Predicting Restaurant Food Cost Using Machine Learning

By Darshik A S

What is your favorite meal? Do you know restaurants that serve your favorite food the way you want it to be? Even if you know, sometimes you might have to reconsider it or get your mind off your food cravings. It's because of one key factor that is the cost. In this blog, we will hop into an end to end machine learning project on predicting the average cost of food in a restaurant in India. You can view the complete Python code/Notebook here.

1. Problem Statement:

The idea of doing this project is to investigate the factors that affect the price of a meal. For the study, we will use the Restaurant Food Cost dataset (It's a dataset provided by MachineHack in a Hackathon) and find answers to the questions as mentioned:

- What are the top 10 expensive cuisines?
- What type of restaurants are people mostly looking for?
- Does the location of the restaurant affect the cost of food?
- What cuisines are commonly available in each city?
- What type of cuisines are mostly available in India?
- How are the restaurants distributed in each city?
- What hours the restaurant operates?

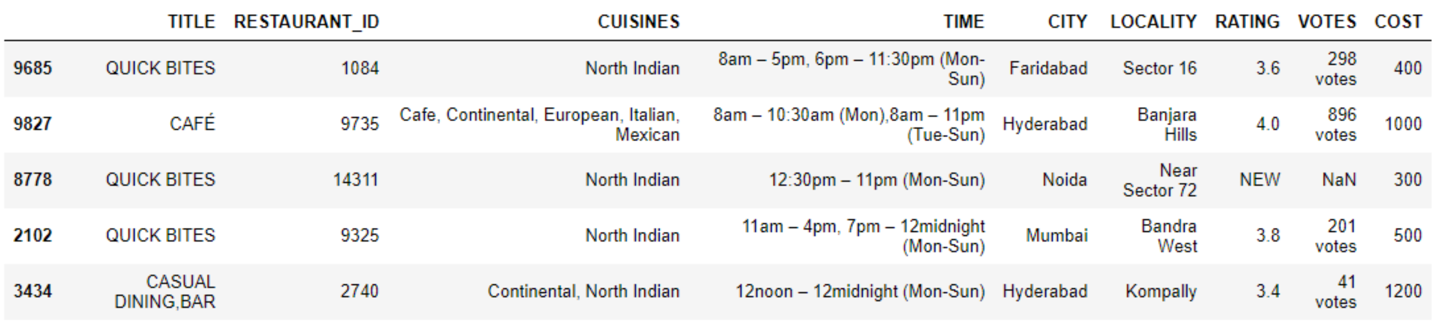
So these are some questions I had in my mind before starting the analysis. Throughout the study, we will bring to light more insights from the data. And later, we will build predictive models for determining the cost.

2. Data Preparation:

First, let's take a glimpse into the dataset. We will use the Pandas data frame to load data and display some samples.

```
import pandas as pd
train_data = pd.read_excel('Data_Train.xlsx')
train_data.sample(5)
```

Output:



	TITLE	RESTAURANT_ID	CUISINES	TIME	CITY	LOCALITY	RATING	VOTES	COST
9685	QUICK BITES	1084	North Indian	8am – 5pm, 6pm – 11:30pm (Mon-Sun)	Faridabad	Sector 16	3.6	298 votes	400
9827	CAFÉ	9735	Cafe, Continental, European, Italian, Mexican	8am – 10:30am (Mon),8am – 11pm (Tue-Sun)	Hyderabad	Banjara Hills	4.0	896 votes	1000
8778	QUICK BITES	14311	North Indian	12:30pm – 11pm (Mon-Sun)	Noida	Near Sector 72	NEW	NaN	300
2102	QUICK BITES	9325	North Indian	11am – 4pm, 7pm – 12midnight (Mon-Sun)	Mumbai	Bandra West	3.8	201 votes	500
3434	CASUAL DINING,BAR	2740	Continental, North Indian	12noon – 12midnight (Mon-Sun)	Hyderabad	Kompally	3.4	41 votes	1200

What does each column mean?

- ***TITLE*** : The feature of the restaurant which can help identify what and for whom it is suitable for.
- ***RESTAURANT_ID*** : A unique ID for each restaurant.
- ***CUISINES*** : The variety of cuisines that the restaurant offers.
- ***TIME*** : The open hours of the restaurant.
- ***CITY*** : The city in which the restaurant is located.
- ***LOCALITY*** : The locality of the restaurant.
- ***RATING*** : The average rating of the restaurant by customers.
- ***VOTES*** : The overall votes received by the restaurant.
- ***COST*** : The average cost of a two-person meal.

So there are 9 feature variables in the dataset. Now we check the type of each variable and see if there are any null values present.

```
train_data.info()
```

Output:

```
RangeIndex: 12690 entries, 0 to 12689
Data columns (total of 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
```

Here we have *RESTAURANT_ID* and *COST* (Target variable) of type integer, and the rest of the variables are of the object type. We can see that the CITY, *LOCALITY*, *RATING,* and *VOTES* have some null values. Now let's remove the duplicates (If exists) and check the proportion of null values present in the dataset and see if we can remove those samples.

```
# Removing duplicates from the dataset
train_data.drop_duplicates(subset=['RESTAURANT_ID','CITY','TITLE','CUISINE
S','LOCALITY'],keep='first',inplace=True)

# Check proportion of null values
data_length = len(train_data)
print('Percentage of null values in CITY :
{:,.2f}%'.format(train_data.CITY.isnull().sum() * 100 / data_length))
print('Percentage of null values in LOCALITY :
{:,.2f}%'.format(train_data.LOCALITY.isnull().sum() * 100 / data_length))
print('Percentage of null values in RATING :
{:,.2f}%'.format(train_data.RATING.isnull().sum() * 100 / data_length))
print('Percentage of null values in VOTES :
{:,.2f}%'.format(train_data.VOTES.isnull().sum() * 100 / data_length))
```

Output:

```
Percentage of null values in CITY: 0.88%
Percentage of null values in LOCALITY: 0.77%
Percentage of null values in RATING: 0.02%
Percentage of null values in VOTES: 9.48%
```

There is a hefty amount of null values in the variables. Removing these will lead to losing information. So we will fill those null values with proper values, which we will do in further analysis.

Feature Extraction:

Looking at the samples, we see that the variables comprise many unnecessary data. Now we will perform feature extraction and generate suitable information out of it.

- Feature extraction on ***TITLE*:**

```
train_data.TITLE.sample(5)
```

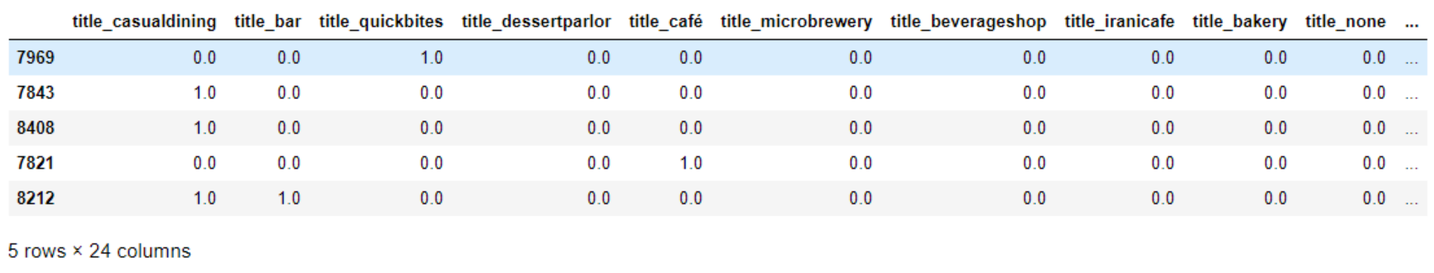
`Output:`

```
10333                          None
10245                        BAKERY
11972    SWEET SHOP, QUICK BITES
1551                     SWEET SHOP
10204                          CAFÉ
Name: TITLE, type: object
```

Sample comprises multiple titles. So it's better to denote each title by creating dummy variables.

```
# Encoding the features from the TITLE column of the train data
train_title = train_data.TITLE.agg(lambda x:
pd.Series(1,str(x).strip().lower().replace(' ','').split(','))).fillna(0)
train_title.columns = ['title_'+i for i in train_title.columns]
train_title.reset_index(drop=True,inplace=True)
train_title.sample(5)
```

`Output:`



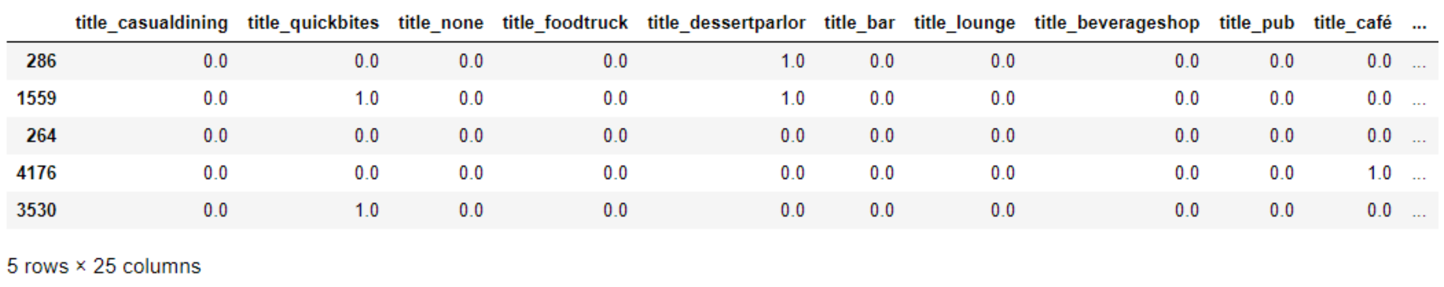
	title_casualdining	title_bar	title_quickbites	title_dessertparlor	title_café	title_microbrewery	title_beverageshop	title_iranicafe	title_bakery	title_none	...
7969	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
7843	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
8408	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
7821	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
8212	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 24 columns

```
# Encoding the features from the TITLE column of the test data
test_title = test_data.TITLE.agg(lambda x:
pd.Series(1,str(x).strip().lower().replace(' ','').split(','))).fillna(0)
```

```
test_title.columns = ['title_'+i for i in test_title.columns]
test_title.reset_index(drop=True,inplace=True)
test_title.sample(5)
```

