Project Evaluation Phase

Project Name

Zomato Restaurants

SME

Mohd. Kashif

Submitted By

Adeeb Naiyer
Batch - DS2402

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to “FLIP ROBO” team who has given me this opportunity work on this insightful dataset helping me to improve my analytical and Data Science skills. Special thanks to my SME, Mr. Mohd Kashif for helping me through the projects through his valuable assistance and for clearing my doubts throughout the projects.

I also extend my appreciation to ‘Data Trained’ for facilitating my internship at FLIP ROBO. The experience has been very important in enhancing my practical knowledge and expertise.

This project ‘Zomato Restaurant Data Analysis’ has been successful with the assistance from training document and live classes recording from Data Trained institute. Some assistance was taken from online websites like Geeks for Geeks, Medium, Stack Overflow etc.

INTRODUCTION

Today the huge quantity of information produced daily can be viewed as the main advantage and disadvantage, particularly in relation to machine learning, data science, and data analytics. Machine learning (ML), which is a branch of AI, involves the development of models that are capable of learning from data and making decisions or predictions without being explicitly programmed to do so, has brought significant changes in various sectors including the food and restaurant business by enhancing customer's satisfaction and organizational efficiency. Data science is basically the use of statistical, computing, and business skills to analyze the data and find out the dining patterns, customer's preferences and organizational efficiency. Data analytics is a systematic way of using data to find out information and patterns with the use of descriptive, diagnostic, predictive and prescriptive analysis. In this project, we analyze the restaurant data of Zomato to get the idea about the trend of cuisines, cost, geographical location and the rating. Here, I apply ML techniques and different algorithms to predict variables like 'Average Cost for two', 'Price range'. The steps that we are going to discuss are exploratory data analysis, data cleaning and transformation, feature creation, model identification and evaluation, tuning, visualisation. In this case, we will apply predictive machine learning algorithms that will enable Zomato to make better recommendations and enhance customers' satisfaction, and thereby, show how these technologies can turn data into insights for business improvement.

1. Problem Definition

This Zomato Restaurant Data Analysis project seeks to offer insights into the restaurant services industry across the world using data from the Zomato company. It is one of the most useful analyses for those people who like to taste food and find out which restaurant has the best taste among the restaurants by taking into account factors such as expenses and budget and exploring the dishes. By this analysis, one can find out which is the most value for money restaurant in specific localities if different countries. It also gives insights which restaurant is nearby and easily accessible by calculating the location.
The main aim of the project is to

- Predict ‘Average cost for two for restaurants
- Predict Price range of the food offered by restaurants

1.1. Objectives

- To analyse the data for missing, duplicate values
- To determine and visualise the unique values
- To carryout visualisations
- To explore and understand cost patterns and affordability across different regions considering the variety of the cuisines available.
- To identify key trends and relationship such as:
  - Popular cuisines in different countries
  - Relationship between location and price range
- To carryout encoding for the categorical columns, minimise outliers and skewness, collinearity in the data.
- To scale and balance the data (for classification tasks)
- To build predictive models that will forecast Average price for two people meal and Price range based on various features.
- To plot RoC curves considering the TPR and TFR

2. Data Analysis

2.1 Overview of the Dataset

The project uses two datasets: Zomato.csv and country_code.csv. Zomato.csv has 3551 rows and 22 columns. The Zomato.csv dataset gas information about restaurants and their geographical locations, cuisines, costs, rating and more. The country_code.csv maps numerical country codes to their respective country names, identifying the location of each restaurants.

2.2 Key Variables in the Dataset

- **Restaurant Id:** Unique id of every restaurant across various cities of the world
- **Restaurant Name:** Name of the restaurant
- **Country Code:** Country in which restaurant is located
- **City:** City in which restaurant is located
- **Address:** Address of the restaurant
- **Locality:** Location in the city
- **Locality Verbose:** Detailed description of the locality
- **Longitude:** Longitude coordinate of the restaurant's location
- **Latitude:** Latitude coordinate of the restaurant's location
- **Cuisines:** Cuisines offered by the restaurant
- **Average Cost for two:** Cost for two people in different currencies □□
- **Currency:** Currency of the country
- **Has Table booking:** yes/no
- **Has Online delivery:** yes/ no
- **Is delivering:** yes/ no
- **Switch to order menu:** yes/no
- **Price range:** range of price of food
- **Aggregate Rating:** Average rating out of 5
- **Rating color:** depending upon the average rating color
- **Rating text:** text on the basis of rating of rating
- **Votes:** Number of ratings casted by people

2.3 Initial data analysis

First, lets import basic libraries and then load and explore the dataset to understand the structure.

2.3.1 Importing necessary libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Fig: Importing the necessary libraries

2.3.2 Loading the Dataset

```
[2]: # Read the CSV file into a DataFrame
zomato = pd.read_csv('zomato.csv', encoding='ISO-8859-1')
zomato
```

Fig: Importing the zomato.csv dataset

```
[3]: country = pd.read_excel('Country-Code.xlsx')
country
```

Fig: Importing the Country-Code.xlsx dataset

Next, we will merge the country code data with the main dataset using pandas and make a final dataset called “df”.

df = pd.merge(zomato, country, on='Country Code', how='left')

Displaying few rows of the Zomato dataset “df” using df.sample()

Restaurant Name	Country Code	City	Locality	Longitude	Latitude	Cuisines	Average Cost for two	Currency	Has Table booking	Has Online delivery	Is delivering now	Price range	Aggregate rating	Rating color	Rating text	Vot
PomoDoro Bistro	1	Gurgaon	Sushant Shopping Arcade, Sushant Lok, Gurgaon	77.079092	28.461071	Italian, Fast Food	700	Indian Rupees(Rs.)	No	Yes	No	2	3.4	Orange	Average	[illegible]
Oh! Calcutta	1	Gurgaon	Cyber Hub, DLF Cyber City	77.088688	28.495208	Bengali, Seafood	1700	Indian Rupees(Rs.)	Yes	Yes	No	3	3.8	Yellow	Good	7[illegible]
Rookery	216	Macon	Macon	-83.627900	32.836100	Burger, Desserts, Bar Food	25	Dollar($)	No	No	No	2	4.5	Dark Green	Excellent	2[illegible]
KB's Kulfi & Icecream	1	Gurgaon	Sector 7	77.018415	28.472672	Ice Cream	200	Indian Rupees(Rs.)	No	Yes	No	1	2.8	Orange	Average	[illegible]
Giani's	1	New Delhi	Prashant Vihar	77.133145	28.710643	Ice Cream, Desserts	400	Indian Rupees(Rs.)	No	No	No	1	3.0	Orange	Average	[illegible]

Fig: Random sample rows of df

After finding out the shape of the dataset using df.shape(), we get to know that the dataset has 9551 rows and 22 columns.

By the problem statement the two dependent variables are the “Average cost for two” and the “Price Range”.

2.3.3 Checking for Missing/negative values and Data type of the columns

It's important to check for and handle missing values appropriately to ensure the integrity of our analysis.

```
# colunm types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9551 entries, 0 to 9550
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Restaurant ID         9551 non-null   int64  
 1   Restaurant Name       9551 non-null   object 
 2   Country Code          9551 non-null   int64  
 3   City                  9551 non-null   object 
 4   Address               9551 non-null   object 
 5   Locality              9551 non-null   object 
 6   Locality Verbose      9551 non-null   object 
 7   Longitude             9551 non-null   float64
 8   Latitude              9551 non-null   float64
 9   Cuisines              9542 non-null   object 
 10  Average Cost for two  9551 non-null   int64  
 11  Currency              9551 non-null   object 
 12  Has Table booking     9551 non-null   object 
 13  Has Online delivery   9551 non-null   object 
 14  Is delivering now     9551 non-null   object 
 15  Switch to order menu  9551 non-null   object 
 16  Price range           9551 non-null   int64  
 17  Aggregate rating      9551 non-null   float64
 18  Rating color          9551 non-null   object 
 19  Rating text           9551 non-null   object 
 20  Votes                 9551 non-null   int64  
 21  Country               9551 non-null   object 
dtypes: float64(3), int64(5), object(14)
memory usage: 1.7+ MB
```

Fig: datatypes/missing/negative values

After analyzing the missing values we got to know that there are no missing values in any columns except 1(Cuisines), which we will deal later as the project advances.

Lets confirm it with the help of the heatmap once

```
# HEATMAP to analyse the missing values
sns.set_style('whitegrid')
sns.set(font_scale=1.1)
sns.set_palette('vlag')
plt.figure(figsize=(16,3))

# Let's visualize it using heatmap
sns.heatmap(df.isnull(), yticklabels=False, cmap='viridis', cbar=False)
plt.title('Missing values in the dataset', fontsize=14);
```

Fig: Heatmap displaying the missing values

2.3.4 Feature Analysis

```
[13]: # Reviewing the number of unique values in each feature and the target
      info_df = df.nunique().to_frame('No. of unique values')
      info_df['type'] = df.dtypes.values
      info_df
```

[13]:

	No. of unique values	type
Restaurant ID	9551	int64
Restaurant Name	7446	object
Country Code	15	int64
City	141	object
Address	8918	object
Locality	1208	object
Locality Verbose	1265	object
Longitude	8120	float64
Latitude	8677	float64
Cuisines	1825	object
Average Cost for two	140	int64
Currency	12	object
Has Table booking	2	object
Has Online delivery	2	object
Is delivering now	2	object
Switch to order menu	1	object
Price range	4	int64
Aggregate rating	33	float64
Rating color	6	object
Rating text	6	object
Votes	1012	int64
Country	15	object

Fig: Checking for unique values in the features

In the above code we will find out the Number of unique values along with the datatype of the feature columns.

Observation:

- Restaurant ID has the same no of values as the no of the rows. So we will go ahead and drop the column as it is no much importance while model building
- Since the latitude and longitude provides us the exact location of the restaurant, we will be not needing the address column. So we will go ahead and drop it.
- The column Switch to order menu has only 1 unique value which will be not of much use while model building, so we will drop it
- Average cost for two: 140 unique values(Regression problem)
- Price Range: 4 unique values(Classification problem)
- Locality verbose column also can be dropped since the information it has is also present in locality and city columns
- Numerical column but considered categorical: Country code, aggregate rating.

After dropping the columns for the above said reasons, the new dataset shape is: (9551,18).

