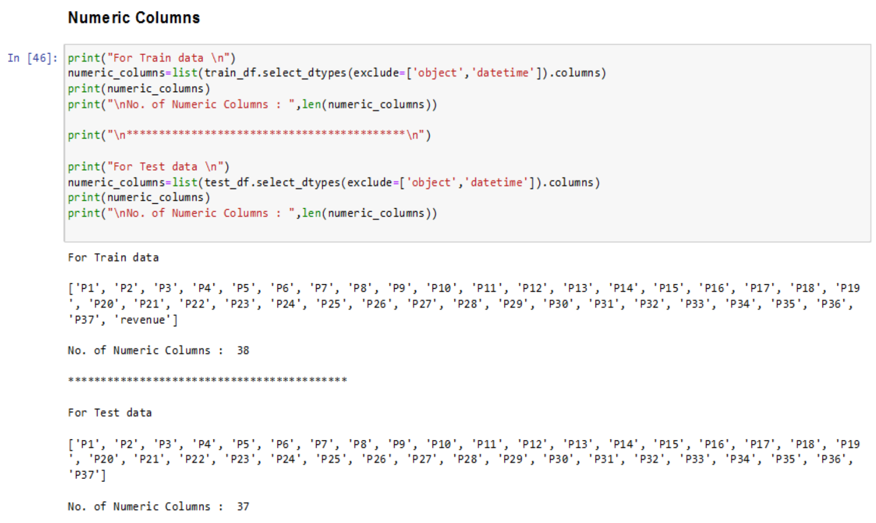
Categorical Features

```
45]: print("For Train data \n")
categorical_columns=list(train_df.select_dtypes(include=['object']).columns)
print(categorical_columns)
print("No. of Categorical Columns : ",len(categorical_columns))

print("\n********************************************\n")

print("For Test data \n")
categorical_columns=list(test_df.select_dtypes(include=['object']).columns)
print(categorical_columns)
print("No. of Categorical Columns : ",len(categorical_columns))
```

```
For Train data

['Open Date', 'City', 'City Group', 'Type']
No. of Categorical Columns :  4

********************************************

For Test data

['Open Date', 'City', 'City Group', 'Type']
No. of Categorical Columns :  4
```

The categorical features in the dataset are – Open Date, City, City Group, Type. These variables has to be processed if required and label encoded to numerical values as the model can only process numbers.

- Numerical Columns,



Numeric Columns

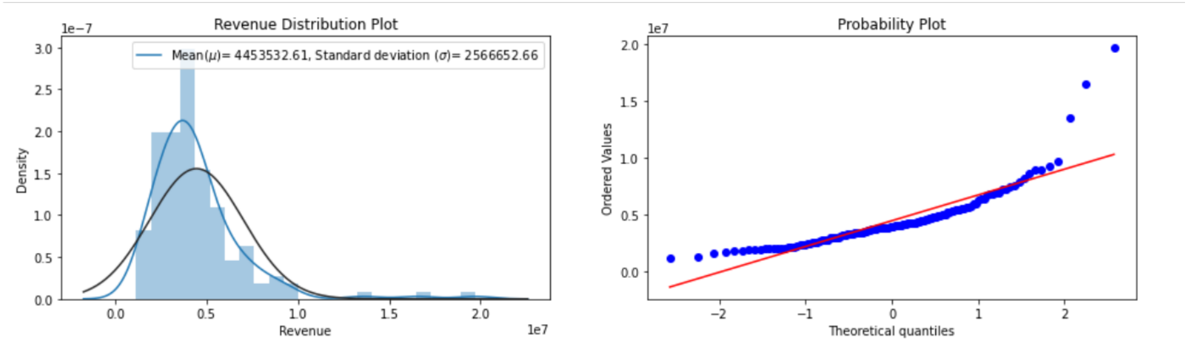
```
In [46]: print("For Train data \n")
numeric_columns=list(train_df.select_dtypes(exclude=['object','datetime']).columns)
print(numeric_columns)
print("\nNo. of Numeric Columns : ",len(numeric_columns))

print("\n*******************************************\n")

print("For Test data \n")
numeric_columns=list(test_df.select_dtypes(exclude=['object','datetime']).columns)
print(numeric_columns)
print("\nNo. of Numeric Columns : ",len(numeric_columns))
```

```
For Train data

['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16', 'P17', 'P18', 'P19
', 'P20', 'P21', 'P22', 'P23', 'P24', 'P25', 'P26', 'P27', 'P28', 'P29', 'P30', 'P31', 'P32', 'P33', 'P34', 'P35', 'P36',
'P37', 'revenue']

No. of Numeric Columns :  38

*******************************************

For Test data

['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16', 'P17', 'P18', 'P19
', 'P20', 'P21', 'P22', 'P23', 'P24', 'P25', 'P26', 'P27', 'P28', 'P29', 'P30', 'P31', 'P32', 'P33', 'P34', 'P35', 'P36',
'P37']

No. of Numeric Columns :  37
```

There are 37 independent numerical features in both train and test dataset. The Obfuscated variables from P1-P37 are all numerical variables.

3) Exploratory Data Analysis & Preprocessing

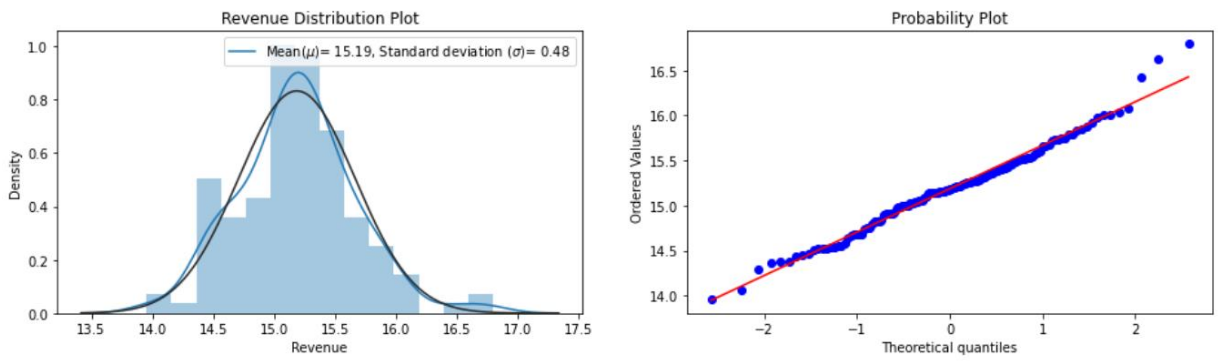
- Log Transforming the Revenue variable,



When the distribution plot of the revenue variable was plotted, it was observed that the distribution of revenue is right skewed. The presence of outliers in the data can be one reason for this.

So, we have done some transformation on the revenue column in order to make it normally distributed so that it can reduce the effect of outliers and improve the model performance.

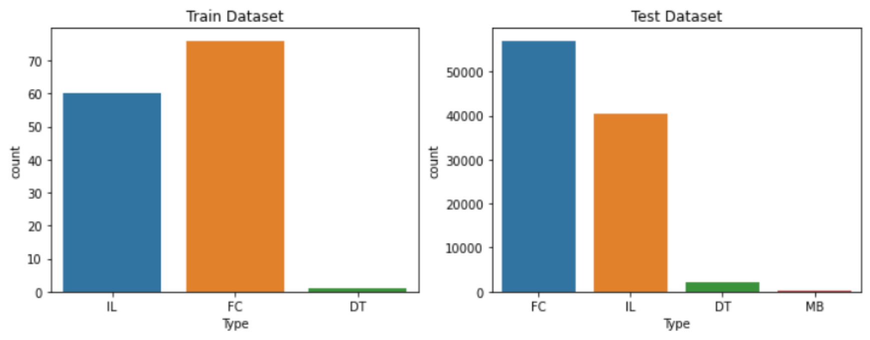
After Log transforming,



- Exploring Features,
  - Univariate Analysis on Categorical Features,

Univariate analysis-This method is perhaps the simplest form of statistical analysis. Like other forms of statistics, it can be inferential or descriptive. The key fact is that only one variable is involved.

- Type,

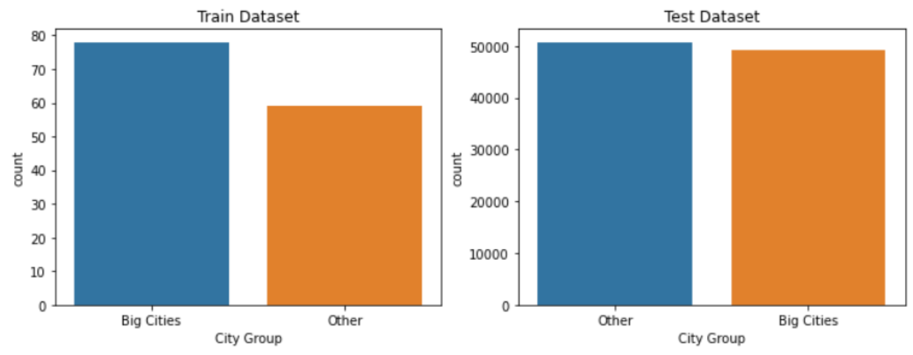


Inference: - From the above plot, it is observed that FC(Food-Courts) are highest in number in both train and test datasets which is followed by Inline Type Restaurants.

The Drive-Thru type restaurants are very small in number in both datasets while Mobile Type is not found in the Train datasets.

This implies that Food courts and Inline Type restaurants are the most preferred by people and generates more revenue since it has many outlets.