Output:



	title_casualdining	title_quickbites	title_none	title_foodtruck	title_dessertparlor	title_bar	title_lounge	title_beverageshop	title_pub	title_café	...
286	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
1559	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
264	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
4176	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...
3530	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

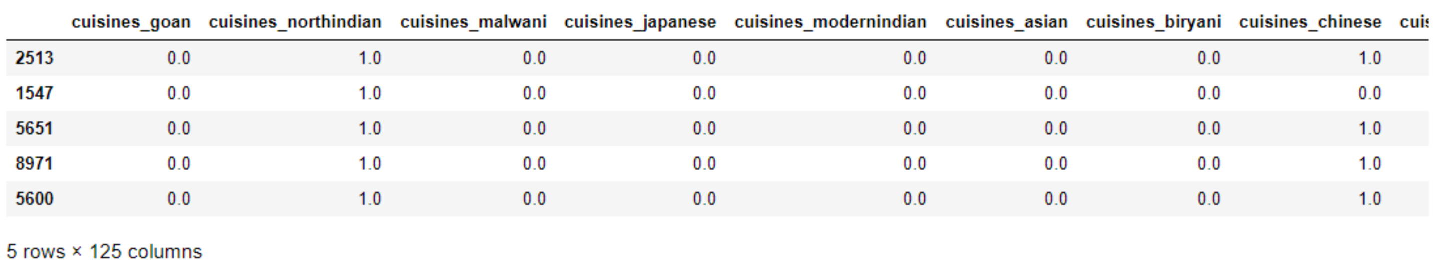
5 rows × 25 columns

There are 24 distinct titles in the training dataset and 25 titles in the test dataset. When we look at the names of the titles, test data contains an extra title named *Bhojanalya*. We can group this title with the *none* title.

- Feature extraction on ***CUISINES***:

Just like we did the feature extraction on TITLE, we can also do the same for CUISINES. The encoded features data frame will look like this:

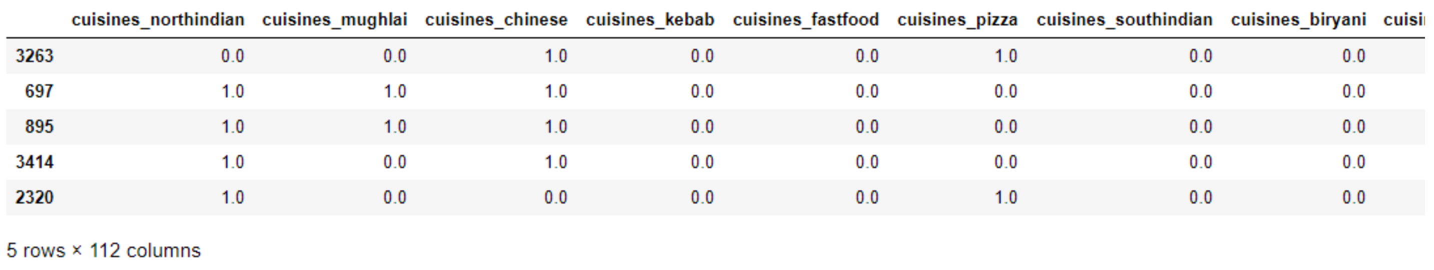
  - Encoded cuisines from the training dataset:



	cuisines_goan	cuisines_northindian	cuisines_malwani	cuisines_japanese	cuisines_modernindian	cuisines_asian	cuisines_biryani	cuisines_chinese	cuis
2513	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
1547	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
5651	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
8971	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	
5600	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	

5 rows × 125 columns

  - Encoded cuisines from the test dataset:



	cuisines_northindian	cuisines_mughlai	cuisines_chinese	cuisines_kebab	cuisines_fastfood	cuisines_pizza	cuisines_southindian	cuisines_biryani	cuisi
3263	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	
697	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
895	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	
3414	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2320	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	

5 rows × 112 columns

Here we see that the number of encoded cuisines from the train and test dataset are not matching. We can ignore cuisines from the test dataset which are not in the training dataset and add columns that are not in the test dataset but exist in the training dataset.

- Feature extraction on ***TIME:***

```
train_data.TIME.sample(10)
```

Output:

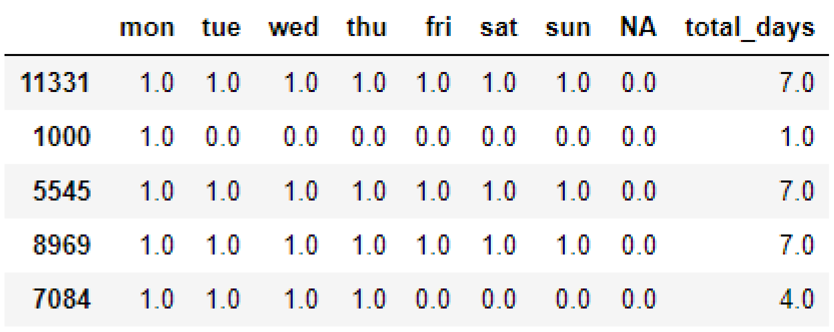
```
6989                                  11am – 11pm (Mon-Sun)
2265                               10:30am – 11pm (Mon-Sun)
10366     7:30am – 10:30am, 1pm – 3:30pm, 7:30pm – 10pm...
9940                             12noon – 12midnight (Mon-Sun)
7797     12noon – 3:30pm, 7pm – 10:45pm (Mon-Sat),9am –...
12371                               12noon – 11:30pm (Mon-Sun)
9907       8:30am – 11:30pm (Mon-Sat),9am – 11:30pm (Sun)
3547                                    12noon – 1am (Mon-Sun)
6962                        8am – 11pm (Mon-Sat),Closed (Sun)
4230                             12noon – 12midnight (Mon-Sun)
Name: TIME, dtype: object
```

When we look into the samples of the TIME variable, there is a lot of information about working hours, days on which the restaurant opens and closes. In some samples, time and days are in ranges and in some information is not available. We will use regular expressions to extract time and days.

```
# (Train data) Extract the days and ranges of days from the TIME feature
using regex and encode it
df_time = train_data.TIME.agg(lambda x:
pd.Series(1,(set(re.findall('([NA]{2}|[c][l][o][s][e][d]\s\(\w+\)|[c][l][o
][s][e][d]|\(\w+?\W\w*\)|[a-z]{3})',str(x).lower().replace('noon','pm').re
place('midnight','am').replace('24 hours','24').replace('not
available','NA')))))).fillna(0)
df_time.reset_index(drop=True,inplace=True)
train_days_open =
pd.DataFrame(np.zeros((len(df_time),8)),columns=['mon','tue','wed','thu','
fri','sat','sun','NA'])
for i in list(df_time.columns):
    l = re.findall('[NA]{2}|[closed]{5}|[a-z]{3}',i)
    if len(l) > 1 and l[0] != 'close':
        train_days_open.loc[df_time[i]==1,l[0]:l[1]] = 1
    elif len(l) > 1 and l[0] == 'close':
        train_days_open.loc[df_time[i]==1,'mon'] = 1
        train_days_open.loc[df_time[i]==1,'tue'] = 1
        train_days_open.loc[df_time[i]==1,'wed'] = 1
        train_days_open.loc[df_time[i]==1,'thu'] = 1
        train_days_open.loc[df_time[i]==1,'fri'] = 1
        train_days_open.loc[df_time[i]==1,'sat'] = 1
        train_days_open.loc[df_time[i]==1,'sun'] = 1
        train_days_open.loc[df_time[i]==1,l[1]] = 0
    elif len(l) == 1 and l[0] not in ['close','NA']:
        train_days_open.loc[df_time[i]==1,l[0]] = 1
    elif len(l) == 1 and l[0] == 'NA':
        train_days_open.loc[df_time[i]==1,'NA'] = 1

# Also calculate the number of days the restaurant will be opened and
concatenate it as an extra feature
train_days_open['total_days'] =
train_days_open.drop(columns=['NA']).sum(axis=1).rename('total_days')
train_days_open.sample(5)
```

Output:



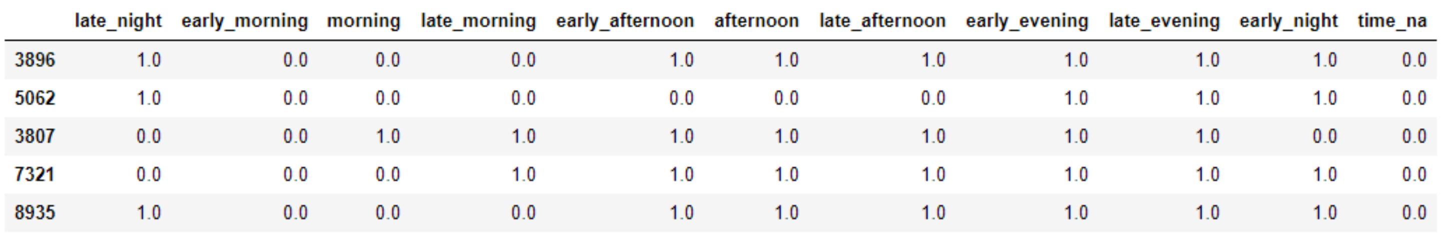
	mon	tue	wed	thu	fri	sat	sun	NA	total_days
11331	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
1000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
5545	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
8969	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	7.0
7084	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	4.0

The regular expression extracts days of the form such as Closed (Mon) or (Mon-Sun) or Wed, Thu, etc. And transforms these into encoded variables. The positive value in each day of a sample denotes that the restaurant will open. There is an extra NA feature column, which is to denote the information that is not available. We can add another feature which is the total days the restaurant opens. It is the sum of positive values of each day of a sample.

Now we will extract the hours of the day the restaurant opens. But there could be a lot of features when creating dummies of individual times. Instead, we will convert these times into part of the day.