We also discover that Average cost for two is a continuous and numeric column with 140 unique values, hence it is a regression problem. While Price range has only 4 unique values, therefore falling under classification problem.

2.3.5 Analysing the unique values

Here, we will define a function to analyse all the unique values

```
[16]: def inspect_column(df, column):
          print(f"Feature {column}:\n{df[column].value_counts()}")
          print(f"Unique values: {df[column].unique()}")
          print(f"# unique values: {df[column].nunique()}")

      inspect_column(df, 'Average Cost for two')
```

Fig: Function to determine unique values

2.3.6 Separating features into numerical and categorical columns

```
[18]: #Separating the numerical discrete variables from the continuous.

      # Separating Numerical and Categorical columns
      cat_col =  df.select_dtypes(include='object').columns.tolist()
      num_cat_col = ['Country Code', 'Aggregate rating']
      num_col =  [col for col in df.select_dtypes(include=np.number).columns.tolist() if col not in num_cat_col]

      # Remove the target variables
      num_col.remove('Average Cost for two')
      num_col.remove('Price range')

      #  Numerical and Categorical columns
      print(f"Categorical Columns:\n {cat_col}\n")
      print(f"Numerical Columns but categorical:\n {num_cat_col}\n")
      print(f"Numerical Columns:\n {num_col}\n")
```

```
Categorical Columns:
 ['Restaurant Name', 'City', 'Locality', 'Cuisines', 'Currency', 'Has Table booking', 'Has Online delivery', 'Is delivering now', 'Rating color', 'Rati
ng text', 'Country']

Numerical Columns but categorical:
 ['Country Code', 'Aggregate rating']

Numerical Columns:
 ['Longitude', 'Latitude', 'Votes']
```

Fig: Dividing columns into categories

Here, the dataset is divided into 3 categorical: Numerical column, categorical column and there is one more category for numerical columns which are classified as categorical columns because of significantly less number of unique values present.

3. EDA – Exploratory Data Analysis

3.1 Statistical summary

```
[22]: # Summary statistics of numerical columns
stats = df.describe()
stats
```

[22]:

	Country Code	Longitude	Latitude	Average Cost for two	Price range	Aggregate rating	Votes
count	9551.000000	9551.000000	9551.000000	9551.000000	9551.000000	9551.000000	9551.000000
mean	18.365616	64.126574	25.854381	1199.210763	1.804837	2.666370	156.909748
std	56.750546	41.467058	11.007935	16121.183073	0.905609	1.516378	430.169145
min	1.000000	-157.948486	-41.330428	0.000000	1.000000	0.000000	0.000000
25%	1.000000	77.081343	28.478713	250.000000	1.000000	2.500000	5.000000
50%	1.000000	77.191964	28.570469	400.000000	2.000000	3.200000	31.000000
75%	1.000000	77.282006	28.642758	700.000000	2.000000	3.700000	131.000000
max	216.000000	174.832089	55.976980	800000.000000	4.000000	4.900000	10934.000000

Fig: Statistical analysis

The Statistical summary present the information about count, mean, median, standard deviation, 25%, 50% and 75% of the numeric data and also the minimum and maximum of data. By this data we can also find insights regarding the skewness, outliers etc which we will explore in the next sections.

3.1.2 Skewness

After analysing the difference between the mean and median, found out that there is skewness in some columns which we need to eliminate later.

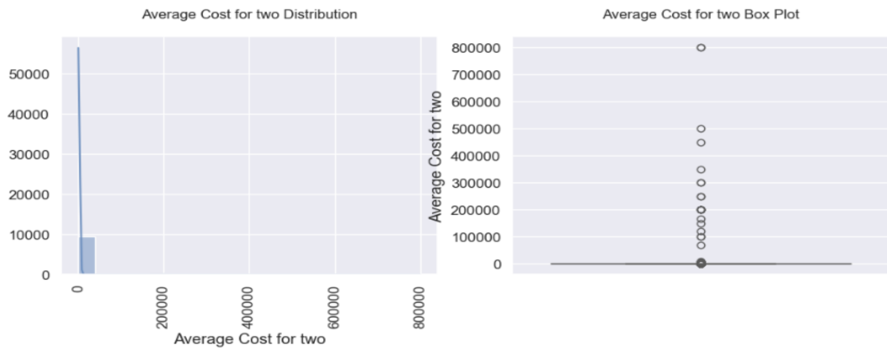
3.1.3 Outliers

The contrast between the maximum value in each column with 2 times the std plus the mean gives the instinct about the potential outliers present in the data.

3.2 Data Visualisation

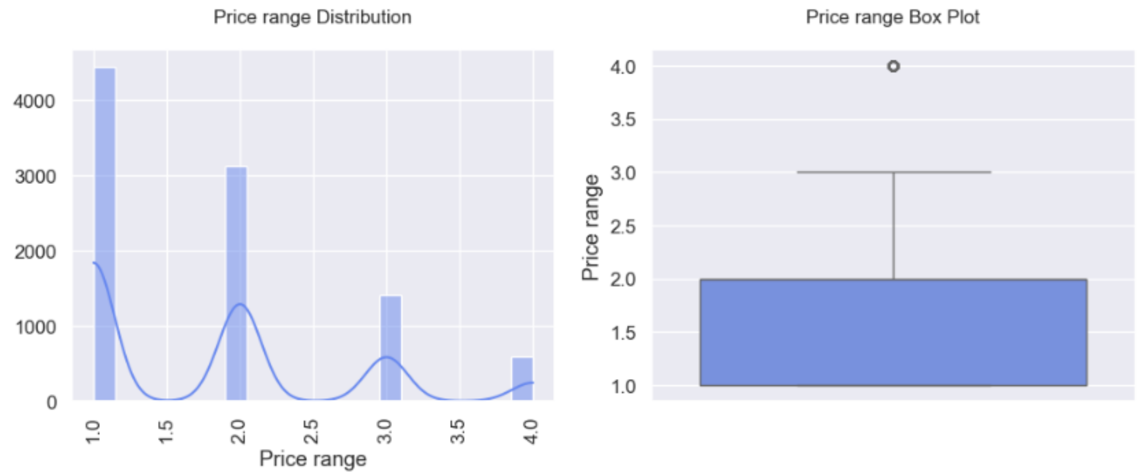
3.2.1 Univariate analysis

We will look at different data distribution across both features and label columns by plotting histogram plot to analyse skewness and box plot for outliers analysis.




```
Skewness of Average Cost for two : 35.477914900871276
```

*Fig: **Regression model target variable** data distribution (Average cost for two)*



Skewness of Price range : 0.8896175931442

*Fig: **Classification model target variable** data distribution*

Observation:

- By looking at both target variables data distribution we can say, the data is right skewed and there are more outliers present in the 'Average cost for two'.

The following were the visualisations for other **independent feature variables**:

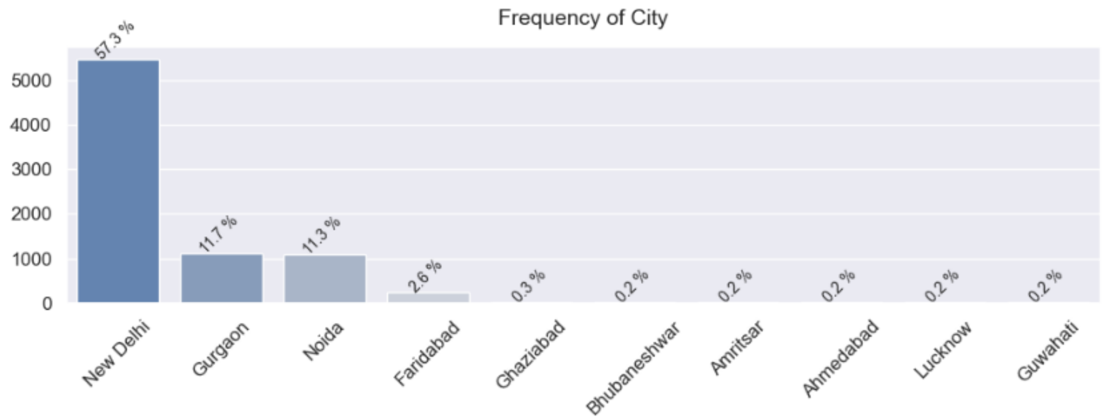


Fig: Top 10 Cities column count for unique values

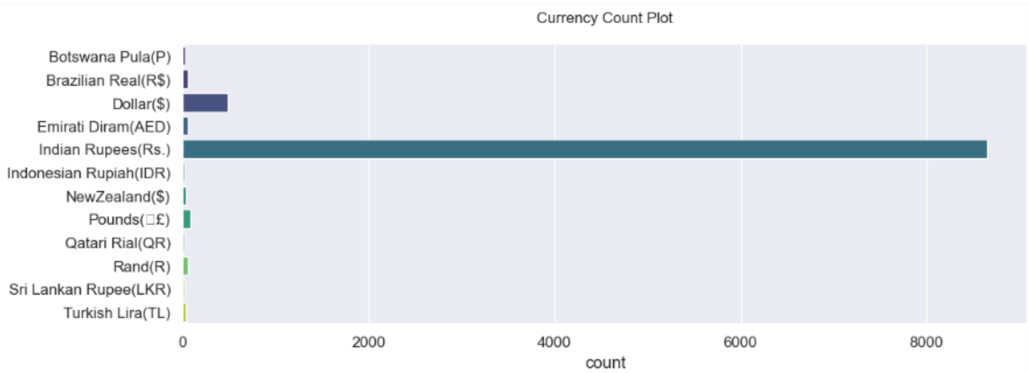


Fig: Currency column count plot

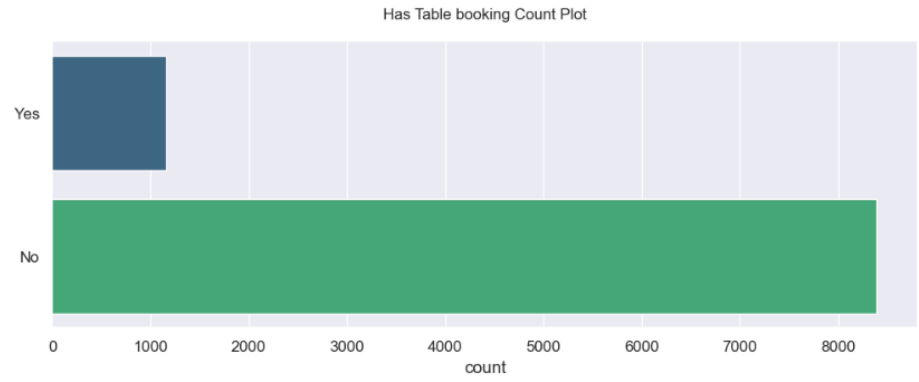


Fig: Has table booking count plot

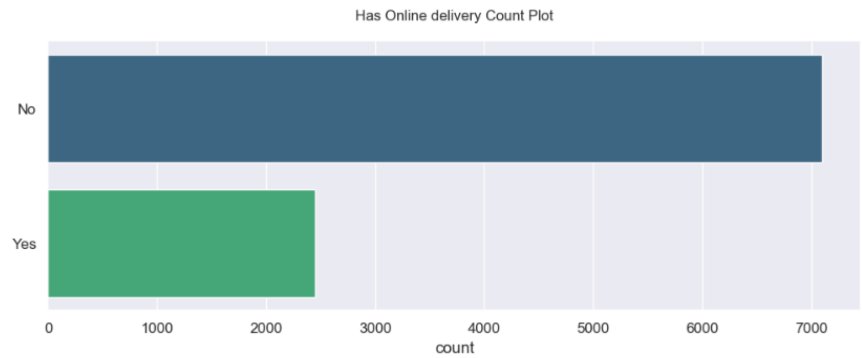


Fig: Has online delivery count plot

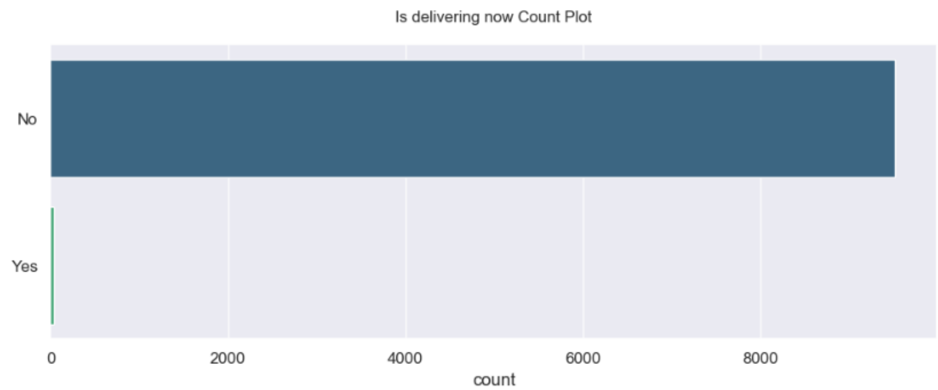


Fig: Is delivering now count plot

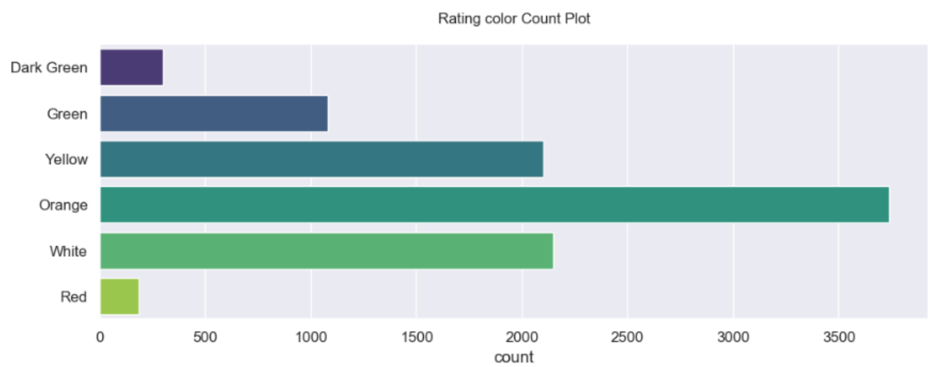


Fig: Rating count plot

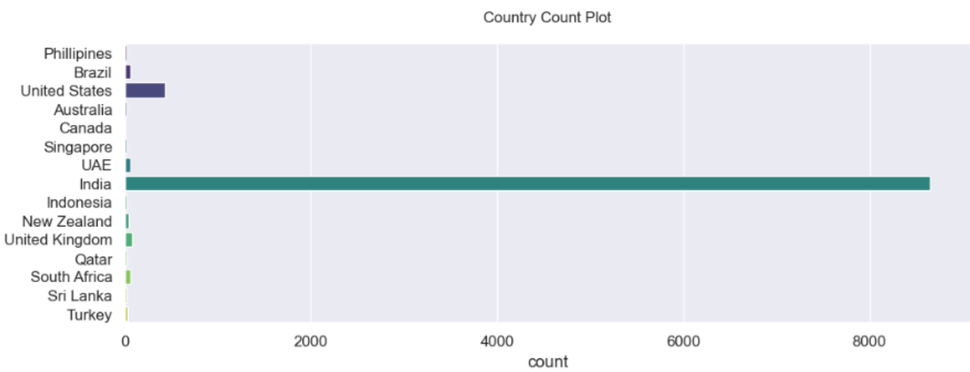


Fig: Country count plot

One of the primary insights from the exploratory data analysis (EDA) is the geographical distribution of restaurants. The majority of the restaurants in the dataset are located in India, followed by the United States and the United Kingdom. This distribution reflects the widespread usage of the Zomato platform in these regions.

3.2.2 Bivariate Analysis

Cuisine Diversity

Cuisine diversity is a critical aspect of the dataset. The analysis reveals that Indian, Chinese, and Continental cuisines are the most common across the dataset. This insight can be useful for understanding global and regional culinary preferences.

Cost Analysis

The cost analysis shows that the average cost for two people varies significantly across different countries and cities. Indian cities generally offer more budget-friendly dining options compared to cities in the United States and the United Kingdom. This analysis can help identify cities where dining out is more economical.

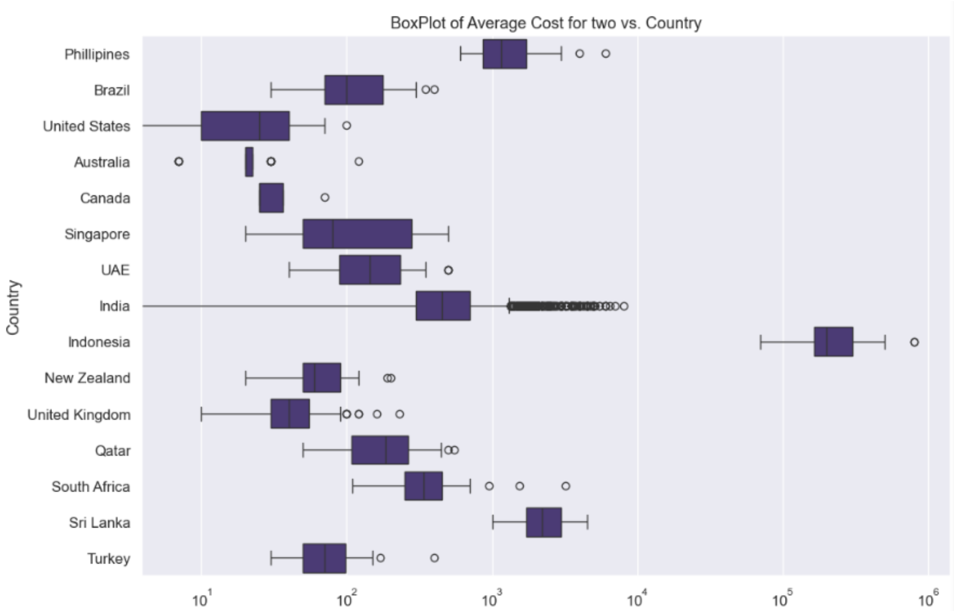


Fig: Average cost for two vs country box plot

Rating Analysis

The rating analysis reveals interesting patterns about the relationship between cost and ratings. Generally, higher ratings are associated with higher costs, indicating that more expensive restaurants tend to receive better reviews. However, there are also high-rated restaurants with moderate costs, indicating good value for money.

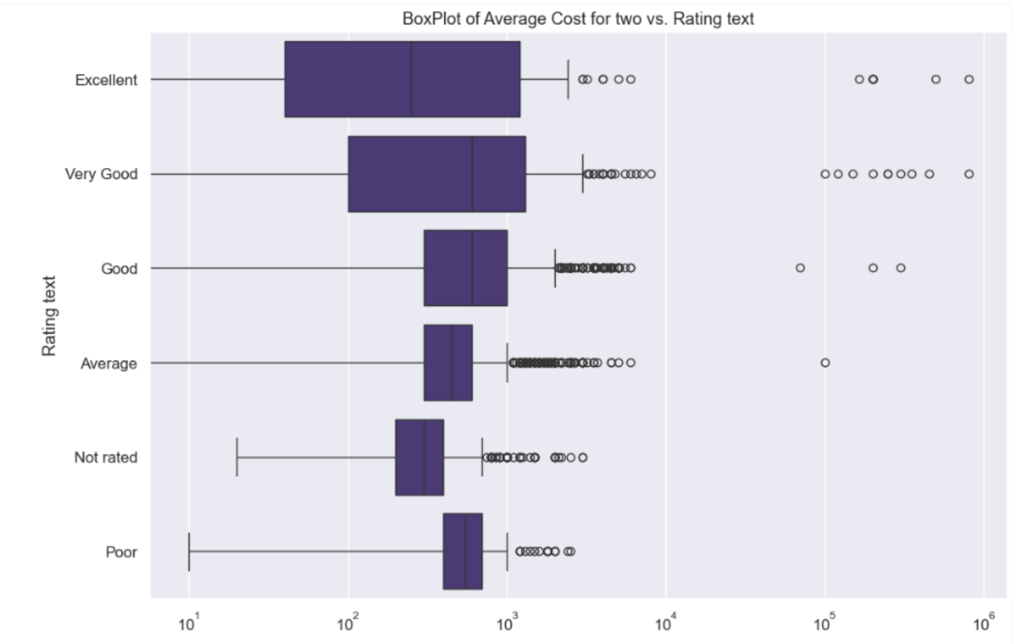


Fig: Average cost for two vs rating text boxplot

The rating analysis can also be done by considering the variables like Rating text and based on the Votes received for the restaurant. By analysing this we can find out the highest rated restaurants are also highly rated.

