Rating Prediction of Zomato Restaurants

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [ ]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

Reading Data

In [ ]:

```
df=pd.read_csv("/content/drive/My Drive/zomato.csv")
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(51717, 17)
```

In [ ]:

```
df.columns
```

Out[ ]:

```
Index(['url', 'address', 'name', 'online_order', 'book_table', 'rate', 'votes',
       'phone', 'location', 'rest_type', 'dish_liked', 'cuisines',
       'approx_cost(for two people)', 'reviews_list', 'menu_item',
       'listed_in(type)', 'listed_in(city)'],
      dtype='object')
```

In [ ]:

```
df.head(2)
```

Out[ ]:

	url	address	name	online_order	book_table	rate	votes	pho
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari,	Jalsa	Yes	Yes	4.1/5	775	080 42297555\r\n 9743772233

	url	address	name	online_order	book_table	rate	votes	pho
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1/5	787	080 41714161

In [ ]:

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

Out[ ]:

```
count     43942
unique       64
top         NEW
freq       2208
Name: rate, dtype: object
```

Checking for Duplicates in dataset

In [ ]:

```
df.duplicated().sum()
```

Out[ ]:

0

Removing the features which don't have any impact on rating of the restaurant

In [ ]:

```
df=df.drop(['phone','address','url'],axis=1)
```

Checking for percentage of NULL values for each features

In [ ]:

```
# https://stackoverflow.com/questions/51070985/find-out-the-percentage-of-missing-values-in-each-c
olumn-in-the-given-dataset
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'percent_missing': percent_missing})
```

In [ ]:

```
missing_value_df
```

Out[ ]:

	percent_missing
name	0.000000
online_order	0.000000
book_table	0.000000
rate	15.033741
votes	0.000000
location	0.040606
rest_type	0.438927

	percent_missing
dish_liked	54.291626
cuisines	0.087012
approx_cost(for two people)	0.669026
reviews_list	0.000000
menu_item	0.000000
listed_in(type)	0.000000
listed_in(city)	0.000000

Approach to fill the Null values -

1. **Rate and dish_liked - Model Based imputation**
2. **Location, cuisines, rest_type - Frequency based imputation**
3. **Approx_cost - Mean based imputation**

approx_cost NULL values

In [ ]:

```
df['approx_cost(for two people)']=df['approx_cost(for two people)'].replace(',','')
df['approx_cost(for two people)'] = df['approx_cost(for two people)'].astype(str).str.replace(',',
'')
df['approx_cost(for two people)'] = df['approx_cost(for two people)'].apply(lambda r: float(r))
```

Replacing NULL value with mean of approx_cost

In [ ]:

```
df['approx_cost(for two people)']=df['approx_cost(for two people)'].fillna(df['approx_cost(for two
people)'].mean())
```

rest_type NULL values

In [ ]:

```
df['rest_type'].isnull().values.sum()
```

Out[ ]:

```
227
```

In [ ]:

```
df['rest_type'].value_counts()
```

Out[ ]:

```
Quick Bites                      19132
Casual Dining                    10330
Cafe                              3732
Delivery                          2604
Dessert Parlor                    2263
                                 ...
Food Court, Beverage Shop            2
Cafe, Food Court                     2
Dessert Parlor, Kiosk                2
Sweet Shop, Dessert Parlor           1
Quick Bites, Kiosk                   1
Name: rest_type, Length: 93, dtype: int64
```

In [ ]:

```
# Replacing rest_type with top 2 occourings rest_type
df['rest_type']=df['rest_type'].fillna('Quick Bites, Casual Dining')
```

Analyzing location column NULL values

In [ ]:

```
df['location'].value_counts()
```

Out[ ]:

```
BTM                      5124
HSR                      2523
Koramangala 5th Block    2504
JP Nagar                 2235
Whitefield               2144
                         ...
Yelahanka                   6
West Bangalore              6
Jakkur                      3
Rajarajeshwari Nagar        2
Peenya                      1
Name: location, Length: 93, dtype: int64
```

In [ ]:

```
df['location'].isnull().values.sum()
```

Out[ ]:

```
21
```

Hence there are only 21 datapoints whose location column is NULL, so lets assign their value as BTM because btm has the maximum no. of restaurants in Bangalore

In [ ]:

```
df['location']=df['location'].fillna('BTM')
```

Cuisnies features

In [ ]:

```
df['cuisines'].isnull().values.sum()
```

Out[ ]:

```
45
```

In [ ]:

```
df['cuisines'].value_counts()
```

Out[ ]:

```
North Indian                                    2913
North Indian, Chinese                           2385
South Indian                                    1828
Biryani                                          918
Bakery, Desserts                                 911
                                                ...
Rolls, Arabian                                     1
Rolls, North Indian                                1
Tea, Beverages, Street Food                        1
Fast Food, Andhra                                  1
Fast Food, Chinese, Burger, Hot dogs, Sandwich     1
Name: cuisines, Length: 2723, dtype: int64
```

Replacing NULL values with top 3 cuisines

In [ ]:

```
df['cuisines']=df['cuisines'].fillna('North Indian, chinese, South Indian')
```

Replacing rate column missing values using Model based imputation

Creating a dataframe having only non null rows

In [ ]:

```
vx=df.copy()
vx=df[(df['rate'].notnull()) & (df['rate']!='NEW') & (df['rate']!='-') ]
vx.shape
```

Out[ ]:

```
(41665, 14)
```

In [ ]:

```
vx['rate'] = vx['rate'].astype(str).str.replace('/5', '')
from sklearn.model_selection import train_test_split
y = vx['rate']
X = vx.drop(['rate'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X_train['online_order'].values)
X_tr_order_ohe = vectorizer.transform(X_train['online_order'].values)
X_test_order_ohe = vectorizer.transform(X_test['online_order'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['book_table'].values)
X_tr_book_table = vectorizer.transform(X_train['book_table'].values)
X_test_book_table = vectorizer.transform(X_test['book_table'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['location'].values)
X_tr_location = vectorizer.transform(X_train['location'].values)
X_test_location = vectorizer.transform(X_test['location'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['rest_type'].values)
X_tr_rest_type = vectorizer.transform(X_train['rest_type'].values)
X_test_rest_type = vectorizer.transform(X_test['rest_type'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['cuisines'].values)
X_tr_cuisines  = vectorizer.transform(X_train['cuisines'].values)
X_test_cuisines  = vectorizer.transform(X_test['cuisines'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['listed_in(type)'].values)
X_tr_listed_in_t = vectorizer.transform(X_train['listed_in(type)'].values)
X_test_listed_in_t = vectorizer.transform(X_test['listed_in(type)'].values)

vectorizer = CountVectorizer()
vectorizer.fit(df['listed_in(city)'].values)
X_tr_listed_in_c = vectorizer.transform(X_train['listed_in(city)'].values)
X_test_listed_in_c = vectorizer.transform(X_test['listed_in(city)'].values)

from sklearn.preprocessing import Normalizer
cost_scalar = Normalizer()
```

```
cost_scalar.fit(X_train['approx_cost(for two people)'].values.reshape(-1,1))
cost_standardized_train = cost_scalar.transform(X_train['approx_cost(for two
people)'].values.reshape(-1, 1))
cost_standardized_test = cost_scalar.transform(X_test['approx_cost(for two
people)'].values.reshape(-1, 1))
```

In [ ]:

```
from scipy.sparse import hstack
X_tr =
hstack((X_tr_order_ohe,X_tr_book_table,X_tr_location,X_tr_rest_type,X_tr_cuisines,X_tr_listed_in_t
,X_tr_listed_in_c,cost_standardized_train)).tocsr()
X_test =
hstack((X_test_order_ohe,X_test_book_table,X_test_location,X_test_rest_type,X_test_cuisines,X_test_
listed_in_t,X_test_listed_in_c,cost_standardized_test)).tocsr()
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
lm= LinearRegression().fit(X_tr,y_train)
```

Now we will create a dataframe that contains all the rows having missing values of rate

In [ ]:

```
missing_values=df[(df['rate'].isnull()) | (df['rate']=='NEW') | (df['rate']=='-')]
```

In [ ]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['online_order'].values)
X_order_ohe_missing_values = vectorizer.transform(missing_values['online_order'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['book_table'].values)
X_book_table_missing_values = vectorizer.transform(missing_values['book_table'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['location'].values)
X_location_missing_values = vectorizer.transform(missing_values['location'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['rest_type'].values)
X_rest_type_missing_values = vectorizer.transform(missing_values['rest_type'].values)



vectorizer = CountVectorizer()
vectorizer.fit(X_train['cuisines'].values)
X_cuisines_missing_values = vectorizer.transform(missing_values['cuisines'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['listed_in(type)'].values)
X_listed_in_t_missing_values = vectorizer.transform(missing_values['listed_in(type)'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['listed_in(city)'].values)
X_listed_in_c_missing_values = vectorizer.transform(missing_values['listed_in(city)'].values)


cost_scalar = Normalizer()
cost_scalar.fit(X_train['approx_cost(for two people)'].values.reshape(-1,1))
cost_standardized_train_missing_values = cost_scalar.transform(missing_values['approx_cost(for two
people)'].values.reshape(-1, 1))
```

In [ ]:

```
X_missing_values =
hstack((X_order_ohe_missing_values,X_book_table_missing_values,X_location_missing_values,X_rest_typ
e_missing_values,X_cuisines_missing_values,X_listed_in_t_missing_values,X_listed_in_c_missing_value
s,cost_standardized_train_missing_values)).tocsr()
```

In [ ]:

```
lm.predict(X_missing_values)
```

Out[ ]:

```
array([3.48164567, 3.48551695, 3.45014085, ..., 3.59386058, 3.59386058,
       3.59386058])
```

In [ ]:

```
y=lm.predict(X_missing_values)
```

In [ ]:

```
missing_values=missing_values.drop(['rate'],axis=1)
missing_values['rate']=y
```

In [ ]:

```
df2=pd.concat([vx,missing_values], axis=0)
```

In [ ]:

```
df2.isna().any()
```

Out[ ]:

```
name                           False
online_order                   False
book_table                     False
rate                           False
votes                          False
location                       False
rest_type                      False
dish_liked                      True
cuisines                       False
approx_cost(for two people)    False
reviews_list                   False
menu_item                      False
listed_in(type)                False
listed_in(city)                False
dtype: bool
```

Now all the missing values of rate column is replaced using model predicted values

Model Based imputation for dish_liked

In [ ]:

```
df_no_null = df2[df2['dish_liked'].notnull()]
```

In [ ]:

```
df_with_null=df2[df2['dish_liked'].isnull()]
```

In [ ]:

```
y=df_no_null['dish_liked']
X=df_no_null.drop(['dish_liked'],axis=1)
```

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X_train['online_order'].values)
X_tr_order_ohe = vectorizer.transform(X_train['online_order'].values)
X_test_order_ohe = vectorizer.transform(X_test['online_order'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['book_table'].values)
X_tr_book_table = vectorizer.transform(X_train['book_table'].values)
X_test_book_table = vectorizer.transform(X_test['book_table'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['location'].values)
X_tr_location = vectorizer.transform(X_train['location'].values)
X_test_location = vectorizer.transform(X_test['location'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['rest_type'].values)
X_tr_rest_type = vectorizer.transform(X_train['rest_type'].values)
X_test_rest_type = vectorizer.transform(X_test['rest_type'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['cuisines'].values)
X_tr_cuisines  = vectorizer.transform(X_train['cuisines'].values)
X_test_cuisines  = vectorizer.transform(X_test['cuisines'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['listed_in(type)'].values)
X_tr_listed_in_t = vectorizer.transform(X_train['listed_in(type)'].values)
X_test_listed_in_t = vectorizer.transform(X_test['listed_in(type)'].values)

vectorizer = CountVectorizer()
vectorizer.fit(df['listed_in(city)'].values)
X_tr_listed_in_c = vectorizer.transform(X_train['listed_in(city)'].values)
X_test_listed_in_c = vectorizer.transform(X_test['listed_in(city)'].values)

from sklearn.preprocessing import Normalizer
cost_scalar = Normalizer()
cost_scalar.fit(X_train['approx_cost(for two people)'].values.reshape(-1,1))
cost_standardized_train = cost_scalar.transform(X_train['approx_cost(for two
people)'].values.reshape(-1, 1))
cost_standardized_test = cost_scalar.transform(X_test['approx_cost(for two
people)'].values.reshape(-1, 1))

from sklearn.preprocessing import Normalizer
rate_scalar = Normalizer()
rate_scalar.fit(X_train['rate'].values.reshape(-1,1))
rate_standardized_train = rate_scalar.transform(X_train['rate'].values.reshape(-1, 1))
rate_standardized_test = rate_scalar.transform(X_test['rate'].values.reshape(-1, 1))
```

In [ ]:

```
from scipy.sparse import hstack
X_tr =
hstack((X_tr_order_ohe,X_tr_book_table,X_tr_location,X_tr_rest_type,X_tr_cuisines,X_tr_listed_in_t
,X_tr_listed_in_c,cost_standardized_train,rate_standardized_train)).tocsr()
X_test =
hstack((X_test_order_ohe,X_test_book_table,X_test_location,X_test_rest_type,X_test_cuisines,X_test_
listed_in_t,X_test_listed_in_c,cost_standardized_test,rate_standardized_test)).tocsr()
```

