**RESTAURANT FOOD COST**

**Developing Machine Learning Model for predicting Restaurant Food Cost**

Table of Contents

[Introduction 2](#_Toc65360539)

[The Problem Statement 2](#_Toc65360540)

[About Dataset 2](#_Toc65360541)

[Load Dataset 2](#_Toc65360542)

[Checking General Information 3](#_Toc65360543)

[Data Analysis 3](#_Toc65360544)

[Exploratory Data Analysis (EDA) and Visualisation Concluding Remarks 11](#_Toc65360545)

[Univariate Analysis 11](#_Toc65360546)

[Bivariate Analysis 13](#_Toc65360547)

[Multi Variate Analysis 14](#_Toc65360548)

[Prepare Dataset for Model Training 15](#_Toc65360549)

[Label Encoding 15](#_Toc65360550)

[Outlier(s): Detection & Removal 16](#_Toc65360551)

[Separate Input and Output/Target Variable 16](#_Toc65360552)

[Skewness: Detection & Treatment 16](#_Toc65360553)

[Scale Data for Model Training 17](#_Toc65360554)

[Model Training: Finding the best model 17](#_Toc65360555)

[Define the function(s) for Automation of Model Training and Testing 17](#_Toc65360556)

[Prepare Model List and Test to get Best Model 20](#_Toc65360557)

[Using PCA (Principle Component Analysis) 21](#_Toc65360558)

[Building Final Model List with PCA(n\_components=100) 24](#_Toc65360559)

[Model Selection: The Final Model 25](#_Toc65360560)

[Conclusion 26](#_Toc65360561)

[Model Prediction 26](#_Toc65360562)

[Pre-Processing Pipeline 26](#_Toc65360563)

[Load Model from Serialized Object/Pickle file 26](#_Toc65360564)

[Save Predicted Values 27](#_Toc65360565)

[Concluding Remarks 27](#_Toc65360566)

# Introduction

Restaurants are an essential part of a country’s economy and society. Whether it may be for social gatherings or a quick bite or to have favourite food, most of us have experienced at least one visit. But there is one factor that will make us reconsider visiting our favourite restaurant, **the** **cost**. Here in, I am going to talk about the complete process of building machine learning model and predicting the cost of the food served by the restaurants across different cities in India.

# The Problem Statement

To investigate the factors that really affect the cost and, on that basis, build a robust model that is capable of predicting the food cost of a restaurant.

# About Dataset

1. **Training Set** – The dataset is in .xlsx (Microsoft Excel) format and consists of ***12,690 records******with******9 features*** as explained below:
   1. **TITLE:** The feature of the restaurant which can help identify what and for whom it is suitable for.
   2. **RESTAURANT\_ID:** A unique ID for each restaurant.
   3. **CUISINES:** The variety of cuisines that the restaurant offers.
   4. **TIME:** The open hours of the restaurant.
   5. **CITY:** The city in which the restaurant is located.
   6. **LOCALITY:** The locality of the restaurant.
   7. **RATING:** The average rating of the restaurant by customers.
   8. **VOTES:** The overall votes received by the restaurant.
   9. **COST:** The average cost of a two-person meal.
2. **Test Set** – Contains ***4,231 records with 8 features***.
3. **Download Files**: <https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects>

# Load Dataset

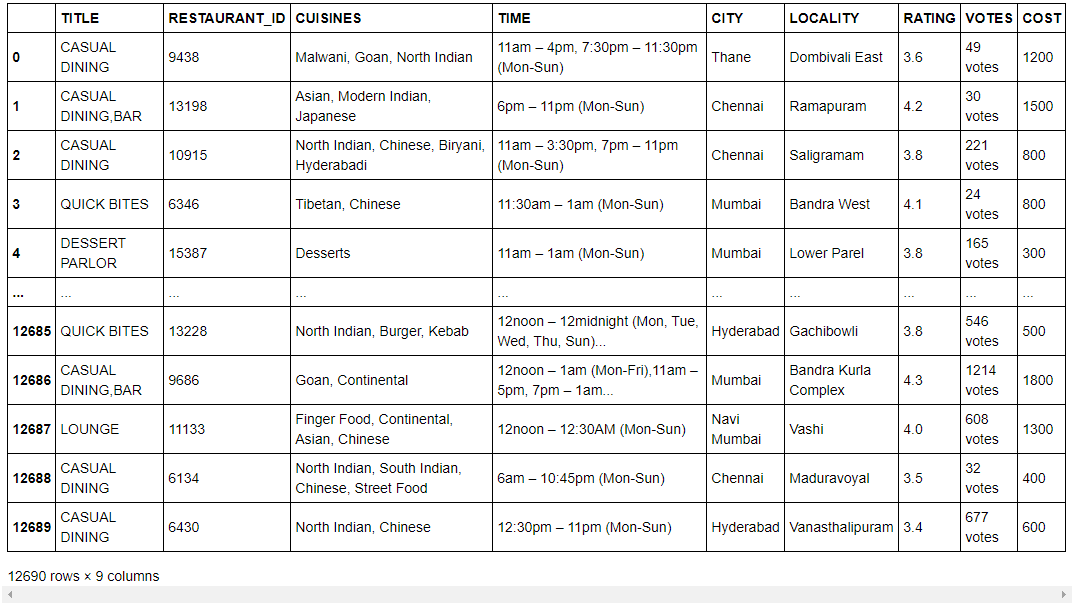
**import** **pandas** **as** **pd**

**import** **re**

df\_rfc = pd.read\_excel('Data\_Train.xlsx',sheet\_name='Sheet1')

*#Interpreting dataset*

df\_rfc



After taking a look at the data, there are 12,690 samples in the training set and the goal here would be to model cost based on 12,690 samples in the training set and see how well the model performs on the 4,231 samples in the test set. Since, the target variable COST is of continuous type, therefore, it is a Regression Problem.

## Checking General Information

*#Checking general information of dataset*

df\_rfc.info()

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

RangeIndex: 12690 entries, 0 to 12689

Data columns (total 9 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 TITLE 12690 non-null object

1 RESTAURANT\_ID 12690 non-null int64

2 CUISINES 12690 non-null object

3 TIME 12690 non-null object

4 CITY 12578 non-null object

5 LOCALITY 12592 non-null object

6 RATING 12688 non-null object

7 VOTES 11486 non-null object

8 COST 12690 non-null int64

dtypes: int64(2), object(7)

memory usage: 892.4+ KB

After going through the general information of dataset, it is found that the feature RESTAURANT\_ID and COST are of continuous (int64) type while others are of discrete (object) type which needs to be handled accordingly. Also, feature CITY, LOCALITY, RATING and VOTES has null values and needs to be treated accordingly.

# Data Analysis

In this part, we will perform an analysis on the given data for each feature and try to get some more insights from there. Also we will perform cleaning of data.