- City Group,

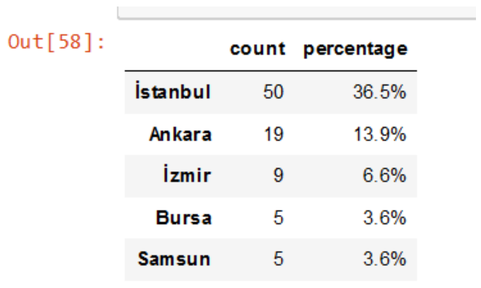


Inference: - The above visualizations shows that the number of restaurants in Big Cities are more than that in other cities.

But both categories have a small difference only in both of the datasets.

- City,

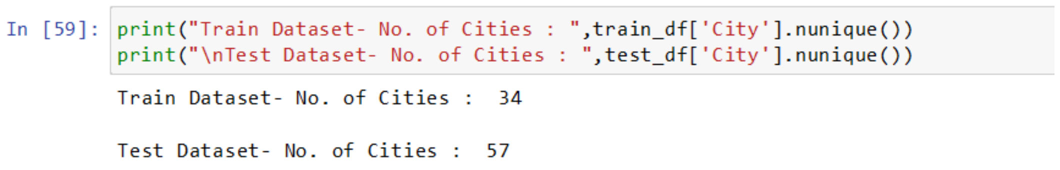
  There are many cities with very few numbers of restaurants. Most restaurants are in some important cities. The major cities with highest number of restaurants are:



Out[58]:

	count	percentage
İstanbul	50	36.5%
Ankara	19	13.9%
İzmir	9	6.6%
Bursa	5	3.6%
Samsun	5	3.6%

  Another key factor which can be noticed in the *city* feature is that the number of unique city names in the train and test dataset is different.



```
In [59]: print("Train Dataset- No. of Cities : ",train_df['City'].nunique())
         print("\nTest Dataset- No. of Cities : ",test_df['City'].nunique())

Train Dataset- No. of Cities :  34

Test Dataset- No. of Cities :  57
```

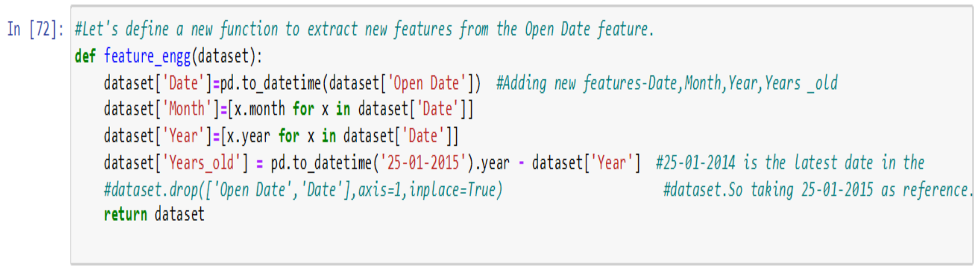
  So, from the above numbers we can understand that the Test dataset contains 23 more number of cities than that listed in the Train dataset.

  This may lead to some issues in model performance, so this must be tackled during the Feature engineering procedures.

- Open Date,

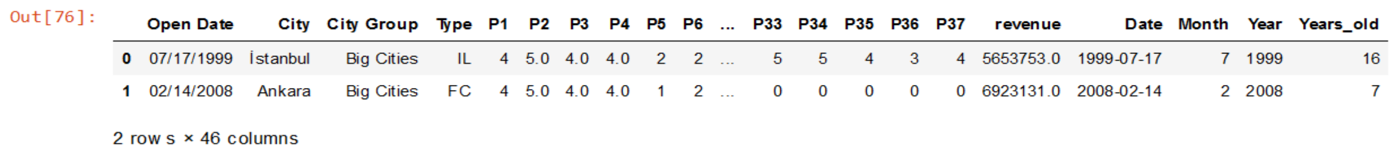
  The Open date feature as in its raw form may not be helpful for drawing out insights.

  But we have extracted some useful insights from the open date feature by doing some transformations.



```
In [72]: #Let's define a new function to extract new features from the Open Date feature.
         def feature_engg(dataset):
             dataset['Date']=pd.to_datetime(dataset['Open Date'])  #Adding new features-Date,Month,Year,Years _old
             dataset['Month']=[x.month for x in dataset['Date']]
             dataset['Year']=[x.year for x in dataset['Date']]
             dataset['Years_old'] = pd.to_datetime('25-01-2015').year - dataset['Year']  #25-01-2014 is the latest date in the
             #dataset.drop(['Open Date','Date'],axis=1,inplace=True)                     #dataset.So taking 25-01-2015 as reference.
             return dataset
```

We have extracted the Date, Month, Year, Years old from the *open date* feature.



Out[76]:

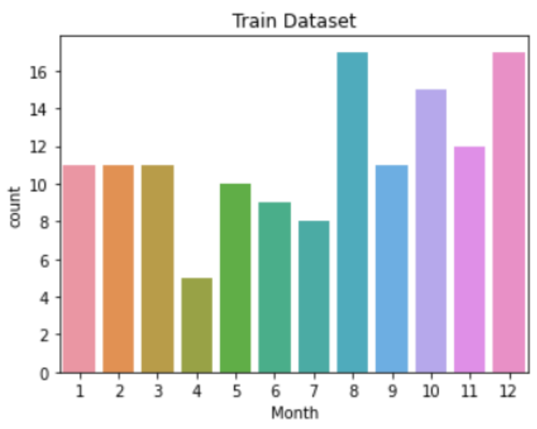
	Open Date	City	City Group	Type	P1	P2	P3	P4	P5	P6	...	P33	P34	P35	P36	P37	revenue	Date	Month	Year	Years_old
0	07/17/1999	İstanbul	Big Cities	IL	4	5.0	4.0	4.0	2	2	...	5	5	4	3	4	5653753.0	1999-07-17	7	1999	16
1	02/14/2008	Ankara	Big Cities	FC	4	5.0	4.0	4.0	1	2	...	0	0	0	0	0	6923131.0	2008-02-14	2	2008	7

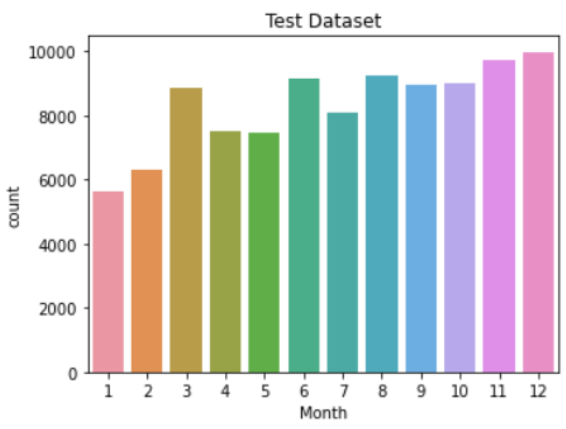
2 row s × 46 columns

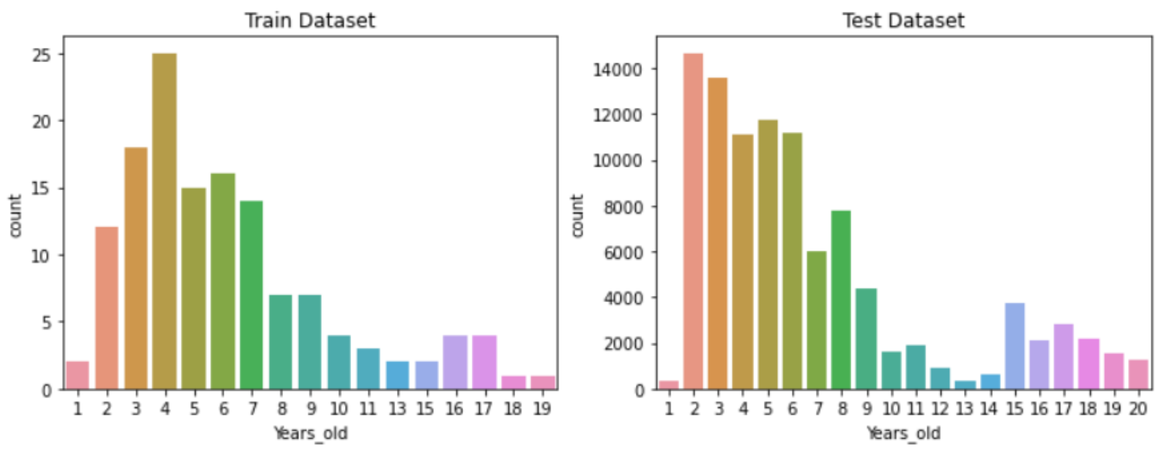
New features have been added to the dataframe-

- Date
- Month
- Year
- Years_old

These features can be made use for further explorations on data to generate meaningful insights.