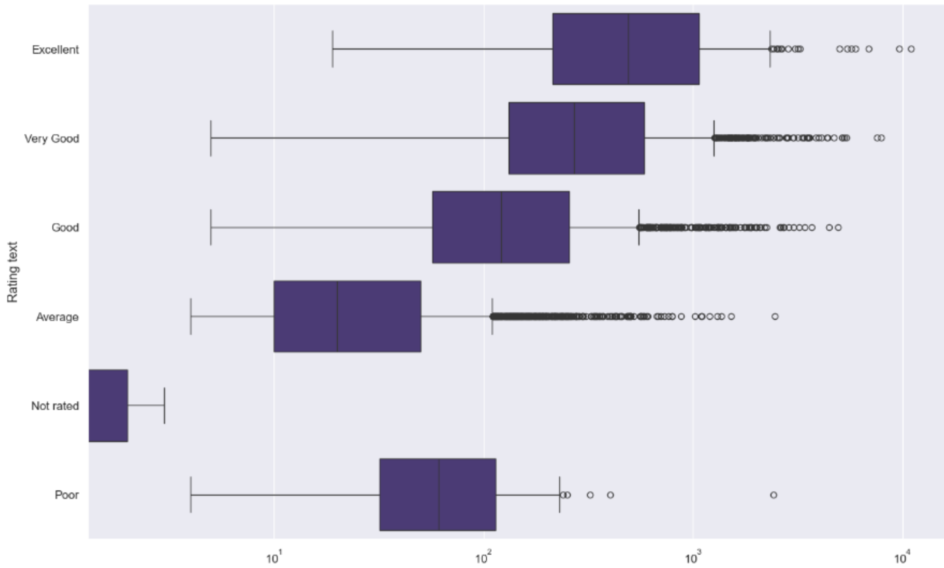


Fig: Rating text vs votes box plot

From the below plot we can determine that most of the Excellent rated restaurants are affordable for two people. Very less restaurants are expensive.

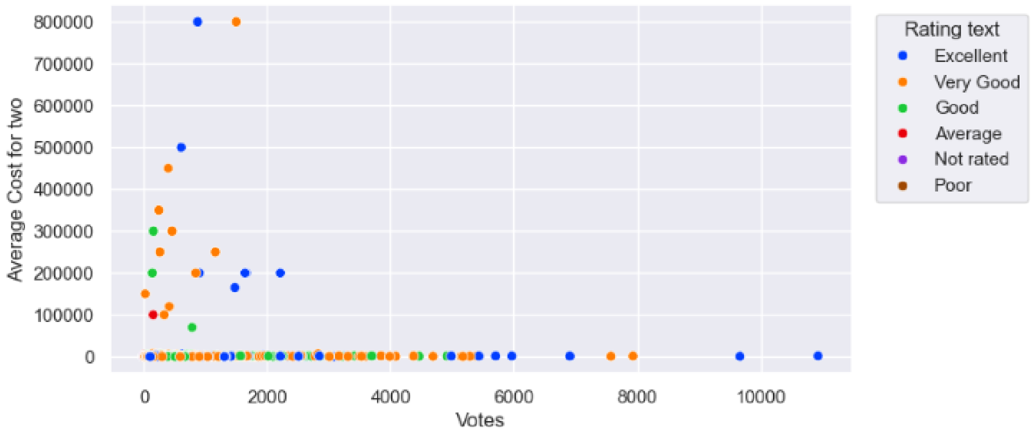


Fig: Scatterplot of Average cost for two vs votes

Location analysis

By plotting a graph between Country and votes, we can observe that there are highest voted and popular restaurants in India followed by USA, UAE

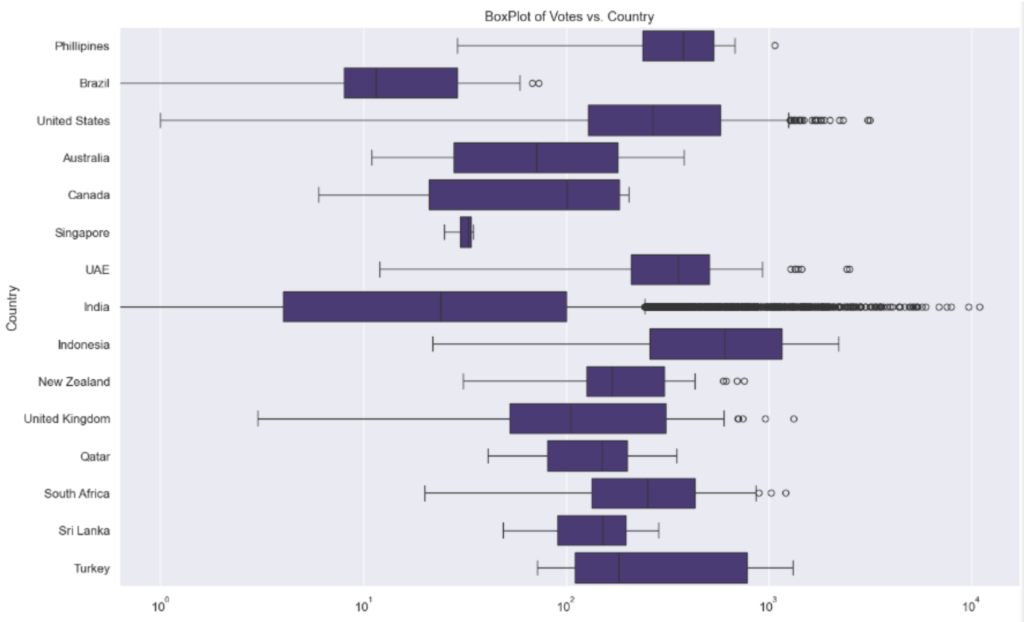


Fig: Country vs votes box plot

From the below plot, we can conclude that there are much expensive restaurants for two people in Indonesia while all other countries are in the affordable price range

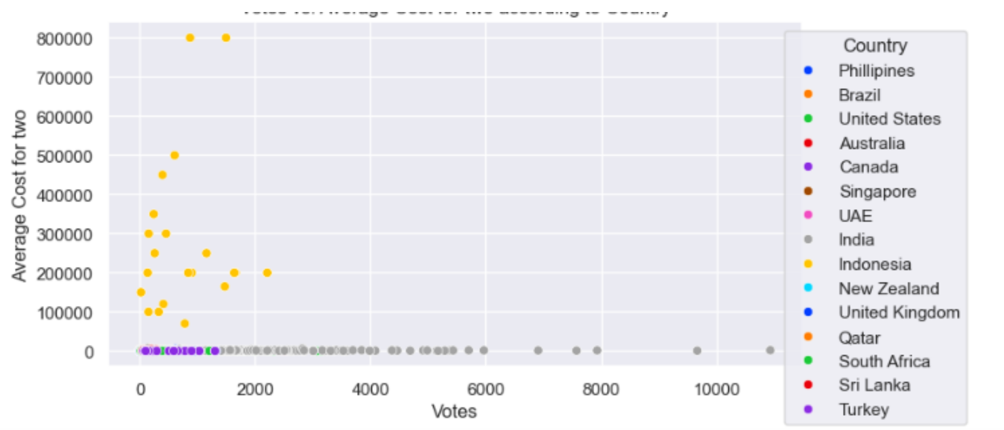


Fig: Scatterplot of average cost for two vs Votes according to country

Price range analysis

We will get insights about the different price ranges in the different parts of the world