Part of the Day	Time
Early Morning	5am - 8am
Morning	8am - 11am
Late Morning	11am - 12pm
Early Afternoon	12pm - 2pm
Afternoon	2pm - 4pm
Late Afternoon	4pm - 5pm
Early Evening	5pm - 7pm
Late Evening	7pm - 9pm
Early Night	9pm - 12am
Late Night	12am - 5am

The output will be:



	late_night	early_morning	morning	late_morning	early_afternoon	afternoon	late_afternoon	early_evening	late_evening	early_night	time_na
3896	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
5062	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0
3807	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0
7321	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0
8935	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Here *time_na* denotes the samples with no times provided.

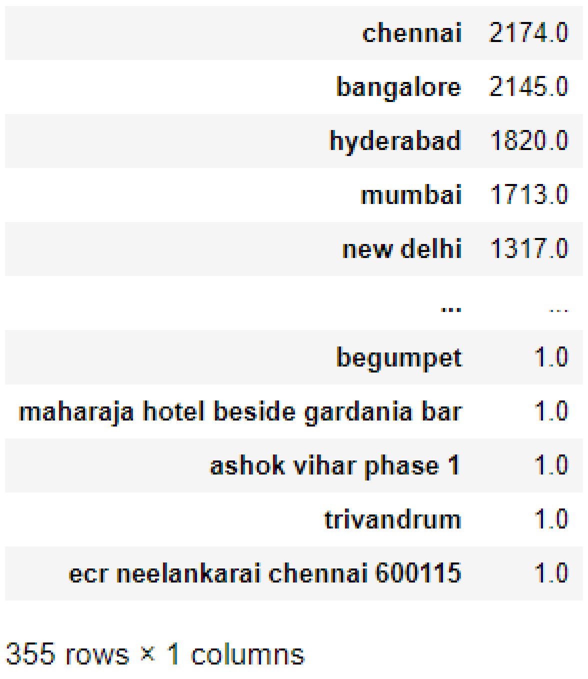
- Feature extraction on ***CITY*** and ***LOCALITY***:

First, we will fill the null values in the CITY variable with 'Unknown'. Then we will find out the most occurring words in the variable.

```
# Find out the most occurring cities from the CITY feature
train_data.CITY.agg(lambda x: pd.Series(1,(set(x.strip().lower().replace('
',' ').split(','))))).fillna(0).sum().sort_values(ascending=False).head(25)
```

```
.to_frame()
```

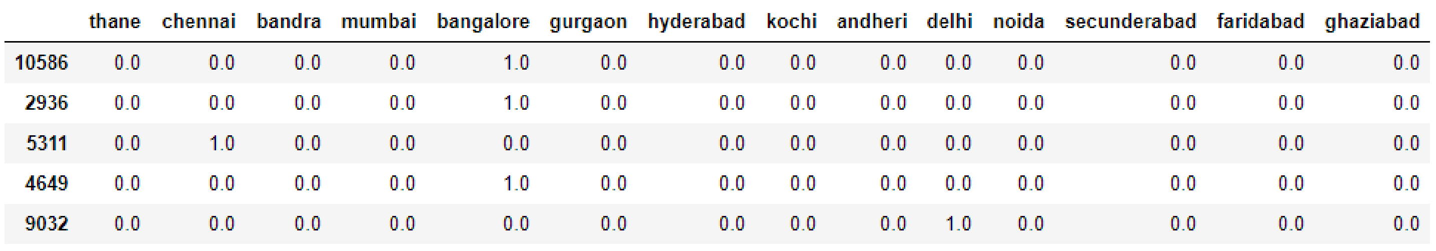
Output:



chennai	2174.0
bangalore	2145.0
hyderabad	1820.0
mumbai	1713.0
new delhi	1317.0
...	...
begumpet	1.0
maharaja hotel beside gardania bar	1.0
ashok vihar phase 1	1.0
trivandrum	1.0
ecr neelankarai chennai 600115	1.0

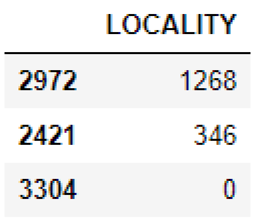
355 rows × 1 columns

Most frequently occurring cities are *Bangalore*, *Thane*, *Bandra*, *Hyderabad*, *Andheri*, *Delhi*, *Chennai*, *Mumbai*, *Kochi*, *Noida*, *Gurgaon*, *Faridabad*, *Ghaziabad,* and *Secunderabad*. We'll try matching these cities with each sample in CITY and LOCALITY using fuzzy-wuzzy. Samples which are not matching with the most occurring cities are put into *others variable*.



	thane	chennai	bandra	mumbai	bangalore	gurgaon	hyderabad	kochi	andheri	delhi	noida	secunderabad	faridabad	ghaziabad
10586	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2936	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5311	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4649	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9032	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

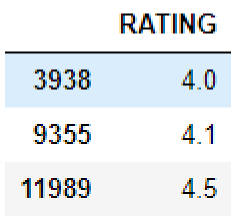
In the LOCALITY variable, remove stop words and digits and label encode the samples.



	LOCALITY
2972	1268
2421	346
3304	0

- Feature extraction on ***RATING***:

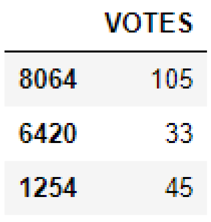
When we look at the unique values in RATING, we can see there are some missing values of the form '-' and 'NEW'. We can replace this value using 1 since the NEW label represents a new restaurant. Then fill the null values with the average rating in each CITY. The resultant rating should be of numeric type.



	RATING
3938	4.0
9355	4.1
11989	4.5

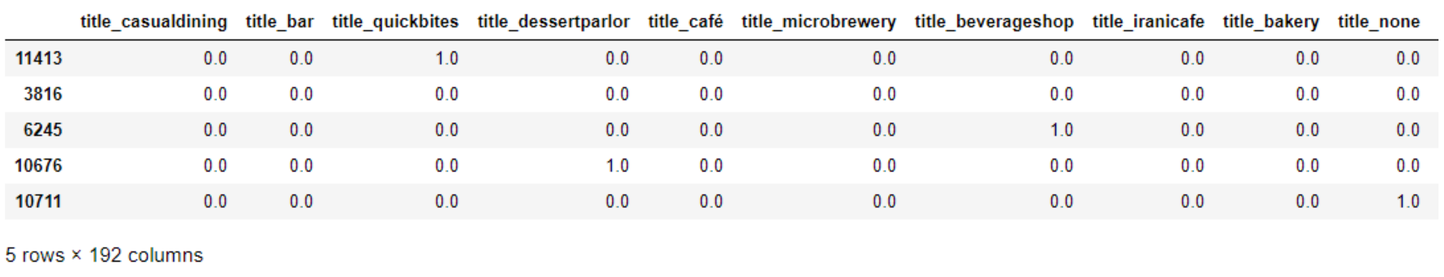
- Feature extraction on ***VOTES***:

Let's fill null values by '0 votes' and convert the variable into numeric type.



	VOTES
8064	105
6420	33
1254	45

So now we have extracted all the features required for further analysis. We can concatenate the extracted feature data frame into a single data frame. The resultant data will look like this:



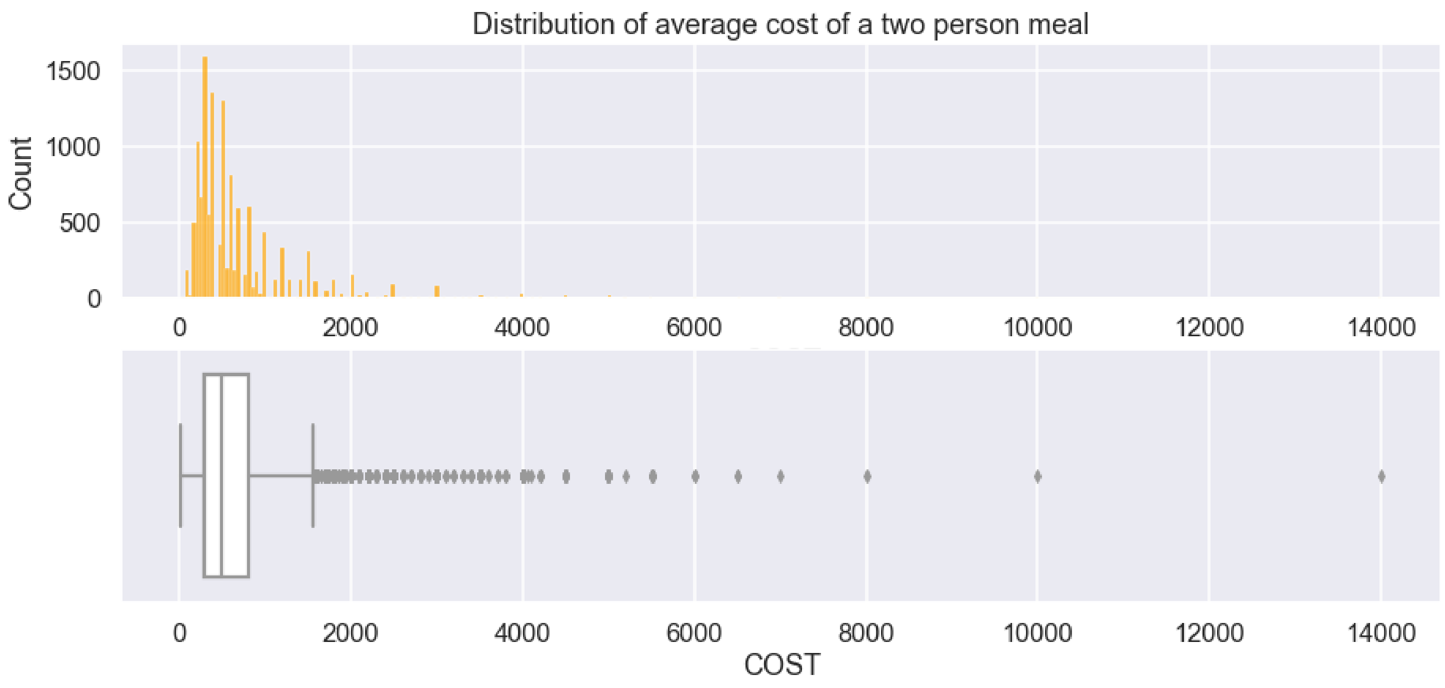
	title_casualdining	title_bar	title_quickbites	title_dessertparlor	title_café	title_microbrewery	title_beverageshop	title_iranicafe	title_bakery	title_none
11413	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3816	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6245	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
10676	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
10711	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

5 rows × 192 columns

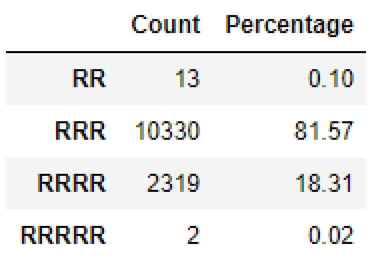
3. Data Analysis

So now we have reached the data analysis part. In this section, we will find out insights from the data through visualization. And find out the answers to the questions we had before starting the project.