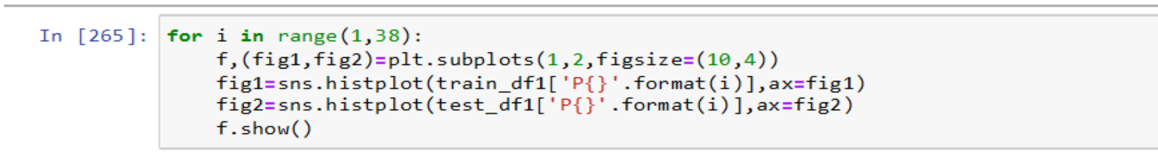




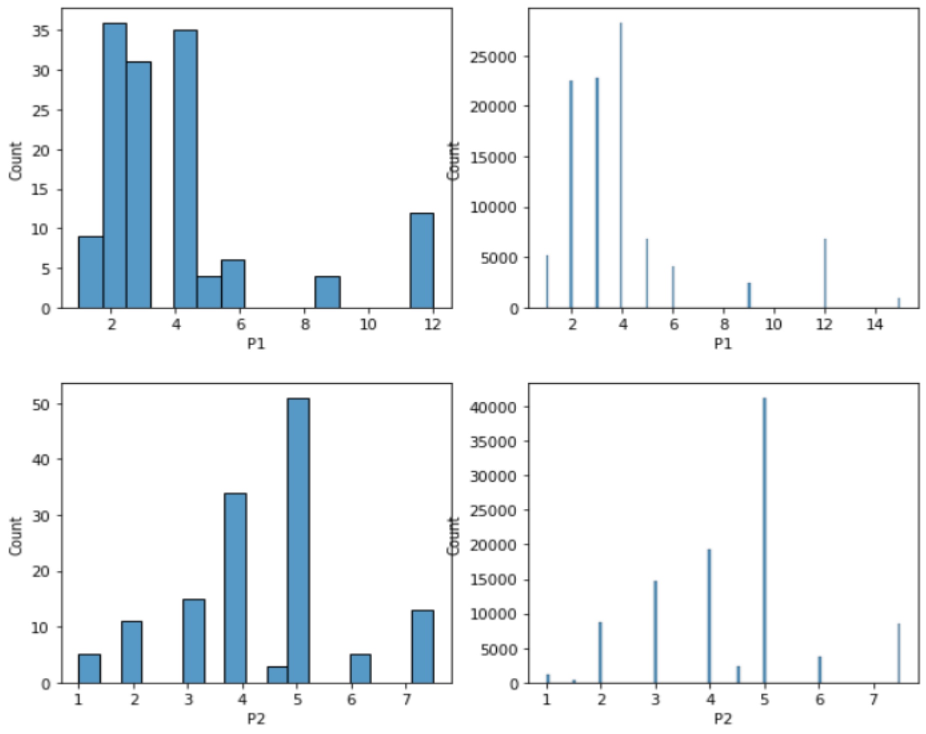


The above bar charts are generated using these new features.

- Univariate Analysis on Numerical Features,



```
In [265]: for i in range(1,38):
              f,(fig1,fig2)=plt.subplots(1,2,figsize=(10,4))
              fig1=sns.histplot(train_df1['P{}'.format(i)],ax=fig1)
              fig2=sns.histplot(test_df1['P{}'.format(i)],ax=fig2)
              f.show()
```

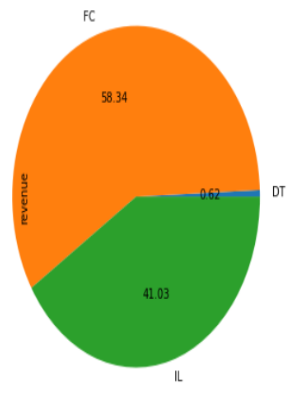


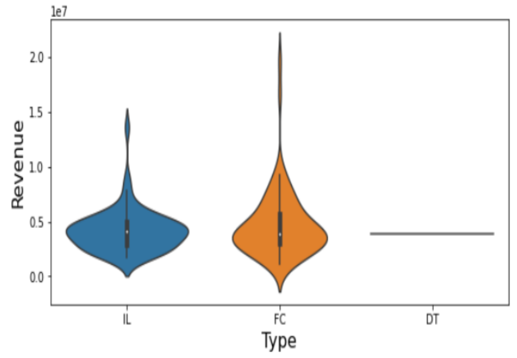
The Count plots of all the obfuscated variables(P1-P37) are plotted.

- Bivariate Analysis,

Bivariate Analysis: - Bi means two and variate means variable, so here there are two variables. The analysis is related to cause and the relationship between the two variables.

  - Type v/s Revenue,



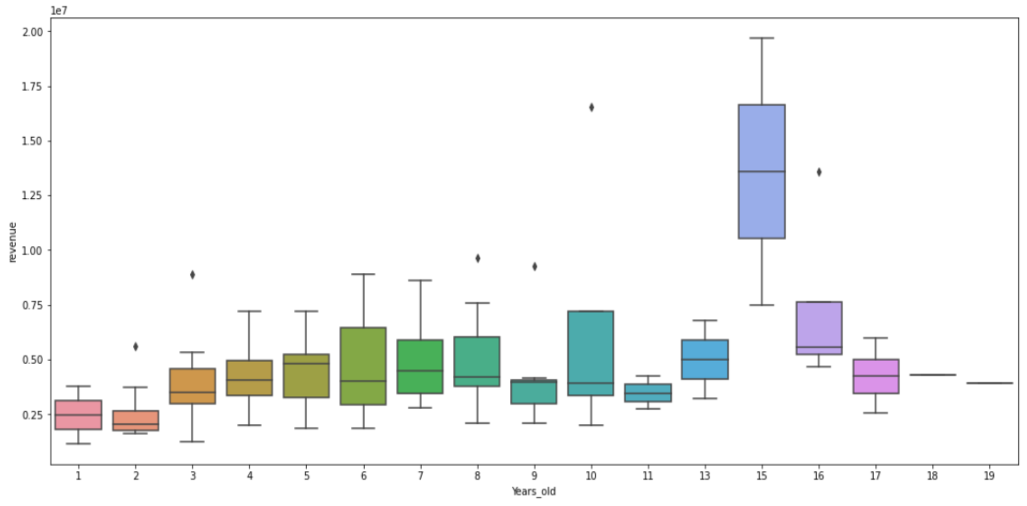


Inference: - From the Type v/s Revenue Pie plot it is evident that the Food Court type restaurants are generating the highest revenue. But this is obvious since the number of Food Court Type restaurants are also the highest.

So, we have plotted the Violin plot for giving more meaningful insights.

From the distributions it is evident that the Food Courts itself is the highest revenue generating type which followed Inline and Drive-Thru type restaurants closely behind.

  - Years old v/s Revenue



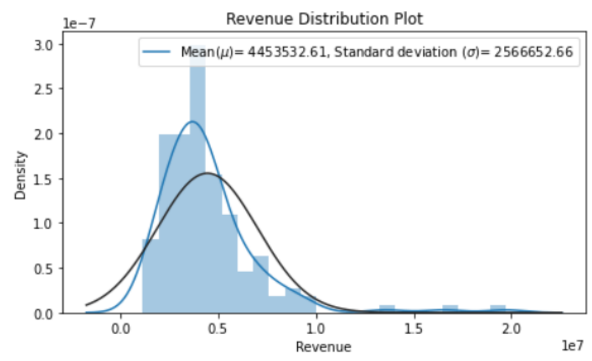
Inference: - From the above plot we see a trend that the restaurants which are established many years back are generating higher revenues when compared to newly launched restaurants.

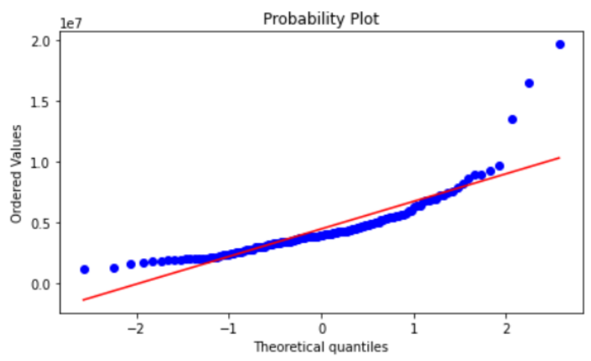
4) **Feature Engineering**

- Outlier Detection and Treatment,

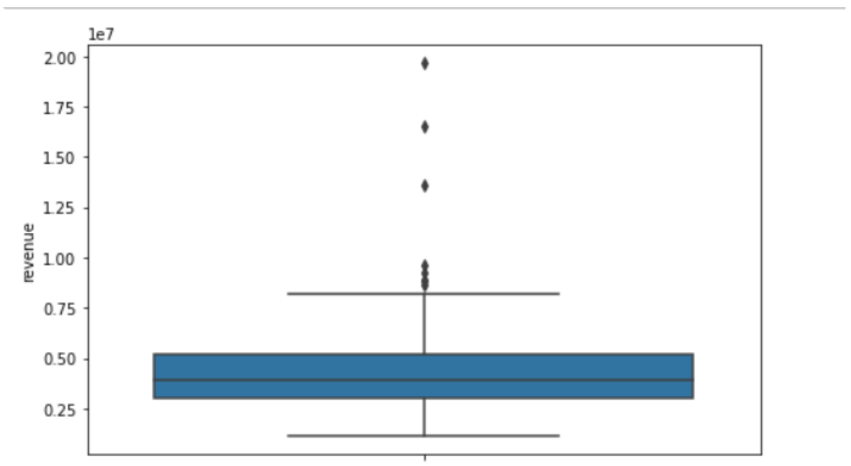
We are checking the presence of outliers from the target variable-*revenue.* The distribution and probplot of this variable are shown below:

Probability plot - A graphical technique for **assessing whether or not a data set follows a given distribution such as the normal or Weibull**. The data are plotted against a theoretical distribution in such a way that the points should form approximately a straight line.





In the above plots, the curve shows some skewness to the right. This implies the effect of outliers persists. This can be explored further using a Boxplot.



Some datapoints could be seen beyond the whiskers of the box. These are the outlier datapoints.