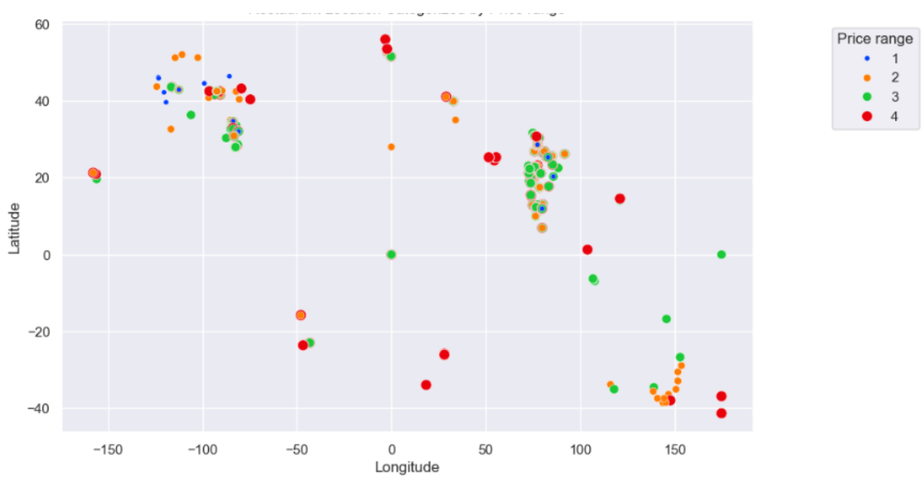


Fig: Restaurant location vs price range scatterplot

3.2.3 Multivariate Analysis

Pair plot

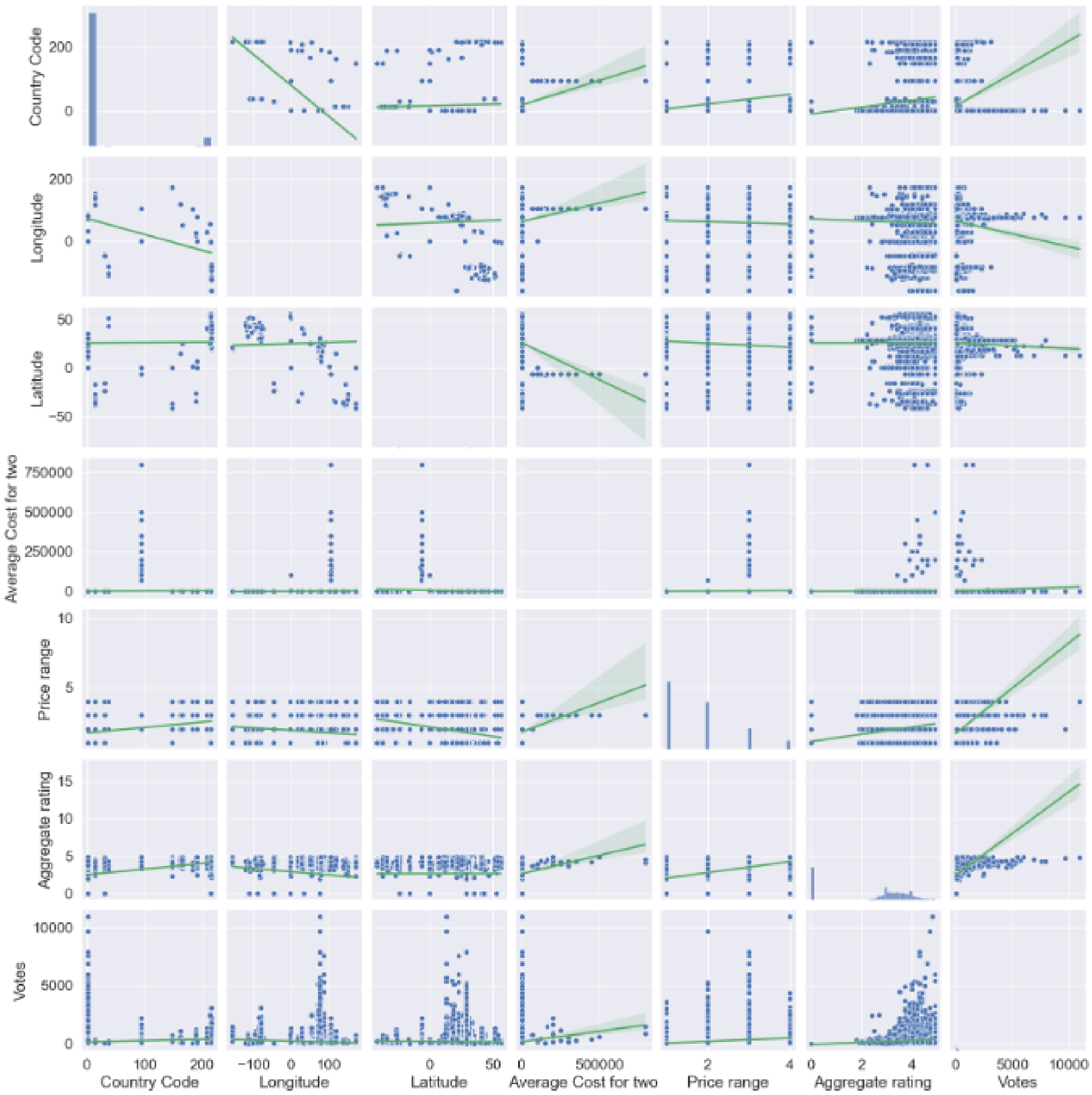


Fig: Pair plot of all numeric columns

3.3 Correlation check

[42]:
```
# Calculate the correlation between features
cor = df.corr()
cor
```

[42]:

	Country Code	Longitude	Latitude	Average Cost for two	Price range	Aggregate rating	Votes
Country Code	1.000000	-0.698299	0.019792	0.043225	0.243327	0.282189	0.154530
Longitude	-0.698299	1.000000	0.043207	0.045891	-0.078939	-0.116818	-0.085101
Latitude	0.019792	0.043207	1.000000	-0.111088	-0.166688	0.000516	-0.022962
Average Cost for two	0.043225	0.045891	-0.111088	1.000000	0.075083	0.051792	0.067783
Price range	0.243327	-0.078939	-0.166688	0.075083	1.000000	0.437944	0.309444
Aggregate rating	0.282189	-0.116818	0.000516	0.051792	0.437944	1.000000	0.313691
Votes	0.154530	-0.085101	-0.022962	0.067783	0.309444	0.313691	1.000000

Fig: Correlation matrix

Visualisation using seaborn

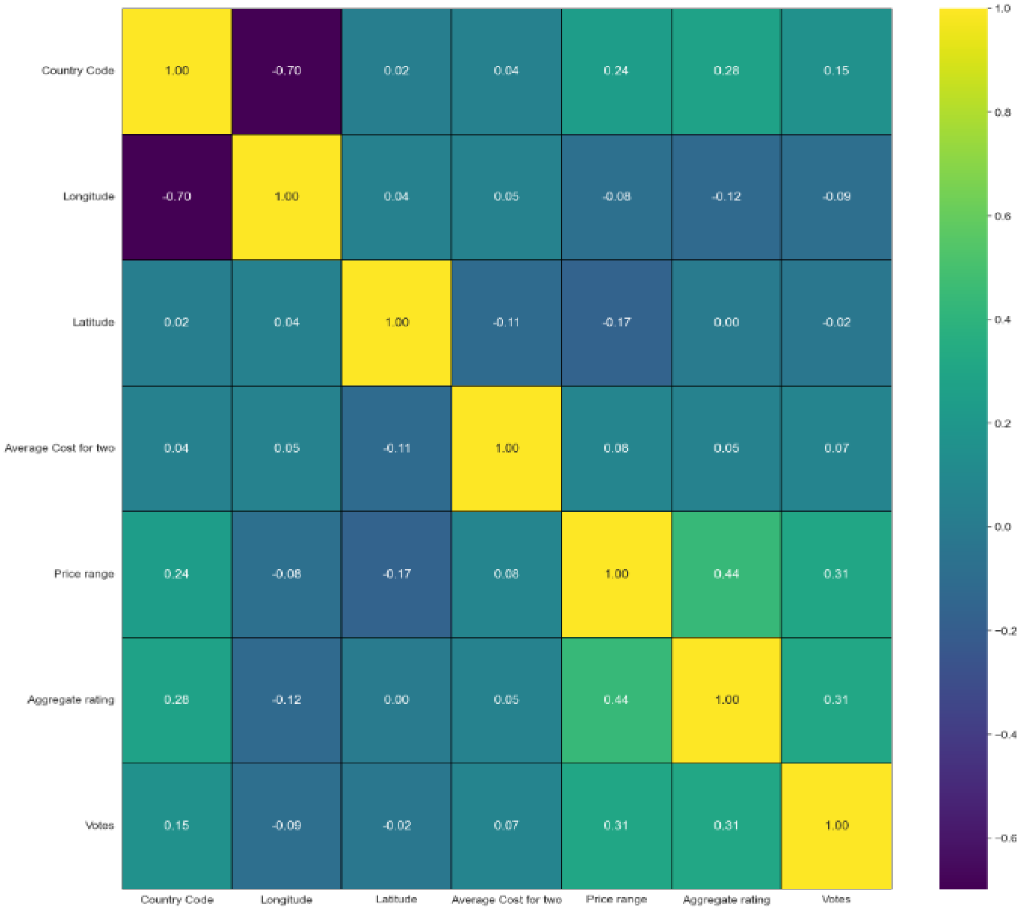


Fig: Correlation matrix visualisation using seaborn

Observation:

- Average cost for two has a weak correlation between other variables.
- Country code shows moderate negative correlation with longitude.

4. Pre-processing Pipeline

Feature Engineering

4.1 Handling Missing Values

Handling missing values is very much important to maintain data integrity. The missing values in the Cuisines will be dropped since there are very less. If there were significantly more missing values then we would have used imputation with the mean or mode values.

4.2 Encoding Categorical Columns

Creating new features from the existing data can enhance the predictive power of the model. The columns in which the values are not easy for further model building will be transformed into versatile values. All the categorical column values will be transformed into numerical through encoding.

4.2.1 Frequency Encoding for columns with high cardinality

Refers to technique of handling categorical values in Machine learning where we replace each of the category with the count of how often it appears in the dataset. Therefore, substituting the category with its frequency.

The columns with high cardinality such as Restaurant name, Locality are encoded with Frequency based encoding and after which the results are:

Restaurant name cardinality reduced to: 30

Locality cardinality reduced to:82

Cuisines cardinality reduced to: 64

```
[49]:
# Calculate the frequency of each category
frequency_map = df['Restaurant Name'].value_counts(normalize=True).to_dict()

# Create a new column with the frequency-based encoding
df['RestaurantName_enc'] = df['Restaurant Name'].map(frequency_map)

# Display the result
print(df[['RestaurantName_enc', 'Restaurant Name']].head())
print(f"\nCounts for Restaurant Name Encoded feature:\n{df['RestaurantName_enc'].value_counts()}\n")
print(f"Unique values in Restaurant Name: {df['Restaurant Name'].nunique()}")
print(f"Unique values in Restaurant Name Encoded: {df['RestaurantName_enc'].nunique()}")
```

```
   RestaurantName_enc         Restaurant Name
0            0.000105        Le Petit Souffle
1            0.000105        Izakaya Kikufuji
2            0.000105  Heat - Edsa Shangri-La
3            0.000105                    Ooma
4            0.000105             Sambo Kojin

Counts for Restaurant Name Encoded feature:
0.000105    6703
0.000210     936
0.000314     324
0.000419     184
0.000524     140
0.000629     108
0.000734      91
0.008698      83
0.008279      79
0.001991      76
0.001886      72
0.002306      66
0.006602      63
0.000838      56
0.001467      56
0.000943      54
0.005345      51
0.005030      48
0.002096      40
0.001362      39
0.001258      36
0.003563      34
0.001153      33
0.001677      32
0.003144      30
0.003039      29
0.002934      28
0.002725      26
0.001572      15
0.001048      10
Name: RestaurantName_enc, dtype: int64

Unique values in Restaurant Name: 7437
Unique values in Restaurant Name Encoded: 30
```

Restaurant name cardinality reduced to 30

Fig: Restaurant name frequency based encoding

4.2.2 Binary encoding for columns with two categories

The columns ‘Has Table booking', 'Has Online delivery', 'Is delivering now' will undergo binary encoding since the categorical values in these columns are yes/no. Hence we will map 1:yes and 0:no.

```
Unique values for Has Table booking: ['Yes' 'No']
Encoded values for Has Table booking: [1 0]

Unique values for Has Online delivery: ['No' 'Yes']
Encoded values for Has Online delivery: [0 1]

Unique values for Is delivering now: ['No' 'Yes']
Encoded values for Is delivering now: [0 1]
```