How are the numeric features distributed? First, let’s see how the target variable (*COST*) is distributed.

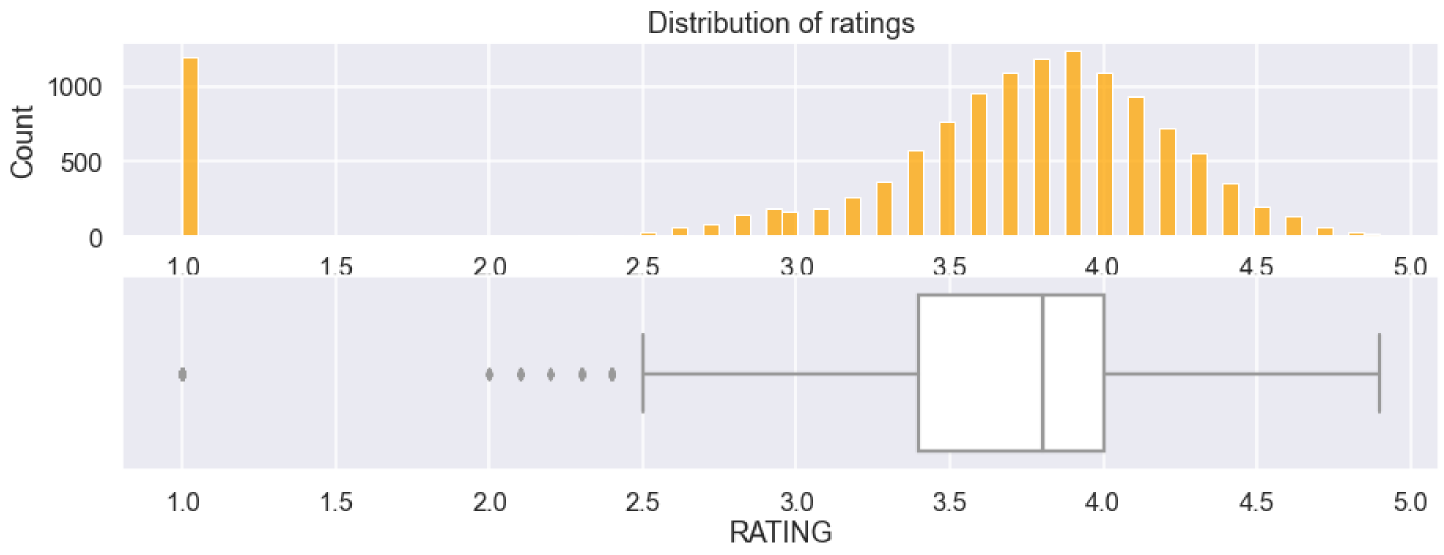


The data is right-skewed and the IQR for the above distribution is 500 Rupees. There are outliers (Cost very far from the central value) which comprises 25% of the entire observations, and about 6.8% are extreme outliers. We can consider these restaurants as really expensive. When we group the prices based on the number of digits in the prices as shown below:

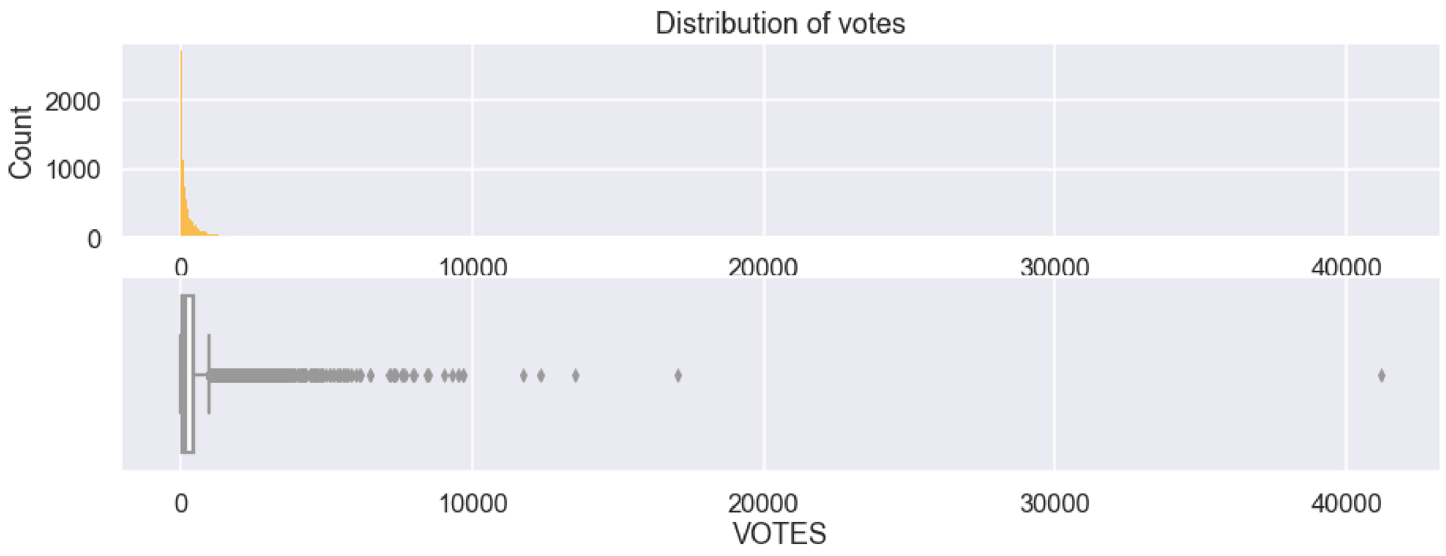


	Count	Percentage
RR	13	0.10
RRR	10330	81.57
RRRR	2319	18.31
RRRRR	2	0.02

Only 2 restaurants are outrageously expensive and 13 restaurants serve food costing less than 100 Rupees. About 81 % of restaurants have prices between 100 - 999 Rupees. Now let's look at the distribution of *RATING*.

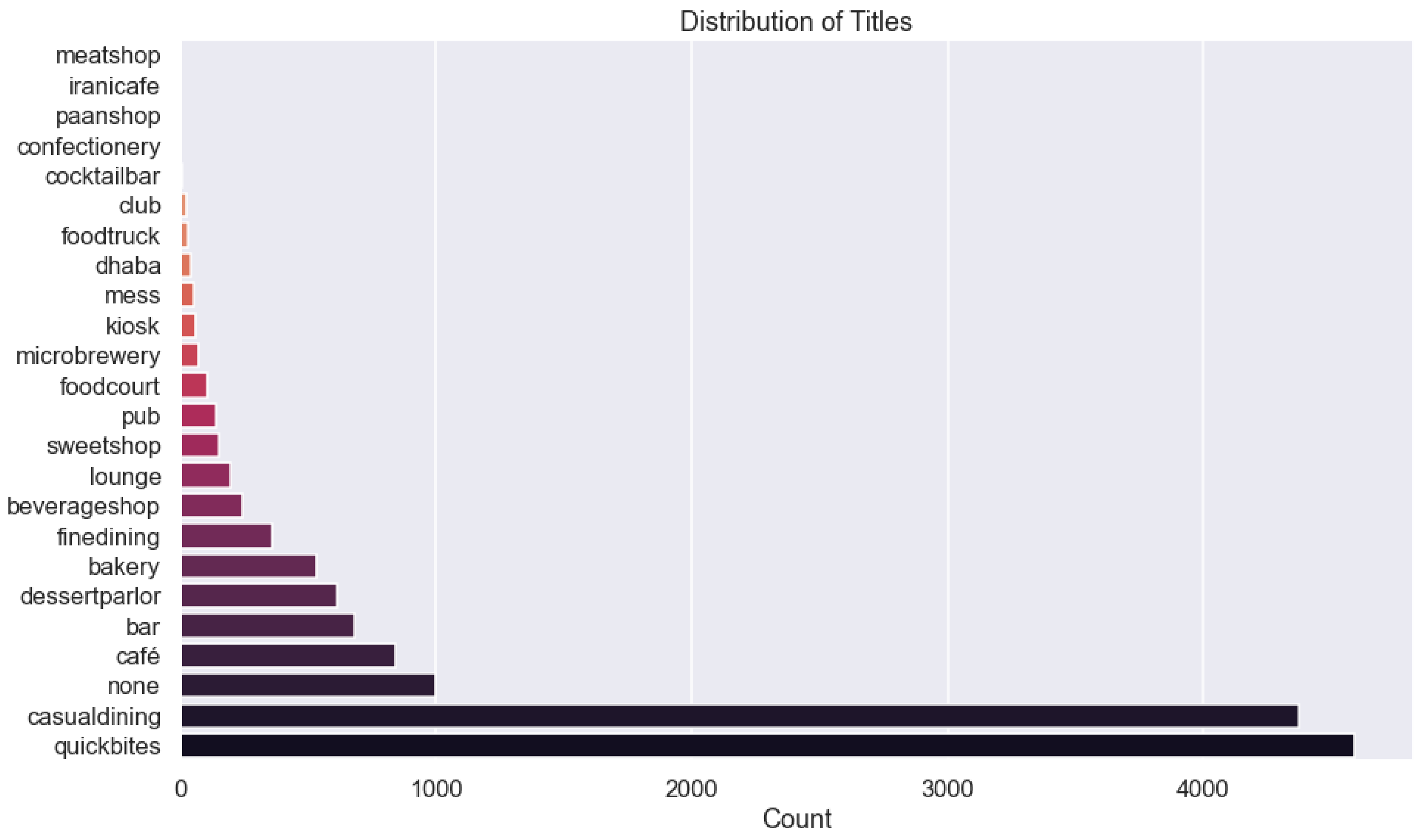


Here we see that the median rating is 3.8. There are more than 1000s of extreme outliers (Rating 1.0). These are the ratings of newly launched restaurants, highly separated from the median rating.