These points have to be treated accordingly. We have checked the percentile values in order to figure out the threshold for removing outliers from the data.

```
for i in range(90,100,1):
  print("At",i,"th percentile :",np.percentile(train_df3['revenue'],i).round(2))
```

```
At 90 th percentile : 7045417.4
At 91 th percentile : 7213830.0
At 92 th percentile : 7506753.6
At 93 th percentile : 7723386.88
At 94 th percentile : 8157828.64
At 95 th percentile : 8683465.2
At 96 th percentile : 8899910.16
At 97 th percentile : 9234060.4
At 98 th percentile : 10750754.72
At 99 th percentile : 15478481.6
```

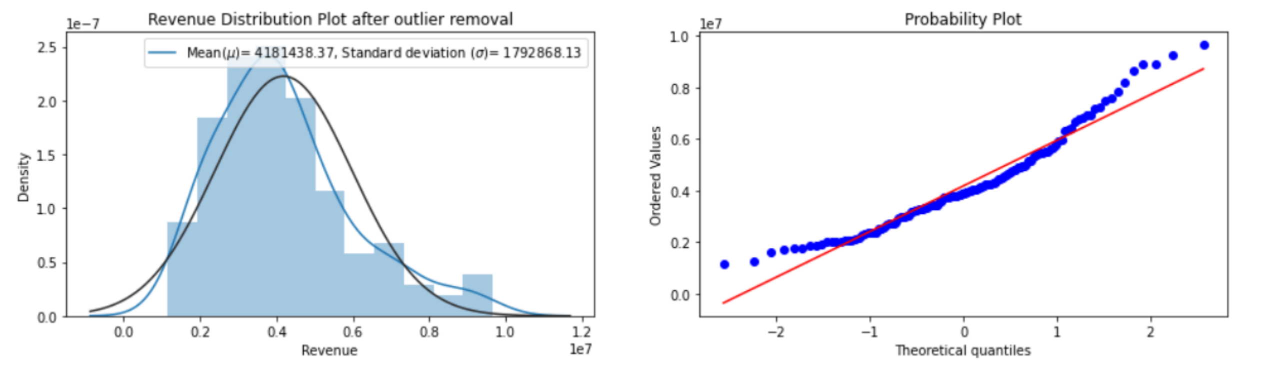
Since it is observed that the values are showing a huge difference at the 98th percentile this can be selected as the threshold and datapoints beyond that can be removed.

```
train_df3=train_df3[train_df3['revenue']<train_df3['revenue'].quantile(.98)]
train_df3.shape
```

```
(134, 46)
```

So, after the removal of outliers the data shape has changed from 137 to 134.

The below figure shows the distribution and probplots after outlier removal.



The skewness of the plot has significantly reduced by removing outliers.

- Dropping unnecessary columns,
  - City,

    As we have already seen in EDA, the number of cities in train and test dataset differs by a large number. So, using this feature may adversely affect the model's performance. So, we have dropped this feature.

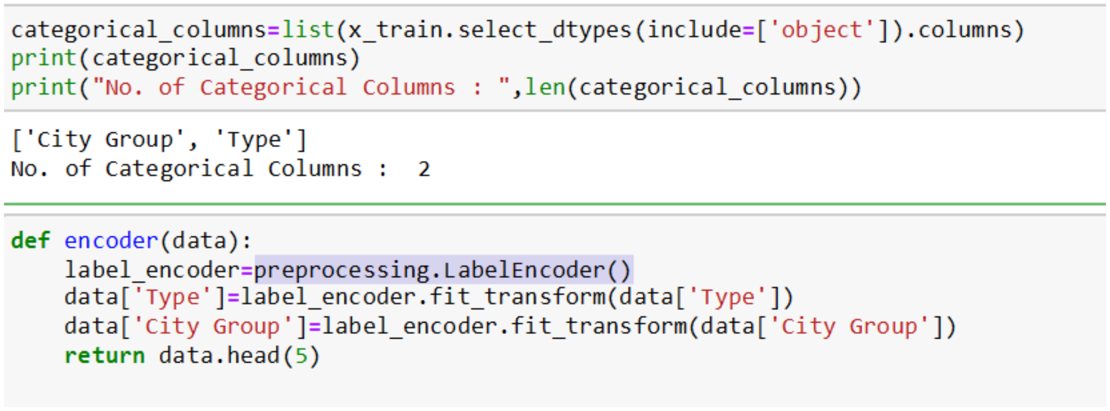
  - Date & Open Date,

    Since we have extracted/generated new features from the Open date feature we can drop these features as well.

- Label Encoding,

We have to encode the categorical features to numbers for the model to work.

LabelEnocoder () function - This converts each value in a categorical column into a numerical value. Each value in a categorical column is called Label.



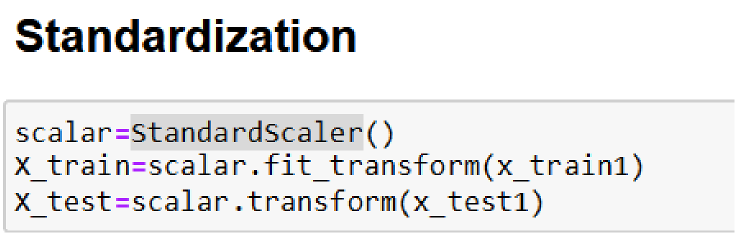
```
categorical_columns=list(x_train.select_dtypes(include=['object']).columns)
print(categorical_columns)
print("No. of Categorical Columns : ",len(categorical_columns))
```

```
['City Group', 'Type']
No. of Categorical Columns :  2
```

```
def encoder(data):
    label_encoder=preprocessing.LabelEncoder()
    data['Type']=label_encoder.fit_transform(data['Type'])
    data['City Group']=label_encoder.fit_transform(data['City Group'])
    return data.head(5)
```

- Standardizing Values,

StandardScaler () function - The idea behind the StandardScaler is that variables that are measured at different scales do not contribute equally to the fit of the model and the learning function of the model and could end up creating a bias. Standard Scaler follows standard normal distribution where mean is centered to zero and scales data to unit variance.



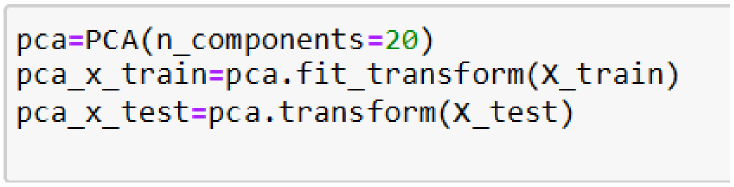
Standardization

```
scalar=StandardScaler()
X_train=scalar.fit_transform(x_train1)
X_test=scalar.transform(x_test1)
```

- Dimensionality Reduction using PCA,

Principal Component Analysis- PCA is a statistical technique for reducing the dimensionality of a dataset. It works by analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information, and enabling the visualization of multidimensional data.

We can choose any number of dimensions that we have to reduce the features to.



```
pca=PCA(n_components=20)
pca_x_train=pca.fit_transform(X_train)
pca_x_test=pca.transform(X_test)
```

5) Key Performance Indicator (KPI),

In this problem, we will be trying to predict the Annual revenue of the restaurants. Since the output we are predicting is a real/continuous value, this is a Regression type problem. For Regression models typically we employ some performance metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Squared Error (MSE) etc.

We can evaluate the performance of the developed model by comparing the Predicted revenue to the Actual revenue given in the dataset. Here, we can implement RMSE as the performance evaluation metric.

Root Mean Squared Error:

RMSE is one of the most popular evaluation metrics for Regression problems. It is the square root of the average squared errors/residuals. The mathematical formula of RMSE is given below:

where,

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Yi = Actual value

Yi hat= Predicted value

n=no. of datapoints

Lower the RMSE value implies the model performs better.

The key advantage of using RMSE as the metric is that the unit of the error, we obtain will be same as that of the output variable that we predict. In our case, if the revenue is expressed in the unit Dollars, then the error or residual value obtained will be also in Dollars. This would make the interpretation easier.

The Disadvantage of RMSE is that it is not much robust to the outliers. So, it is critical to process or handle the outliers when we are employing this metric for evaluation.

6) Predictive Modelling,

Our aim is to predict the revenue generated by a restaurant given the independent variables/features.

Hence this is a Regression type of problem.

Regression- It is **a technique for investigating the relationship between independent features or variables and a dependent variable or response**. It's used as a method for predictive modelling in machine learning, in which an algorithm/model is used to predict continuous outcomes.

We are using the below Regression models to predict the outcome.

I. Linear Regression
II. KNN Regressor
III. Random Forest
IV. Gradient Boosting Regressor
V. XGBoost

Let's go through each of these algorithms,

I. Linear Regression,

In Regression, we plot a graph between the variables which best fit the given data points. The machine learning model can deliver predictions regarding the data. In naïve words, ***"Regression plots a line or***

curve that passes through all the data points on a target-predictor graph in such a way that the vertical distance between the data points and the regression line is minimum." It is used principally for prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.

II. KNN Regressor,

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood. A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors.

The size of the neighborhood can be selected by using cross validation.

GridSearchCV- It is **a technique to search through given set of the grid of parameters for the best parameter values**. It is basically a cross-validation method. The model and the parameters are required to be fed into this function. Best parameter values are extracted and then the predictions are made.

III. Random Forest,

Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model. **Random Forest Regression** is a supervised machine learning algorithm that uses **Ensemble learning** method for regression.

IV. Gradient Boosting Regressor,

Gradient boosting Regression **calculates the difference between the prediction and the known target value**. This difference is called residual. After that Gradient boosting Regression trains a weak model that maps features to that residual.