1. **TITLE:** Checking for unique values and separating them into different columns.

x = 'TITLE'

*#checking for unique values*

df[x].unique()

array(['CASUAL DINING', 'CASUAL DINING,BAR', 'QUICK BITES',

'DESSERT PARLOR', 'CAFÉ', 'MICROBREWERY',

'QUICK BITES,BEVERAGE SHOP', 'CASUAL DINING,IRANI CAFE',

'BAKERY,QUICK BITES', 'None', 'BAR,CASUAL DINING', 'BAR', 'PUB',

'BEVERAGE SHOP', 'FINE DINING', 'CAFÉ,QUICK BITES',

'BEVERAGE SHOP,DESSERT PARLOR', 'SWEET SHOP,QUICK BITES',

'DESSERT PARLOR,SWEET SHOP', 'BAKERY', 'BAKERY,DESSERT PARLOR',

'BAR,LOUNGE', 'FOOD COURT', 'LOUNGE',

'DESSERT PARLOR,BEVERAGE SHOP', 'LOUNGE,CASUAL DINING',

'FOOD TRUCK', 'QUICK BITES,FOOD COURT', 'SWEET SHOP',

'BEVERAGE SHOP,FOOD COURT', 'PUB,CASUAL DINING', 'MESS',

'MICROBREWERY,CASUAL DINING', 'CASUAL DINING,SWEET SHOP', 'KIOSK',

'QUICK BITES,KIOSK', 'CLUB', 'FINE DINING,BAR',

'DESSERT PARLOR,QUICK BITES', 'FOOD COURT,QUICK BITES',

'LOUNGE,CAFÉ', 'BAKERY,CONFECTIONERY', 'CASUAL DINING,CAFÉ',

'DHABA', 'CAFÉ,DESSERT PARLOR', 'QUICK BITES,DESSERT PARLOR',

'PUB,MICROBREWERY', 'LOUNGE,BAR', 'DESSERT PARLOR,CAFÉ',

'CAFÉ,BAR', 'SWEET SHOP,CONFECTIONERY', 'CASUAL DINING,PUB',

'MICROBREWERY,BAR', 'DESSERT PARLOR,BAKERY',

'QUICK BITES,SWEET SHOP', 'BEVERAGE SHOP,QUICK BITES',

'CASUAL DINING,LOUNGE', 'CASUAL DINING,CLUB', 'QUICK BITES,CAFÉ',

'BAR,CAFÉ', 'CAFÉ,CASUAL DINING', 'QUICK BITES,CASUAL DINING',

'CASUAL DINING,MICROBREWERY', 'CASUAL DINING,BAKERY',

'CAFÉ,BAKERY', 'MEAT SHOP', 'QUICK BITES,BAKERY',

'BAR,FINE DINING', 'SWEET SHOP,CASUAL DINING',

'MEAT SHOP,QUICK BITES', 'PUB,LOUNGE', 'BAKERY,CAFÉ',

'COCKTAIL BAR', 'FINE DINING,LOUNGE', 'CONFECTIONERY',

'QUICK BITES,BAR', 'BAKERY,FOOD COURT', 'PUB,BAR',

'DESSERT PARLOR,FOOD COURT', 'QUICK BITES,FOOD TRUCK',

'BAKERY,BEVERAGE SHOP', 'CLUB,BAR', 'BAKERY,SWEET SHOP',

'SWEET SHOP,BAKERY', 'CASUAL DINING,FOOD COURT', 'PAAN SHOP',

'BEVERAGE SHOP,CAFÉ', 'FOOD COURT,DESSERT PARLOR',

'CLUB,MICROBREWERY', 'CAFÉ,BEVERAGE SHOP',

'DESSERT PARLOR,PAAN SHOP', 'MICROBREWERY,LOUNGE', 'LOUNGE,CLUB',

'SWEET SHOP,DESSERT PARLOR', 'BAR,PUB',

'CONFECTIONERY,QUICK BITES', 'DESSERT PARLOR,KIOSK', 'LOUNGE,PUB',

'SWEET SHOP,BEVERAGE SHOP', 'FINE DINING,CAFÉ',

'BEVERAGE SHOP,CASUAL DINING', 'KIOSK,QUICK BITES',

'CASUAL DINING,DESSERT PARLOR', 'LOUNGE,FINE DINING', 'PUB,CAFÉ',

'CAFÉ,LOUNGE', 'BAR,CLUB', 'COCKTAIL BAR,CASUAL DINING',

'MICROBREWERY,PUB', 'CAFÉ,FINE DINING', 'KIOSK,FOOD COURT',

'LOUNGE,MICROBREWERY', 'BAR,MICROBREWERY'], dtype=object)

After having a close look it is found that values are stored in comma separated form, therefore, splitting these values into different columns and extracting the unique values will give us the unique number of title present in the dataset which we will use for encoding before providing to model training.

*#Splitting feature TITLE*

x = 'TITLE'

temp\_df = df[x].str.split(',',expand=**True**)

unique\_values = []

**for** i **in** temp\_df.columns:

unique\_values = np.append(unique\_values,temp\_df[i].str.strip().unique())

unique\_title = pd.unique(unique\_values)

unique\_title = unique\_title[(unique\_title != 'None')&(unique\_title != **None**)&(unique\_title !='')]

print(unique\_title)

*#Seperating unique values into different columns*

**for** i **in** unique\_title:

col = "is "+i

df[col] = df[x].apply(**lambda** y: 'Yes' **if** i **in** y **else** 'No')

*#Interpreting first 5 rows after extraction*

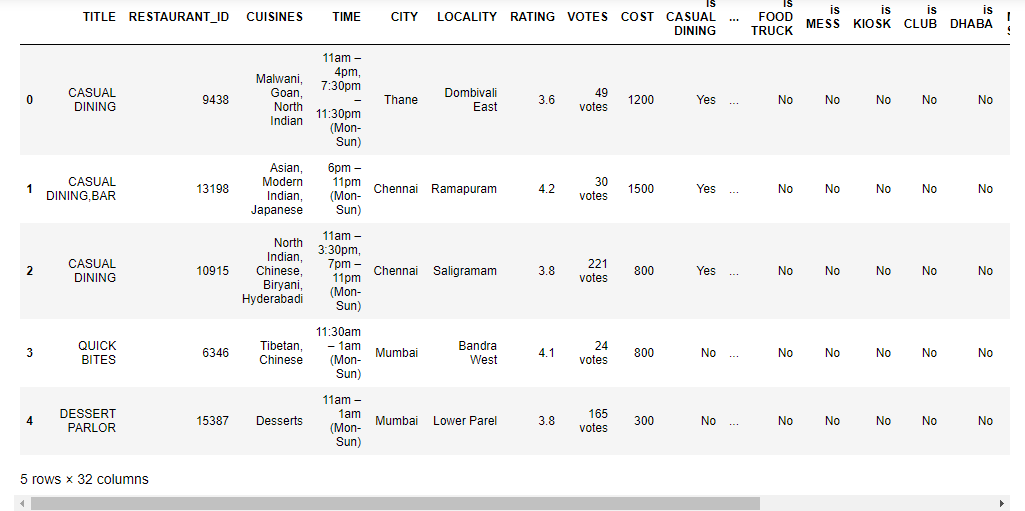
df.head(5)

['CASUAL DINING' 'QUICK BITES' 'DESSERT PARLOR' 'CAFÉ' 'MICROBREWERY'

'BAKERY' 'BAR' 'PUB' 'BEVERAGE SHOP' 'FINE DINING' 'SWEET SHOP'

'FOOD COURT' 'LOUNGE' 'FOOD TRUCK' 'MESS' 'KIOSK' 'CLUB' 'DHABA'

'MEAT SHOP' 'COCKTAIL BAR' 'CONFECTIONERY' 'PAAN SHOP' 'IRANI CAFE']



1. **CUISINES:** Checking for unique values and separating them into different columns.

x = 'CUISINES'

*#checking for unique values*

df[x].unique()

array(['Malwani, Goan, North Indian', 'Asian, Modern Indian, Japanese',

'North Indian, Chinese, Biryani, Hyderabadi', ...,

'North Indian, Burger, Kebab', 'Goan, Continental',

'Finger Food, Continental, Asian, Chinese'], dtype=object)

*#Splitting feature CUISINES*

x = 'CUISINES'

temp\_df = df[x].str.split(',',expand=**True**)

unique\_values = []

**for** i **in** temp\_df.columns:

unique\_values = np.append(unique\_values,temp\_df[i].str.strip().unique())

unique\_cuisines = pd.unique(unique\_values)

unique\_cuisines = unique\_cuisines[(unique\_cuisines != 'None')&(unique\_cuisines != **None**)&(unique\_cuisines !='')]

print(unique\_cuisines)

*#Seperating unique values into different columns*

**for** i **in** unique\_cuisines:

col = "is "+i

df[col] = df[x].apply(**lambda** y: 'Yes' **if** i **in** y **else** 'No')

*#Interpreting first 5 rows after extraction*

df.head(5)

['Malwani' 'Asian' 'North Indian' 'Tibetan' 'Desserts' 'Cafe' 'Bar Food'