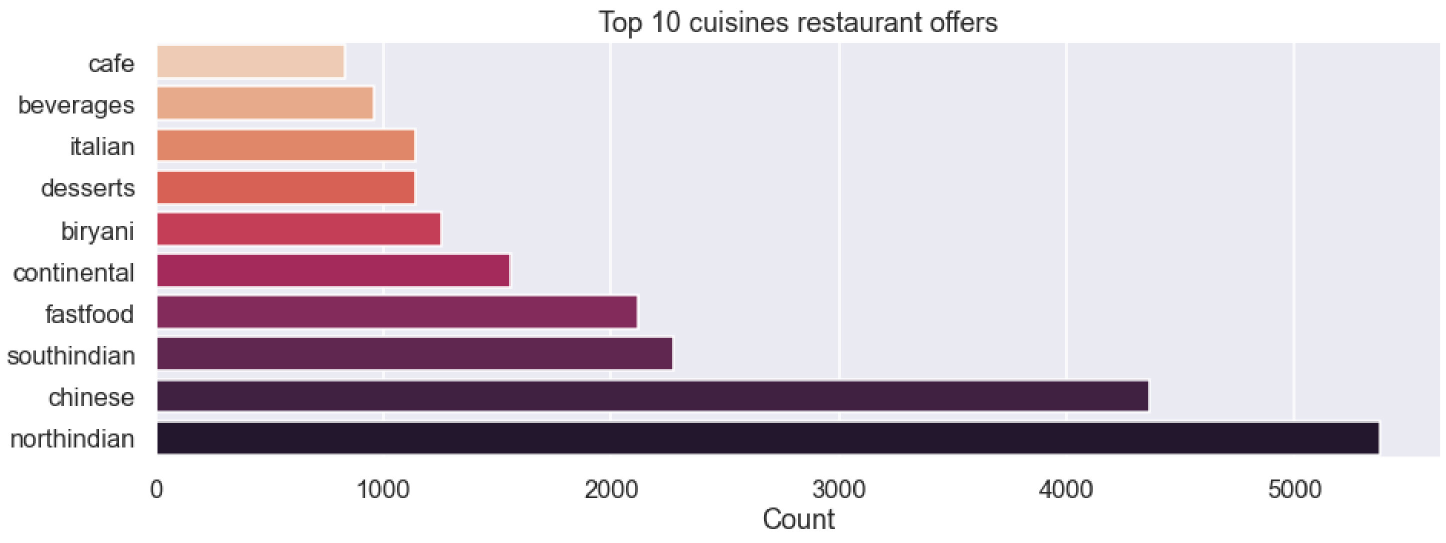
The median number of votes received by the restaurant is 132. The VOTES is extremely right skewed, and almost 50% of the observations are outliers.

What type of restaurants are people mostly looking for?

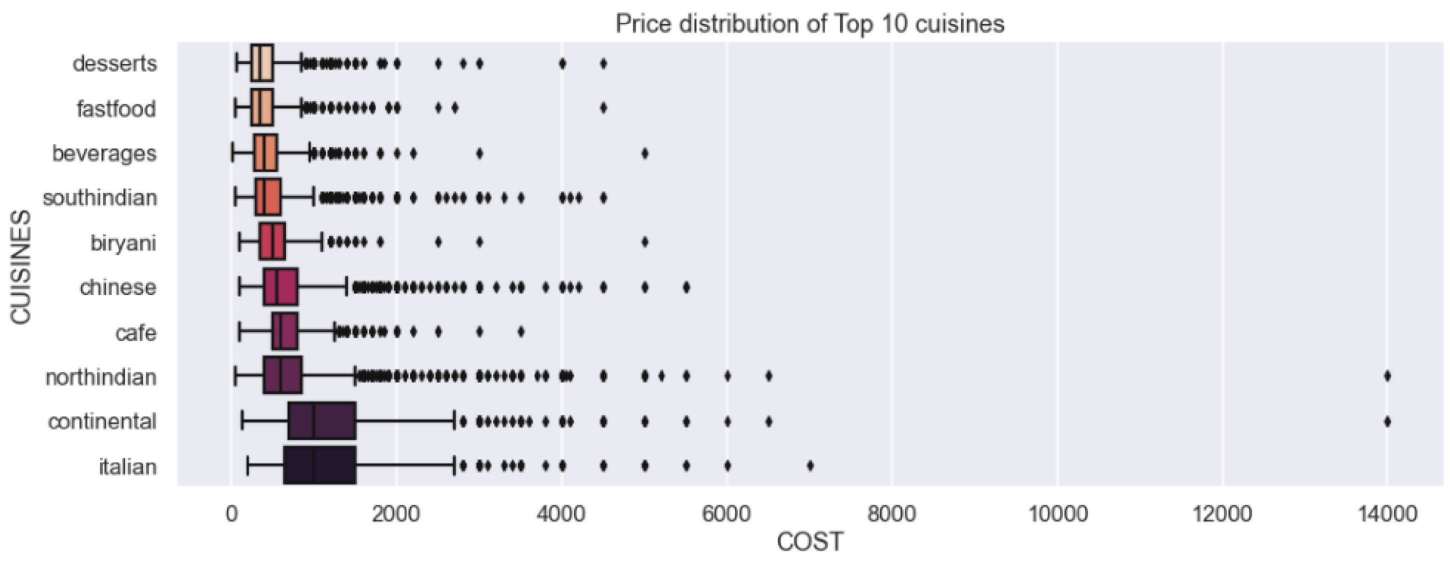


From the graph, we see that restaurants for casual dining and quick bites are widely available. It seems most Indians prefer to get quick bites and go out for casual dining.

What types of cuisines are mostly available in India? We'll now try to find out the top 10 available cuisines in India and the price range of those cuisines.

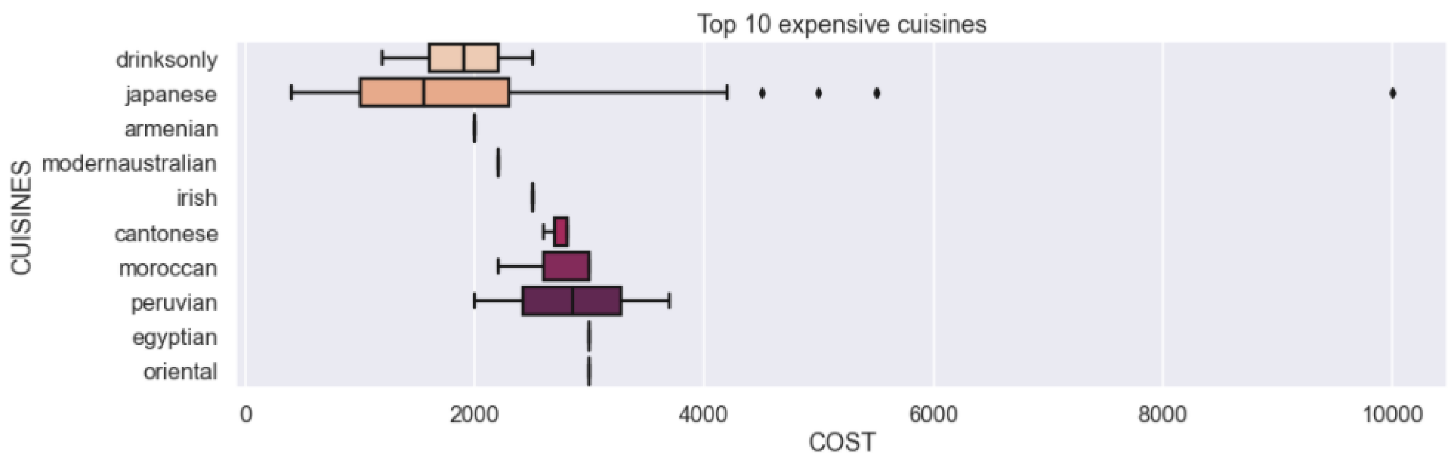


The above graph shows the top 10 cuisines available in India based on the number of observations from the data. It seems most Indians prefer North Indian cuisine over other types. Second, it comes to Chinese, followed by South Indian, etc. When we look at the count of North Indian and Chinese, these cuisines are enormously available in the country. It is more than twice the number of South Indian or Fast-food restaurants available. Now let's see the distribution of the average cost of a meal of the top 10 cuisines.