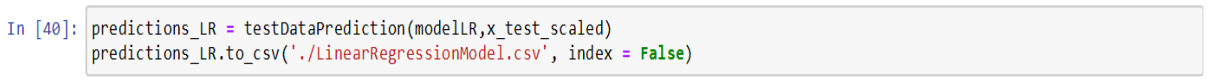
V. XGBoost,

XGBoost stands for Extreme Gradient Boost. It is a tree-based ensemble machine learning algorithm which has **higher predicting power and performance** and it is achieved by improvisation on Gradient Boosting framework by introducing some accurate approximation algorithms.

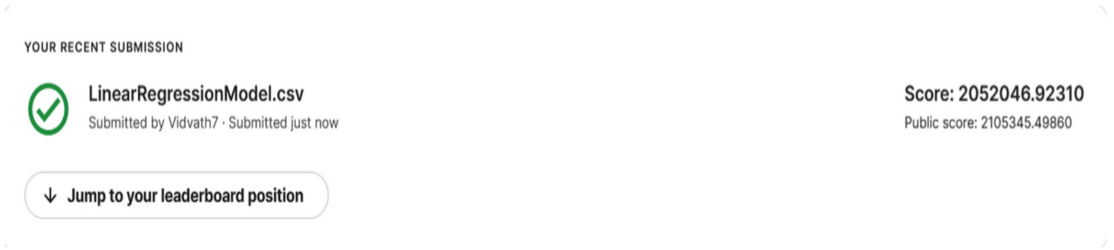
7) Comparing Performance of Models,

i. Logistic Regression,

a. Standard scaled data,



```
In [40]: predictions_LR = testDataPrediction(modelLR,x_test_scaled)
         predictions_LR.to_csv('./LinearRegressionModel.csv', index = False)
```



YOUR RECENT SUBMISSION

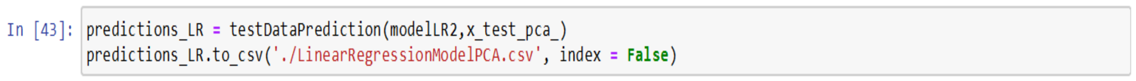
LinearRegressionModel.csv

Submitted by Vidvath7 · Submitted just now

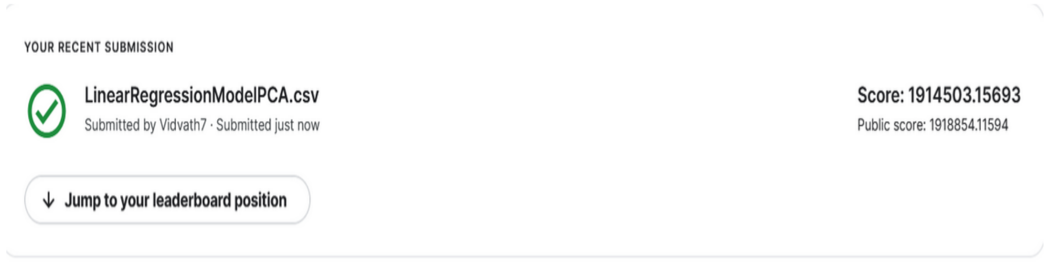
Score: 2052046.92310

Public score: 2105345.49860

b. PCA,



```
In [43]: predictions_LR = testDataPrediction(modelLR2,x_test_pca_)
         predictions_LR.to_csv('./LinearRegressionModelPCA.csv', index = False)
```



YOUR RECENT SUBMISSION

LinearRegressionModelPCA.csv

Submitted by Vidvath7 · Submitted just now

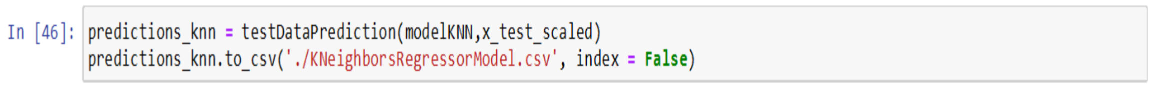
Score: 1914503.15693

Public score: 1918854.11594

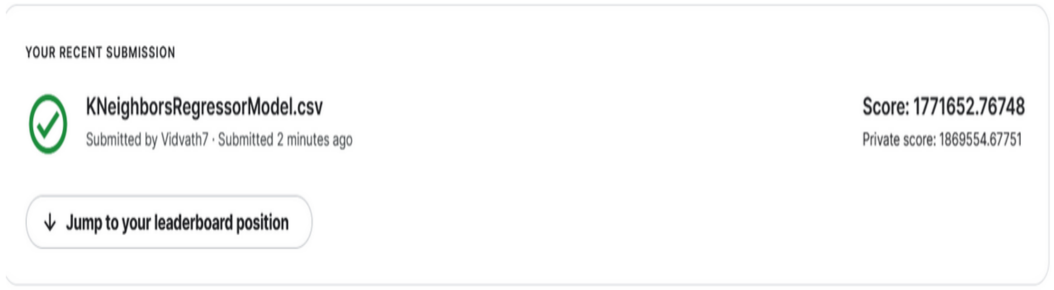
Inference: The PCA scaled data gives better results for Linear Regression.

ii. KNN Regressor,

a. Standard scaled data,



```
In [46]: predictions_knn = testDataPrediction(modelKNN,x_test_scaled)
         predictions_knn.to_csv('./KNeighborsRegressorModel.csv', index = False)
```



YOUR RECENT SUBMISSION

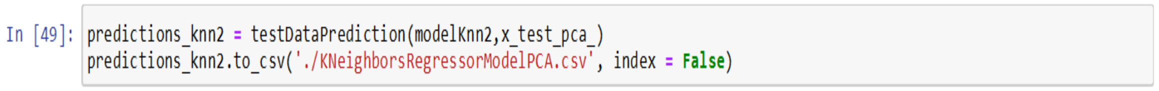
KNeighborsRegressorModel.csv

Submitted by Vidvath7 · Submitted 2 minutes ago

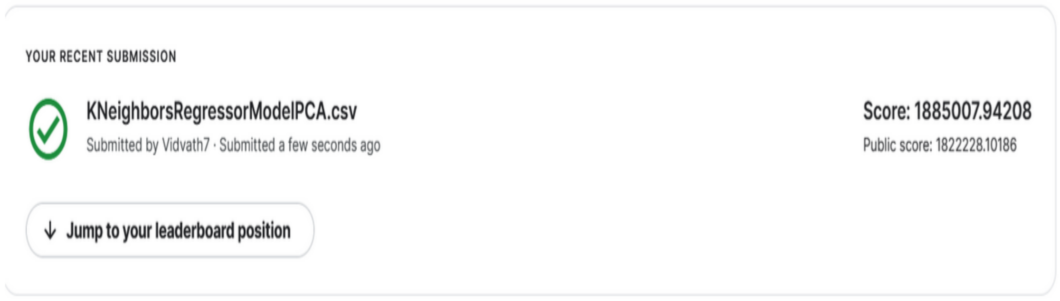
Score: 1771652.76748

Private score: 1869554.67751

b. PCA**,**



```
In [49]: predictions_knn2 = testDataPrediction(modelKnn2,x_test_pca_)
         predictions_knn2.to_csv('./KNeighborsRegressorModelPCA.csv', index = False)
```



YOUR RECENT SUBMISSION

KNeighborsRegressorModelPCA.csv

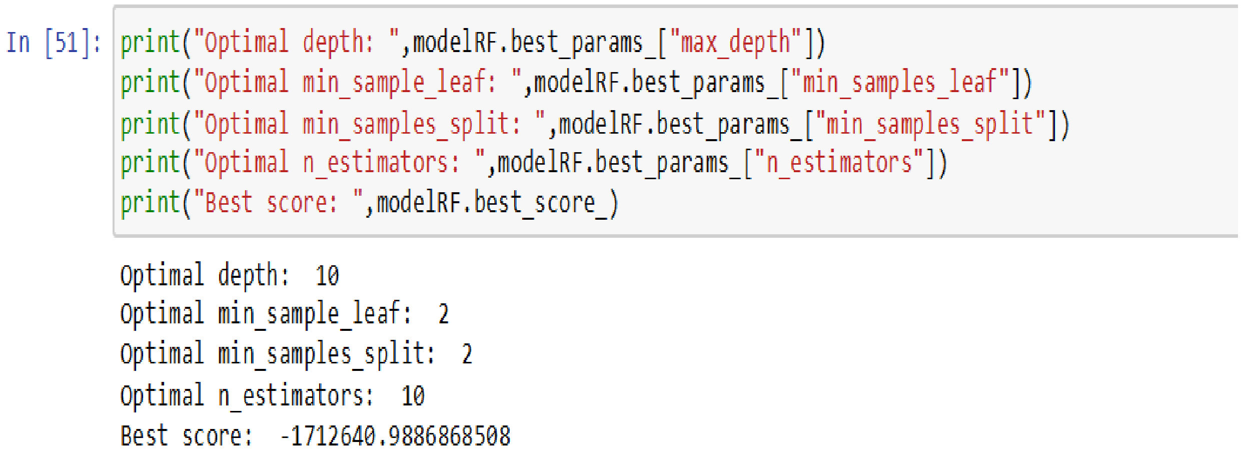
Submitted by Vidvath7 · Submitted a few seconds ago

Score: 1885007.94208

Public score: 1822228.10186

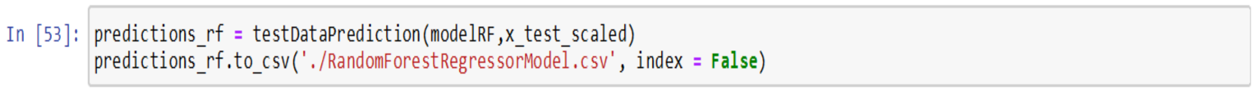
Inference: The Standard scaled data gives better results for KNN Regressor.