In [ ]:

```
from sklearn.multiclass import OneVsRestClassifier
```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier

classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l1'), n_jobs=-1)
classifier.fit(X_tr, y_train)
#predictions = classifier.predict(x_test_multilabel)
```

Out[ ]:

```
OneVsRestClassifier(estimator=SGDClassifier(alpha=1e-05, average=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=0.1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal', loss='log',
                                            max_iter=1000, n_iter_no_change=5,
                                            n_jobs=None, penalty='l1',
                                            power_t=0.5, random_state=None,
                                            shuffle=True, tol=0.001,
                                            validation_fraction=0.1, verbose=0,
                                            warm_start=False),
                    n_jobs=-1)
```

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(X_train['online_order'].values)
X_tr_order_ohe = vectorizer.transform(df_with_null['online_order'].values)
# X_test_order_ohe = vectorizer.transform(X_test['online_order'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['book_table'].values)
X_tr_book_table = vectorizer.transform(df_with_null['book_table'].values)
# X_test_book_table = vectorizer.transform(X_test['book_table'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['location'].values)
X_tr_location = vectorizer.transform(df_with_null['location'].values)
# X_test_location = vectorizer.transform(X_test['location'].values)

vectorizer = CountVectorizer()
vectorizer.fit(X_train['rest_type'].values)
X_tr_rest_type = vectorizer.transform(df_with_null['rest_type'].values)
# X_test_rest_type = vectorizer.transform(X_test['rest_type'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['cuisines'].values)
X_tr_cuisines  = vectorizer.transform(df_with_null['cuisines'].values)
# X_test_cuisines  = vectorizer.transform(X_test['cuisines'].values)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['listed_in(type)'].values)
X_tr_listed_in_t = vectorizer.transform(df_with_null['listed_in(type)'].values)
# X_test_listed_in_t = vectorizer.transform(X_test['listed_in(type)'].values)

vectorizer = CountVectorizer()
vectorizer.fit(df['listed_in(city)'].values)
X_tr_listed_in_c = vectorizer.transform(df_with_null['listed_in(city)'].values)
# X_test_listed_in_c = vectorizer.transform(X_test['listed_in(city)'].values)

from sklearn.preprocessing import Normalizer
cost_scalar = Normalizer()
cost_scalar.fit(X_train['approx_cost(for two people)'].values.reshape(-1,1))
cost_standardized_train = cost_scalar.transform(df_with_null['approx_cost(for two people)'].values
.reshape(-1, 1))
# cost_standardized_test = cost_scalar.transform(X_test['approx_cost(for two
people)'].values.reshape(-1, 1))

from sklearn.preprocessing import Normalizer
rate_scalar = Normalizer()
rate_scalar.fit(X_train['rate'].values.reshape(-1,1))
rate_standardized_train = rate_scalar.transform(df_with_null['rate'].values.reshape(-1, 1))
```

```
rate_standardized_train = rate_scalar.transform(df_with_null['rate'].values.reshape(-1, 1))
# rate_standardized_test = rate_scalar.transform(X_test['rate'].values.reshape(-1, 1))
```

In [ ]:

```
X_tr =
hstack((X_tr_order_ohe,X_tr_book_table,X_tr_location,X_tr_rest_type,X_tr_cuisines,X_tr_listed_in_t
,X_tr_listed_in_c,cost_standardized_train,rate_standardized_train)).tocsr()
```

In [ ]:

```
y=classifier.predict(X_tr)
df_with_null=df_with_null.drop(['dish_liked'],axis=1)
df_with_null['dish_liked']=y
```

In [ ]:

```
df=pd.concat([df_with_null,df_no_null], axis=0)
```

In [ ]:

```
df.head(2)
```

Out[ ]:

	name	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_lis
6	Rosewood International Hotel - Bar & Restaurant	No	No	3.6	8	Mysore Road	Casual Dining	North Indian, South Indian, Andhra, Chinese	800.0	[('Rated 5.0', 'RATED\n Awesome food ?? Great ...
19	360 Atoms Restaurant And Cafe	Yes	No	3.1	13	Banashankari	Cafe	Cafe, Chinese, Continental, Italian	400.0	[('Rated 5.0', 'RATED\n Friendly staffs , nic...

In [ ]:

```
df.isna().any()
```

Out[ ]:

```
name                           False
online_order                   False
book_table                     False
rate                           False
votes                          False
location                       False
rest_type                      False
cuisines                       False
approx_cost(for two people)    False
reviews_list                   False
menu_item                      False
listed_in(type)                False
listed_in(city)                False
dish_liked                     False
dtype: bool
```

From the above table, we can see that all the null values are replaced

In [ ]:

```
df['rate']=df['rate'].apply(lambda r: float(r))
```

In [ ]:

```
decimals = 1
df['rate'] = df['rate'].apply(lambda x: round(x, decimals))
```

Exploratory Data Analysis

Univariate Analysis

Analysis on location

In [ ]:

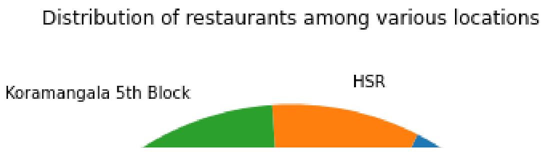
```
df['listed_in(city)'].value_counts()
```

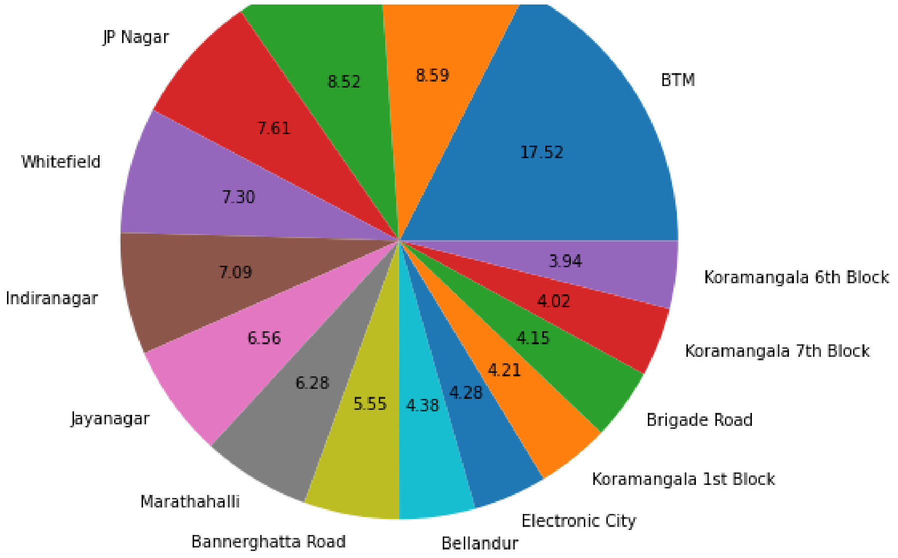
Out[ ]:

```
BTM                      3279
Koramangala 7th Block    2938
Koramangala 5th Block    2836
Koramangala 4th Block    2779
Koramangala 6th Block    2623
Jayanagar                2371
JP Nagar                 2096
Indiranagar              1860
Church Street            1827
MG Road                  1811
Brigade Road             1769
Lavelle Road             1744
HSR                      1741
Marathahalli             1659
Whitefield               1620
Residency Road           1620
Bannerghatta Road        1617
Brookefield              1518
Old Airport Road         1425
Kammanahalli             1329
Kalyan Nagar             1309
Basavanagudi             1266
Sarjapur Road            1261
Electronic City          1229
Bellandur                1227
Frazer Town              1185
Malleshwaram             1096
Rajajinagar              1079
Banashankari              863
New BEL Road              740
Name: listed_in(city), dtype: int64
```

In [ ]:

```
import matplotlib. pyplot as plt
plt.figure(figsize=(10,8))
x = df.location.value_counts()[:15]
y = df['location'].value_counts()[:15].index
plt.pie(x, labels=y, autopct='%.2f')
plt.title('Distribution of restaurants among various locations')
plt.show()
```



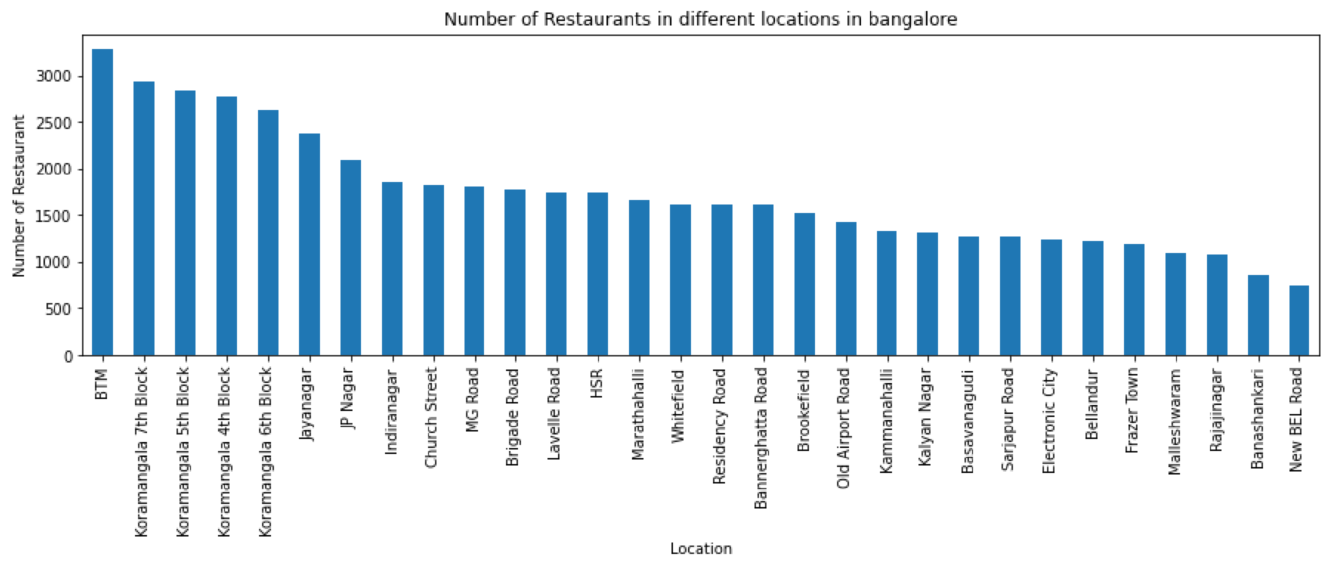


In [ ]:

```
plt.figure(figsize=(15,4))
k =df['listed_in(city)'].value_counts()
k.plot(kind='bar')
plt.title('Number of Restaurants in different locations in bangalore')
plt.xlabel('Location')
plt.ylabel('Number of Restaurant')
```

Out[ ]:

```
Text(0, 0.5, 'Number of Restaurant')
```



Conclusion- There is a variation in restaurants as per the locations. BTM has the highest number of the restaurants in Bangalore that 3108 restarants. New BEL Road contains the least number of resturants followed by banashankari. Btm has 17.24% of the total restaurants in bangalore

Analysis on online order

In [ ]:

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

Out[ ]:

```
Yes     30444
No      21273
```

```
Name: online_order, dtype: int64
```

In [ ]:

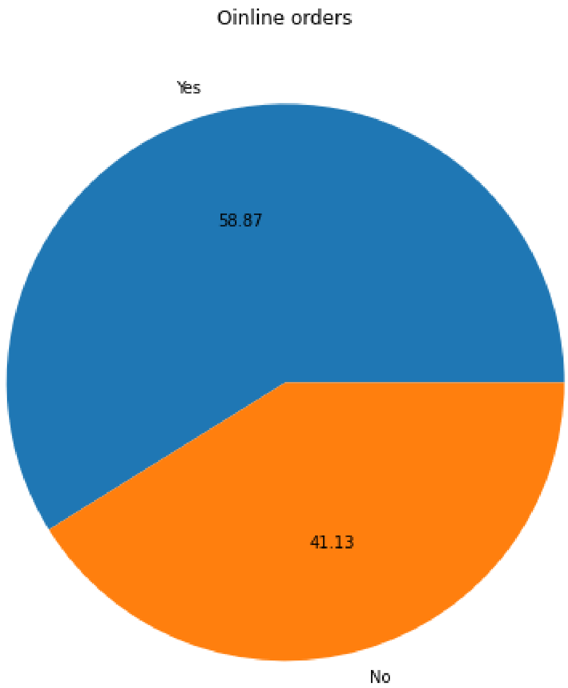
```
plt.figure(figsize=(10,5))
k =df['online_order'].value_counts()
k.plot(kind='bar')
plt.title('Number of Restaurants that accepts online orders')
plt.xlabel('online orders')
plt.ylabel('counts')
```

Out[ ]:

```
Text(0, 0.5, 'counts')
```

In [ ]:

```
import matplotlib. pyplot as plt
plt.figure(figsize=(10,8))
x = df.online_order.value_counts()[:15]
y = df['online_order'].value_counts()[:15].index
plt.pie(x, labels=y, autopct='%.2f')
plt.title('Oinline orders')
plt.show()
```



Conclusion- Number of restaurants that allows online order are more than those restaurants who don't allows online order. There are 29342 restaurants in bangalore which are accpeting the online orders and 20098 restaurants which don't accepts the online order. There are 59.65% of restaurants that allows online ordering

Analysis on ratings

In [ ]:

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

Out[ ]:

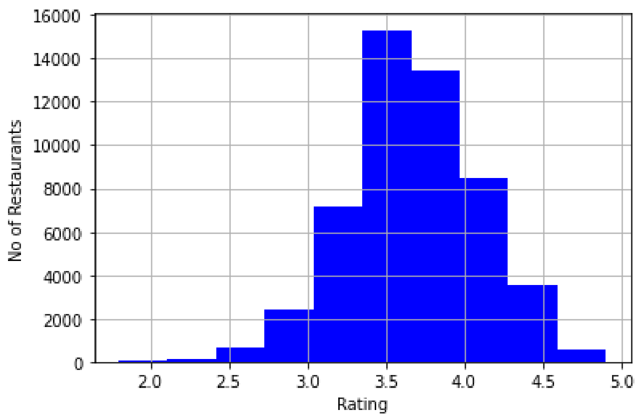
```
3.5    5450
3.6    5137
3.7    4871
3.4    4713
3.8    4337
3.9    4200
3.3    3402
4.0    3302
4.1    2991
4.2    2198
3.2    2113
4.3    1707
3.1    1614
4.4    1147
3.0    1029
2.9     802
4.5     656
2.8     600
2.7     307
4.6     302
2.6     260
4.7     170
2.5     101
2.4      70
4.8      66
4.9      55
2.3      51
2.2      26
2.1      24
2.0      11
1.8       5
Name: rate, dtype: int64
```

In [ ]:

```
df['rate'] = df['rate'].apply(lambda r: float(r))
df['rate'].hist(color='blue')
plt.xlabel('Rating')
plt.ylabel('No of Restaurants')
```

Out[ ]:

```
Text(0, 0.5, 'No of Restaurants')
```



In [ ]:

```
print(df['rate'].min())
print(df['rate'].max())
print(df['rate'].mean())
```

```
1.8
4.9
3.6652841425448277
```

Conclusion- Majority of restaurants has ratings between 3.6 to 3.9. 15% of the restaurnts have an approx rating of 3.7 . Minimum rating for the restaurants is 1.8 . There is not even a single restaurant in bangalore where rating is equal to 5.

How many stores are there for each restaurants

In [ ]:

```
df['name'].value_counts()
```

Out[ ]:

```
Cafe Coffee Day       96
Onesta                85
Just Bake             73
Empire Restaurant     71
Five Star Chicken     70
                      ..
Lucky Singh & Co       1
Natis By Wings         1
HVR Veg                1
Curry Chutney          1
Flavors                1
Name: name, Length: 8792, dtype: int64
```

In [ ]:

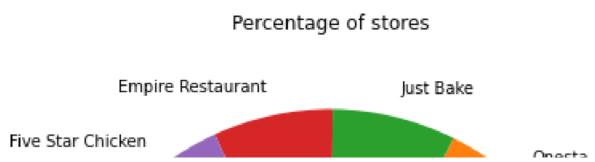
```
df['name'].value_counts()
```

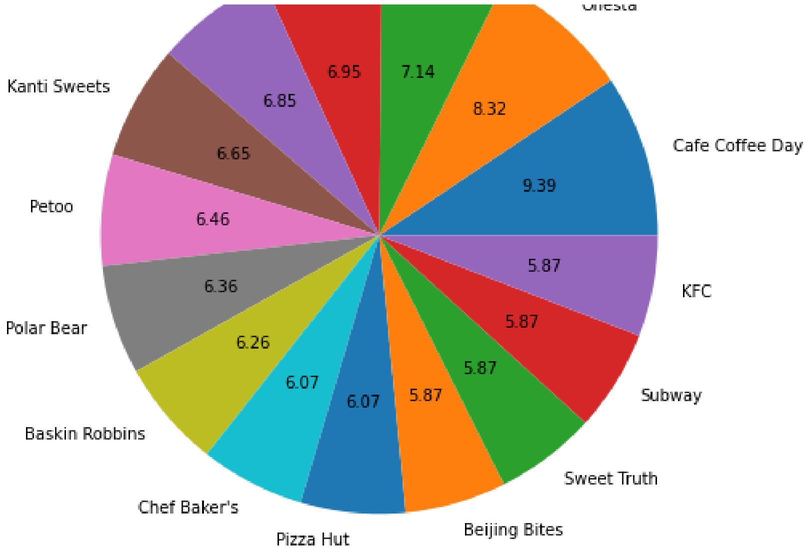
Out[ ]:

```
Cafe Coffee Day       96
Onesta                85
Just Bake             73
Empire Restaurant     71
Five Star Chicken     70
                      ..
Lucky Singh & Co       1
Natis By Wings         1
HVR Veg                1
Curry Chutney          1
Flavors                1
Name: name, Length: 8792, dtype: int64
```

In [ ]:

```
import matplotlib. pyplot as plt
plt.figure(figsize=(10,8))
x = df.name.value_counts()[:15]
y = df['name'].value_counts()[:15].index
plt.pie(x, labels=y, autopct='%.2f')
plt.title('Percentage of stores')
plt.show()
```



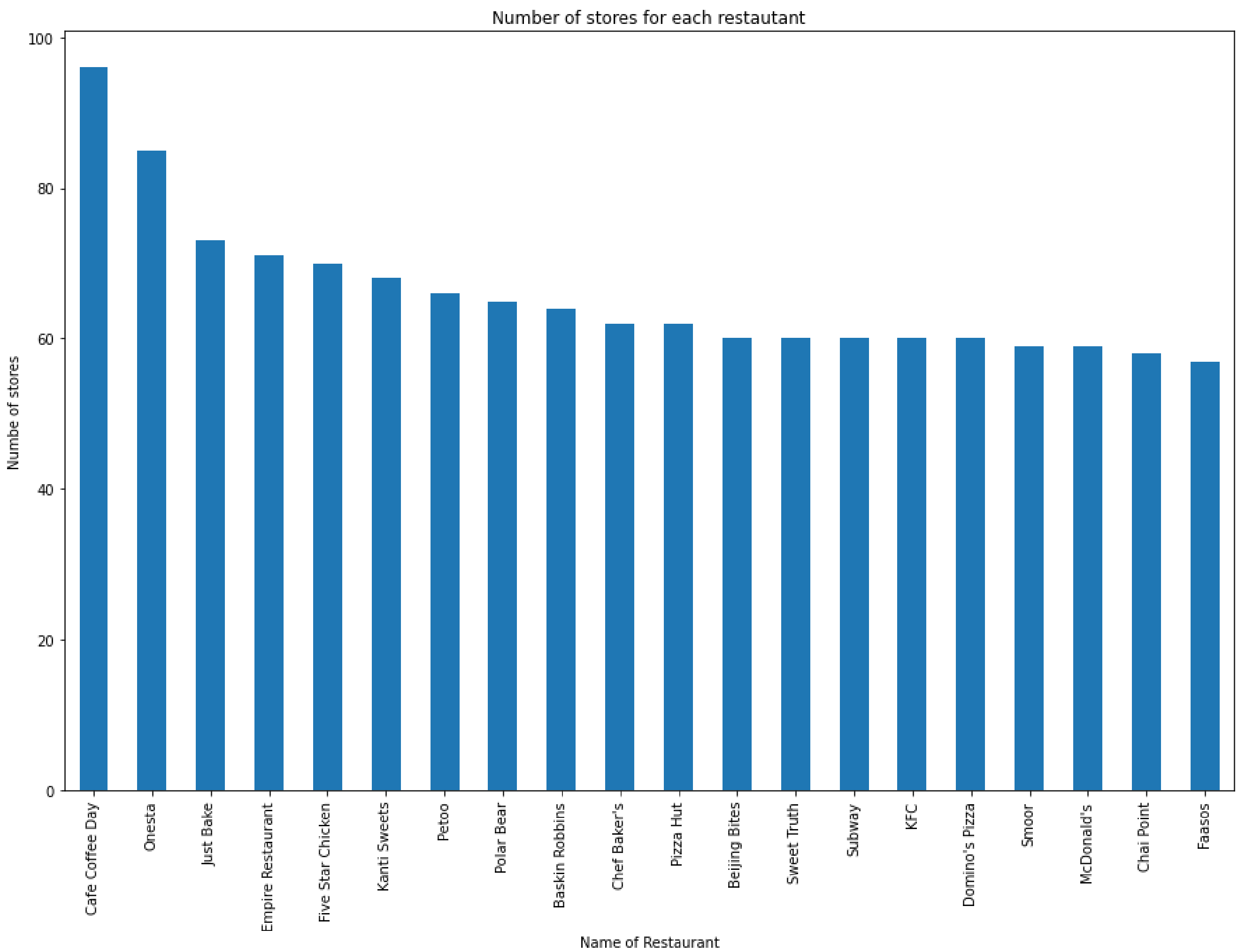


In [ ]:

```
plt.figure(figsize = (15,10))
k = df.name.value_counts()[:20]
k.plot(kind = 'bar')
plt.xlabel("Name of Restaurant")
plt.ylabel("Numbe of stores")
plt.title("Number of stores for each restautant")
```

Out[ ]:

```
Text(0.5, 1.0, 'Number of stores for each restautant')
```



Conclusion- There is a variation in the number of stores in bangalore. CCD has maximum number of stores in bangalore followed by onesta and just bake. There are various restaurants that are having only 1 stores such as SV Juice Corner Tiffun, Brown box etc. The total no. of stores of CCD compromised of 9.26 % of the entire stores present in bangalore

Restaurants allows booking of tables

In [ ]:

```
df['book_table'].value_counts()
```

Out[ ]:

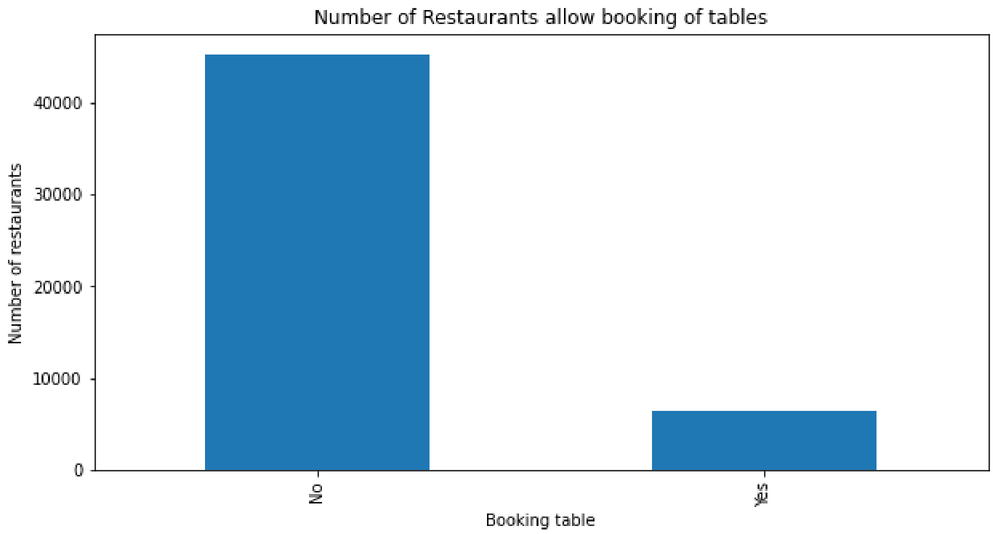
```
No     45268
Yes     6449
Name: book_table, dtype: int64
```

In [ ]:

```
plt.figure(figsize=(10,5))
k =df['book_table'].value_counts()
k.plot(kind='bar')
plt.title('Number of Restaurants allow booking of tables')
plt.ylabel('Number of restaurants')
plt.xlabel('Booking table')
```

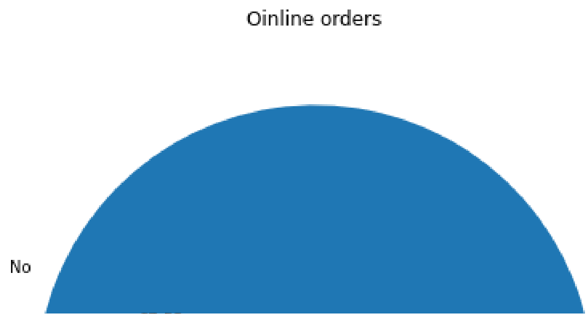
Out[ ]:

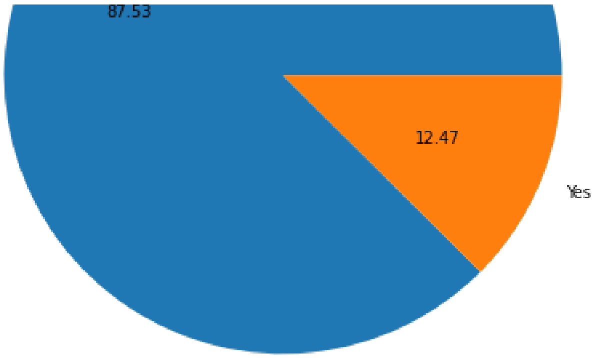
```
Text(0.5, 0, 'Booking table')
```



In [ ]:

```
import matplotlib. pyplot as plt
plt.figure(figsize=(10,8))
x = df.book_table.value_counts()
y = df['book_table'].value_counts().index
plt.pie(x, labels=y, autopct='%.2f')
plt.title('Oinline orders')
plt.show()
```





Conclusion- There are 43120 restaurants that are accepting the booking of table and 6320 restarants that are not accepting the booking of table. Majority of restaurants may be street food type restaurant as it is not allowing booking of table. 87.22% of the restaurants are not allowing the booking of tables