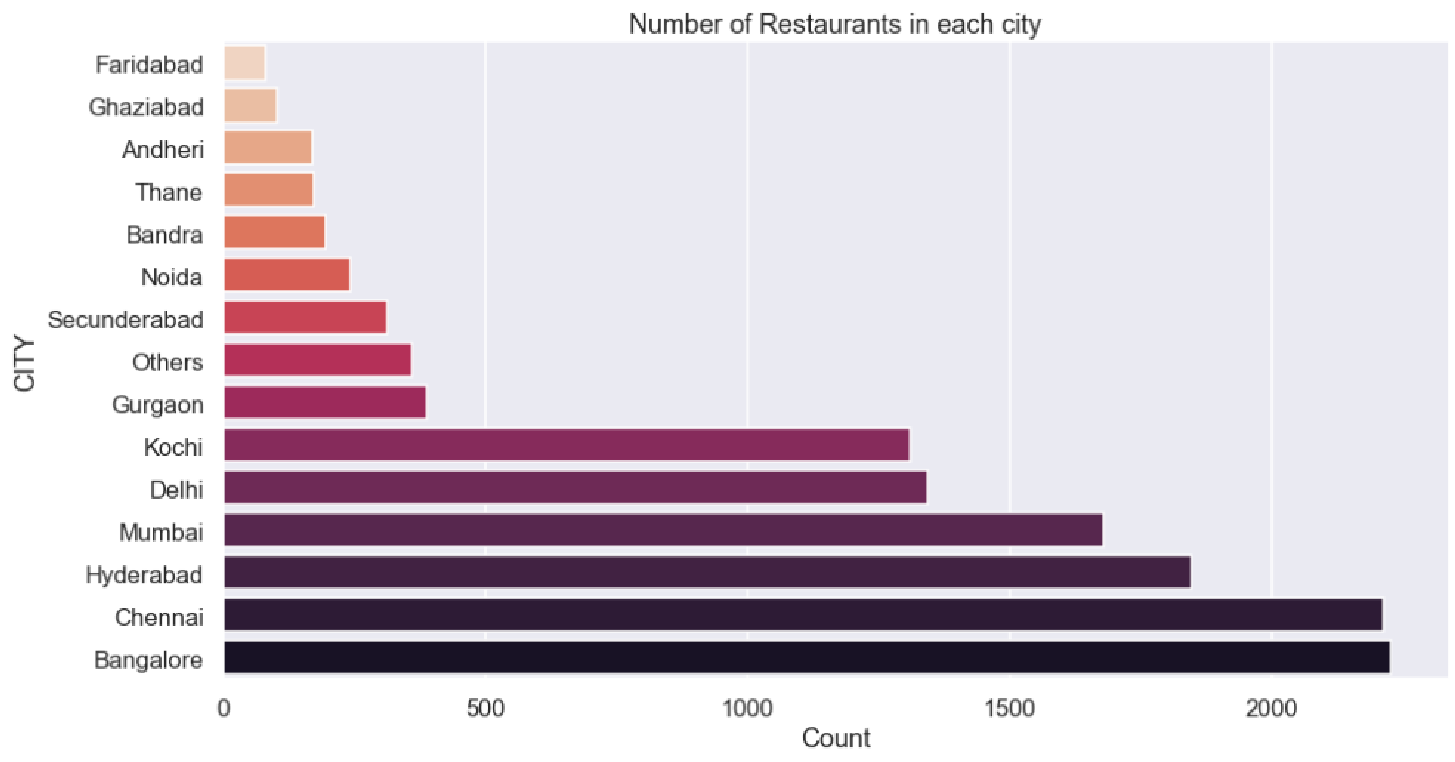
The box plot shows us the average cost of the two-person meal of the top 10 cuisines, which is in the order of lowest to the highest median cost. Out of the top 10 cuisines, we find Italian to be the most expensive and the lowest for the desserts. When we look at the range of cost, a few of the restaurants serving North Indian and Continental cuisines are extremely expensive. These restaurants charge food at 5 digit prices.

What are the top 10 most expensive cuisines? Previously we analyzed the cost of the Top 10 cuisines. Now we'll see the top 10 expensive cuisines.



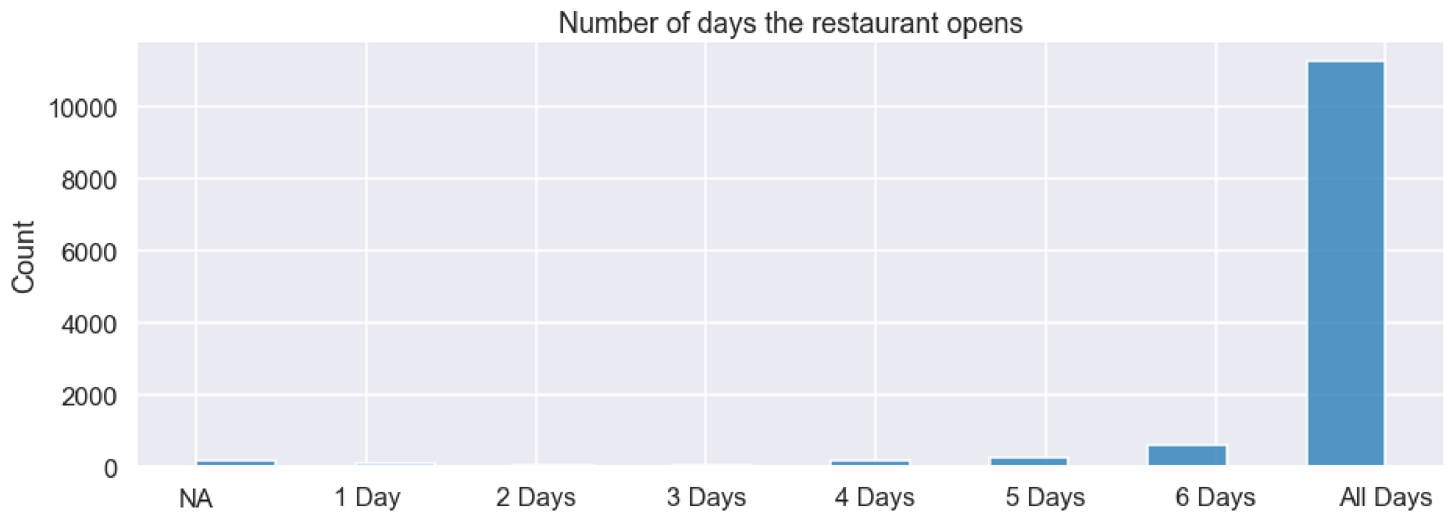
The above graph shows the distribution of the cost of the most expensive cuisines, arranged in the order of mean cost. We see that the Oriental type cuisines are the most expensive with a mean cost of 3000 Rupees, followed by Egyptian, Peruvian, etc. The cost of Japanese cuisine is spread widely, its cost ranges from 500 to over 10000 Rupees.

How many restaurants are there in each city? Until now we broadly looked at the most expensive and top cuisines. Now let's analyze each city.



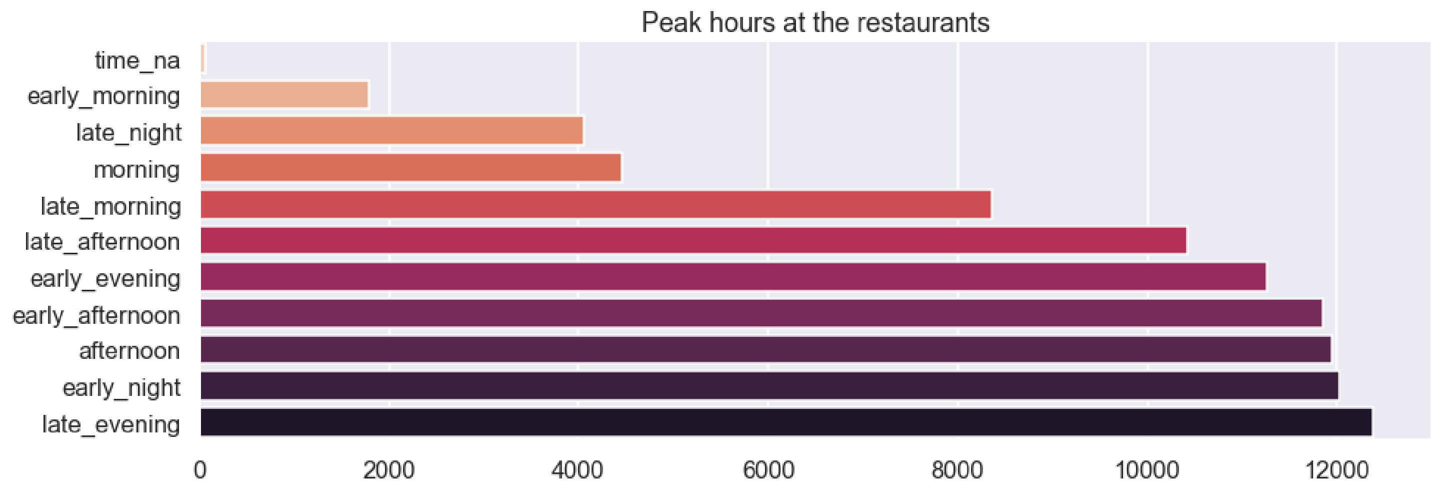
Bangalore and Chennai are having the largest number of restaurants than any other city, with a count exceeding 2000 restaurants. The lowest number of restaurants is in Faridabad. There are other cities, which together comprise almost 400 restaurants.

Is the restaurant open on all days? Let's visualize the number of restaurants and how many days a restaurant opens in a week.



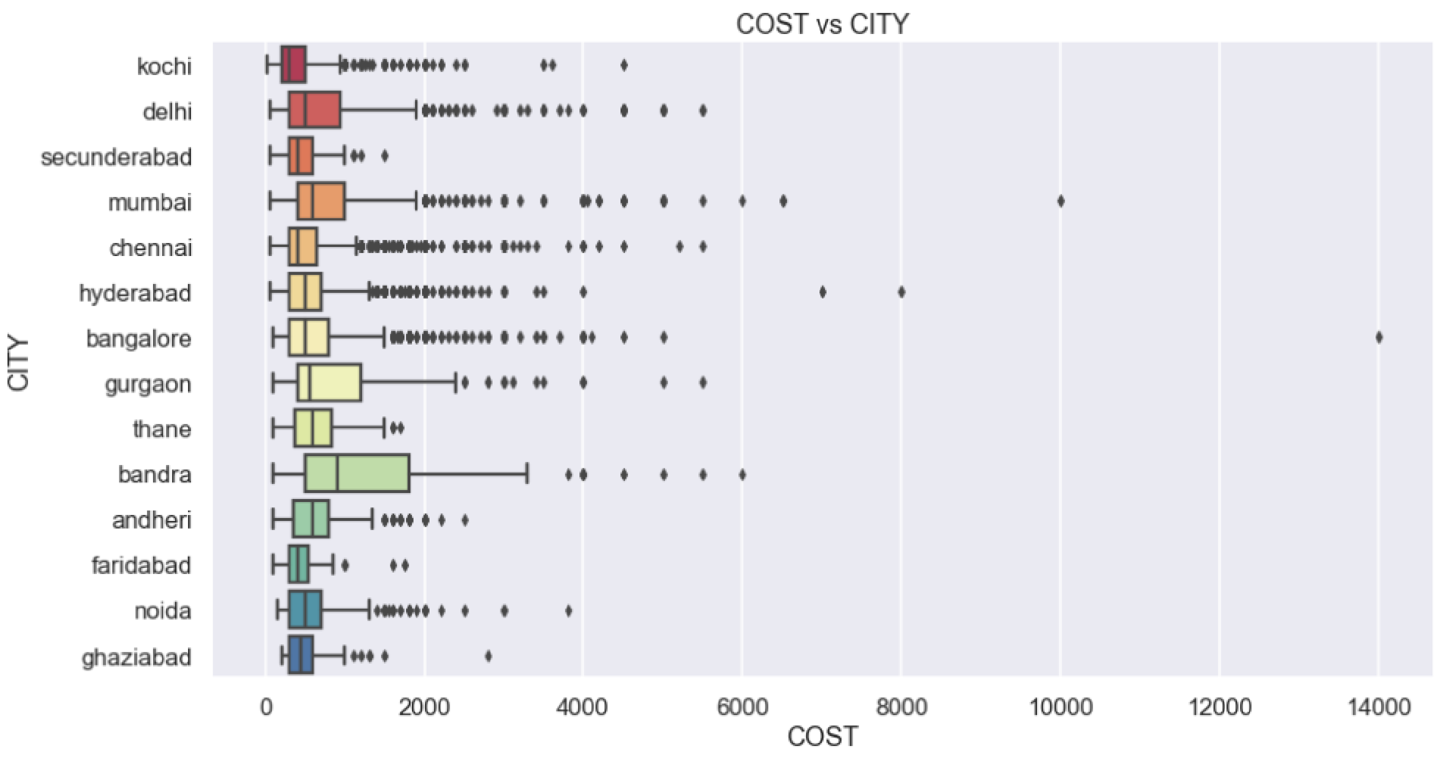
From the above graph, we see that almost 11000 restaurants operate on all days of the week. The rest of the restaurants operate on a lesser number of days. Some restaurants open for a very short number of days, maybe it’s true or it could be because of partial information given in the TIME variable, e.g. some samples end with ‘…’ in the dataset, which are incomplete.