iii. Random Forest,

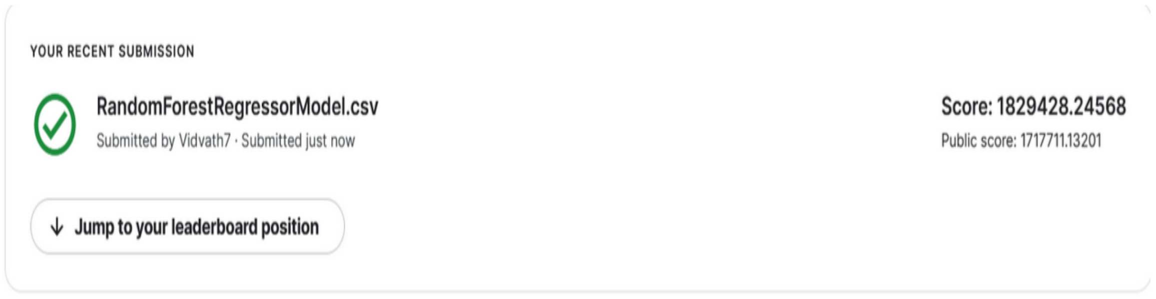


```
In [51]: print("Optimal depth: ",modelRF.best_params_["max_depth"])
         print("Optimal min_sample_leaf: ",modelRF.best_params_["min_samples_leaf"])
         print("Optimal min_samples_split: ",modelRF.best_params_["min_samples_split"])
         print("Optimal n_estimators: ",modelRF.best_params_["n_estimators"])
         print("Best score: ",modelRF.best_score_)

         Optimal depth:  10
         Optimal min_sample_leaf:  2
         Optimal min_samples_split:  2
         Optimal n_estimators:  10
         Best score:  -1712640.9886868508
```

The above snap shows the optimal parameters for the Random Forest for our data.



```
In [53]: predictions_rf = testDataPrediction(modelRF,x_test_scaled)
         predictions_rf.to_csv('./RandomForestRegressorModel.csv', index = False)
```



YOUR RECENT SUBMISSION

RandomForestRegressorModel.csv
Submitted by Vidvath7 · Submitted just now

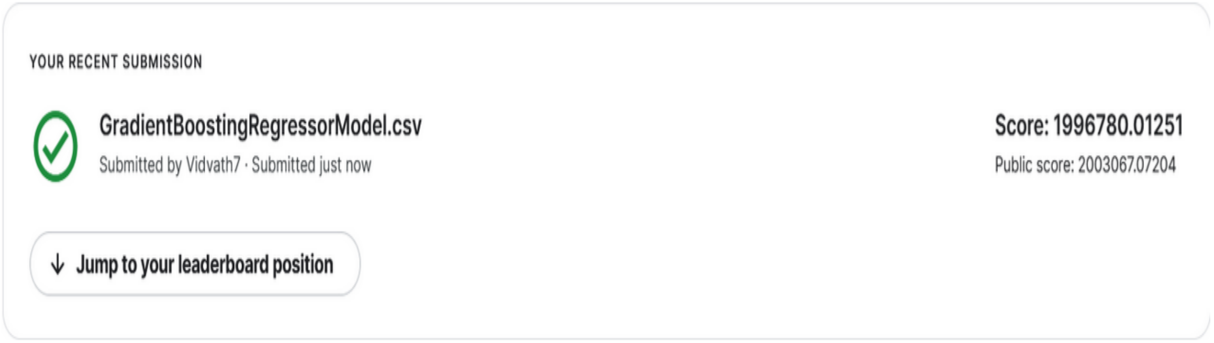
Score: 1829428.24568
Public score: 1717711.13201

iv. Gradient Boosting Regressor,

```
In [55]: print("Optimal depth: ",modelGBR.best_params_["max_depth"])
         print("Optimal Learning rate: ",modelGBR.best_params_["learning_rate"])
         print("Optimal n_estimators: ",modelGBR.best_params_["n_estimators"])
         print("Best score: ",modelGBR.best_score_)

         Optimal depth:  8
         Optimal Learning rate:  0.01
         Optimal n_estimators:  100
         Best score:  -1661974.6809895881
```

The above snap shows the optimal parameters for the Gradient Boosting Regressor for our data.

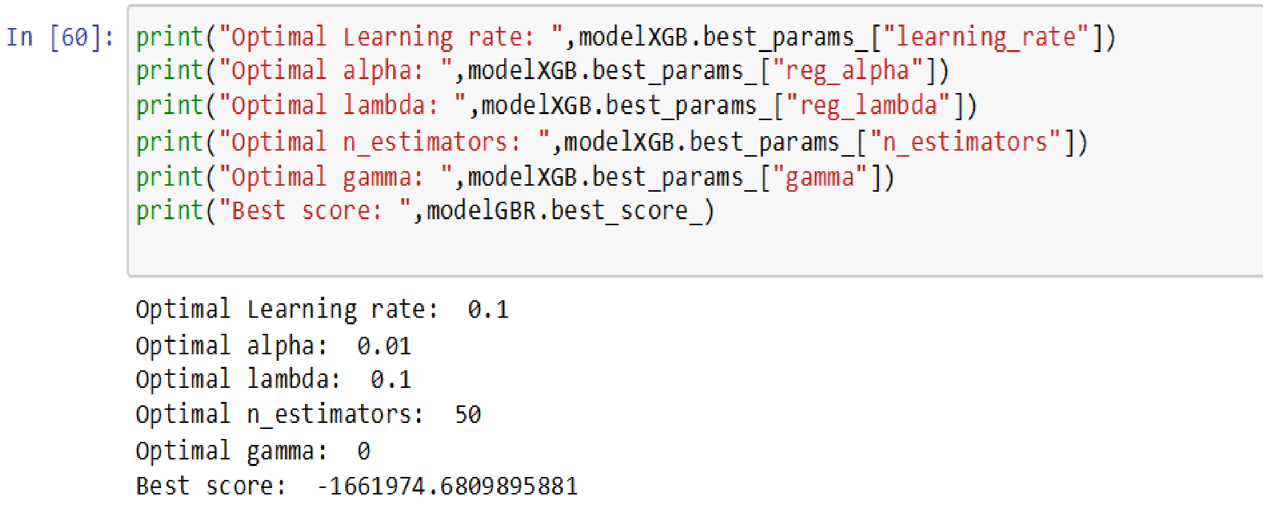
```
In [57]: predictions_gbr = testDataPrediction(modelGBR,x_test_scaled)
         predictions_gbr.to_csv('./GradientBoostingRegressorModel.csv', index = False)
```


YOUR RECENT SUBMISSION

GradientBoostingRegressorModel.csv
Submitted by Vidvath7 · Submitted just now

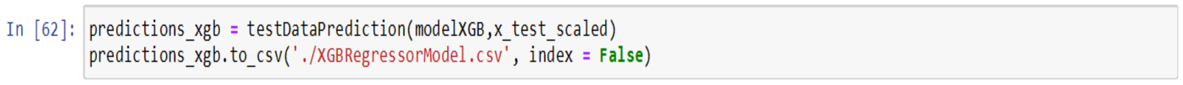
Score: 1996780.01251
Public score: 2003067.07204

v. XGBoost,

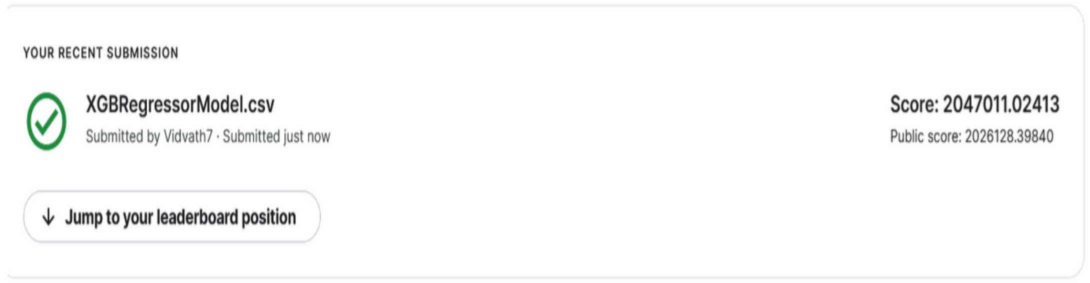


```
In [60]: print("Optimal Learning rate: ",modelXGB.best_params_["learning_rate"])
         print("Optimal alpha: ",modelXGB.best_params_["reg_alpha"])
         print("Optimal lambda: ",modelXGB.best_params_["reg_lambda"])
         print("Optimal n_estimators: ",modelXGB.best_params_["n_estimators"])
         print("Optimal gamma: ",modelXGB.best_params_["gamma"])
         print("Best score: ",modelGBR.best_score_)

         Optimal Learning rate:  0.1
         Optimal alpha:  0.01
         Optimal lambda:  0.1
         Optimal n_estimators:  50
         Optimal gamma:  0
         Best score:  -1661974.6809895881
```

The above snap shows the optimal parameters for the XGBoost Regressor for our data.



```
In [62]: predictions_xgb = testDataPrediction(modelXGB,x_test_scaled)
         predictions_xgb.to_csv('./XGBRegressorModel.csv', index = False)
```