'South Indian' 'Fast Food' 'Arabian' 'Maharashtrian' 'Parsi' 'Chinese'

'Bakery' 'Continental' 'Andhra' 'Biryani' 'Italian' 'Finger Food'

'Beverages' 'American' 'European' 'Ice Cream' 'Kerala' 'Seafood' 'Pizza'

'Mithai' 'Rolls' 'Thai' 'Juices' 'Burger' 'Hyderabadi' 'Mediterranean'

'Gujarati' 'Mexican' 'Healthy Food' 'Sandwich' 'Indian' 'Coffee'

'Indonesian' 'BBQ' 'Bihari' 'Lebanese' 'Bengali' 'Chettinad' 'Mughlai'

'Street Food' 'Rajasthani' 'Portuguese' 'Oriya' 'Japanese' 'Ethiopian'

'Modern Indian' 'Spanish' 'Russian' 'Mangalorean' 'Turkish' 'Steak'

'Kebab' 'Wraps' 'Momos' 'Naga' 'Burmese' 'Malaysian' 'Korean' 'Tea'

'Sindhi' 'Vietnamese' 'Goan' 'French' 'Raw Meats' 'Salad'

'Middle Eastern' 'Kashmiri' 'North Eastern' 'Sri Lankan' 'Sushi'

'Tex-Mex' 'Afghan' 'Konkan' 'Bubble Tea' 'African' 'German' 'Drinks Only'

'Paan' 'Assamese' 'Nepalese' 'Hot dogs' 'Cafe Food' 'Awadhi' 'British'

'Bohri' 'Armenian' 'South American' 'Iranian' 'Lucknowi'

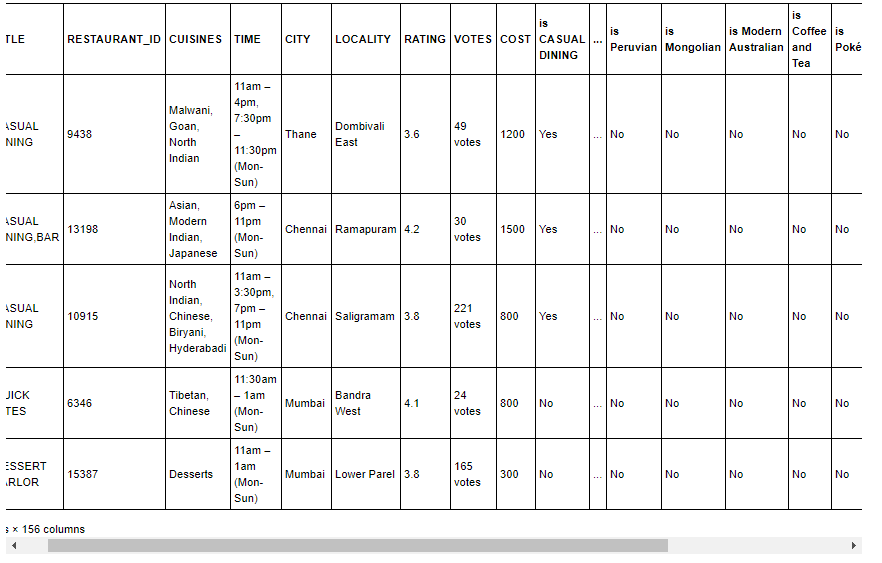
'Charcoal Chicken' 'Tamil' 'Pakistani' 'Irish' 'Multi Cuisine'

'Roast Chicken' 'Singaporean' 'Egyptian' 'Cantonese' 'Belgian' 'Panini'

'Greek' 'Pan Asian' 'Oriental' 'Grill' 'Hot Pot' 'Moroccan' 'Fusion'

'Peruvian' 'Mongolian' 'Modern Australian' 'Coffee and Tea' 'Poké'

'Cuisine Varies' 'Jewish' 'Satay' 'Vegan' 'Dumplings']



After looking into feature cuisines, we have separated the unique values into different columns by assigning each restaurant with Yes (if feature is present) or No (if feature is not present) value.

1. **TIME:** Checking feature for unique values and separating them into different columns.

*#Checking feature TIME for unique values*

x = 'TIME'

df[x].value\_counts()

11am – 11pm (Mon-Sun) 1415

12noon – 12midnight (Mon-Sun) 469

12noon – 11pm (Mon-Sun) 407

11am – 12midnight (Mon-Sun) 310

10am – 10pm (Mon-Sun) 221

...

11:45am – 1am (Mon-Sun) 1

Closed (Mon, Sat), 6am – 10pm (Tue, Wed, Thu, Fri... 1

11:45am – 12:45AM (Mon-Sun) 1

6am – 8:30pm (Mon-Sat),6am – 12noon (Sun) 1

1:30pm – 4pm, 6pm – 11:30pm (Mon, Wed, Thu, Fri... 1

Name: TIME, Length: 2689, dtype: int64

This feature is quite complex as it contains **2689 unique values**. So, **dropping** this feature from dataset and moving further with other features.

1. **CITY:** Checking feature for unique values and separating them into different columns.

*#checking for unique values*

x = 'CITY'

df[x].value\_counts()

Chennai 2174

Bangalore 2149

Hyderabad 1819

Mumbai 1722

New Delhi 1321

...

Karol Bagh 1

Karnataka 560102 1

Chembur. 1

New Delhi-110024 1

chennai 1

Name: CITY, Length: 359, dtype: int64

As we can see, this feature has 359 unique values and also contains similar values with certain changes, therefore, splitting them into different columns and then extracting the unique values to get the actual number of unique values.

*#Splitting feature CITY into different columns by space*

x = 'CITY'

temp\_df = df[x].str.split(' ',expand=**True**)

temp\_df.replace(np.nan,'Unkown',inplace=**True**)

**for** i **in** temp\_df.columns:

temp\_df[i]=temp\_df[i].apply(**lambda** y: y.title() **if** len(y)>3 **and** y.isalpha() **else** 'Unkown')

unique\_values = []

**for** i **in** temp\_df.columns:

unique\_values = np.append(unique\_values,pd.unique(temp\_df[i]))

unique\_city = pd.unique(unique\_values)

print(unique\_city)

*#Seperating unique values into different columns*

df[x].replace(np.nan,'Unkown',inplace=**True**)

**for** i **in** unique\_city:

col = "in "+i

df[col] = df[x].apply(**lambda** y: 'Yes' **if** i **in** y.title() **else** 'No')

['Thane' 'Chennai' 'Mumbai' 'Bangalore' 'Gurgaon' 'Hyderabad' 'Kochi'

'Andheri' 'Unkown' 'Malad' 'Bangalor' 'Navi' 'Bandra' 'Delhi' 'Noida'

'Secunderabad' 'India' 'Madhuranagar' 'Faridabad' 'Maharashtra'

'Telagana' 'Ghaziabad' 'Karnataka' 'Kerala' 'Edappally' 'Kadavanthra'

'Ernakulam' 'Bengalore' 'Near' 'Kilpauk' 'Bengaluru' 'Kothaguda'

'Goregaon' 'Banglore' 'Tamil' 'Kakkanad' 'Outer' 'Mulund' 'Telangana'

'Ponnuruni' 'Gachibowli' 'Semmancheri' 'Powai' 'Dombivali' 'Kandivali'

'Dewan' 'Gurugram' 'Sector' 'Kaloor' 'Besant' 'Arumbakkam' 'Adjacent'

'Dwarka' 'Kalyan' 'Avadi' 'Kondapur' 'Mehdipatnam' 'Gandipet' 'Velachery'

'Pallavaram' 'Vijaya' 'Metro' 'Madhapur' 'Sarjapur' 'Whitefield' 'Rohini'

'Karol' 'Perungudi' 'Thykoodam' 'Greater' 'Khairatabad' 'Chullickal'

'Grant' 'Hitech' 'West' 'Chander' 'Nedumbassery' 'Naya' 'Pitampura'