What hours does the restaurant operate? Now we know almost every restaurant open on all days of the week. But at what time they operate.



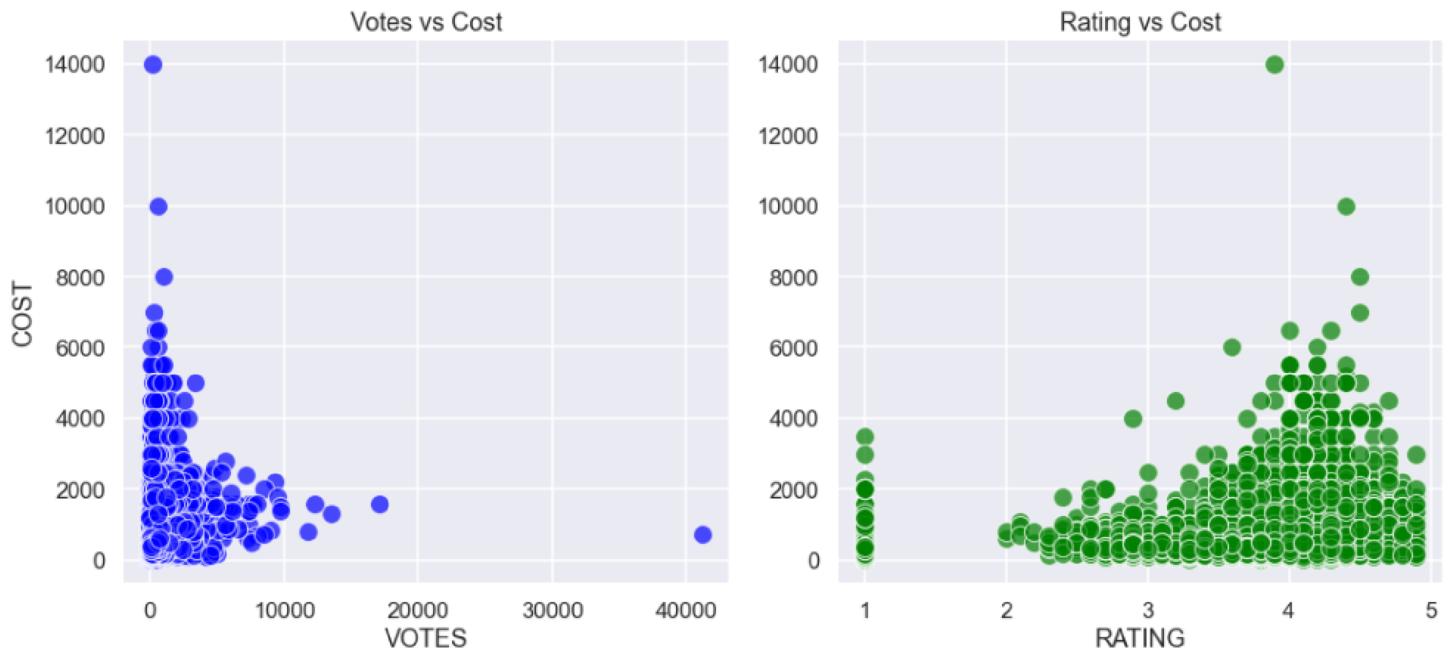
The above graph shows the number of restaurants operating at different hours of the day. We see that the peak hours are the late evening. In this hour, almost all the restaurants are busy serving meals to the customers. There are at least 4000 restaurants which operate in the late nights and almost 2000 in the early morning.

Which city is the least expensive and the most expensive?



Bangalore and Mumbai have the highest priced restaurants. The highest median price and spread are in Bandra. The price range in Faridabad and Secunderabad is the lowest. Kochi has the least median price.

VOTES vs COST vs Rating. Now let's find out how votes and ratings related to cost.



In the Votes vs Cost plot, we see that the expensive restaurants have the least number of votes and the average priced restaurant has the highest number of votes. Well, it makes sense that the number of people visiting the expensive restaurants is comparatively lower than people visiting the average prices restaurants, therefore the number of votes will be less.

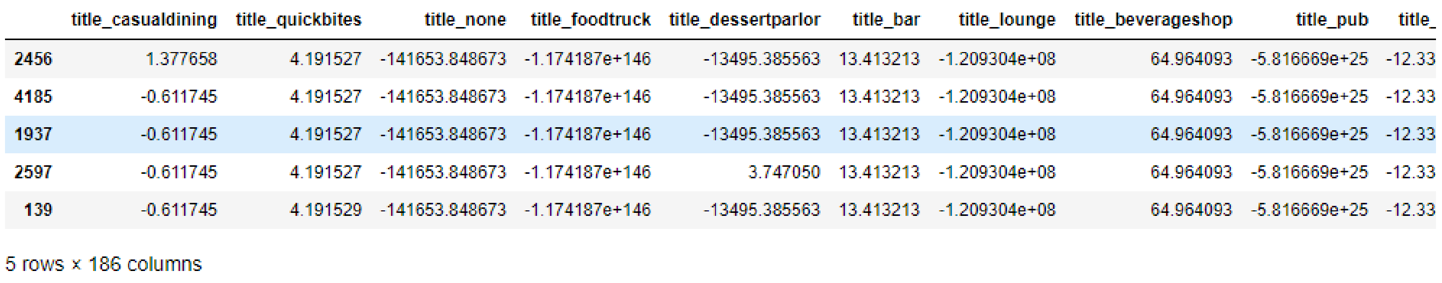
In the Rating vs Cost plot, the newly launched restaurants are having prices between 0 - 4000 Rupees. Most of the expensive restaurants have ratings between 4.0 - 5.0.

EDA Remarks

The number of restaurants and price of food varies from city to city. Factors that could affect these numbers are the population of the city. We have seen that places have a huge role in influencing prices. North Indian cuisines are most liked and most served in the restaurants. There are a huge number of people liking exotic or foreign cuisines such as the Continental, Italian, and many more. We found that Bangalore and Mumbai have restaurants which are exceptionally expensive. There is scope for more analysis on this dataset. We can find out the most popular destination or the cuisines in each city based on ratings. Also, the most popular cuisines in different price categories, etc.

4. Feature Engineering and preprocessing:

First, we scale the data using the Standard scaler function from the Sklearn module. Then check for skewness and remove skewness from the data. For that, we will apply Box-Cox transformation on the data, with a threshold of 0.5 (Used skew-autotransform method). The resulting data is :



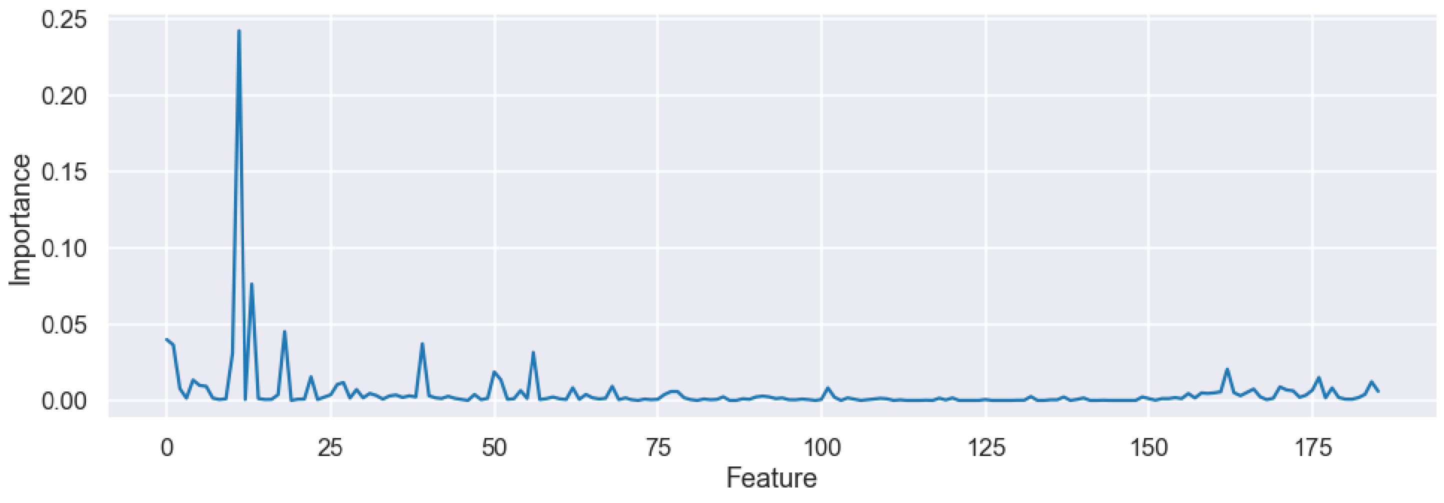
	title_casualdining	title_quickbites	title_none	title_foodtruck	title_dessertparlor	title_bar	title_lounge	title_beverageshop	title_pub	title_
2456	1.377658	4.191527	-141653.848673	-1.174187e+146	-13495.385563	13.413213	-1.209304e+08	64.964093	-5.816669e+25	-12.33
4185	-0.611745	4.191527	-141653.848673	-1.174187e+146	-13495.385563	13.413213	-1.209304e+08	64.964093	-5.816669e+25	-12.33
1937	-0.611745	4.191527	-141653.848673	-1.174187e+146	-13495.385563	13.413213	-1.209304e+08	64.964093	-5.816669e+25	-12.33
2597	-0.611745	4.191527	-141653.848673	-1.174187e+146	3.747050	13.413213	-1.209304e+08	64.964093	-5.816669e+25	-12.33
139	-0.611745	4.191529	-141653.848673	-1.174187e+146	-13495.385563	13.413213	-1.209304e+08	64.964093	-5.816669e+25	-12.33