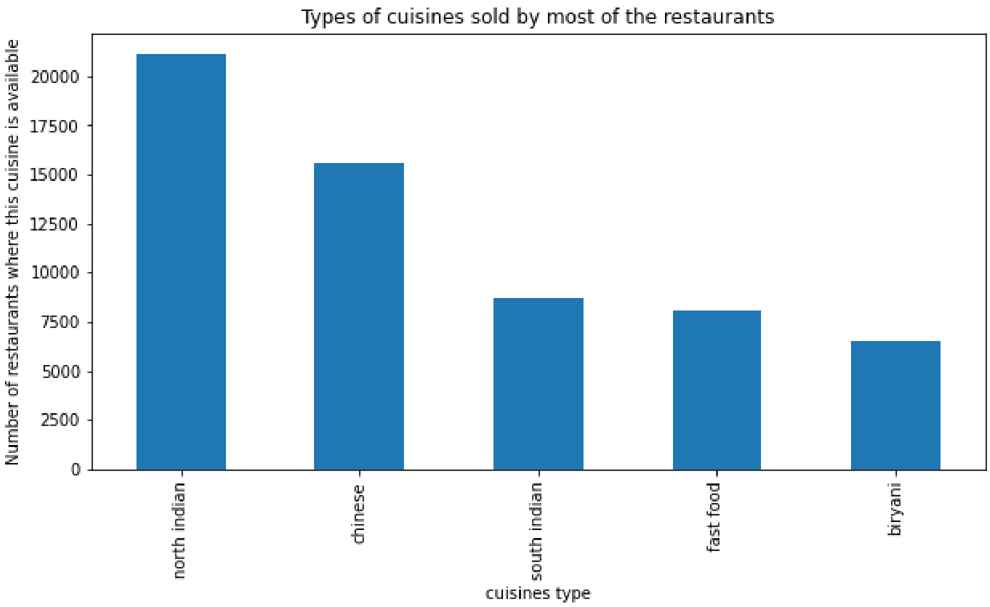
Types of cuisines sold by most of the restaurants

In [ ]:

```
total=[]
k = df[df['cuisines'].notnull()]
k['cuisines'] = k['cuisines'].apply(lambda x:x.lower().strip())
for i in k['cuisines']:
    for j in i.split(','):
        j = j.strip()
        total.append(j)
plt.figure(figsize=(10,5))
a=pd.Series(total).value_counts()[:5]
a.plot(kind='bar')
plt.title('Types of cuisines sold by most of the restaurants')
plt.xlabel('cuisines type')
plt.ylabel('Number of restaurants where this cuisine is available')
```

Out[ ]:

```
Text(0, 0.5, 'Number of restaurants where this cuisine is available')
```



Conclusion - North indian and chinese are the two most sold cusines in bangalore. Number of restaurants where north indian cuisine is available is close to 20,000 and number of restaurants where chinese food is available is close to 14,000.

Items liked by peoples in Bangalore

Items liked by peoples in Bangalore

In [ ]:

```
df['dish_liked'].nunique()
```

Out[ ]:

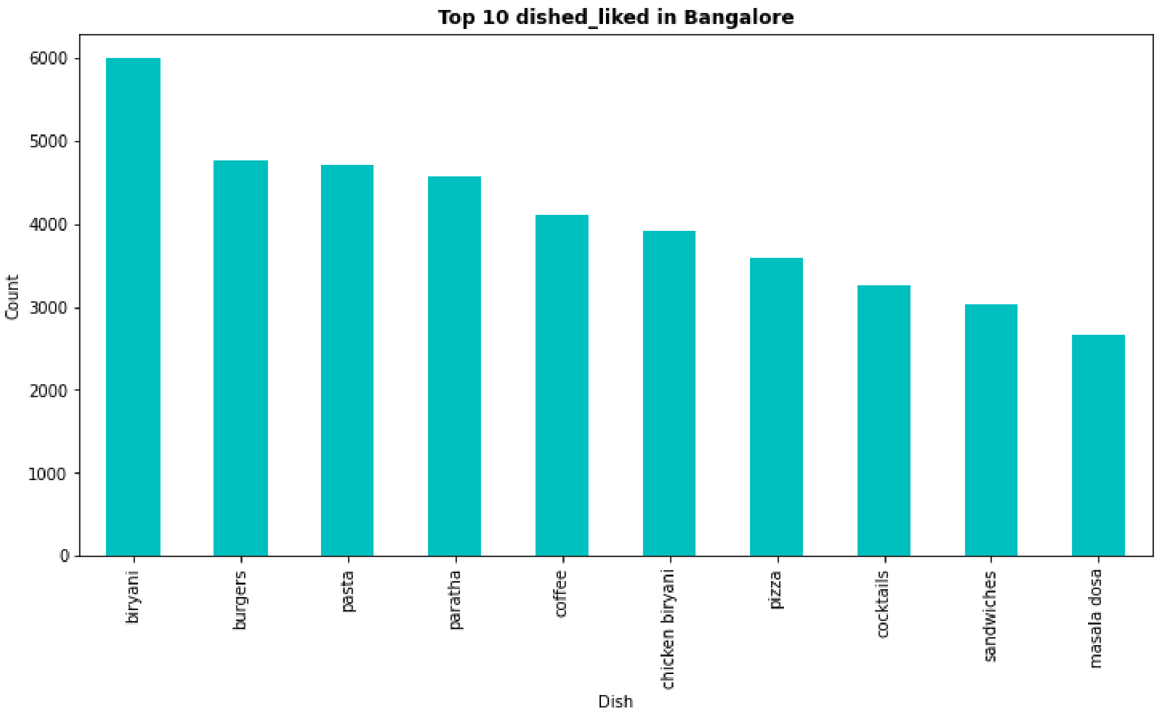
```
5271
```

In [ ]:

```
dishes_data = df[df.dish_liked.notnull()]
dishes_data.dish_liked = dishes_data.dish_liked.apply(lambda x:x.lower().strip())
dish_count = []
for i in dishes_data.dish_liked:
    for t in i.split(','):
        t = t.strip() # remove the white spaces to get accurate results
        dish_count.append(t)
plt.figure(figsize=(12,6))
pd.Series(dish_count).value_counts()[0:10].plot(kind='bar',color= 'c')
plt.title('Top 10 dished_liked in Bangalore',weight='bold')
plt.xlabel('Dish')
plt.ylabel('Count')
```

Out[ ]:

```
Text(0, 0.5, 'Count')
```



Conclusion- Biryani is the most liked dish by the peoples of bangalore. There are around 12000 restaunts where biryani is one of the mpst famous recipe. Chicken is the second most famous dish liked in bangalore

Analysis on cost of dining

In [ ]:

```
df['approx_cost(for two people)']
```

Out[ ]:

```
6        800.0
19       400.0
22       900.0
24       300.0
25       600.0
```

```
        ...
50032    250.0
50050    200.0
50135    650.0
50571    500.0
51386    500.0
Name: approx_cost(for two people), Length: 51717, dtype: float64
```

In [ ]:

```
df['approx_cost(for two people)'].max()
```

Out[ ]:

```
6000.0
```

In [ ]:

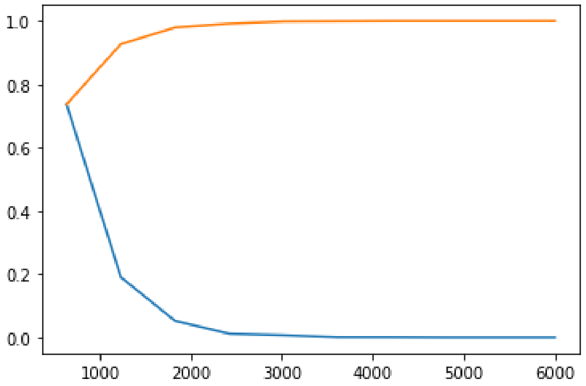
```
counts, bin_edges = np.histogram(df['approx_cost(for two people)'], bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

```
[7.36798345e-01 1.90285593e-01 5.24779086e-02 1.18336330e-02
 7.07697662e-03 7.73440068e-04 6.57424058e-04 3.86720034e-05
 1.93360017e-05 3.86720034e-05]
[  40.   636. 1232. 1828. 2424. 3020. 3616. 4212. 4808. 5404. 6000.]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fd2cc609390>]
```

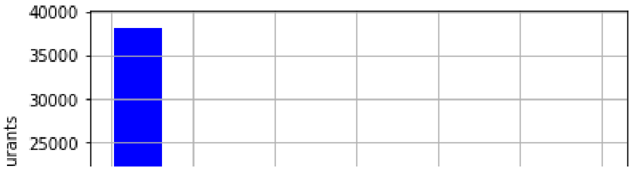


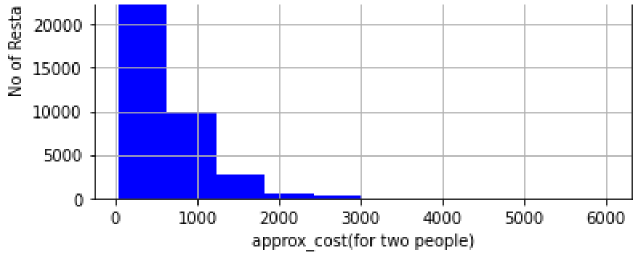
In [ ]:

```
df['approx_cost(for two people)'] = df['approx_cost(for two people)'].apply(lambda r: float(r))
import matplotlib.pyplot as plt
df['approx_cost(for two people)'].hist(color="blue")
plt.xlabel('approx_cost(for two people)')
plt.ylabel('No of Restaurants')
```

Out[ ]:

```
Text(0, 0.5, 'No of Restaurants')
```





In [ ]:

```
df['approx_cost(for two people)'].mean()
```

Out[ ]:

```
555.4315664479955
```

Conclusion- Majority of restaurants in bangalore has average cost for 2 person is 561. The minimum cost for the dining is 40 and maximum cost is 6000. It concludes that there are all sorts of food at different prices are available in bangalore

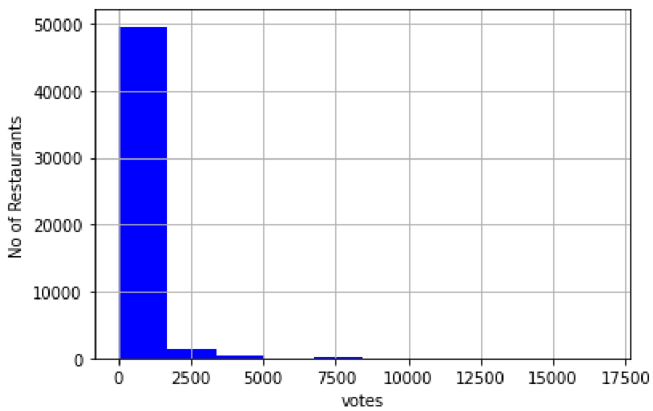
Votes

In [ ]:

```
import matplotlib.pyplot as plt
df['votes'].hist(color="blue")
plt.xlabel('votes')
plt.ylabel('No of Restaurants')
```

Out[ ]:

```
Text(0, 0.5, 'No of Restaurants')
```



In [ ]:

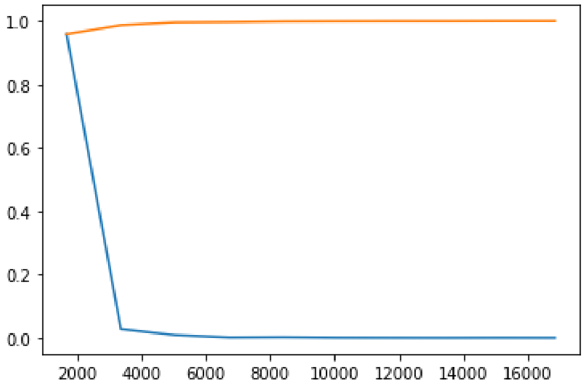
```
counts, bin_edges = np.histogram(df['votes'], bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

```
[9.58794980e-01 2.80372025e-02 8.75920877e-03 1.19883211e-03
 1.93360017e-03 5.41408048e-04 2.51368022e-04 3.86720034e-05
 3.28712029e-04 1.16016010e-04]
[    0.   1683.2  3366.4  5049.6  6732.8  8416.  10099.2 11782.4 13465.6
 15148.8 16832. ]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fd2cbf7ea20>]
```



In [ ]:

```
print(df['votes'].mean())
print(df['votes'].min())
print(df['votes'].max())
```

```
283.69752692538236
0
16832
```

Conclusion The restaurants in Bangalore has an average vote of 296.76 . Minimum vote for the restaurant is 0 and the maaximum votes are 16832. Very few restaurants in bangalore has no. of votes greater than 1700

Rating of restaurants vs online_order

In [ ]:

```
df['rate']
```

Out[ ]:

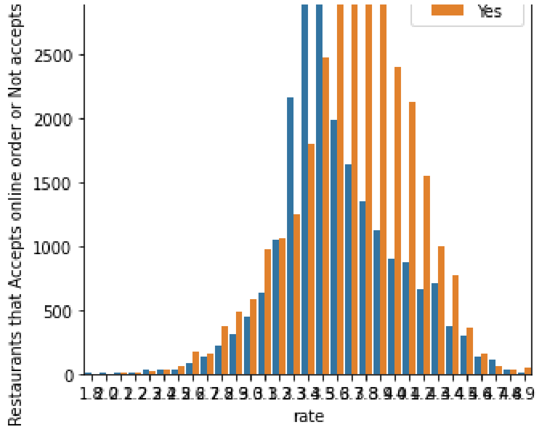
```
6        3.6
19       3.1
22       3.6
24       3.7
25       3.2
        ...
50032    3.4
50050    3.5
50135    3.5
50571    3.5
51386    3.5
Name: rate, Length: 51717, dtype: float64
```

In [ ]:

```
import seaborn as sns
plt.figure(figsize = (5,5))
sns.countplot(x=df['rate'], hue = df['online_order'])
plt.ylabel("Restaurants that Accepts online order or Not accepts online orders")
plt.xlabel('rate')
plt.title("rating of restaurant vs oline order")
```

Out[ ]:

```
Text(0.5, 1.0, 'rating of restaurant vs oline order')
```



Conclusion - Only for those restaurants whose rating is 3.7, the number of restaurants accepting online order is more than the restaurants who don't accepts the online order. For all the restaurants (whose rating is other than 3.7), there are mpre no. of restaurants that accepts online order rather than the restaurants who don't accepts the online order.