'Lower' 'Rajiv' 'Medavakkam' 'Sathya' 'Behind' 'Palarivattom' 'Brigade'

'Virar' 'Aluva' 'Marine' 'Nallathambi' 'Citypark' 'Bhayandar'

'Thammenahalli' 'Khar' 'Road' 'Kukatpally' 'Faridabd' 'Dilsukhnagar'

'Potheri' 'Mahim' 'Lingampally' 'Vasai' 'Banaswadi' 'Ward' 'Perumbavoor'

'Mira' 'Pokhran' 'Uttar' 'Naharpar' 'Hosur' 'Sriram' 'Vyttila' 'Banjara'

'Malapallipuram' 'Panampilly' 'Borivali' 'Ecil' 'Jubilee' 'Amrit'

'Telengana' 'Rajanpada' 'Mahabalipuram' 'Gurgoan' 'Elamakkara' 'Kolathur'

'Rodeo' 'Pallimukku' 'Champapet' 'Andavar' 'Nungambakkam' 'Jogeshwari'

'Kukatapally' 'Navallur' 'Beside' 'Begumpet' 'Maharaja' 'Ashok'

'Trivandrum' 'Narayanguda' 'Thevera' 'Palm' 'East' 'Ramapuram' 'Nandanam'

'Saket' 'Indiranagar' 'Thiruvanmiyur' 'Ambattur' 'Banglaore' 'Anna'

'Kanakapura' 'Serilingampally' 'Himayath' 'Nallala' 'Wagle' 'First'

'Chenn' 'Perambur' 'Vaishali' 'Thanisandra' 'Block' 'Opposite'

'Vadapalani' 'Badlapur' 'Kalamassery' 'Palavakkam' 'Mahadevpura'

'Veliaveetil' 'Sholinganallur' 'Tripunithura' 'Mogappair' 'Marathahalli'

'Haridwar' 'Indirapuram' 'Manikonda' 'Rajarajeshwari' 'Fort' 'Lahari'

'Ramanthapur' 'Phase' 'Uppal' 'Nizampet' 'Ulsoo' 'Chromepet' 'Janakpuri'

'Tambaram' 'Malleshwaram' 'Kadubesanahalli' 'Haryana' 'Golf' 'Masab'

'Lokhandwala' 'Teynampet' 'Gurudwara' 'Land' 'Circle' 'Reliance' 'Nadu'

'Elamkulam' 'Ring' 'Main' 'Rama' 'Nagar' 'Layout' 'Pillar' 'Neerus'

'Bagh' 'City' 'Maredpally' 'Padur' 'Parel' 'Gandhi' 'Ramalayam' 'Stage'

'Kovalam' 'Bank' 'Drive' 'Galleria' 'Village' 'Palya' 'Shangrilla'

'Munrshwara' 'Thuraipakkam' 'Karapakkam' 'Thousand' 'Santosh' 'Kailash'

'Koramangala' 'Neelankarai' 'Raod' 'Pradesh' 'Bharat' 'Hills' 'Kaur'

'Ravipuram' 'Street' 'Chakala' 'Mukteshwar' 'Guda' 'Excellency' 'Hotel'

'Vihar' 'Beach' 'Coast' 'Chrompet' 'Railway' 'Ramlila' 'Salai' 'Estate'

'International' 'Tavarekere' 'Barathi' 'Synergy' 'House' 'Apartments'

'This' 'Avenue' 'Plaza' 'Center' 'Housing' 'Course' 'Jyothinivas' 'Tank'

'Shakurpur' 'Fresh' 'Teachers' 'Commercial' 'Vacs' 'Temple' 'Kamala'

'Vasanth' 'Lights' 'Harlur' 'Market' 'Have' 'Ashram' 'Gardens' 'Ground'

'Airport' 'Park' 'Vivekananda' 'Mark' 'Cinema' 'Society' 'College'

'Colony' 'Pastries' 'Stop' 'Indian' 'Thrissur' 'Gardania' 'Landmark'

'Company' 'Above' 'Building' 'Signal' 'Mettuguda' 'Stand' 'Mahaveer'

'Delivery' 'Connaught' 'Nmrec' 'Subhash' 'Hard' 'Place' 'Loyola' 'Ware'

'Ozone' 'Station' 'Restaurant' 'Ever' 'Paharganj' 'Degree' 'Green'

'Apartment']

After inspecting the feature CITY, we have extracted the unique values from it and then separated them into different columns by assigning Yes (if feature is present) or No (if feature is not present) with corresponding restaurant.

1. **LOCALITY:** Checking feature for unique values and separating them into different columns.

*#Checking feature LOCALITY for unique values*

x = 'LOCALITY'

u\_locality = df[x].value\_counts()

u\_locality

Gachibowli 166

Indiranagar 138

Edappally 122

Kakkanad 121

HSR 120

...

Manjapetty Aluva Perumbavoor Road 1

Tambaram Senatorium 1

South Extension Part-2 1

Near City Centre Metro 1

Chettipunyam 1

Name: LOCALITY, Length: 1416, dtype: int64

As we can see, this feature contains 1416 unique values and also have some mixed values therefore separating them into different column would help in finding actual number of unique values.

*#Checking null values*

x='LOCALITY'

print(‘Null Values: ‘,)df[x].isnull().sum()

Null Values: 98

This feature has some null values and we will replace them with Other to give them a different importance.

*#Replacing null values with 'Other'*

x='LOCALITY'

df[x].replace(np.nan,'Other',inplace=**True**)

*#Replacing all unique values which occurs less than 2 times with 'Other' and storing it into another variable*

x = 'LOCALITY'

df[x]=df[x].apply(**lambda** y: y **if** y != 'Other' **and** u\_locality[y]>1 **else** 'Other')

unique\_locality = df[x].value\_counts()

unique\_locality.index

Index(['Other', 'Gachibowli', 'Indiranagar', 'Edappally', 'Kakkanad', 'HSR',

'Madhapur', 'Kukatpally', 'Marathahalli', 'Jubilee Hills',

...

'Muttukadu', '4 Bunglows', 'Bolgatty', 'Sector 55', 'Sahakar Nagar',

'Ejipura', 'Vazhakkala', 'Rajendra Place', 'Okhla Phase 2',

'Punjabi Bagh West'],

dtype='object', length=715)

Now, we have replaced the null values with Other and it is reduced to 715, moving further with splitting of feature.

*#Seperating unique values into different columns with Yes or No values*

x = 'LOCALITY'

**for** i **in** unique\_locality.index:

col = "in "+i

df[col] = df[x].apply(**lambda** y: 'Yes' **if** i **in** y **else** 'No')

After inspecting the feature LOCALITY, we have extracted the unique values from it and then separated them into different columns by assigning Yes (if feature is present) or No (if feature is not present) with corresponding restaurant.

1. **RATING:** Inspecting this feature to gain more information on it.

*#Checking for unique values*

x = 'RATING'

df[x].unique()

array(['3.6', '4.2', '3.8', '4.1', '4.0', '4.3', '3.9', '3.3', '3.4', '-',

'4.5', '3.5', '4.4', '2.7', '3.7', '4.7', 'NEW', '3.1', '2.5',

'4.6', '2.8', nan, '3.0', '3.2', '2.6', '2.9', '4.9', '4.8', '2.4',

'2.3', '2.0', '2.1', '2.2'], dtype=object)

After looking into the unique values we found that this feature contains numeric as well as alphabetic values, therefore, extraction as well as proper conversion is required.

*#Extracting rating and storing as float*

x = 'RATING'

rating=df[x].str.extract(r'([-+]?\d\*\.\d+|d+)').astype('float64')

*#Checking for null values and replacing them with mean value as the rating lies in a specific range.*

print("Null Values: ",rating[0].isnull().sum())

rating[0].replace(np.nan,rating[0].mean(),inplace=**True**)

*#Replacing feature RATING with rating[0]*

df[x] = rating[0]

print("Null Values after Replacement: ",df[x].isnull().sum())

Null Values: 1204

Null Values after Replacement: 0

As we can see that after extraction and conversion, 1204 null values were present, therefore, replaced the null value with mean of the rating to remove null values.