Fig: Binary encoding

4.2.3 Label encoding for columns with more than 2 unique values

The columns City, Currency, Rating text will undergo Label encoding

```
City:
{'Abu Dhabi': 0, 'Agra': 1, 'Ahmedabad': 2, 'Albany': 3, 'Allahabad': 4, 'Amritsar': 5, 'Ankara': 6, 'Armidale': 7, 'Athens': 8, 'Auckland': 9, 'August
a': 10, 'Aurangabad': 11, 'Balingup': 12, 'Bandung': 13, 'Bangalore': 14, 'Beechworth': 15, 'Bhopal': 16, 'Bhubaneshwar': 17, 'Birmingham': 18, 'Bogo
r': 19, 'Boise': 20, 'Brasí_lia': 21, 'Cape Town': 22, 'Cedar Rapids/Iowa City': 23, 'Chandigarh': 24, 'Chatham-Kent': 25, 'Chennai': 26, 'Clatskanie':
27, 'Cochrane': 28, 'Coimbatore': 29, 'Colombo': 30, 'Columbus': 31, 'Consort': 32, 'Dalton': 33, 'Davenport': 34, 'Dehradun': 35, 'Des Moines': 36, 'D
icky Beach': 37, 'Doha': 38, 'Dubai': 39, 'Dubuque': 40, 'East Ballina': 41, 'Edinburgh': 42, 'Faridabad': 43, 'Fernley': 44, 'Flaxton': 45, 'Forrest':
46, 'Gainesville': 47, 'Ghaziabad': 48, 'Goa': 49, 'Gurgaon': 50, 'Guwahati': 51, 'Hepburn Springs': 52, 'Huskisson': 53, 'Hyderabad': 54, 'Indore': 5
5, 'Inner City': 56, 'Inverloch': 57, 'Jaipur': 58, 'Jakarta': 59, 'Johannesburg': 60, 'Kanpur': 61, 'Kochi': 62, 'Kolkata': 63, 'Lakes Entrance': 64,
'Lakeview': 65, 'Lincoln': 66, 'London': 67, 'Lorn': 68, 'Lucknow': 69, 'Ludhiana': 70, 'Macedon': 71, 'Macon': 72, 'Makati City': 73, 'Manchester': 7
4, 'Mandaluyong City': 75, 'Mangalore': 76, 'Mayfield': 77, 'Mc Millan': 78, 'Middleton Beach': 79, 'Mohali': 80, 'Monroe': 81, 'Montville': 82, 'Mumba
i': 83, 'Mysore': 84, 'Nagpur': 85, 'Nashik': 86, 'New Delhi': 87, 'Noida': 88, 'Ojo Caliente': 89, 'Orlando': 90, 'Palm Cove': 91, 'Panchkula': 92, 'P
asay City': 93, 'Pasig City': 94, 'Patna': 95, 'Paynesville': 96, 'Penola': 97, 'Pensacola': 98, 'Phillip Island': 99, 'Pocatello': 100, 'Potrero': 10
1, 'Pretoria': 102, 'Princeton': 103, 'Puducherry': 104, 'Pune': 105, 'Quezon City': 106, 'Ranchi': 107, 'Randburg': 108, 'Rest of Hawaii': 109, 'Rio d
e Janeiro': 110, 'San Juan City': 111, 'Sandton': 112, 'Santa Rosa': 113, 'Savannah': 114, 'Secunderabad': 115, 'Sharjah': 116, 'Singapore': 117, 'Siou
x City': 118, 'Surat': 119, 'Sí£o Paulo': 120, 'Tagaytay City': 121, 'Taguig City': 122, 'Tampa Bay': 123, 'Tangerang': 124, 'Tanunda': 125, 'Trentham
East': 126, 'Vadodara': 127, 'Valdosta': 128, 'Varanasi': 129, 'Vernonia': 130, 'Victor Harbor': 131, 'Vineland Station': 132, 'Vizag': 133, 'Waterlo
o': 134, 'Weirton': 135, 'Wellington City': 136, 'Winchester Bay': 137, 'Yorkton': 138, 'ÛÁstanbul': 139}

Currency:
{'Botswana Pula(P)': 0, 'Brazilian Real(R$)': 1, 'Dollar($)': 2, 'Emirati Diram(AED)': 3, 'Indian Rupees(Rs.)': 4, 'Indonesian Rupiah(IDR)': 5, 'NewZea
land($)': 6, 'Pounds(\x8c£)': 7, 'Qatari Rial(QR)': 8, 'Rand(R)': 9, 'Sri Lankan Rupee(LKR)': 10, 'Turkish Lira(TL)': 11}

Rating text:
{'Average': 0, 'Excellent': 1, 'Good': 2, 'Not rated': 3, 'Poor': 4, 'Very Good': 5}
```

Fig: Label encoding the City, currency and rating text column

4.3 Eliminating outliers

Outliers are the data that differ significantly from other data. Outliers must be treated by eliminating the values. The elimination must be in such a way that there is no much loss of the data in the original dataset. In this project, we will calculate the Z Score of the records in the dataset and whichever value is greater then **3.8** will deleted. Therefore 780 rows with outliers were deleted. This represent **8.17%** of the data. In the new dataset there are **8762 rows and 16 columns.**

```
[65]: # threshold = 3.8
      df_new = df[(z<3.8).all(axis=1)]

      print(f"{df.shape[0] - df_new.shape[0]} rows with outliers were deleted.")
      print(f"This represent {round((df.shape[0] - df_new.shape[0]) / df.shape[0] *100, 2)}% of the data.")
      print(f"In the new dataset there are {df_new.shape[0]} rows and {df.shape[1]} columns.")

      df = df_new.copy()
      df

      780 rows with outliers were deleted.
      This represent 8.17% of the data.
      In the new dataset there are 8762 rows and 16 columns.
```

Fig: Handling outliers

4.4 Skewness Correction

Skewness is the degree of asymmetry observed in a probability distribution. We have to carryout skewness correction as it can affect the performance and accuracy of many data science models, especially those that assume normality or use mean-based metrics

[66]:

	Skew
RestaurantName_enc	4.374703
Country Code	4.077681
Votes	3.710843
Average Cost for two	3.557952
Has Table booking	2.259133
Cuisines_enc	1.575493
Has Online delivery	1.062437
Price range	0.966022
Rating text	0.418737
Locality_enc	0.346096
Is delivering now	0.000000
Currency	-0.502153
Aggregate rating	-0.898995
City	-1.435432
Latitude	-2.314064
Longitude	-3.333051

Fig: Skewness check

Normally the skewness over 0.5 must be treated accordingly. Here we will consider that metric and apply all the skewness transformation methods like log, sqrt, cbrt to check which method eliminates the skewness significantly.

We can see 'Aggregate rating', 'Average Cost for two', 'City', 'Country Code, 'Cuisines_enc', 'Currency', 'Has Online delivery', 'Has Table booking', 'Latitude', 'Longitude', 'Price range', 'RestaurantName_enc', 'Votes' has skewness greater than 0.5

[69]:

	index	Skewness	feature	Skewness_abs
48	Aggregate rating_sqrt	-1.122926	Aggregate rating	1.122926
46	Average Cost for two_cbrt	0.574186	Average Cost for two	0.574186
40	City_sqrt	-2.432530	City	2.432530
37	Country Code_log	3.996050	Country Code	3.996050
35	Cuisines_enc	1.575493	Cuisines	1.575493
34	Cuisines_enc_cbrt	0.393310	Cuisines_enc	0.393310
28	Currency_sqrt	-2.412280	Currency	2.412280
24	Has Online delivery_sqrt	1.062437	Has Online delivery	1.062437
20	Has Table booking_sqrt	2.259133	Has Table booking	2.259133
16	Latitude_sqrt	-3.347294	Latitude	3.347294
14	Longitude_cbrt	-3.263973	Longitude	3.263973
9	Price range_log	0.413940	Price range	0.413940
7	RestaurantName_enc	4.374703	RestaurantName	4.374703
5	RestaurantName_enc_log	2.064310	RestaurantName_enc	2.064310
2	Votes_cbrt	0.843845	Votes	0.843845

Fig: Skewness transformation

In the above transformation for skewness there is significant reduction in skewness of the columns Average Cost for two(cbrt), Country Code(log), Cuisines_enc(cbrt), RestaurantName_enc(log), Votes. We will go ahead and apply the respective transformation methods for these columns.

[75]:

	Skew
Country Code	3.996050
Has Table booking	2.259133
RestaurantName_enc	2.064310
Has Online delivery	1.062437
Price range	0.966022
Votes	0.843845
Average Cost for two	0.574186
Rating text	0.418737
Cuisines_enc	0.393310
Locality_enc	0.346096
Is delivering now	0.000000
Currency	-0.502153
Aggregate rating	-0.898995
City	-1.435432
Latitude	-2.314064
Longitude	-3.333051

Fig: Final skewness check after applying transformation

4.5 Separating features for regression model to predict Average Cost for two

[76]:
```
# Separating the independent and target variables into x and y
x = df.drop(['Average Cost for two'], axis=1)
y = df['Average Cost for two']

print(f"Feature Dimension = {x.shape}")
print(f"Label Dimension = {y.shape}")
display(x.head())
display(y.head())
```
```
Feature Dimension = (8762, 15)
Label Dimension = (8762,)
```
Fig: Separating features and labels for Regression model

```
[77]: # Separating the independent and target variables into x and y
x2 = df.drop(['Price range'], axis=1)
y2 = df['Price range']

print(f"Feature Dimension = {x2.shape}")
print(f"Label Dimension = {y2.shape}")
display(x2.head())
display(y2.unique())
```

```
Feature Dimension = (8762, 15)
Label Dimension = (8762,)
```

Fig: Separating features and labels for classification model

4.6 Scaling Data using Standard scaler for predicting Average cost for two

Standard scaling the data using standard scaler() since all the data is dispersed and differ from each other in large variations.

```
[78]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Scaling data
x = pd.DataFrame(scaler.fit_transform(x), columns = x.columns)

display(x.head())
```

	Country Code	City	Longitude	Latitude	Currency	Has Table booking	Has Online delivery	Is delivering now	Price range	Aggregate rating	Rating text	Votes	RestaurantName_enc	Locality_enc
0	2.615675	-2.667425	-3.313615	-5.417011	-5.903607	-0.379098	-0.601169	0.0	0.288915	0.276470	-1.049025	-0.626733	-0.516062	-1.288029
1	2.615675	-2.667425	-3.313640	-5.418713	-5.903607	-0.379098	-0.601169	0.0	-0.854995	0.800949	0.158722	-0.521668	-0.516062	-1.288029
2	2.615675	-2.667425	-3.313625	-5.416161	-5.903607	-0.379098	-0.601169	0.0	0.288915	0.735389	0.158722	-0.464176	-0.516062	-1.288029
3	2.615675	-2.667425	-3.313800	-5.416335	-5.903607	-0.379098	-0.601169	0.0	1.432825	0.800949	0.158722	-0.464176	-0.516062	-1.288029
4	2.615675	-2.667425	-3.314596	-5.425551	-5.903607	-0.379098	-0.601169	0.0	0.288915	0.407590	-1.049025	-0.464176	-0.516062	-1.216287

Fig: Scaling data using standard scaler

4.7 Multicollinearity Analysis

Multicollinearity is a statistical concept where several independent variables in a model are correlated. We need to eliminate the columns which are more collinear to other columns since only one column is enough for model building.