In [ ]:

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

Out[ ]:

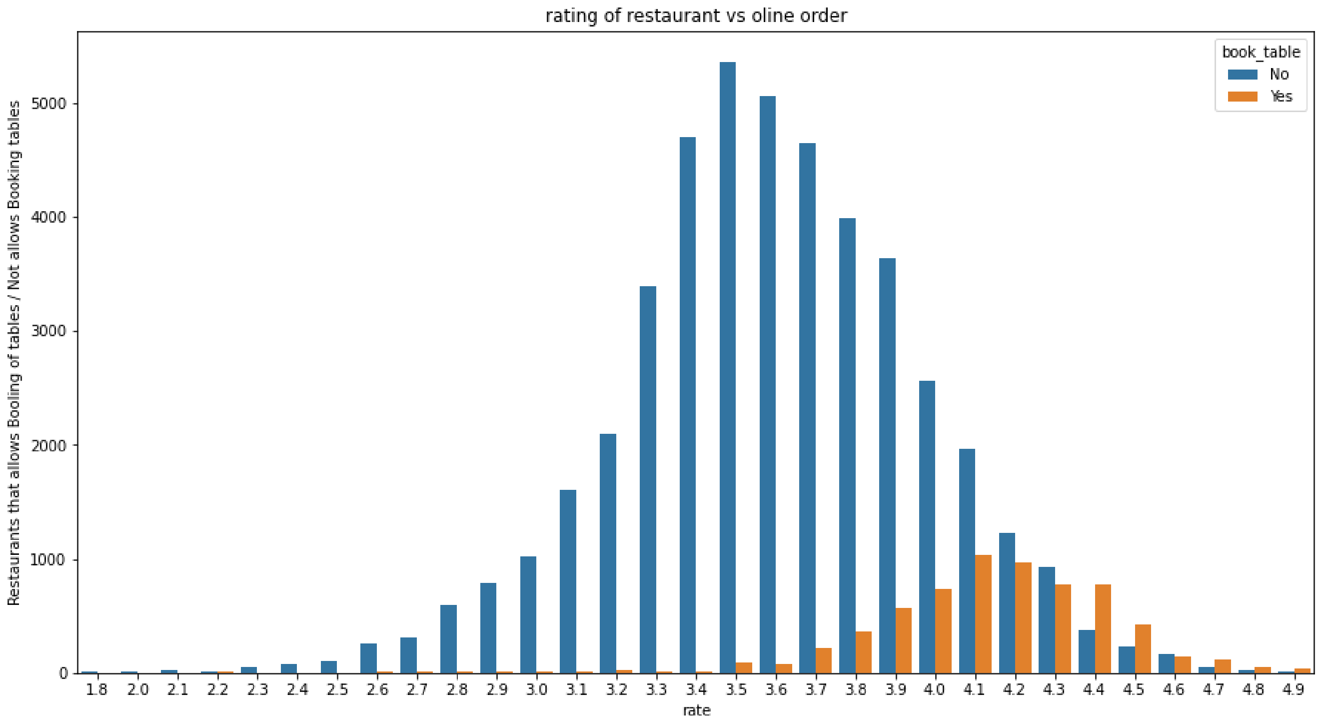
```
3.6652841425448277
```

In [ ]:

```
import seaborn as sns
plt.figure(figsize = (15,8))
sns.countplot(x=df['rate'], hue = df['book_table'])
plt.ylabel("Restaurants that allows Booling of tables / Not allows Booking tables")
plt.xlabel('rate')
plt.title("rating of restaurant vs oline order")
```

Out[ ]:

```
Text(0.5, 1.0, 'rating of restaurant vs oline order')
```



Coclusion- The maximum no. restaurants that allows table booking has an average rating of 4.2 . The maximum number of restaurants, which dont allows table booking has an average rating of 3.7 . Irrespective of ratings, the number of restaurants

that allows booking of tables are less than the restaurants which don;t allows that.

Type of restaurant

In [ ]:

```
df['listed_in(type)'].value_counts()
```

Out[ ]:

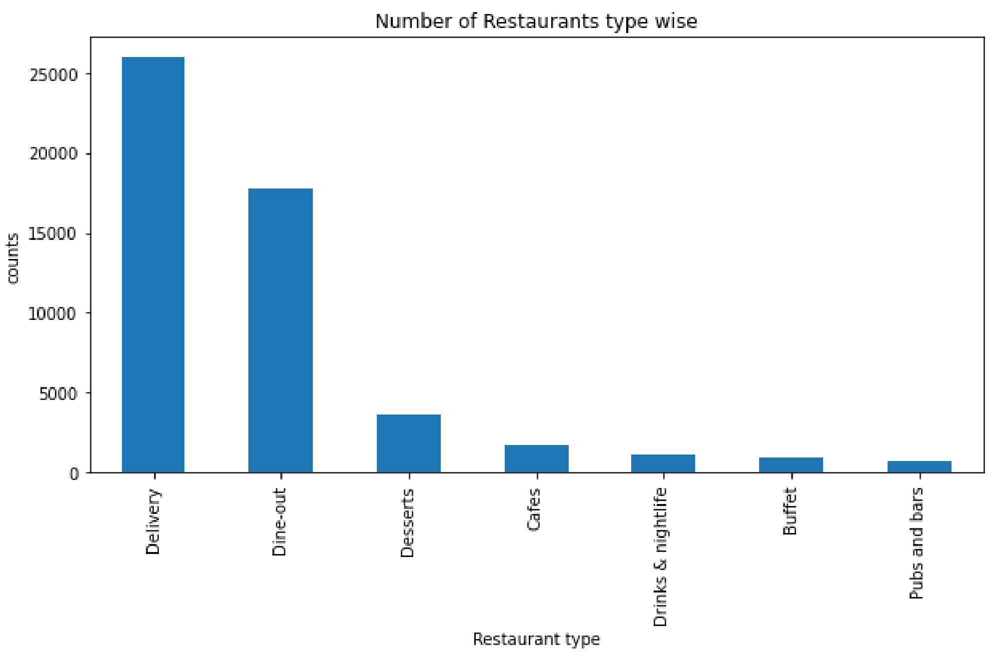
```
Delivery              25942
Dine-out              17779
Desserts               3593
Cafes                  1723
Drinks & nightlife     1101
Buffet                  882
Pubs and bars           697
Name: listed_in(type), dtype: int64
```

In [ ]:

```
plt.figure(figsize=(10,5))
k =df['listed_in(type)'].value_counts()
k.plot(kind='bar')
plt.title('Number of Restaurants type wise')
plt.xlabel('Restaurant type')
plt.ylabel('counts')
```

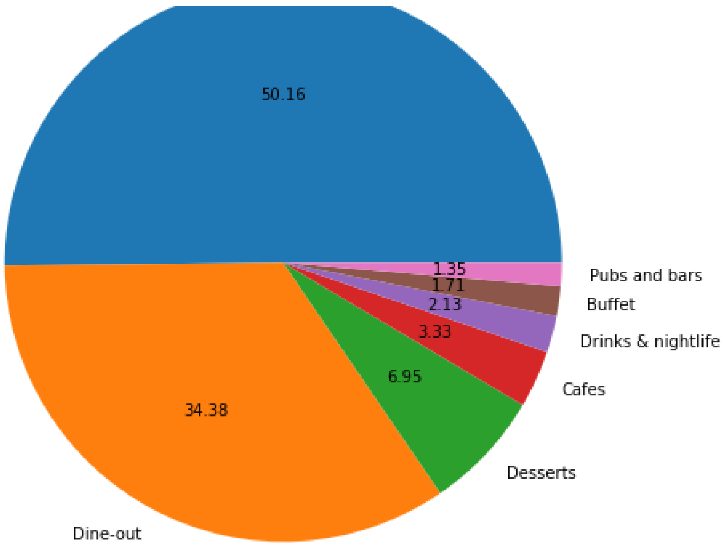
Out[ ]:

```
Text(0, 0.5, 'counts')
```



In [ ]:

```
import matplotlib. pyplot as plt
plt.figure(figsize=(10,8))
x = df['listed_in(type)'].value_counts()[:15]
y = df['listed_in(type)'].value_counts()[:15].index
plt.pie(x, labels=y, autopct='%.2f')
plt.title('Distribution of restaurants based on its types')
plt.show()
```



Conclusion - Aound 50% of the restaurants in bangalore belongs to the delivery type of restaurants. The least type of restaurants in bangalore belongs to pubs and bars, buffet, drinks and nightlife. Also there are lot of restaurants (34%) which allows dine-out service. In total there ar 24728 restaurants that belongs to delivery type. The number of Pubs and bar is 669 whihc the minimum among all the types of restaurants

Pairplot

In [ ]:

```
df2=df.copy()
```

In [ ]:

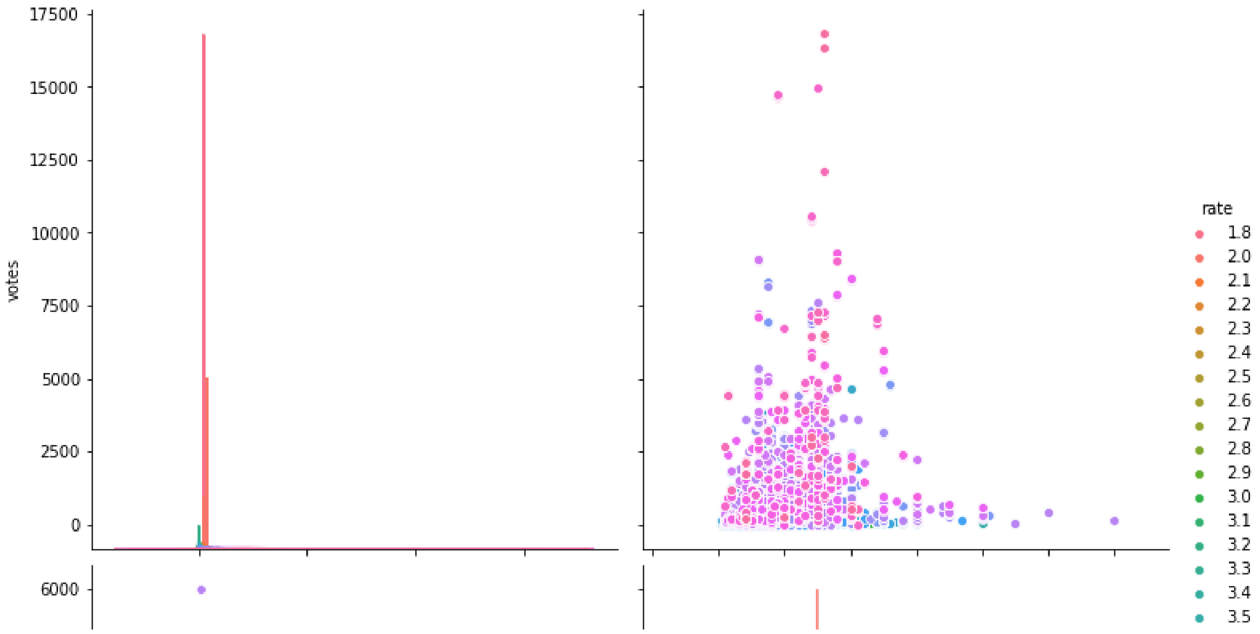
```
df2['approx_cost(for two people)'].dtype
```