1. **VOTES:** Inspecting this feature to gain more information on it.

*#checking for unique values*

x = 'VOTES'

df[x].value\_counts()

44 votes 71

29 votes 66

28 votes 66

38 votes 65

35 votes 64

..

2310 votes 1

3765 votes 1

3048 votes 1

2795 votes 1

6508 votes 1

Name: VOTES, Length: 1847, dtype: int64

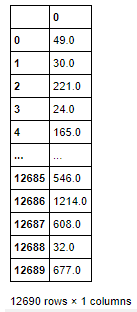
After looking into this feature, we found that there are 1847 unique values but they are stored in xxxx votes format, therefore, extracting the xxxx would give us more information on this feature.

*#Extracting VOTES and storing as float*

x = 'VOTES'

votes=df[x].str.extract(r'(\d+)').astype('float64')

votes



As we can see, after extraction of values, it is now in the form of continuous data type. Also there might be some null values present therefore replacing them with mean value as the votes lies in specific range.

*#Checking for null values and replacing them with mean value as the votes lies in a specific range.*

print("Null Values: ",votes[0].isnull().sum())

votes[0].replace(np.nan,votes[0].mean(),inplace=**True**)

*#Replacing feature VOTES with votes[0]*

df[x] = votes[0]

print("Null Values after Replacement: ",df[x].isnull().sum())

Null Values: 1204

Null Values after Replacement: 0

As we can see, there were 1204 null values present and has been replaced with mean of the value.

1. **COST:** Checking for null values as we already know this feature is of continuous type.

*#Checking for null values*

x = 'COST'

print("Null Values: ",df[x].isnull().sum())

Null Values: 0

As we can see, there are no null values present in this feature therefore moving further with dataset cleaning.

After inspecting all the features, we have extracted the relevant information from them and stored them into different columns. Now it’s time to drop features TITLE, RESTAURANT\_ID, CUISINES, TIME, CITY, LOCALITY to get the final shape of the dataset and also these were splitted into different columns and has no use further.

*#Dropping features*

drop\_feature = ['TITLE','RESTAURANT\_ID','CUISINES','TIME','CITY','LOCALITY']

df\_new = df.drop(columns=drop\_feature)

df\_new.head()



After dropping features, we can see that **the final dataset contains 1,081 features with 12,690 records**. Now moving further with Exploratory Data Analysis (EDA) and Visualisation to obtain more specific information.

# Exploratory Data Analysis (EDA) and Visualisation Concluding Remarks

As we know there are 1,081 features in the final dataset and it is difficult to visualize each of them, therefore, considering feature **RATING, VOTES and COST for EDA & Visualisation** as all other features are derived features and has only two values i.e. Yes or No.

**Importing visualisation libraries**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

## Univariate Analysis

Univariate Analysis means analysis of one variable or one feature at a time and it basically tells us how data in each feature is distributed and also tells us about central tendencies like mean, median, and mode as well as presence of outliers in the dataset.

**Data Distribution using distplot:** Distribution of data basically tells us about the mean, median, mode, maximum, minimum, standard deviation and skewness of data. We can get and visualize them using distribution plot of searborn library as shown below:

*#Checking data distribution in contineous feature*

cont\_feature = df\_new.columns[df\_new.dtypes != object]

cols = 2

rows = len(cont\_feature)//cols

**if** rows % cols != 0:

rows += 1

fig = plt.figure(figsize=(16,8))

plt.subplots\_adjust(hspace=0.8)

k=1

**for** i **in** cont\_feature:

axes = plt.subplot(rows,cols,k)

sns.distplot(df\_new[i],ax=axes)

axes.set\_title(f"Distribution Plot: **{**i**}**")

me = round(df[i].mean(),2)

mn = round(df[i].min(),2)

mx = round(df[i].max(),2)

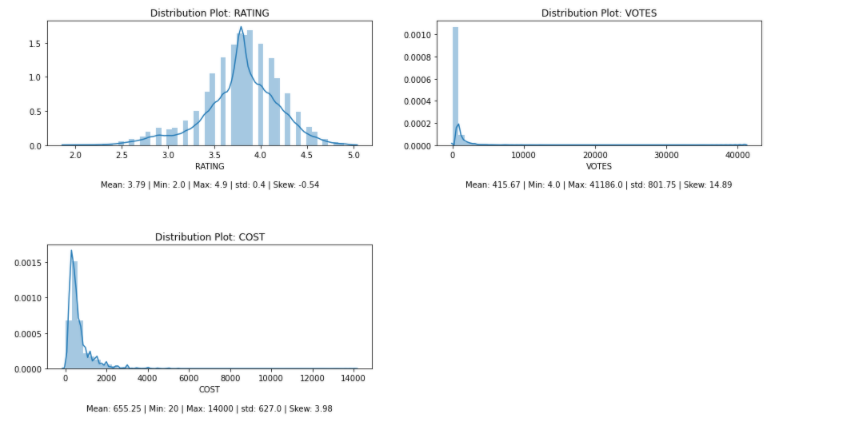
st = round(df[i].std(),2)

sk = round(df[i].skew(),2)

axes.set\_xlabel(f"**{**i**}\n\n**Mean: **{**me**}** | Min: **{**mn**}** | Max: **{**mx**}** | std: **{**st**}** | Skew: **{**sk**}**")

k += 1

plt.show()



After giving a look to the above plot, we clearly infer the following information for each of the feature as stated below:

**for feature RATING:**

* Data is somewhat distributed normally but not in bell curve.
* Average rating is: 3.79
* Minimum rating is: 2.0
* Maximum rating is: 4.9
* Standard deviation is lower (0.4) which indicates data is not spread.
* Skewness (-0.54) indicates, data is negatively skewed but is in negligible range.

**for feature VOTES:**

* Data is not distributed normally or not in bell curve.
* Average votes are: 415
* Minimum votes are: 4
* Maximum votes are: 41186
* Standard deviation is normal (801.75) which indicates data is not much spreaded.
* Skewness (14.89) indicates, data is right skewed and needs to be treated accordingly.

**for feature COST:**

* Data is somewhat distributed normally but not in bell curve.
* Average cost is: 655.25
* Minimum cost is: 20
* Maximum cost is: 14000
* Standard deviation is normal (627) which indicates data is not much spreaded.
* Skewness (3.98) indicates, data is positively skewed but is in negligible range.

**Box-Plot:** A box plot is a graphical rendition of statistical data based on the minimum, first quartile, median, third quartile, and maximum. It helps in identifying data distribution, skewness and presence of outliers in data. Now, plot the features using box-plot to get more information:

*#Checking contineous features with box-plot*

cont\_feature = df\_new.columns[df\_new.dtypes != object]

cols = 2

rows = len(cont\_feature)//cols

**if** rows % cols != 0:

rows += 1

fig = plt.figure(figsize=(16,8))

plt.subplots\_adjust(hspace=0.8)

k=1

**for** i **in** cont\_feature:

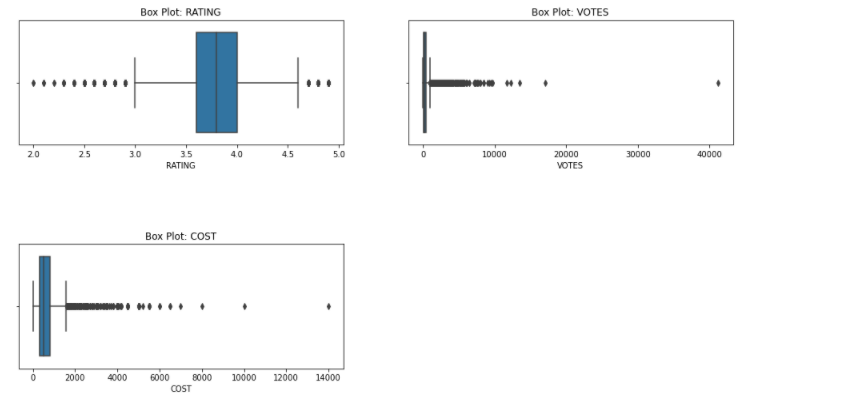
axes = plt.subplot(rows,cols,k)

sns.boxplot(df\_new[i],ax=axes)

axes.set\_title(f"Box Plot: **{**i**}**")

k += 1

plt.show()



From above plot, it clearly shows presence of outliers in all continuous features and needs to be treated accordingly. Now moving further with Bivariate Analysis to check the relationship between features.