The VIF values which have more than 10 must be eliminated but in our case there are no values more than 10 so we are retaining all the columns as it is

```
[80]: from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif['Features'] = x.columns
vif['VIF values'] = [variance_inflation_factor(x.values, i) for i in range(len(x.columns))]

vif.sort_values(by='VIF values', ascending=False)
```

[80]:

	Features	VIF values
0	Country Code	4.484625
2	Longitude	4.043782
11	Votes	3.348845
9	Aggregate rating	3.209660
8	Price range	2.032472
3	Latitude	1.864181
5	Has Table booking	1.544788
10	Rating text	1.472991
4	Currency	1.446210
13	Locality_enc	1.250860
14	Cuisines_enc	1.218126
6	Has Online delivery	1.198256
1	City	1.143111
12	RestaurantName_enc	1.070044
7	Is delivering now	NaN

Fig: Checking the VIF values

4.8 Balancing the dataset for Classification task

Fig: Unbalanced dataset

We will use the SMOTE technique to balance the dataset. SMOTE is an oversampling technique where the synthetic samples are generated for the minority class.

```
[84]: # Oversampling the data
      from imblearn.over_sampling import SMOTE
      SM = SMOTE()
      X, Y = SM.fit_resample(x2, y2)
```

Fig: Applying SMOTE

After applying SMOTE, the dataset will be balanced

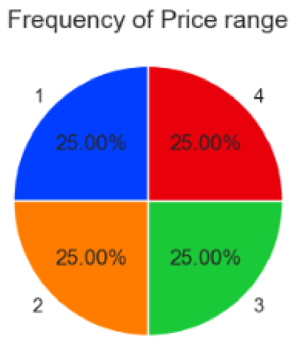


Fig: Balanced Data

Finally, We will proceed with model building.

5. Building Machine Learning Models

5.1. Regression Model (Average cost for two)

5.1.1 Importing the necessary regression algorithms

```
[86]: # Import Regression Algorithms
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import Lasso, Ridge
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR

from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.model_selection import train_test_split
```

Fig: Importing algorithms

5.1.2 Finding out best random state

```
[88]:
%%time
# Build the model
model = LinearRegression()
random_state, acc = find_best_random_state(model, x, y)
print(f"Maximum r2 score is {round(acc,4)} at random_state {random_state}")

Maximum r2 score is 0.8743 at random_state 108
CPU times: total: 46.9 ms
Wall time: 689 ms
```

Fig: best random state

The best random state I achieved was 108 at r2 score of 87.43% for the Linear regression model.

5.1.3 Model Training

Train various models, including Linear Regression, Decision Tree, and Random Forest, KNN, Lasso, Ridge, Gradient boosting, SVR to predict Average cost for two.

5.1.4 Cross-Validation and performance measure using r2 score

Perform cross-validation to assess the model performance. The cross validation is calculate keeping cv=5 value and the metric for calculation of accuracy is r2 score.

```
# Perform cross-validation and measure performance using R-squared (R2)
scores = cross_val_score(svr, x, y, cv=5, scoring='r2')
mse = mean_squared_error(y_test, y_pred)
r2_score_val = svr.score(x_test, y_test)
r_mse = np.sqrt(mse)
```

Fig: CV and r2 score

Display cross-validation results

[101]:

	id	Model	RMSE	R2_Score(test)	CV_Mean	Dif_R2_CVmean
1	Random Forest Regressor	(DecisionTreeRegressor(max_depth=70, max_featu...	0.685863	0.910305	0.901060	0.009246
4	Gradient Boosting Regressor	([DecisionTreeRegressor(criterion='friedman_ms...	0.696700	0.907449	0.763334	0.144114
7	SVR	SVR()	0.752543	0.892017	0.652713	0.239304
2	Decision Tree Regressor	DecisionTreeRegressor(random_state=108)	0.973338	0.819358	0.588506	0.230852
3	K Neighbors Regressor	KNeighborsRegressor()	0.830938	0.868348	0.503549	0.364799
5	Lasso	Lasso()	1.828854	0.362251	0.321357	0.040894
6	Ridge	Ridge()	0.811881	0.874317	-0.971199	1.845516
0	LinearRegression	LinearRegression()	0.811875	0.874319	-6.086404	6.960723

Fig: CV results

Based on the cross-validation results, we can select the best-performing model. **Random Forest** model performs the best with the highest CV score at **90.1%**

The lease performing model after Cross validation is Linear Regression

5.1.5 Hyperparameter Tuning

Performing hyperparameter tuning for the best model (Random Forest) to optimize its performance.

```
[103]: %%time
# Random Forest Regressor
param_dist = {
    'n_estimators': np.arange(10, 200, 5),  # Vary the number of trees
    'max_depth': [None] + list(np.arange(10, 110, 5)),  # Vary the maximum depth of trees
    'min_samples_split': np.arange(2, 11),  # Vary the minimum samples required to split
    'min_samples_leaf': np.arange(1, 11),  # Vary the minimum samples required for a leaf
    'bootstrap': [True, False]  # Bootstrap sampling
}

# Build the model
model = RandomForestRegressor()

# Perform search with cross validation
random_search = RandomizedSearchCV(
    model, param_distributions=param_dist, n_iter=10, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, random_state=random_state)
random_search.fit(x_train, y_train)

CPU times: total: 984 ms
Wall time: 9.93 s

[103]: ▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

Fig: Hyperparameter tuning for best performance

Getting the best parameters which give highest accuracy

```
[104]: # Get the best hyperparameters and the best model
best_params = random_search.best_params_
best_model = random_search.best_estimator_

print("Best Parameters for RandomForestRegressor model:")
best_params

Best Parameters for RandomForestRegressor model:
[104]: {'n_estimators': 60,
 'min_samples_split': 9,
 'min_samples_leaf': 6,
 'max_depth': 70,
 'bootstrap': True}
```

Fig: best parameters

Final run for accuracy check after setting the best parameters

```
[106]: # Create the model with the best parameters
       best_model = RandomForestRegressor(max_depth=70, min_samples_leaf=6, min_samples_split=9,
                                          n_estimators=60)

       best_model.fit(x_train, y_train)
       y_pred = best_model.predict(x_test)

       # Check the r2 score
       r2_score_val = r2_score(y_test, y_pred)
       print(f"Maximum R2 score: {r2_score_val*100:.2f}%")

       Maximum R2 score: 90.98%
```

Fig: Final run for accuracy check

For the final run for accuracy check I obtained 90.98% accuracy which was my best accuracy for the Regression model.

5.1.6 Saving the Model

We will save the model for future use using Joblib in .pkl extension

```
[107]: # Saving the model using .pkl
       import joblib
       joblib.dump(best_model, "avg_cost_regressor_model.pkl")

[107]: ['avg_cost_regressor_model.pkl']
```

Fig: Model saving

5.2 Classification Model (Price Range)

5.2.1 Defining function to find best random state

```
[108]: from sklearn.model_selection import train_test_split
       from sklearn.ensemble import RandomForestClassifier
       from sklearn.metrics import accuracy_score

       def find_best_random_state(model, x, y):
           best_acc = 0
           best_random_state = 0

           for i in range(1,200):
               x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=i)
               model.fit(x_train, y_train)
               y_pred = model.predict(x_test)
               acc = accuracy_score(y_test, y_pred)
               if acc > best_acc:
                   best_acc = acc
                   best_random_state = i

           return [best_random_state, best_acc]
```

Fig: Function to find best random state

The Best accuracy is 0.9942 at random_state 139

```
[109]:
%%time
# Build the model
model = RandomForestClassifier()
random_state, acc = find_best_random_state(model, X, Y)
print(f"Best accuracy is {round(acc,4)} at random_state {random_state}")

Best accuracy is 0.9942 at random_state 139
CPU times: total: 2min 9s
Wall time: 3min 36s
```

Fig: Best accuracy at best random state

5.2.2 Creating train and test split

```
x_train shape:  (12017, 15)
x_test shape:   (5151, 15)
y_train shape:  (12017,)
y_test shape:   (5151,)
```

Fig: train/test split

5.2.3 Importing classification algorithms

```
[111]:
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, accuracy_score, auc
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

Fig: Classification algorithms

5.2.4 Defining function to calculate the accuracy of model

Defining the function to calculate the accuracy, Confusion matrix and y_pred

```
[112]:
# Functions
def calc_accuracy(model, id_model):
    ''' Calculate the accuracy of the model. Return the accuracy, training accuracy, and the predicted values.'''
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    acc = accuracy_score(y_test, y_pred)
    acc_train = model.score(x_train, y_train)

    print(f"\nModel: {id_model}")
    print(f"Confusion matrix: \n {confusion_matrix(y_test, y_pred)}")
    print(f"Classification report: \n {classification_report(y_test, y_pred)}" )
    print(f"Training Accuracy using {id_model} is {acc_train*100.0:.2f}%")
    print(f"The accuracy score using {id_model} is {round(acc*100.0, 2)}%")

    return [acc, acc_train, y_pred]
```

Fig: Accuracy calculation function

Now lets define the models and create a data frame to store the values like id, Model, Training Accuracy, Model Accuracy Score which we obtain after running the models.

```
[113]:
models = {'RandomForestClassifier' : RandomForestClassifier(),
          'ExtraTreesClassifier': ExtraTreesClassifier(),
          'LogisticRegression': LogisticRegression(),
          'SVC': SVC(),
          'GradientBoostingClassifier': GradientBoostingClassifier(),
          'AdaBoostClassifier': AdaBoostClassifier(),
          'BaggingClassifier': BaggingClassifier()}

# Setting up for saving the results of each model
df_model_accuracy = pd.DataFrame(columns=['id','Model','Training Accuracy','Model Accuracy Score'])
y_pred = {}
```