YOUR RECENT SUBMISSION

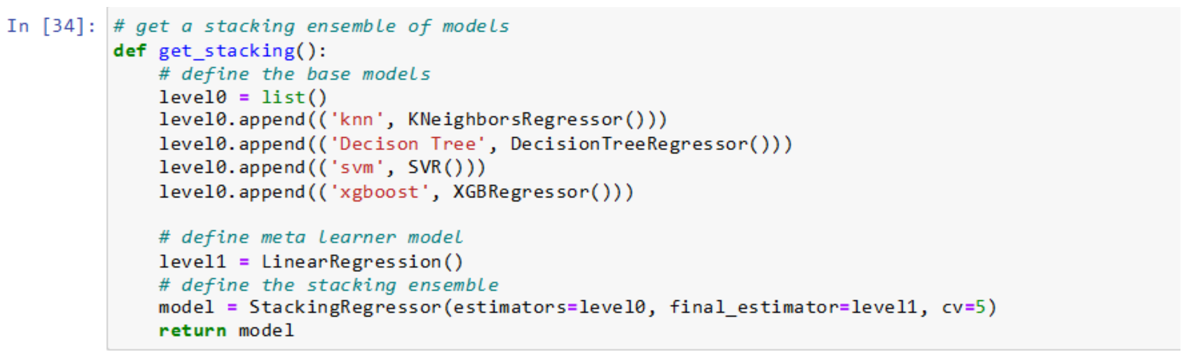
XGBRegressorModel.csv

Submitted by Vidvath7 · Submitted just now

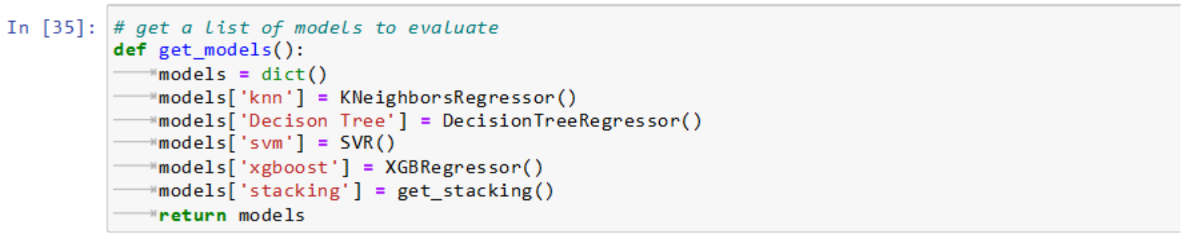
Score: 2047011.02413

Public score: 2026128.39840

vi. Stacking Model,

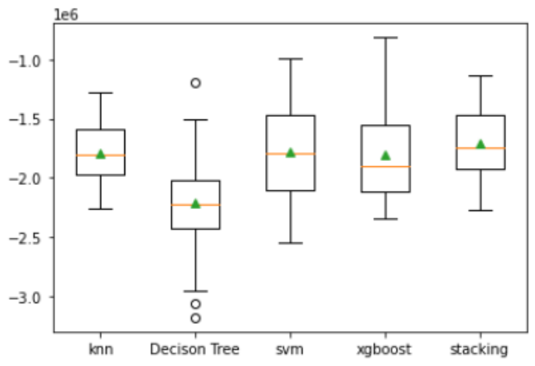


```
In [34]: # get a stacking ensemble of models
         def get_stacking():
             # define the base models
             level0 = list()
             level0.append(('knn', KNeighborsRegressor()))
             level0.append(('Decison Tree', DecisionTreeRegressor()))
             level0.append(('svm', SVR()))
             level0.append(('xgboost', XGBRegressor()))

             # define meta learner model
             level1 = LinearRegression()
             # define the stacking ensemble
             model = StackingRegressor(estimators=level0, final_estimator=level1, cv=5)
             return model
```



```
In [35]: # get a list of models to evaluate
         def get_models():
             models = dict()
             models['knn'] = KNeighborsRegressor()
             models['Decison Tree'] = DecisionTreeRegressor()
             models['svm'] = SVR()
             models['xgboost'] = XGBRegressor()
             models['stacking'] = get_stacking()
             return models
```

We have used the knn, Decision Tree, SVM Regressor, XGBoost algorithms in the stacking model.



It is evident that the Stacking model performs better than the individual models.

```
In [51]: predictions_stack = testDataPrediction(model,x_test_scaled)
         predictions_stack.to_csv('./StackingEnsembleModel.csv', index = False)
```

YOUR RECENT SUBMISSION

StackingEnsembleModel.csv
Submitted by Vidvath7 · Submitted just now

Score: 1836228.52632
Public score: 1774579.34132

Jump to your leaderboard position

We have obtained a RMSE score of 1836228.52632 for the test data using the stacking ensemble model.

8) Summarizing Model Performances

```
In [66]: from tabulate import tabulate
         dict = {'Models'     : ['Linear Regression - Standard scaler','Linear Regression - PCA','KNN Regressor- Standard scaler',
                                  'KNN Regressor - PCA','Random Forest Regressor-GridSearchCV','Gradient Boosting Regressor -GridSearc
                 'Train RMSE' : [rms_LR_,rms_LR_pca,rms_knn,rms_knn2,rms_rf,rms_gbr,rms_xgb],
                 'Test RMSE'  : ['2052046.92310','1914503.15693','1771652.76748','1885007.94208',
                                  '1829428.24568','1996780.01251','2047011.02413']
                 }
         df = pd.DataFrame(dict)
         print(tabulate(df, headers = 'keys', tablefmt = 'psql'))
```

```
+----+--------------------------------------+--------------+-------------+
|    | Models                               |   Train RMSE |   Test RMSE |
|----+--------------------------------------+--------------+-------------|
|  0 | Linear Regression - Standard scaler  |  1.42203e+06 | 2.05205e+06 |
|  1 | Linear Regression - PCA              |  1.61423e+06 | 1.9145e+06  |
|  2 | KNN Regressor- Standard scaler       |  1.5645e+06  | 1.77165e+06 |
|  3 | KNN Regressor - PCA                  |  1.54253e+06 | 1.88501e+06 |
|  4 | Random Forest Regressor-GridSearchCV | 911052       | 1.82943e+06 |
|  5 | Gradient Boosting Regressor -GridSearchCV | 760511  | 1.99678e+06 |
|  6 | XGBoost                              | 167556       | 2.04701e+06 |
+----+--------------------------------------+--------------+-------------+
```

The model with the least TEST RMSE, i.e **KNN Regressor- Standard Scaler** values proves to be the best model among the ones that we have implemented.

6) Results

The RMSE score was used to measure the performance of the model. The submissions were done on Kaggle, and the model was evaluated based upon the performance on the test dataset. Even though the Gradient Boosting Regressor, XGBoost performed well with the training data, it was not up to the mark for the unseen test data.

The KNN Regressor with the standard scaled data performed the best followed by the Random Forest model for the unseen test data. GridSearchCV was used to find the best/optimal parameters for each model and predictions were done by the optimized model.

The Random Forest model is chosen for the productionization and is deployed to Heroku.

7) Conclusion

We have successfully predicted the revenue of restaurants and have achieved a decent RMSE score on Kaggle submissions. Various Data preprocessing steps were done, and different machine learning models were created to predict the results. The best model was identified using the Key Performance Indicator(RMSE) and was deployed to Cloud and Productionized.

6) Deployment and Productionization

We will be using the streamlit framework to create the Restaurant Revenue Predictor application and deploying to Heroku.

Heroku- Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage and scale modern apps.

We have created the app.py file, which is the root of our streamlit application where we define the inputs, functions and other details.

We have also created other files which are required for deployment in Heroku . They are :

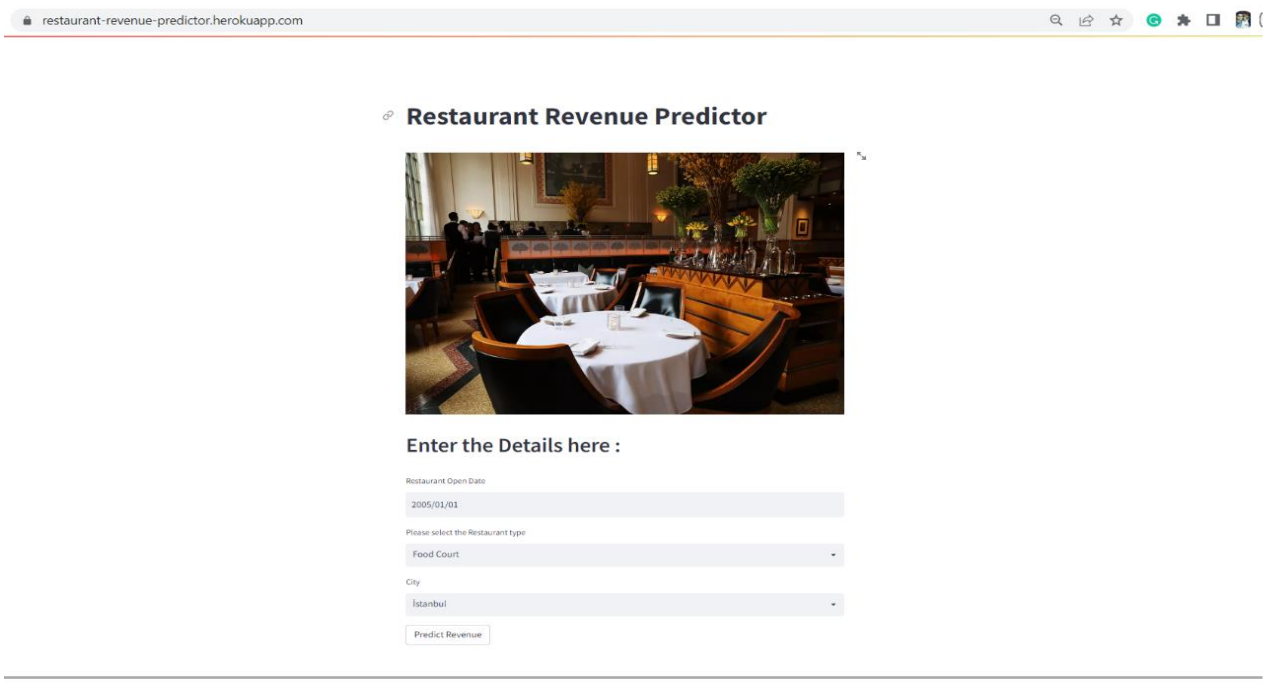
- setup.sh
- Procfile
- requirements.txt