## Bivariate Analysis

Bivariate analysis is one of the simplest forms of quantitative analysis. It involves the analysis of two variables (namely X and Y), for the purpose of determining the empirical relationship between them. Therefore, checking the relationship of feature RATING and VOTES with target variable COST to get more information on it.

*#Checking relationship between rating and cost.*

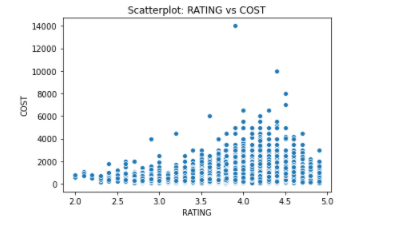
x = 'RATING'

y = 'COST'

sns.scatterplot(x,y,data=df\_new)

plt.title(f"Scatterplot: **{**x**}** vs **{**y**}**")

plt.show()



After looking into the graph we can clearly infer that As rating increase up to 4.5, cost also increases and then starts decreases towards rating up to 5.

*#Checking relationship between votes and cost.*

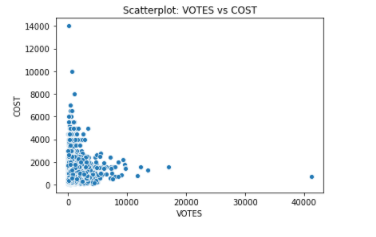
x = 'VOTES'

y = 'COST'

sns.scatterplot(x,y,data=df\_new)

plt.title(f"Scatterplot: **{**x**}** vs **{**y**}**")

plt.show()



Votes is an also interesting feature as from graph one can clearly infer that cost tends to decrease as the votes increases.

Now moving further with multi variate analysis to get the complete inference of data and their relationship with each other.

## Multi Variate Analysis

Multivariate analysis refers to any statistical technique used to analyse more complex sets of data.

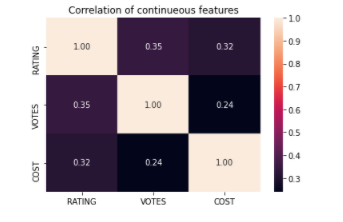
*#Checking correlation of contineous features and interpreting using heatmap*

df\_corr = df\_new.corr()

sns.heatmap(df\_corr,annot=**True**,fmt=".2f")

plt.title("Correlation of continueous features")

plt.show()



From above representation we can clearly say that all the continuous features have positively good correlation with each other. Hence, moving further with to check the correlation of features with target COST.

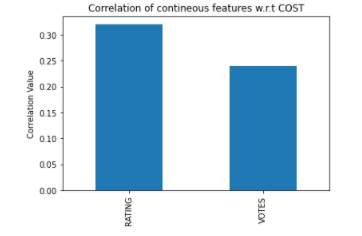
*#Checking correlation w.r.t cost using bar plot*

df\_corr['COST'].sort\_values(ascending=**False**).drop('COST').plot.bar()

plt.title("Correlation of contineous features w.r.t COST")

plt.ylabel("Correlation Value")

plt.show()



Correlation of bar-plot shows that feature RATING has highest correlation with COST as compared to VOTES.

# Prepare Dataset for Model Training

Preparing dataset Model training is the one of core part of machine learning model building and includes different types data modification and transformation to achieve the better model performance.

**Importing Libraries:**

**from** **sklearn.preprocessing** **import** OrdinalEncoder, StandardScaler, power\_transform

**from** **scipy.stats** **import** zscore

## Label Encoding

To deal with object types in the data, label encoding is used to transform these features into numerical form which can be provided to the machine learning models for training. Here, we are using OrdinalEncoder, which encodes categorical features as an integer array, to transform these features into numerical form. The features, except RATING, VOTES and COST, will be encoded using OrdinalEncoder as they are of object type in the dataset.

*#Encoding all categorical features using OrdinalEncoder*

cat\_feature = df\_new.columns[df\_new.dtypes == object]

df\_enc = df\_new.copy()

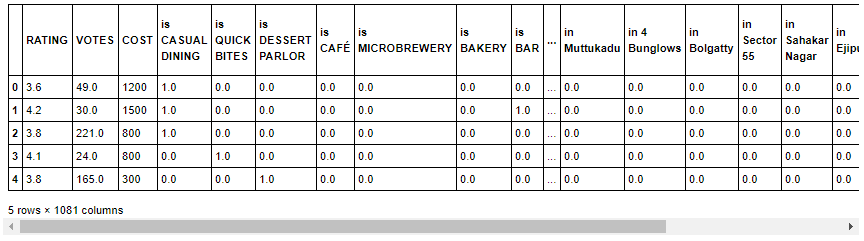
ec = OrdinalEncoder()

**for** i **in** cat\_feature:

df\_enc[i] = ec.fit\_transform(np.array(df\_new[i]).reshape(-1,1))

*#Interpreting dataset after encoding*

df\_enc.head()



As we can see, after applying OrdinalEncoder, all the categorical features has been transformed to continuous type with values 0 and 1.

## Outlier(s): Detection & Removal

Outliers are extreme values that fall a long way outside of the other observations. It can be detected and removed using either Z-Score or Interquartile Range (IQR) methods. Here we are going to use z-score for this purpose.

*#Checking and removing outliers using zscore (threshold value -3 to +3)*

z = np.abs(zscore(df\_enc))

*#printing location of outliers*

np.where(z>3)

(array([ 0, 0, 0, ..., 12688, 12688, 12689], dtype=int64),

array([ 26, 94, 542, ..., 72, 674, 610], dtype=int64))

*#Removing outliers*

df\_wo = df\_enc[(z<=3).all(axis=1)]

print(f"Original Shape: **{**df\_enc.shape**}**")

print(f"New Shape: **{**df\_wo.shape**}**")

print(f"% Loss: **{**(len(df\_enc)-len(df\_wo))\*100/len(df\_enc)**}**%")

Original Shape: (12690, 1081)

New Shape: (0, 1081)

% Loss: 100.0%

From above, we can see that, with this method of outlier removal, all of the records has been removed therefore discarding the outlier removal and proceeding with original dataset.

## Separate Input and Output/Target Variable

Now, we can separate the features into input as X and output/target as Y to continue further with data preparation.

X = df\_enc.drop(columns=['COST']) *#Input variables*

Y = df\_enc['COST'] *#Output/Target Variables*

print(X.shape)

print(Y.shape)

(12690, 1080)

(12690,)

After separating input and output variable we can see that the variable X has 12690 records with 1080 features while variable Y has 12690 records with a single target feature.

## Skewness: Detection & Treatment

*#Checking and treating skewness in contineous features (optimum value -0.5 to +0.5)*

cont\_f = ['RATING','VOTES']

*#Skewness*

X[cont\_f].skew()

RATING -0.537352

VOTES 14.890494

dtype: float64

**for** i **in** cont\_f:

**if** np.abs(X[i].skew()) > 0.5:

X[i] = power\_transform(np.array(X[i]).reshape(-1,1))

*#Re-Checking after treatment*

X[cont\_f].skew()

RATING 0.028430

VOTES -0.007247

dtype: float64

From the above, we can see that, the feature RATING & VOTES has highly skewed data and has been treated using power\_transform to reduce the skewness which is needed for model training.