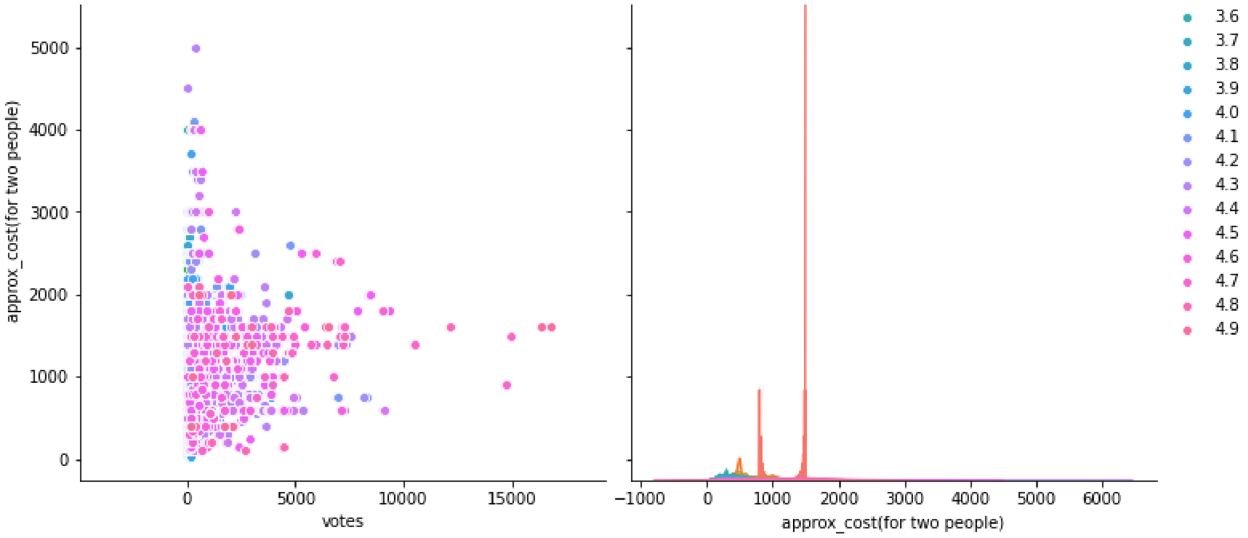
Out[ ]:

```
dtype('float64')
```

In [ ]:

```
sns.pairplot(df,hue="rate",size=5)
plt.show()
```





Conclusion from this pairplot

1. **In the plot of votes vs rate, most of restaurants having higher no. of votes has better ratings also**
2. **In the plot of approx_cost vs rate, the restaurant whose rating is high has more price.**
3. **In the graph of rate vs cost, rate vs votes, the data points are linearly separable**

EDA Summary

1. **BTM alone has 3108 restarants which is the highest number of Restaurants in Bangalore as compared to any other location. BEL has the least Number of restaurants ie. 725. Number of restaurants in BTM comprise of 17% of total restaurants**
2. **The number of restaurants that takes online order is more than those which don't accepts online order. There are more 29342 restaurants that are accepting online orders and there are 20098 restaurants that are not accepting online order**
3. **There is a variation in ratings of restarants between 1.8 to 4.9. The average rating of restaurants is 3.7.**
4. **CCD has 93 stores in bangalore which the highest number of stores for any restaurant in bangalore follwed by onesta having 85 restaurants.**
5. **There are 43120 restaurants that are accepting the booking of table and 6320 restarants that are not accepting the booking of table. Majority of restaurants may be street food type restaurant as it is not allowing booking of table**
6. **North Indian, Chinese and South indian are the top 3 cuisines avalable in the most of restaurants.**
7. **Chicken is the most liked dish by the peoples of bangalore followed by Biryani and rice.**
8. **The average cost of restaurants for the dining is 561. Minimum cost is 40 and max cost is 4000. Overall, 87.22% of the restaurants are not allowing the booking of tables**
9. **Only for those restaurants whose rating is 3.7, the number of restaurants accepting online order is more than the restaurants who don't accepts the online order. For all the other restaurants (whose rating is other than 3.7), there are mpre no. of restaurants that accepts online order rather than the restaurants who don't accepts the online order.**
10. **Aound 50% of the restaurants in bangalore belongs to the delivery type of restaurants. The least type of restaurants in bangalore belongs to pubs and bars, buffet, drinks and nightlife. Also there are lot of restaurants (34%) which allows dine-out service. In total there ar 24728 restaurants that belongs to delivery type. The number of Pubs and bar is 669 whihc the minimum among all the types of restaurants**
11. **The maximum no. restaurants that allows table booking has an average rating of 4.2 . The maximum number of restaurants, which dont allows table booking has an average rating of 3.7 . Irrespective of ratings, the number of restaurants that allows booking of tables are less than the restaurants which don;t allows that.**

Checking for multicolliearity

In [ ]:

```
#https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
```

```
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [ ]:

```
k=df.copy()
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
T = LabelEncoder()
k['location'] = T.fit_transform(k['location'])
k['rest_type'] = T.fit_transform(k['rest_type'])
k['cuisines'] = T.fit_transform(k['cuisines'])
```

In [ ]:

```
k['dish_liked'] = T.fit_transform(k['dish_liked'])
```

In [ ]:

```
k['online_order'] = T.fit_transform(k['online_order'])
k['listed_in(city)'] = T.fit_transform(k['listed_in(city)'])
k['book_table'] = T.fit_transform(k['book_table'])
k['listed_in(type)'] = T.fit_transform(k['listed_in(type)'])
```

In [ ]:

```
k=k.drop(['menu_item','reviews_list'],axis=1)
```

In [ ]:

```
k=k.drop(['name','rate'],axis=1)
```

In [ ]:

```
k.head(1)
```

Out[ ]:

	online_order	book_table	votes	location	rest_type	cuisines	approx_cost(for two people)	listed_in(type)	listed_in(city)	dish_liked
6	0	0	8	57	27	2210	800.0	0	1	3767

In [ ]:

```
calc_vif(k)
```

Out[ ]:

	variables	VIF
0	online_order	2.159482
1	book_table	1.936241
2	votes	1.399276
3	location	3.056287
4	rest_type	4.780746
5	cuisines	4.492455
6	approx_cost(for two people)	4.388961

	variables	VIF
7	listed_in(type)	5.970487
8	listed_in(city)	4.022720
9	dish_liked	3.281583

Conclusion - Hence by ananlyzing the vif values, we can conclude that there is no multicollinearlity between any independent variables beacause the vif values are very small for each of the independent variables.

Feature Engineering

In [ ]:

```
df.head(2)
```

Out[ ]:

	name	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_lis
6	Rosewood International Hotel - Bar & Restaurant	No	No	3.6	8	Mysore Road	Casual Dining	North Indian, South Indian, Andhra, Chinese	800.0	[('Rated 5.0', 'RATED\n Awesome food ?? Great ...
19	360 Atoms Restaurant And Cafe	Yes	No	3.1	13	Banashankari	Cafe	Cafe, Chinese, Continental, Italian	400.0	[('Rated 5.0', 'RATED\n Friendly staffs , nic...

1. **Total No. of cuisines available in each of the restaurant**

In [ ]:

```
df['number_of_cuisines']=df['cuisines'].str.split(',').apply(len)
```

1. **Total number of dishes liked by the customers. It may be directly proportional to the rating**

In [ ]:

```
df['number_of_liked_dishes']=df['dish_liked'].str.split(',').apply(len)
```

1. **Facilities offered by restaurants - there are 2 major facilities that a restaurant can proive is obline order and booking tables. so, here we are summing both of them to find the overall quality of service by the restaurant.**

In [ ]:

```
df['Facilities_offered']=k['online_order']+k['book_table']
```

In [ ]:

```
df.head(2)
```

Out[ ]:

	name	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_lis

	name	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_list
6	Rosewood International Hotel - Bar & Restaurant	No	No	3.6	8	Mysore Road	Casual Dining	North Indian, South Indian, Andhra, Chinese	800.0	[('Rated 5.0', 'RATED\n Awesome food ?? Great ...
19	360 Atoms Restaurant And Cafe	Yes	No	3.1	13	Banashankari	Cafe	Cafe, Chinese, Continental, Italian	400.0	[('Rated 5.0', 'RATED\n Friendly staffs , nic...

1. This function is used to convert categorical features into response coded features. It simply perform MEAN VALUE REPLACEMENT.

In [ ]:

```
# # https://www.geeksforgeeks.org/python-creating-a-pandas-dataframe-column-based-on-a-given-condi
tion/

key_dict = dict()
def provide_response_coded_features(groupByVal,columnName, df):
        ## distribute values group by n take mean of rate column
    mean_df = df.groupby([groupByVal]).mean()
    ## stored in dict data type.. key is column name and values in mean value of rate column.
    mean_dict =mean_df['rate'].to_dict()
    key_dict.update([ (groupByVal, mean_dict) ] )
    for k, v in mean_dict.items():
            mean_dict[k] = round(v,2)
    df[columnName] = df[groupByVal].map(mean_dict)
    return df
```

In [ ]:

```
# create response coded feature for dish_liked feature.

mean_dish_liked =provide_response_coded_features('dish_liked','mean_dish_liked',df)
mean_dish_liked[['rate','dish_liked','mean_dish_liked']][:10]
```

Out[ ]:

	rate	dish_liked	mean_dish_liked
6	3.6	Pav Bhaji, Masala Dosa, Idli Vada, Filter Coff...	3.70
19	3.1	Pasta, Gelato, Garlic Bread, Mojito, Nachos, P...	3.59
22	3.6	Sandwich, Omelette, Ice Tea, Virgin Mojito, Ho...	3.62
24	3.7	Waffles, Pasta, Crispy Chicken, Honey Chilli C...	3.64
25	3.2	Waffles, Pasta, Crispy Chicken, Honey Chilli C...	3.64
26	3.8	Masala Dosa, Idli, Filter Coffee, Medu Dosa	3.65
27	3.3	Cappuccino, Sausage Roll, Chicken Sandwich, Ch...	3.52
28	3.3	Sandwich, Omelette, Ice Tea, Virgin Mojito, Ho...	3.62
32	3.9	Cup Cake, Chocolate Cake	3.78
36	2.8	Chicken Biryani	3.47

In [ ]:

```
mean_cuisines =provide_response_coded_features('cuisines','mean_cuisines',df)
mean_cuisines[['rate','cuisines','mean_cuisines']][:10]
```

Out[ ]:

	rate	cuisines	mean_cuisines
6	3.6	North Indian, South Indian, Andhra, Chinese	3.90
19	3.1	Cafe, Chinese, Continental, Italian	3.69
22	3.6	Cafe, Fast Food	3.39
24	3.7	Cafe	3.64
25	3.2	Cafe, Bakery	3.82
26	3.8	Cafe, South Indian	3.78
27	3.3	Cafe, Fast Food, Beverages	3.70
28	3.3	Cafe, Fast Food	3.39
32	3.9	Bakery, Desserts	3.65
36	2.8	North Indian, Chinese, Fast Food	3.46

In [ ]:

```
def return_dict_mean_value(query_feature):

    result_dict=dict()

    for feature_name, values in key_dict.items():
        if feature_name == query_feature:
            for key in values:
                result_dict.update([ (key, values[key]) ] )

                print(key + ':', values[key])
    return result_dict
return_dict_mean_value('online_order')
```

Out[ ]:

```
{}
```

In [ ]:

```
dict_cuisines = return_dict_mean_value('cuisines')
dict_dish_liked = return_dict_mean_value('dish_liked')
```

In [ ]:

```
df['mean_cuisines'] = df['cuisines'].map(dict_cuisines)
df['mean_dish_liked'] = df['dish_liked'].map(dict_dish_liked)
```

In [ ]:

```
df[['rate','mean_dish_liked']]
```

Out[ ]:

	rate	mean_dish_liked
6	3.6	3.70
19	3.1	3.59
22	3.6	3.62
24	3.7	3.64
25	3.2	3.64
...	...	...
50032	3.4	3.40

	rate	mean_dish_liked
50050	3.5	3.50
50135	3.5	3.50
50571	3.5	3.41
51386	3.5	3.41

51717 rows × 2 columns

Feature Engineering Summary

1. Mean value replacement for dish_liked - Here, first we have done response coding followed by mean value replacement for dish_liked colimn. We found its value is almost simillar to the rate column
2. Mean value replacement for cuisines - Here also, first we have done response coding followed by mean value replacement for cuisisnes column.
3. Number of cuisines available- This column contains the total number of cuisines available in each restaurants
4. Number of dish_liked - This column contains the total number of dishes liked by the customers in each restaurants.
5. Facilities offered - If the restarant is allowing both online_order and booking_table, then we have given the facilities offered values as 2. If restarant is allowing either of the them, then we've given the values as 1. If the restarant os not allowing any of the faciilities, then we've given the value as 0.

In [ ]:

```
df=df.drop(['name','menu_item'],axis=1)
```

In [ ]:

```
df.head(2)
```

Out[ ]:

	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(typ
6	No	No	3.6	8	Mysore Road	Casual Dining	North Indian, South Indian, Andhra, Chinese	800.0	[('Rated 5.0', 'RATED\n Awesome food ?? Great ...	Buffet
19	Yes	No	3.1	13	Banashankari	Cafe	Cafe, Chinese, Continental, Italian	400.0	[('Rated 5.0', 'RATED\n Friendly staffs , nic...	Cafes

Preprocessing of Features

In [ ]:

```
df['dish_liked'] = df['dish_liked'].str.replace(',','')
df['cuisines'] = df['cuisines'].str.replace(',','')
```

In [ ]:

```
df.head(1)
```

Out[ ]:

	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(type)	listed_

	online_order	book_table	rate	votes	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(type)	listed_
6	No	No	3.6	8	Mysore Road	Casual Dining	North Indian South Indian Andhra Chinese	800.0	[('Rated 5.0', 'RATED\n Awesome food ?? Great ...	Buffet	Banash

In [ ]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:

```
from bs4 import BeautifulSoup
# Combining all the above stundents
from tqdm import tqdm
import re
# tqdm is for printing the status bar
word_counter = []

from nltk.corpus import stopwords

def filterised_text(text):
    preprocessed_text = []
    for sentance in tqdm(text):
        sentance = re.sub('[0-9]+', '', sentance)
        sentance = re.sub('[^A-Za-z0-9]+', ' ', sentance)
        sentance = re.sub(r"http\S+", "", sentance)
        sentance = BeautifulSoup(sentance, 'lxml').get_text()
        sentance = decontracted(sentance)
        sentance = re.sub("\S*\d\S*", "", sentance).strip()
        sentance = re.sub('[^A-Za-z]+', ' ', sentance)
        sentance = ' '.join(word.lower() for word in sentance.split() if len(word)>1 and word.lower
() not in stopwords.words('english'))
        sentance = re.sub(r"rated", "", sentance)
        count = len(sentance.split())
```

```
        word_counter.append(count)
        preprocessed_text.append(sentance.strip())
    return preprocessed_text



def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

preprocessed_reviews = filterised_text(df['reviews_list'].astype(str).values)
df['preprocessed_reviews']  = preprocessed_reviews
preprocessed_reviews[1822]
```

```
100%|██████████| 51717/51717 [2:03:16<00:00,  6.99it/s]
```

Out[ ]:

```
'must try chicken biryani ngot delivery ubereats promotion going regarding biryani taste albeit go
od biryani quantity wise extremely less consider ordering starter usually finish starter biryani a
lone ur ordering biryani may quantity less coz promotion nanyhow interaction staff fast delivery g
ood food   worst restuatant went check place green glen none even bothered ask needed interested
finally call take order take mins get food items pathetic service plates dirty cleanliness
maintained food also worst everything half boiled inspite taking mins   honestly restaurant great
potential one coolest green glen layout next lot apartments spacious interiors well covered menu n
however always felt short staff ni tried lunch dinner nbreakfast yes maybe times nthey make
delicious cheese masala dosa bread omelette poori sabji masala chai   npathetic place mosquitoes e
at well food arrives page menu half things available takes lifetime serve even customers half plac
e empty guess reason   went place nearby bad earlier review ndecent place eating economical food
cost around   decided check new restaurant bellandur thoroughly disappointed empty staff even
vaguely interested food also good make pathetic service providing feedback matter guy reception un
moved never coming back would recommend place'
```

In [ ]:

```
y=df['rate']
df=df.drop(['rate'],axis=1)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

Vectorizing categorical fearures

In [ ]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_online_order = CountVectorizer()
vectorizer_online_order.fit(df['online_order'].values)
print(vectorizer_online_order.get_feature_names())

X_tr_order_ohe = vectorizer_online_order.transform(X_train['online_order'].values)
X_cv_order_ohe = vectorizer_online_order.transform(X_cv['online_order'].values)
X_test_order_ohe = vectorizer_online_order.transform(X_test['online_order'].values)
print("Shape of training dataset one hot encoding & corresponding class label ",X_tr_order_ohe.sha
pe, y_train.shape)
```

```
pe, y_train.shape)
print("\nShape of cv dataset one hot encoding & corresponding class label ",X_cv_order_ohe.shape, y_cv.shape)
print("\nShape of test dataset one hot encoding & corresponding class label ",X_test_order_ohe.shape, y_test.shape)
```

```
['no', 'yes']
Shape of training dataset one hot encoding & corresponding class label  (23215, 2) (23215,)

Shape of cv dataset one hot encoding & corresponding class label  (11435, 2) (11435,)

Shape of test dataset one hot encoding & corresponding class label  (17067, 2) (17067,)
```

In [ ]:

```
vectorizer_location = CountVectorizer()
vectorizer_location.fit(df['location'].values)
print(vectorizer_location.get_feature_names())

X_tr_location_ohe = vectorizer_location.transform(X_train['location'].values)
X_cv_location_ohe = vectorizer_location.transform(X_cv['location'].values)
X_test_location_ohe = vectorizer_location.transform(X_test['location'].values)
```

```
['1st', '2nd', '3rd', '4th', '5th', '6th', '7th', '8th', 'airport', 'banashankari', 'banaswadi', 'bangalore', 'bannerghatta', 'basavanagudi', 'basaveshwara', 'bel', 'bellandur', 'bhima', 'block', 'bommanahalli', 'brigade', 'brookefield', 'btm', 'central', 'church', 'city', 'commercial', 'course', 'cunningham', 'cv', 'domlur', 'east', 'ejipura', 'electronic', 'frazer', 'garden', 'hbr', 'hebbal', 'hennur', 'hosur', 'hsr', 'indiranagar', 'infantry', 'itpl', 'jakkur', 'jalahalli', 'jayanagar', 'jeevan', 'jp', 'kaggadasapura', 'kalyan', 'kammanahalli', 'kanakapura', 'kengeri', 'koramangala', 'kr', 'kumaraswamy', 'langford', 'lavelle', 'layout', 'madras', 'magadi', 'main', 'majestic', 'malleshwaram', 'marathahalli', 'market', 'marks', 'mg', 'mysore', 'nagar', 'nagarbhavi', 'nagawara', 'new', 'north', 'old', 'peenya', 'puram', 'race', 'rajajinagar', 'rajarajeshwari', 'raman', 'rammurthy', 'residency', 'richmond', 'road', 'rt', 'sadashiv', 'sahakara', 'sanjay', 'sankey', 'sarjapur', 'seshadripuram', 'shanti', 'shivajinagar', 'south', 'st', 'street', 'thippasandra', 'town', 'ulsoor', 'uttarahalli', 'varthur', 'vasanth', 'vijay', 'west', 'whitefield', 'wilson', 'yelahanka', 'yeshwantpur']
```

In [ ]:

```
vectorizer_book_table = CountVectorizer()
vectorizer_book_table.fit(df['book_table'].values)
print(vectorizer_book_table.get_feature_names())

X_tr_book_table_ohe = vectorizer_book_table.transform(X_train['book_table'].values)
X_cv_book_table_ohe = vectorizer_book_table.transform(X_cv['book_table'].values)
X_test_book_table_ohe = vectorizer_book_table.transform(X_test['book_table'].values)
```

```
['no', 'yes']
```

In [ ]:

```
vectorizer_rest_type = CountVectorizer()
vectorizer_rest_type.fit(df['rest_type'].values)
print(vectorizer_rest_type.get_feature_names())

X_tr_rest_type_ohe = vectorizer_rest_type.transform(X_train['rest_type'].values)
X_cv_rest_type_ohe = vectorizer_rest_type.transform(X_cv['rest_type'].values)
X_test_rest_type_ohe = vectorizer_rest_type.transform(X_test['rest_type'].values)
```

```
['bakery', 'bar', 'beverage', 'bhojanalya', 'bites', 'cafe', 'cafee', 'casual', 'club', 'confectionery', 'court', 'delivery', 'dessert', 'dhaba', 'dining', 'fine', 'food', 'irani', 'kiosk', 'lounge', 'meat', 'mess', 'microbrewery', 'parlor', 'pop', 'pub', 'quick', 'shop', 'sweet', 'takeaway', 'truck', 'up']
```

In [ ]:

```
vectorizer_dish_liked = CountVectorizer()
vectorizer_dish_liked.fit(df['dish_liked'].values)
print(vectorizer_dish_liked.get_feature_names())

X_tr_dish_liked_ohe = vectorizer_dish_liked.transform(X_train['dish_liked'].values)
X_cv_dish_liked_ohe = vectorizer_dish_liked.transform(X_cv['dish_liked'].values)
```

```
X_cv_dish_liked_ohe = vectorizer_dish_liked.transform(X_cv['dish_liked'].values)
X_test_dish_liked_ohe = vectorizer_dish_liked.transform(X_test['dish_liked'].values)
```

```
['65', 'aalo', 'aam', 'aamras', 'abbabi', 'achari', 'adrak', 'afghan', 'afghani', 'aglio',
'ajwaini', 'akki', 'akuri', 'al', 'alfam', 'alfredo', 'almond', 'almonds', 'aloo', 'alpham',
'alur', 'ambience', 'ambur', 'american', 'americano', 'amritsari', 'anardana', 'ande', 'andhra', '
angara', 'anjal', 'anjeer', 'apollo', 'appam', 'apple', 'appletini', 'arabian', 'arabic',
'arrabiata', 'arrangement', 'arsalan', 'arugula', 'asparagus', 'au', 'augratin', 'avakai',
'avalanche', 'avocado', 'baati', 'babaganush', 'baby', 'babycorn', 'bacon', 'baconator', 'badam',
'badami', 'bagel', 'baguette', 'baida', 'baigan', 'baingan', 'bajji', 'baked', 'baklava', 'ball',
'balls', 'balti', 'bamboo', 'banana', 'bananas', 'bang', 'bangkok', 'bangla', 'banh', 'bannoffee',
'banoffee', 'bao', 'bar', 'barbaresca', 'barbecue', 'barbeque', 'barfi', 'barone', 'bartha', 'basa
', 'basanti', 'basil', 'basket', 'basmati', 'bath', 'bati', 'batter', 'bbq', 'beans', 'beda', 'bee
f', 'beer', 'beetroot', 'begun', 'beijing', 'bele', 'belgian', 'belgium', 'bella', 'bellini',
'benedict', 'bengali', 'benne', 'bento', 'berry', 'berryblast', 'bhaath', 'bhaja', 'bhaji', 'bhaji
ya', 'bhajji', 'bhalle', 'bhara', 'bharta', 'bharwan', 'bhath', 'bhatti', 'bhature', 'bheja', 'bhe
l', 'bhetki', 'bhindi', 'bhuna', 'bhurji', 'bibimbap', 'big', 'bingsu', 'biriyani', 'biryani', 'bi
scuit', 'biscuits', 'bisi', 'biskoot', 'bisque', 'bites', 'blackberry', 'blackcurrant', 'blanc', '
blast', 'bloody', 'blueberry', 'bomb', 'bombay', 'bombil', 'bonanza', 'bonda', 'bone', 'boneless',
'boondi', 'boti', 'bowl', 'bowls', 'box', 'brain', 'bravas', 'bread', 'breads', 'breakfast', 'brea
st', 'brew', 'brewed', 'brinjal', 'broccoli', 'brosted', 'broth', 'brown', 'brownie', 'brulee', 'b
runch', 'brunches', 'bruschetta', 'bruschettas', 'bubble', 'bucket', 'buddha', 'buffalo',
'buffet', 'bukhara', 'bulgogi', 'bun', 'bunglow', 'burger', 'burgers', 'burmese', 'burrito',
'burritos', 'burst', 'butter', 'butterfly', 'buttermilk', 'butterscotch', 'button', 'by',
'cabbage', 'caesar', 'cafe', 'cafreal', 'cajun', 'cake', 'cakes', 'calamari', 'calcutta',
'california', 'calzone', 'candy', 'cannelloni', 'cantonese', 'cappuccino', 'capsicum', 'caramel',
'caramelized', 'caramello', 'carbonara', 'carrot', 'cashew', 'caviar', 'ceaser', 'ceylon',
'chaach', 'chaai', 'chaap', 'chaat', 'chai', 'chamcham', 'chana', 'chapa', 'chapati', 'charcoal',
'charcuterie', 'chat', 'chatni', 'chatpata', 'chawal', 'cheddar', 'cheese', 'cheeseballs',
'cheesecake', 'cheesiano', 'cheesy', 'chelo', 'chenin', 'chestnuts', 'chettinad', 'chhole',
'chicken', 'chiller', 'chilli', 'chimichangas', 'chinese', 'chingri', 'chip', 'chips', 'choco', 'c
hocochip', 'chocoholic', 'chocolate', 'chocolava', 'chokha', 'chole', 'chop', 'chops', 'chopsuey',
'chorizo', 'choux', 'chowmein', 'chukka', 'chunks', 'chur', 'chura', 'churma', 'churros',
'chutney', 'cider', 'cigars', 'cinnamon', 'clear', 'club', 'cobb', 'cocktail', 'cocktails', 'cocon
ut', 'cod', 'coffee', 'coke', 'cola', 'colada', 'cold', 'coleslaw', 'combo', 'cone', 'cookie', 'co
okies', 'cooler', 'coorg', 'coriander', 'corn', 'cosmopolitan', 'cotta', 'cottage', 'cotton', 'cou
ntry', 'courteous', 'crab', 'crabmeat', 'crackers', 'crackling', 'craft', 'craze', 'cream', 'cream
y', 'creme', 'crepe', 'crepes', 'crisp', 'crispy', 'croissant', 'croissants', 'croquettes',
'crumble', 'crunch', 'crunchy', 'crust', 'cuban', 'cucumber', 'cuisine', 'cuon', 'cup', 'cupcake',
'curd', 'curry', 'custard', 'cut', 'cutlet', 'cutlets', 'cutting', 'da', 'daab', 'daal', 'dabeli',
'daddy', 'dahi', 'dahipuri', 'daiquiri', 'dak', 'dal', 'dalna', 'dance', 'dark', 'date', 'death',
'decadence', 'decor', 'deep', 'dehati', 'del', 'delight', 'deluxe', 'designer', 'dessert',
'detox', 'devils', 'devotion', 'dhania', 'dhaniya', 'dhansak', 'dhokar', 'dhokla', 'diced', 'dim',
'dimsum', 'dimsums', 'dindigul', 'disco', 'dish', 'diwani', 'dj', 'dog', 'doi', 'dom', 'donburi',
'doner', 'donne', 'donut', 'dorai', 'dosa', 'double', 'doughnut', 'draft', 'dragon', 'draught', 'd
reamcake', 'drink', 'drizzle', 'drums', 'drumstick', 'drumsticks', 'drunken', 'dry', 'duck',
'dum', 'dumpling', 'dumplings', 'dumpukht', 'dunkaccinos', 'dutch', 'dynamite', 'eastern',
'eater', 'eclair', 'eclairs', 'egg', 'eggless', 'eggplant', 'eggs', 'elaichi', 'elaneer',
'empanadas', 'enchiladas', 'english', 'espresso', 'executive', 'exotic', 'extravaganza', 'fafda',
'faham', 'fajita', 'fajitas', 'falafal', 'falafel', 'faluda', 'fantasy', 'farm', 'farmhouse',
'farsan', 'fatayer', 'fattoush', 'feast', 'ferrero', 'feta', 'fiery', 'fiesta', 'fig', 'filet', 'f
ilter', 'finger', 'fingers', 'fire', 'firewood', 'firni', 'fish', 'flakes', 'flan', 'float', 'floo
r', 'florentine', 'flurry', 'fondue', 'food', 'frankie', 'frappe', 'frappuccino', 'freak',
'french', 'fried', 'friendly', 'fries', 'frittata', 'fritters', 'fruit', 'fruits', 'fry', 'fudge',
'funda', 'fung', 'gadwal', 'gajar', 'galauti', 'ganache', 'ganna', 'gaon', 'gappe', 'garden', 'gar
lic', 'gassi', 'gateau', 'gatte', 'gelato', 'german', 'ghamandi', 'ghar', 'ghee', 'ghewar', 'ghont
o', 'ghost', 'gimbap', 'ginger', 'gini', 'gnocchi', 'goan', 'gobhi', 'gobi', 'gol', 'golden', 'gol
i', 'gong', 'gongura', 'gooey', 'goreng', 'gosht', 'gourmet', 'grain', 'grand', 'granola',
'grape', 'gratin', 'gravy', 'greek', 'green', 'grill', 'grilled', 'guava', 'gujarati', 'gulab', 'g
ulkand', 'gum', 'gundappa', 'guntur', 'gur', 'gurer', 'gyoza', 'hakka', 'haleem', 'halwa', 'ham',
'handi', 'hara', 'hariyali', 'hash', 'hatti', 'hawaian', 'hazelnut', 'healthy', 'heaven',
'hefeweizen', 'herbed', 'hibiscus', 'highway', 'holige', 'honey', 'hookah', 'hot', 'hotdog', 'hous
e', 'hrc', 'hummus', 'hunan', 'hush', 'hyderabadi', 'hydrabadi', 'ice', 'icecream', 'icecreams', '
iced', 'idiyappam', 'idli', 'idlis', 'ilish', 'imli', 'indian', 'indori', 'irani', 'irish', 'islan
d', 'italian', 'italiano', 'jackfruit', 'jaipuri', 'jal', 'jalapeno', 'jalebi', 'jam', 'jamun', 'j
apanese', 'japchae', 'jar', 'jasmine', 'java', 'jeera', 'jerk', 'jhol', 'joint', 'jolada',
'jowar', 'juice', 'juices', 'juicy', 'jujeh', 'jumbo', 'ka', 'kaapi', 'kaaram', 'kabab', 'kachori'
, 'kadai', 'kadak', 'kadala', 'kadam', 'kadhai', 'kadhi', 'kaffir', 'kahwa', 'kai', 'kaju', 'kala'
, 'kali', 'kalia', 'kalimirch', 'kallappam', 'kalmi', 'kamikaze', 'kanda', 'kanti', 'kappa', 'kara
chi', 'karam', 'karara', 'karela', 'karimeen', 'kasha', 'kashmiri', 'kastoori', 'kat', 'kathi', 'k
atla', 'katli', 'katsu', 'ke', 'kebab', 'kebabs', 'keema', 'kerala', 'kesar', 'kesari', 'kesaria',
'key', 'kha', 'khali', 'khamiri', 'khao', 'khara', 'kharabath', 'khau', 'kheema', 'kheer',
'khichda', 'khichdi', 'khubani', 'khurchan', 'ki', 'kichadi', 'kimchi', 'king', 'kit', 'kitkat', '
kiwi', 'kizhi', 'kodi', 'kofta', 'kokum', 'kola', 'kolhapuri', 'korean', 'kori', 'korma', 'kosha',
'kothu', 'kozhi', 'krisper', 'kshatriya', 'kudu', 'kulcha', 'kulche', 'kulfi', 'kullad', 'kuluki',
'kunafa', 'kung', 'kurkure', 'kurkuri', 'kutchi', 'lababdar', 'laccha', 'ladoo', 'lager', 'laham',
'lahori', 'lajawab', 'laksa', 'lal', 'lamb', 'lasagna', 'lasagne', 'lash', 'lassi', 'latte', 'lauk
```