5 rows × 186 columns

We will split train data into training and validation sets. Where training consists of 80% samples and 20% for the validation.

```
x_train, x_test, y_train, y_test =
train_test_split(X,y,test_size=0.2,random_state=37)
```

Now the data is ready for training, but the final data comprises 186 columns, which is quite many features. But we don't require the entire features for training because most of them might be insignificant for the prediction, and the second thing is the increasing time and space complexity. So before getting into training, we will try reducing the features by finding the feature importance of each variable. We will use the XGBRegressor algorithm to find out the importance of features.



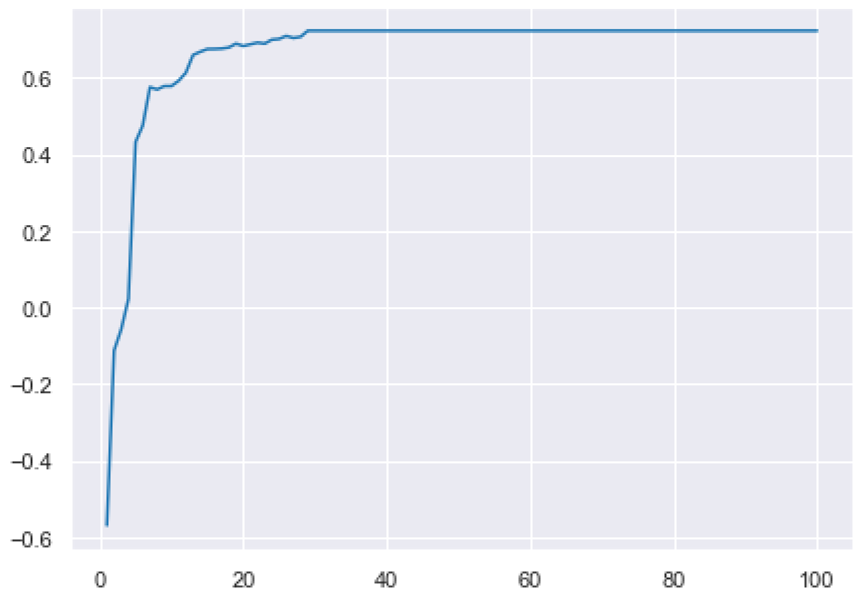
The spikes in the above graph denote the scores of highly important features. Feature selection work based on dimensionality reduction. We'll select the k best features by training the booster on different values of k.

```
from sklearn.feature_selection import SelectFromModel
from xgboost import XGBRegressor
from sklearn.metrics import r2_score

scores = []
for i in range(1,101):
    fs = SelectFromModel(XGBRegressor(), max_features=i)
    fs.fit(x_train, y_train)
    x_train_fs = fs.transform(x_train)
    x_test_fs = fs.transform(x_test)
    model = XGBRegressor().fit(x_train_fs,y_train)
    pred = model.predict(x_test_fs)
    scores.append(r2_score(pred,y_test))
    print(i,end='.')

# Plot R2 Score vs the max_features
plt.figure(figsize=(7,5))
sns.lineplot(x=np.arange(1,101),y=scores)
plt.show()
```

Output :



The R2 score remains the same after k > 28. So we should select at least 29 features to attain the best score.

The best 29 features are :

Titles : Casual Dining, bar, Quickbites, Cafe, Microbrewery, Pub, Fine Dining, Lounge, Club, Cocktail Bar.

Cuisines : Japanese, Asian, BarFood, Continental, European, Italian, FingerFood, Mediterranean, Sushi.

Part of the day : Late Night, Early Afternoon, Late Evening

City : Bandra, Mumbai, Gurgaon, Kochi, Delhi

Rating and Votes.

5. Building machine learning models:

Now we build models using the Sklearn algorithms. For the training, we are going to build multiple regressors and find out which model performs the best. For that, we will choose linear models such as Ridge and Lasso, Tree algorithms such as decision trees, Boosting algorithms such as Light gradient boost and Cat Boosting algorithms and some are ensemble methods which are Random forest and extra tree regressor.

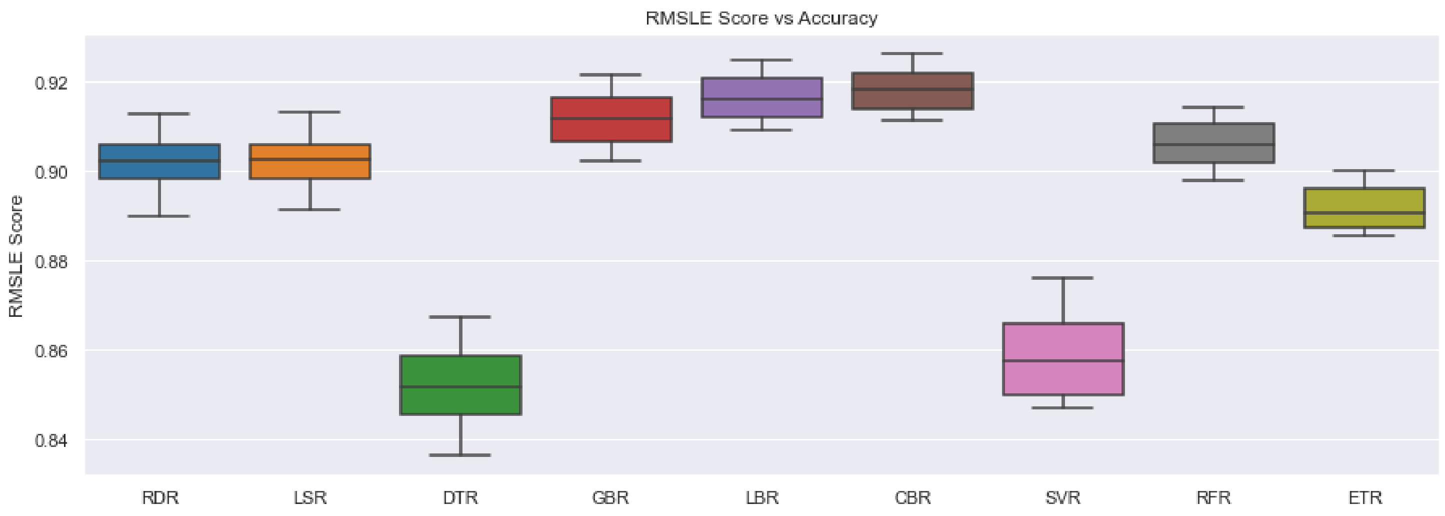
```
from sklearn.linear_model import Ridge, Lasso
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor,
RandomForestRegressor, ExtraTreesRegressor
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor

# Create dictionary of models to be trained
model_dict = {
    'RDR':Ridge(),
    'LSR':Lasso(),
    'DTR':DecisionTreeRegressor(),
    'GBR':GradientBoostingRegressor(),
    'LBR':LGBMRegressor(),
    'CBR':CatBoostRegressor(verbose=0),
    'SVR':SVR(),
    'RFR':RandomForestRegressor(),
    'ETR':ExtraTreesRegressor()
}
```

Apply cross-validation on the training set and evaluate on the validation set. For the evaluation, I've used the Root Mean Squared Log Error (RMSLE).

```
# Lets use Root Mean Squared Log error for the evaluation of the models
def rmsle_score(pred,y):
    pred = np.log(pred)
    y = np.log(y)
    return 1-((np.sum((pred-y)**2))/len(y))**1/2
scorer = make_scorer(rmsle_score,greater_is_better=True)
```

If either the prediction and actual values is big, root mean squared error (RMSE) scoring yields large error. Whereas RMSLE doesn't penalize vast differences in the predicted and the actual values.



The above graph shows the average RMSLE score of each regressor from the cross validation. We will take the models with median scores greater than 0.90 and apply hyperparameter tuning to improve model prediction.

Algorithm	**Train RMSLE**	**Validation RMSLE**
Ridge	0.902	0.904
Lasso	0.902	0.904
LGBM	0.915	0.917
boost	0.913	0.915
GradientBoost	0.912	0.913
Random Forest	0.916	0.918

- Based on the RMSLE score, the best performing models are:
  - **Random Forest Regressor**: 0.918
  - **LGBM Regressor** : 0.917

Though the Random Forest regressor has achieved the best score, the inference time for this model is higher than the rest of the model. Finally, we can use either of the models for predicting the test data.

What if we use the entire features on LGMB Regressor? We will use the same hyper-parameters from the already trained LGBM model to fit the train data.

Training RMSLE score : 0.928

Validation RMSLE score : 0.93

Well, there were about 186 features initially and after reduction, we got 29 features. So that is almost an 85% reduction in input variables. Which significantly improves training speed and inference.

6. Conclusion :

So we explored the restaurant food prices dataset and investigated the factors affecting the prices with various visualizations. We also did training different regression models using the most important features and found out that Random forest Regressor and Light Gradient Boosted regressor was able to achieve the highest RMSLE scores. These models can further be improved by tuning with wide ranges of hyper-parameters. The dataset actually consists of two files, training and testing. The test dataset undergoes the similar preprocessing steps explained in this blog. The complete code for preprocessing and predicting test dataset is available within the notebook itself.