## Scale Data for Model Training

Scaling data for model training is refers to normalize the range of independent variables or features of data. Here we are using StandardScaler for this purpose:

*#scaling contineous features*

sc = StandardScaler()

**for** i **in** cont\_f:

X[i] = sc.fit\_transform(np.array(X[i]).reshape(-1,1))

# Model Training: Finding the best model

The models that I have decided to train for this dataset are LinearRegression and Lasso models. The goal here is to find the best hyper-tuned models for further processing.

**Importing Libraries:**

**from** **sklearn.model\_selection** **import** train\_test\_split, GridSearchCV, cross\_val\_score

**from** **sklearn.metrics** **import** r2\_score, mean\_squared\_error, mean\_absolute\_error

## Define the function(s) for Automation of Model Training and Testing

**import** **timeit**

*#Defining function for best random state*

**def** get\_best\_rstate(r,model,x,y,test\_size=0.25):

best\_rState = 0

best\_mScore = 0

**for** i **in** r:

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=test\_size,random\_state=i)

model.fit(x\_train,y\_train)

predict\_y = model.predict(x\_test)

temp\_r2Score = r2\_score(y\_test,predict\_y)

**if** temp\_r2Score>best\_mScore:

best\_mScore = temp\_r2Score

best\_rState = i

**return** best\_rState,best\_mScore

*#Defining function for best CV*

**def** get\_best\_cv(model,parameters,x\_train,y\_train,r=range(2,20)):

best\_cv = 0

best\_cvScore = 0

**for** i **in** r:

gscv = GridSearchCV(model,parameters)

gscv.fit(x\_train,y\_train)

temp\_cvScore = cross\_val\_score(gscv.best\_estimator\_,x\_train,y\_train,cv=i).mean()

**if** i == 2:

best\_cvScore = temp\_cvScore

best\_cv = i

**if** temp\_cvScore>best\_cvScore:

best\_cvScore = temp\_cvScore

best\_cv = i

**return** best\_cv,best\_cvScore

*#Defining function for building models*

**def** build\_model(models,x,y,r\_range=range(100),t\_size=0.25,cv\_range=range(2,20)):

**for** i **in** models:

print(f"Processing **{**i**}**...")

*#Start time*

start\_time = timeit.default\_timer()

*#Finding the best random\_state for train test split*

best\_rState, best\_mScore = get\_best\_rstate(r\_range,models[i]["name"],x,y)

*#Splitting train test data with best random\_state*

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=t\_size,random\_state=best\_rState)

*#Hypertuning Parameters*

*#Finding best CV*

best\_cv, best\_cvScore = get\_best\_cv(models[i]["name"],models[i]["parameters"],x\_train,y\_train,cv\_range)

*#Building final model with hypertuned parameters*

gscv = GridSearchCV(models[i]["name"],models[i]["parameters"],cv=best\_cv)

gscv.fit(x\_train,y\_train)

*#End time*

end\_time = timeit.default\_timer()

*#Checking Final Performance of the model*

predict\_y = gscv.best\_estimator\_.predict(x\_test)

r2Score = r2\_score(y\_test,predict\_y)

mse = mean\_squared\_error(y\_test,predict\_y)

mae = mean\_absolute\_error(y\_test,predict\_y)

*#Storing metrics*

models[i]['initial\_r2\_score'] = best\_mScore

models[i]['r2\_score'] = r2Score

models[i]['mse'] = mse

models[i]['mae'] = mae

models[i]['rmse'] = np.sqrt(mse)

*#Storing model specs.*

models[i]["random\_state"] = best\_rState

models[i]["x\_train"] = x\_train

models[i]["x\_test"] = x\_test

models[i]["y\_train"] = y\_train

models[i]["y\_test"] = y\_test

models[i]["cv"] = best\_cv

models[i]["cross\_val\_score"] = best\_cvScore

models[i]["gscv"] = gscv

models[i]["predict\_y"] = predict\_y

models[i]["build\_time"] = end\_time - start\_time

print(f"**\t**Completed: [in **{**end\_time-start\_time**}**s]**\n**")

**return** models.copy();

*#Function Displaying model performance and comparing it to select the best model.*

**def** model\_performance(build\_models):

model\_names = []

model\_r2Scores = []

model\_cvScores = []

model\_buildTimes = []

**for** i **in** build\_models:

model = build\_models[i]

print(f"START: **{**i**}\n**")

print(f"**\t**Best random\_state: **{**model['random\_state']**}** with best r2\_score: **{**model['initial\_r2\_score']**}\n**")

print(f"**\t**Best CV: **{**model['cv']**}** with best cross\_val\_score: **{**model['cross\_val\_score']**}\n**")

print(f"**\t**Best Parameters: **{**model['gscv'].best\_params\_**}\n\n**")

print(f"----Final Performance----")

print(f"R2 Score: **{**round(model['r2\_score']\*100,2)**}**%**\n**")

print(f"MSE: **{**model['mse']**}\n**")

print(f"RMSE:**{**model['rmse']**}\n**")

print(f"MAE: **{**model['mae']**}\n**")

print(f"BuildTime: **{**model['build\_time']**}\n**")

print(f"END: **{**i**}\n\n\n**")

model\_names.append(i)

model\_r2Scores.append(model['r2\_score'])

model\_cvScores.append(model['cross\_val\_score'])

model\_buildTimes.append(model['build\_time'])

d\_cmp = pd.DataFrame({"Name":model\_names,"r2\_score":model\_r2Scores,"cross\_val\_score":model\_cvScores,"build\_time(sec)":model\_buildTimes})

d\_cmp['Difference'] = d\_cmp['r2\_score']-d\_cmp['cross\_val\_score']

print(d\_cmp)

## Prepare Model List and Test to get Best Model

This part includes the preparation of model list with parameters and then passed them to defined functions for getting best parameter and hyper-tuned model performances.

**import** **warnings**

warnings.simplefilter('ignore')

**from** **sklearn.linear\_model** **import** LinearRegression, Lasso

*#Preparing List of Models with parameters*

models = {

"LinearRegression":{

"name": LinearRegression(),

"parameters":{

"fit\_intercept": [**True**,**False**],

"normalize": [**True**,**False**]

}

},

}

*#Building and Testing models*

build\_models = build\_model(models,X,Y)

Processing LinearRegression...

Completed: [in 4269.2861654s]

*#Displaying model performance*

model\_performance(build\_models)

START: LinearRegression

Best random\_state: 5 with best r2\_score: 0.732308057697959

Best CV: 17 with best cross\_val\_score: -3.4166731841123533e+19

Best Parameters: {'fit\_intercept': False, 'normalize': True}

----Final Performance----

R2 Score: 72.96%

MSE: 102645.63390161823

RMSE:320.38357308329375

MAE: 202.08756511112045

BuildTime: 4269.2861654

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.729639 -3.416673e+19 4269.286165 3.416673e+19

We have trained and tested here LinearRegression model which takes approx**. 4269 seconds** to complete the training and testing process. It gives the **R2 Score: 72.96% with RMSE: 320.38.**

Since, this dataset contains 1080 training features which takes lots of time to train a model, therefore, we are moving to Principal Component Analysis to reduce the curse of dimensionality and check the model performances with different number of n\_components to achieve the optimum model score.

## Using PCA (Principle Component Analysis)

**from** **sklearn.decomposition** **import** PCA

*#Decomposing Input Variable X with different n\_components and testing model for best performance*

n = [10,50,100,150,200]

**for** i **in** n:

pca = PCA(n\_components=i)

pca\_X = pca.fit\_transform(X)

build\_pca = build\_model(models,pca\_X,Y)

print(f"For n\_components: **{**i**}**======")

model\_performance(build\_pca)

Processing LinearRegression...

Completed: [in 10.011553600000298s]

For n\_components: 10======

START: LinearRegression

Best random\_state: 0 with best r2\_score: 0.3306018471234975

Best CV: 18 with best cross\_val\_score: 0.29106731204130765

Best Parameters: {'fit\_intercept': True, 'normalize': False}

----Final Performance----

R2 Score: 33.06%

MSE: 232370.328896443

RMSE:482.0480566255226

MAE: 284.07733106540513

BuildTime: 10.011553600000298

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.330602 0.291067 10.011554 0.039535

Processing LinearRegression...