➢ Deploying on Heroku,

Go to this link to access the Restaurant revenue prediction application on web.

https://restaurant-revenue-predictor.herokuapp.com/

Let's see the home page of our application.


restaurant-revenue-predictor.herokuapp.com

Restaurant Revenue Predictor

Enter the Details here :

Restaurant Open Date

2005/01/01

Please select the Restaurant type

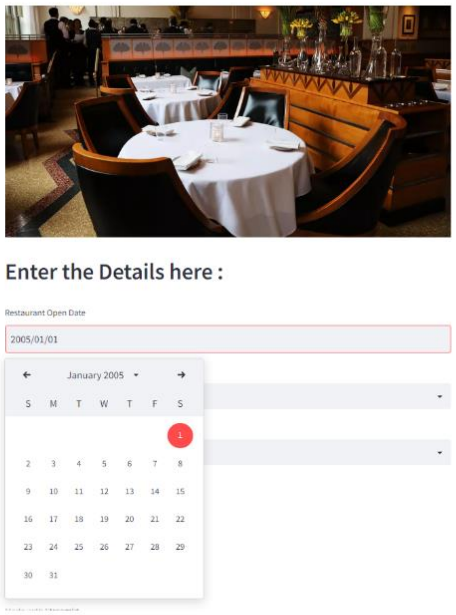
Food Court

City

İstanbul

Providing Date Input,

Providing date input

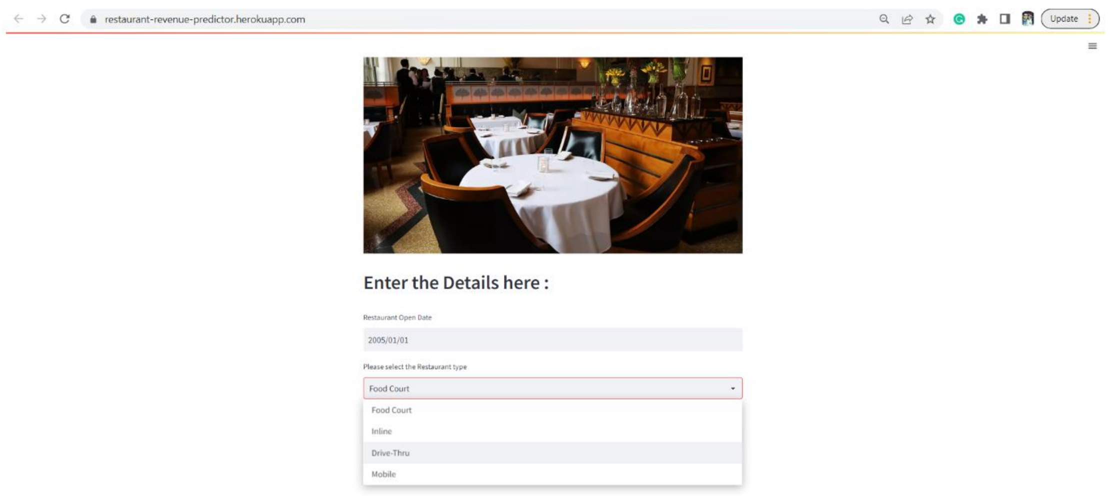

Enter the Details here :

Restaurant Open Date

2005/01/01

Providing Restaurant Type,

Providing Restaurant type details



Enter the Details here :

Restaurant Open Date

2005/01/01

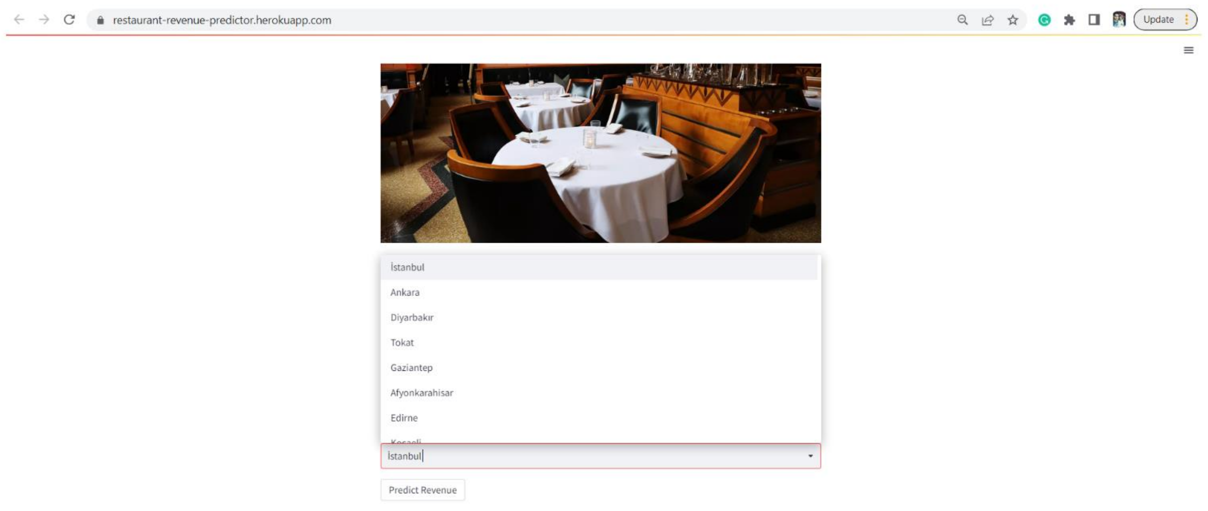
Please select the Restaurant type

Food Court

- Food Court
- Inline
- Drive-Thru
- Mobile

Providing City,

Selecting the City



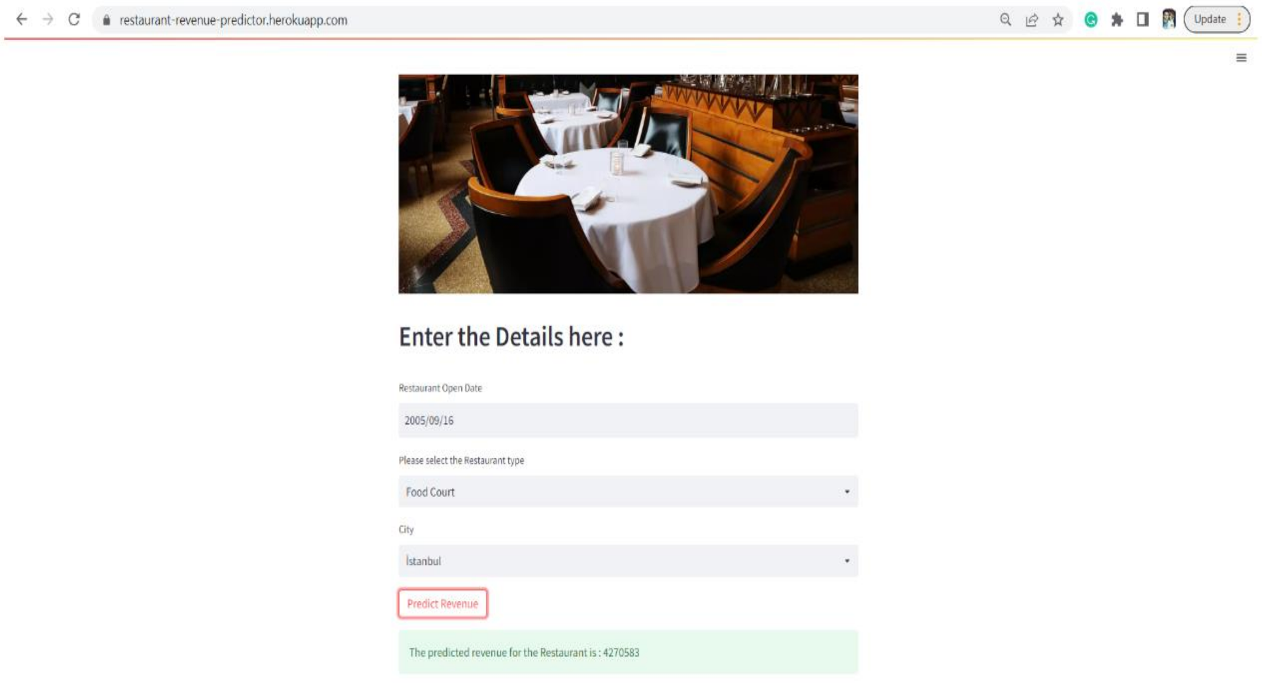
- İstanbul
- Ankara
- Diyarbakır
- Tokat
- Gaziantep
- Afyonkarahisar
- Edirne

İstanbul

Predict Revenue

Predicting the Revenue,

Predicting and displaying the Revenue


restaurant-revenue-predictor.herokuapp.com

Enter the Details here :

Restaurant Open Date

2005/09/16

Please select the Restaurant type

Food Court

City

İstanbul

The predicted revenue for the Restaurant is : 4270583

Here, we could see the output/Predicted Revenue when clicked on 'Predict Revenue' button after providing all the required inputs.