# End to End Deployment of Zomato Restaurant ratings

## Part 1: PERFORMING EDA

### Import the Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import plotly.offline as py
import seaborn as sns

import matplotlib.ticker as mtick
plt.style.use('fivethirtyeight')
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

### Importing the DataSet to Perform EDA

```
df=pd.read_csv('Zomato_df.csv')
df.head(5)
```

```
df.shape
```

```
df.dtypes
```

```
df.isnull().sum()

#In case null values are present, then use
#df.dropna(how='any', inplace=True)
#df.isnull().sum()
```

```
# Checking for Duplicate Values
df.duplicated().sum()

#In case duplicate values are present, then use
#df.drop_duplicates(inplace=True)
#df.duplicated().sum()
```

```
#Renaming of columns if required
#df.columns()
#df = df.rename(columns())
```

### Cleaning the Dataset

```
df['cost'].unique()
```

```
# If the above cost column contains any special characters, then same need to be removed by replacing with space
# Also, if rquired data type needs to be changed to Int or float type.

# zomato['cost']=zomato['cost'].astype(str)    changing cost to str datatype.
```

```
# df['cost']=df['cost'].apply(lambda x:x.replace(",",""))
# df['cost']=df['cost'].astype(float)
```

```
# Reading unique value from the rate Column

df['rate'].unique()
# If the column contains some alphabeical objects, same need to be removed by
# df = df.loc[df.rate!='new']   getting rid of "new"
```

```
array([4.1, 3.8, 3.7, 4.6, 4. , 4.2, 3.9, 3. , 3.6, 2.8, 4.4, 3.1, 4.3,
       2.6, 3.3, 3.5, 3.2, 4.5, 2.5, 2.9, 3.4, 2.7, 4.7, 2.4, 2.2, 2.3,
       4.8, 4.9, 2.1, 2. , 1.8])
```

```
# if the rating format is given as : 4.2/5, then we wish to remove /5.
#df['rate']=df['rate'].apply(lambda x:x.replace("/5",""))
```

## Visualization

#### 1. MOST FAMOUS RESTAURANT CHAIN IN BANGLORE

```
plt.figure (figsize = (17,10))
chains=df['name'].value_counts()[:20]
sns.barplot(x=chains, y=chains.index, palette='deep')
plt.title('MOST FAMOUS RESTAURANT CHAIN IN BANGLORE')
plt.xlabel('number of outlets')
plt.show()
```

#### 2. % of Customers who do table booking

```
x=df('book_table').value_counts()
trace = go.pie(label=x.index, value=x, textinfo='value')
layout = go.layout(title="Table Booking", width=400, height=600)
fig = go.figure(data=[trace], layout=layout)
py.iplot(fig, filename = 'pie_chart_subplots')
```

#### 3. WHETHER RESTAURANTS DELIVER ONLINE OR NOT

```
sns.countplot(df['online_order'])
fig = plot.gcf()
fig.set_size_inches(10,10)
plt.title('whether restaurants deliver online or not')
plt.show()
```

#### 4. RATING DISTRIBUTION

```
plt.figure(figsize(9,7))
sns.distplot(df['rate'],bins=20)
```

#### 5. DISTRIBUTION OF COST VS RATING IN PARALLEL WITH ONLINE ORDER

```
plt.figure(figsize(9,7))
sns.scatterplot(x='rate', y='cost', hue='online_order', data=df)
plt.show()
```

## Rating Distribution

#### COUNT OF RATINGS AS BETWEEN '1 AND 2', '2 AND 3', '3 AND 4', '4 AND 5'

```
df['rate'].unique()
```

```
df['rate'].min()
```

```
df['rate'].max()
```

```
((df['rate']>=1) & (df['rate']<2)).sum()
```

```
((df['rate']>=2) & (df['rate']<3)).sum()
```

```
((df['rate']>=3) & (df['rate']<4)).sum()
```

```
((df['rate']>=4) & (df['rate']<5)).sum()
```

**PLOTTING THE COUNT WITH THE HELP OF PIE CHART**

```
slices = [((df['rate']>=1) & (df['rate']<2)).sum(),
          ((df['rate']>=2) & (df['rate']<3)).sum(),
          ((df['rate']>=3) & (df['rate']<4)).sum(),
          ((df['rate']>=4) & (df['rate']<5)).sum()
          ]

labels = ['1<rate<2', '2<rate<3', '3<rate<4', '4<rate<5']
plt.pie(slices, labels=labels, colors=colors)
fig = plt.gcf()
plt.title('% of restaurants according to their rating')
fig.set_size_inches(10,10)
plt.show()
```

**SERVICE TYPES**

```
sns.countplot(df['type']).set_xticklabels(sns.countplot(df['type']).get_xticklabels, rotation=90, ha='right')
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.show()
```

**DISTRIBUTION OF COST OF FOOD FOR 2 PEOPLE**

```
from plotly.offline import iplot
trace0= go.Box(y=df['cost'], name = 'accepting online order')
data=[trace0]
layout=go.layout(title = 'Box plot of approximate cost', width=800, height=800, yaxis=dict(title='Price'))
fig=go.Figure(data=data, layout=layout)
py.iplot(fig)
```

**DISTRIBUTION OF CHARGES**

```
plt.figure(figsize(8,8))
sns.distplot(df['cost'])
plt.show()
```

## Part 2: Building ML Model

```
df.head()
```

### Convert the online categorical variables into a numeric format

```
df.online_order[df.online_order == 'Yes'] = 1
df.online_order[df.online_order == 'No'] = 0
```

```
df.online_order.value_counts()
```

```
df.online_order = pd.to_numeric(df.online_order)
```

## Change the string categorical into to a categorical int

```
df.book_table[df.book_table == 'Yes'] = 1
df.book_table[df.book_table == 'No'] = 0
```

```
df.book_table = pd.to_numeric(df.book_table)
```

```
df.book_table.value_counts()
```

Label encode the categorical variables to make it easier to build algorithm

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
df.location = le.fit_transform(df.location)
df.rest_type = le.fit_transform(df.rest_type)
df.cuisines = le.fit_transform(df.cuisines)
df.menu_item = le.fit_transform(df.menu_item)
```

```
df.head()
```

```
my_data=df.iloc[:,[2,3,4,5,6,7,9,10,12]]
my_data.to_csv('Zomato_df.csv')
```

```
x = df.iloc[:,[2,3,5,6,7,9,10,12]]
x.head()
```

```
y = df['rate']
y
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.3,random_state=10)
```

## LINEAR REGRESSION

```
lr_model=LinearRegression()
lr_model.fit(x_train,y_train)
```

```
from sklearn.metrics import r2_score
y_pred=lr_model.predict(x_test)
r2_score(y_test,y_pred)
```

## RANDOM FOREST

from sklearn.tree import DecisionTreeRegressor

```
from sklearn.ensemble import RandomForestRegressor
RF_Model=RandomForestRegressor(n_estimators=650,random_state=245,min_samples_leaf=.0001)
RF_Model.fit(x_train,y_train)
y_predict=RF_Model.predict(x_test)
r2_score(y_test,y_predict)
```

## EXTRA TREE REGRESSOR

```
#Preparing Extra Tree Regression
from sklearn.ensemble import  ExtraTreesRegressor
ET_Model=ExtraTreesRegressor(n_estimators = 120)
ET_Model.fit(x_train,y_train)
y_predict=ET_Model.predict(x_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_predict)
```

```
Use pickle to save our model so that we can use it later

import pickle
# Saving model to disk
pickle.dump(ET_Model, open('model.pkl','wb'))
model=pickle.load(open('model.pkl','rb'))
```