Completed: [in 37.53752160000022s]

For n\_components: 50======

START: LinearRegression

Best random\_state: 42 with best r2\_score: 0.7041918109431684

Best CV: 14 with best cross\_val\_score: 0.6619850694706325

Best Parameters: {'fit\_intercept': True, 'normalize': True}

----Final Performance----

R2 Score: 70.42%

MSE: 104584.92390948448

RMSE:323.395924386014

MAE: 207.20342243856118

BuildTime: 37.53752160000022

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.704192 0.661985 37.537522 0.042207

Processing LinearRegression...

Completed: [in 89.46722809999937s]

For n\_components: 100======

START: LinearRegression

Best random\_state: 59 with best r2\_score: 0.7321118661610824

Best CV: 18 with best cross\_val\_score: 0.6772494272348752

Best Parameters: {'fit\_intercept': True, 'normalize': False}

----Final Performance----

R2 Score: 73.21%

MSE: 92649.96038316516

RMSE:304.38456002755

MAE: 193.1555890748179

BuildTime: 89.46722809999937

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.732112 0.677249 89.467228 0.054862

Processing LinearRegression...

Completed: [in 162.98078270000042s]

For n\_components: 150======

START: LinearRegression

Best random\_state: 59 with best r2\_score: 0.7356603426897884

Best CV: 18 with best cross\_val\_score: 0.6821127123280722

Best Parameters: {'fit\_intercept': True, 'normalize': False}

----Final Performance----

R2 Score: 73.57%

MSE: 91422.70852585482

RMSE:302.3618833878616

MAE: 192.24991035448735

BuildTime: 162.98078270000042

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.73566 0.682113 162.980783 0.053548

Processing LinearRegression...

Completed: [in 228.24133789999996s]

For n\_components: 200======

START: LinearRegression

Best random\_state: 59 with best r2\_score: 0.7345639589687407

Best CV: 18 with best cross\_val\_score: 0.6902169300424117

Best Parameters: {'fit\_intercept': True, 'normalize': False}

----Final Performance----

R2 Score: 73.46%

MSE: 91801.89631168226

RMSE:302.9882775152898

MAE: 194.190459038754

BuildTime: 228.24133789999996

END: LinearRegression

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.734564 0.690217 228.241338 0.044347

From above results, we can infer the following:

* **Model score with decomposition is optimum at *n\_components=100* with r2\_score: 73.21% which almost equivalent to model score with original input.**
* **Continuing with PCA(n\_components=100)**

## Building Final Model List with PCA(n\_components=100)

pca = PCA(n\_components=100)

pca\_X = pca.fit\_transform(X)

*#Model lists*

models = {

"LinearRegression":{

"name": LinearRegression(),

"parameters":{

"fit\_intercept": [**True**],

"normalize": [**True**]

}

},

"Lasso":{

"name": Lasso(),

"parameters":{

"alpha": [0.0001,0.001],

"fit\_intercept": [**True**],

"selection": ['cyclic','random']

}

}

}

*#Building and Testing Final model using PCA*

final\_build = build\_model(models,pca\_X,Y)

Processing LinearRegression...

Completed: [in 52.72074889999931s]

Processing Lasso...

Completed: [in 66.06136200000037s]

*#Displaying model performances*

model\_performance(final\_build)

START: LinearRegression

Best random\_state: 59 with best r2\_score: 0.7324243543594329

Best CV: 18 with best cross\_val\_score: 0.6770463182088224

Best Parameters: {'fit\_intercept': True, 'normalize': True}

----Final Performance----

R2 Score: 73.24%

MSE: 92541.8853490735

RMSE:304.2069778112815

MAE: 192.92895593949933

BuildTime: 52.72074889999931

END: LinearRegression

START: Lasso

Best random\_state: 59 with best r2\_score: 0.7188173645598405

Best CV: 18 with best cross\_val\_score: 0.677050342573987

Best Parameters: {'alpha': 0.001, 'fit\_intercept': True, 'selection': 'random'}

----Final Performance----

R2 Score: 73.24%

MSE: 92542.43636509177

RMSE:304.20788346966253

MAE: 192.93235261809804

BuildTime: 66.06136200000037

END: Lasso

Name r2\_score cross\_val\_score build\_time(sec) Difference

0 LinearRegression 0.732424 0.677046 52.720749 0.055378

1 Lasso 0.732423 0.677050 66.061362 0.055372

From above tested models with PCA, we concludes that **Lasso performs better with r2\_score: 73.24% and cross\_val\_score: 67.71%. Therefore, proceeding with Lasso Model.**

# Model Selection: The Final Model

In this step, we will save or serialize the final model which gives the highest performance into an object or pickle file.

**Importing Libraries:**

**import** **joblib**

*#Saving final model*

final\_model = final\_build['Lasso']

filename = 'restaurant\_food\_cost.obj'

joblib.dump(final\_model['gscv'].best\_estimator\_,open(filename,'wb'))

## Conclusion

The final model performance is good with **r2\_score: 73.24% and cross\_val\_score: 67.71%** and can be improved further by training with more specific data.

This can be depicted using scatterplot as shown below:

*#Ploting Original and Predicted data on scatterplot*

original = final\_model['y\_test']

predict = final\_model['predict\_y']

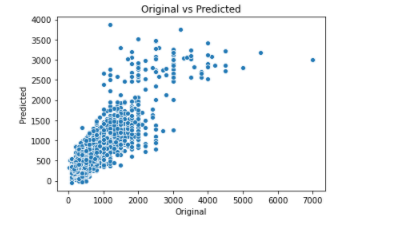
sns.scatterplot(original,predict)

plt.title("Original vs Predicted")

plt.xlabel("Original")

plt.ylabel("Predicted")

plt.show()



We can see that the original and predicted values are directly proptional with certain difference which is a good sign and also it scattered at some points which needs to be improve to get the better model.

# Model Prediction

In this phase we will supply the set of test data to the serialized model to get the predicted value by using following steps:

## Pre-Processing Pipeline

Pipelines are a simple way to keep data pre-processing and modelling code organized. Specifically, a pipeline bundles pre-processing and modelling steps so that one can use the whole bundle as if it were a single step. This process includes the following steps which has been already discussed above:

* 1. Load Test Data
  2. Extract Information from features TITLE, CUISINES, CITY, LOCALITY, RATING and VOTES
  3. Treat Null Values
  4. Encode Discrete Features (using OrdinalEncoder)
  5. Remove Outliers using zscore
  6. Treat skewness in contineous features using power\_transform
  7. Scale contineous feature data using StandardScaler
  8. Apply PCA(n\_components=100)
  9. Load Serialized Model and Make Prediction for Test Data

## Load Model from Serialized Object/Pickle file

model\_file\_name = 'restaurant\_food\_cost.obj'

loaded\_model = joblib.load(model\_file\_name)

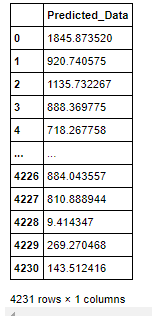
*#Predicting Test Data*

predict\_test = loaded\_model.predict(pca\_test\_x)

*#Displaying Predicted Data*

pred\_t = pd.DataFrame({"Predicted\_Data": predict\_test})

pred\_t



## Save Predicted Values

In this step, we will save the predicted value to a .csv (Comma Separated Values) file.

*## Saving predicted data to .csv file*

pred\_t.to\_csv('predicted\_test\_data.csv')

# Concluding Remarks

During the process of data analysis, I came across feature TIME which is quite complex in nature and I dropped it from dataset but somewhere I am thinking that diving into this feature would give some more useful aspects of the dataset which might play a key role in improving model performance. But for now I am resting this article here and will come up with some more dataset problems in future.