Fig: Creating model instances

5.2.5 Running the models and comparing the model scores

	id	Model	Training Accuracy	Model Accuracy Score
0	RandomForestClassifier	(DecisionTreeClassifier(max_features='sqrt', r...	1.000000	0.994370
6	BaggingClassifier	(DecisionTreeClassifier(random_state=165362422...	0.999168	0.990487
4	GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.989681	0.985634
1	ExtraTreesClassifier	(ExtraTreeClassifier(random_state=1525629565),...	1.000000	0.983887
3	SVC	SVC()	0.947741	0.944865
2	LogisticRegression	LogisticRegression()	0.940251	0.934382
5	AdaBoostClassifier	(DecisionTreeClassifier(max_depth=1, random_st...	0.740701	0.742186

Fig: Model comparison

5.2.6 Defining the function to calculate CV Score

```
[122]: def checking_cvscore(id_model, model, y_pred):
           score = cross_val_score(model, X, Y, cv=5, scoring='accuracy')

           score_mean = score.mean()
           diff = accuracy_score(y_test, y_pred) - score_mean

           print(f"\n::: Model: {id_model}::: \nscore:{score}")
           print(f"Score mean: {score_mean:.4f}")
           print(f"Difference between Accuracy score and cross validation score is {diff:.4f}")
           return [score_mean, diff]
```

Fig: Function to calculate Cv score

5.2.7 Comparing the models after Cross validation

After applying cross validation, we an find that the highest performance is for the Random forest classifier model

[125]:

id	Model	Training Accuracy	Model Accuracy Score	CV score mean	Diff Acc and cv score
RandomForestClassifier	(DecisionTreeClassifier(max_features='sqrt', r...	1.000000	0.994370	0.967324	0.027046
BaggingClassifier	(DecisionTreeClassifier(random_state=165362422...	0.999168	0.990487	0.950202	0.040285
GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.989681	0.985634	0.972158	0.013476
ExtraTreesClassifier	(ExtraTreeClassifier(random_state=1525629565),...	1.000000	0.983887	0.935523	0.048364
SVC	SVC()	0.947741	0.944865	0.935230	0.009635
LogisticRegression	LogisticRegression()	0.940251	0.934382	0.930977	0.003405
AdaBoostClassifier	(DecisionTreeClassifier(max_depth=1, random_st...	0.740701	0.742186	0.721519	0.020667

Fig: Models comparison after CV

Now lets apply the hyper parameter tuning and check the accuracy improvement.

5.2.8 Hyperparameter tuning

Defining the parameter for random forest regression model

```
[128]: # RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
            'n_estimators': np.arange(100, 305, 5),  # 1.Vary the number of trees default 100
            'max_features': ['sqrt', 'log2', None], # 2. default 'sqrt'
            'max_depth': [None] + list(np.arange(80, 110, 5)),  # 3.maximum depth of trees default None
            'max_leaf_nodes': [9, 12, 30], # 4. Restrics the grow of each tree default None,
            'min_samples_split': np.arange(2, 5),  # 6.minimum samples required to split default=2

#             'criterion':['gini','entropy'],
#             'random_state': [random_state, 50, 500, 1000],
#             'n_jobs': [-1, 1]
            }

# Build the model
model = RandomForestClassifier()
```

Fig: Hyper parameter tuning

```
[130]: # Get the best hyperparameters and the best model
best_params = random_search.best_params_
best_model_cl = random_search.best_estimator_

print("Best Parameters for RandomForestClassifier model:")
display(best_params)

Best Parameters for RandomForestClassifier model:
{'n_estimators': 265,
 'min_samples_split': 2,
 'max_leaf_nodes': 30,
 'max_features': None,
 'max_depth': 95}
```

Fig: Best parameters

5.2.9 Final model

```
[132]: %%time
# Create the model with the best parameters
best_model = RandomForestClassifier (
                                    max_depth = 105,
                                    max_features = None,
                                    max_leaf_nodes = 30,
                                    min_samples_split = 3,
                                    n_estimators = 160,
)

best_model.fit(x_train, y_train)
y_pred = best_model.predict(x_test)

# Check the accuracy
acc = accuracy_score(y_test, y_pred)
print(f"accuracy_score: {round(acc*100,2)}%")

accuracy_score: 97.22%
CPU times: total: 2.62 s
Wall time: 3.86 s
```

Lets adjust the parameters a bit and check fro the accuracy

```
[133]: # Create the model with the best parameters
best_model = RandomForestClassifier (
                                    max_depth = None,
                                    max_features = 'sqrt',
                                    max_leaf_nodes = None,
                                    min_samples_split = 3,
                                    n_estimators = 160,
)

best_model.fit(x_train, y_train)
y_pred = best_model.predict(x_test)

# Check the accuracy
acc = accuracy_score(y_test, y_pred)
print(f"accuracy_score: {round(acc*100,2)}%")

accuracy_score: 99.4%
```

Fig: Final model accuracy calculation

By running the model with the best parameters the obtained accuracy was **97.22%** but after tweaking the parameter a bit, obtained an accuracy of **99.4%.**

5.2.10 Applying ROC curve on the best performing model

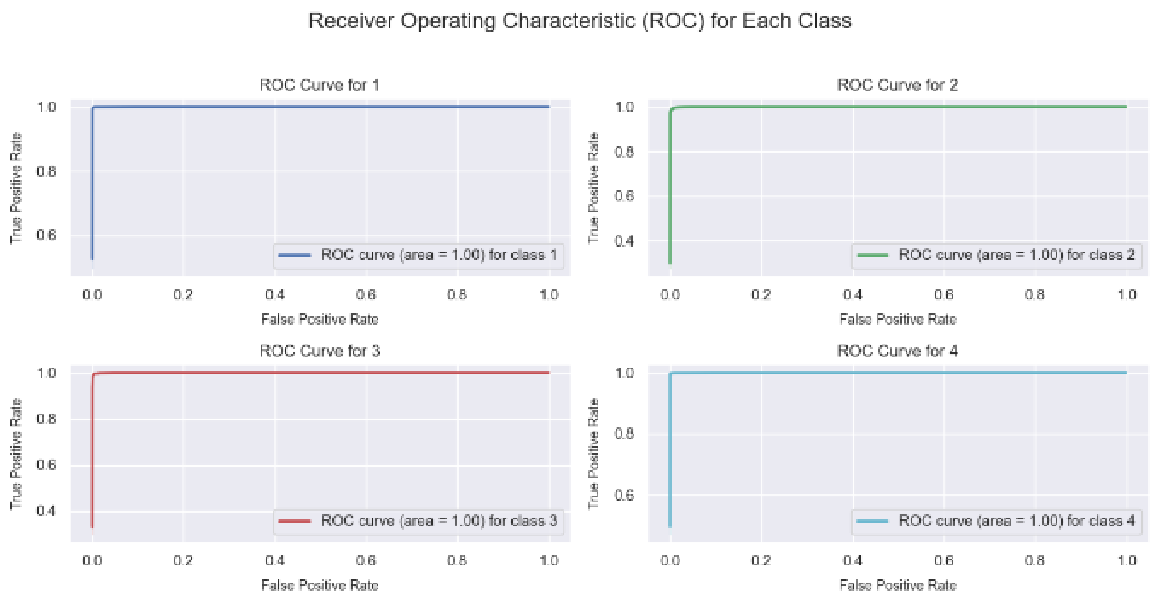


Fig: ROC curve for all the categories of target Price range

5.2.11 Saving the model

```
[151]: # Saving the model using .pkl
       import joblib
       joblib.dump(best_model, "clf_model.pkl")

[151]: ['clf_model.pkl']
```

Fig: Model saving

6 Concluding Remarks

Findings and Insights

From the data analysis and model building process, several key insights were uncovered:

- Geographical Insights: The majority of the restaurants are located in India, followed by the United States and the United Kingdom. This reflects the widespread usage of the Zomato platform in these regions.
- Cuisine Preferences: Indian, Chinese, and Continental cuisines are the most popular across the dataset. This insight can help understand global and regional culinary preferences.
- Cost Analysis: Indian cities generally offer more budget-friendly dining options compared to cities in the United States and the United Kingdom. This analysis can help identify cities where dining out is more economical.
- Rating Analysis: Higher restaurant costs generally correlate with better ratings. However, there are also high-rated restaurants with moderate costs, indicating good value for money.
- Best Model: The Random Forest model, after hyperparameter tuning, showed the best performance in predicting restaurant ratings. This model can be used to provide personalized recommendations to users on platforms like Zomato.

Future Work

Future work can include:

- Incorporating additional features such as user demographics and dining preferences to further enhance the predictive power of the model.
- Exploring deep learning models for more complex and nuanced predictions.
- Developing a recommendation system to suggest restaurants based on user preferences and past behavior.

Conclusion

- The dataset comprises 9551 rows and 22 columns
- Columns such as 'Restaurant ID', 'Address', and 'Switch to order menu' were dropped after univariate analysis due to no much significance
- There are two target variables: 'Average Cost for Two', which is continuous(regression model) and 'Price Range,' a categorical variable with four possible values(classification model)
- The best regression model is Randomforestclassifier with a accuracy of 90.98%
- The best classification model is Randomforestclassifier with a accuracy of 99.4%

- During outlier handling, 8.17% of the data was deleted
- The dataset has no duplicates
- During handling missing values the 9 rows were deleted which represented very less constitution of the original dataset

References

- Scikit-Learn Library Documentation
- Blogs from Towards Data Science, Analytics Vidhya, Medium
- Andrew Ng's Notes on Machine Learning (GitHub)
- Data Science Projects with Python, Second Edition by Packt
- Hands-On Machine Learning with Scikit-Learn and TensorFlow by Aurélien Géron
- Lackermair, G., Kailer, D. & Kanmaz, K. (2013). Importance of online product reviews from a consumer's perspective. Horizon Research Publishing, 1-5. doi: 10.13189/aeb.2013.010101
- Baccianella, S., Esuli, A. & Sebastiani, F. (2009). Multi-facet rating of product reviews. Proceedings of the 31st European Conference on Information Retrieval (ECIR), 461-472
- Chevalier, J. & Mayzlin, D. (2006). The Effect of Word of Mouth on Sales: Online Book Reviews. Journal of Marketing Research, 43, 345-354. doi: 10.3386/w10148
- Pang, B., Lee, L. & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. Proceedings of EMNLP, 271-278. doi: 10.3115/1118693.1118704