```
i', 'lava', 'lazeez', 'leaf', 'lebanese', 'legendary', 'lemon', 'lemonade', 'lemongrass', 'liege',
'lime', 'litchi', 'liti', 'litti', 'liver', 'lobster', 'lollipop', 'long', 'lotus', 'luchi', 'luck
nowi', 'lunch', 'lung', 'lychee', 'maans', 'maaz', 'mac', 'macaroni', 'macaroon', 'macchiato', 'ma
ckerel', 'madagascar', 'maddur', 'madfoon', 'maggi', 'maharaja', 'mai', 'majestic', 'makhani', 'ma
khanwala', 'makhni', 'maki', 'makkai', 'makke', 'mala', 'malabar', 'malabari', 'malacca', 'malai',
'malaikari', 'malaysian', 'malgoum', 'malhooth', 'malpua', 'malt', 'malwani', 'manakeesh',
'manali', 'manchow', 'manchurian', 'mandi', 'mango', 'mangsho', 'mapo', 'margarita', 'margherita',
'marshmallow', 'martini', 'marwai', 'mary', 'marzano', 'masala', 'mascarpone', 'mash', 'mashed',
'maskas', 'masoor', 'matar', 'matka', 'maxx', 'mayo', 'mc', 'mcaloo', 'mcflurry', 'mcmuffin', 'mcs
picy', 'meal', 'meat', 'meatballs', 'mediterranean', 'medu', 'meen', 'meetha', 'meishan',
'melted', 'memon', 'menu', 'methi', 'mex', 'mexican', 'mezze', 'middle', 'milk', 'milkshake', 'mil
ky', 'millet', 'minced', 'minestrone', 'mini', 'minotaur', 'mint', 'mirch', 'mirchi', 'misal', 'mi
shti', 'miso', 'missi', 'mississippi', 'mix', 'mixed', 'mla', 'mocha', 'mochar', 'mochi', 'mocktai
l', 'mocktails', 'mojito', 'molten', 'momo', 'momos', 'monster', 'moo', 'moong', 'morning',
'mosambi', 'motichoor', 'moussaka', 'mousse', 'mozzarella', 'mud', 'mudcake', 'mudde', 'mudpie', '
muffin', 'mughlai', 'multani', 'multi', 'mumbai', 'muradabadi', 'murg', 'murgh', 'murgi',
'mushroom', 'mushrooms', 'mustard', 'mutton', 'mysore', 'naan', 'nacho', 'nachos', 'naga',
'nalli', 'nannari', 'nasi', 'nati', 'nawabi', 'neer', 'nepali', 'new', 'nihari', 'nippat',
'nirvana', 'nitash', 'nitrogen', 'nizami', 'nolen', 'non', 'noodle', 'noodles', 'nuggets', 'nut',
'nutella', 'obbattu', 'octopus', 'of', 'okra', 'olio', 'omelette', 'onion', 'open', 'opera',
'orange', 'oregano', 'oreo', 'outdoor', 'overdose', 'overload', 'oysters', 'paan', 'pad',
'paella', 'pahadi', 'pahari', 'pai', 'pak', 'pakoda', 'pakora', 'pakwan', 'palak', 'pallipalayam',
'palya', 'pan', 'pancake', 'pancakes', 'panchmel', 'pandhra', 'pandi', 'pane', 'paneer', 'pani', '
panipuri', 'panje', 'panna', 'pannacotta', 'panneer', 'panner', 'pao', 'papad', 'papadi',
'papaya', 'papdi', 'paper', 'paprika', 'parantha', 'paratha', 'parfait', 'parmigiana', 'parotta',
'pasanda', 'passion', 'pasta', 'pastry', 'patata', 'patiala', 'patisserie', 'patiyala',
'pattanam', 'patthar', 'pattice', 'pattis', 'patty', 'paturi', 'pav', 'pavbhaji', 'pavs', 'paya',
'payasam', 'payassam', 'pazham', 'peach', 'peanut', 'peanuts', 'pearl', 'peas', 'peda', 'peking',
'penne', 'pepper', 'pepperoni', 'peppy', 'pepsi', 'peri', 'pesarattu', 'peshawari', 'pesto', 'peth
a', 'phad', 'phirni', 'pho', 'photo', 'phulkas', 'phulke', 'picante', 'pickle', 'pickled', 'pie',
'piece', 'pilaf', 'pili', 'pinacolada', 'pineapple', 'pink', 'pista', 'pita', 'pitika', 'pizza', '
plain', 'plate', 'platter', 'plum', 'pocket', 'pockets', 'podi', 'poha', 'poli', 'pollichathu', 'p
omegranate', 'pomfret', 'pongal', 'pool', 'poori', 'popcorn', 'poppers', 'pops', 'pora', 'pori', '
poricha', 'pork', 'posto', 'pot', 'potato', 'potatoes', 'pothichoru', 'potli', 'prawn', 'prawns',
'primavera', 'prompt', 'protein', 'pudding', 'pudina', 'puff', 'pulao', 'pulav', 'pulled',
'pulpy', 'pulusu', 'pumpkin', 'punjabi', 'puppies', 'puran', 'puri', 'puttu', 'pyaz',
'quesadilla', 'quesedillas', 'quiche', 'quinoa', 'raagi', 'raan', 'rabri', 'radhaballavi',
'ragda', 'ragi', 'ragout', 'railway', 'rainbow', 'raita', 'raj', 'rajasthani', 'rajdhani',
'rajma', 'rajugari', 'ramen', 'rara', 'rasam', 'rasgulla', 'rasmalai', 'raspberry', 'rassa', 'rata
touille', 'rava', 'ravioli', 'rawas', 'red', 'rendang', 'reshmi', 'rezala', 'ribs', 'rice',
'rich', 'ricotta', 'rings', 'risotto', 'roast', 'roasted', 'rocher', 'roganjosh', 'roll', 'rolls',
'rooftop', 'rose', 'roti', 'rotis', 'rotti', 'rottis', 'royal', 'royale', 'rumali', 'rusk', 'russi
an', 'rustica', 'saag', 'saagu', 'saalan', 'sabji', 'sabudana', 'sabzi', 'saffron', 'sagu', 'saha'
, 'sake', 'salad', 'salads', 'salami', 'sali', 'salmon', 'salsa', 'salt', 'salted', 'sambar', 'sam
osa', 'sandwich', 'sandwiches', 'sangria', 'sarbat', 'sarso', 'sarson', 'sashimi', 'satay',
'sattu', 'sauce', 'sausage', 'sausages', 'sauteed', 'savoury', 'scallops', 'schezwan',
'schnitzel', 'scrambled', 'sea', 'seafood', 'seaweed', 'seekh', 'sekuwa', 'sensation', 'service',
'sesame', 'set', 'sev', 'shaadi', 'shahi', 'shake', 'shakes', 'shakshouka', 'shami', 'shammi', 'sh
anghai', 'shanmukha', 'shaptak', 'shaptra', 'sharbat', 'shashlik', 'shawarama', 'shawarma',
'sheermal', 'shephards', 'shikanji', 'shitake', 'sholay', 'sholey', 'shoot', 'shorba', 'shot', 'sh
otgun', 'shots', 'shrikhand', 'shrimp', 'shwarma', 'sicilian', 'sikandari', 'singapore',
'sitaphal', 'sitting', 'sizzler', 'sizzlers', 'skewer', 'skillet', 'skins', 'slider', 'sling', 'sm
oked', 'smokey', 'smoothie', 'snacker', 'snapper', 'snicker', 'soan', 'soba', 'soda', 'sol',
'som', 'sooji', 'sorbet', 'souffle', 'soup', 'sour', 'sourdough', 'souvlaki', 'soya', 'spaghetti',
'spice', 'spicy', 'spinach', 'split', 'sponge', 'spring', 'spritzer', 'sprout', 'squid', 'staff',
'starter', 'steak', 'steam', 'steamed', 'stem', 'stew', 'stick', 'sticks', 'stir', 'stout', 'straw
berry', 'strips', 'stroganoff', 'stromboli', 'stuffed', 'style', 'sub', 'subz', 'subzi', 'suey', '
sugarcane', 'sukha', 'sukka', 'sulemani', 'sum', 'sums', 'sundae', 'sunday', 'supreme', 'sushi', '
suzette', 'sweet', 'sweets', 'swiss', 'tabakh', 'tabasco', 'tabbouleh', 'table', 'taco', 'tacos',
'tadka', 'tai', 'tak', 'taka', 'takoyaki', 'tam', 'tamarind', 'tamatar', 'tamda', 'tandoor', 'tand
oori', 'tangdi', 'tango', 'taquitos', 'tarkari', 'tart', 'tarts', 'tashkent', 'tava', 'tawa', 'tea
', 'tempura', 'tender', 'tenderloin', 'tenders', 'tennessee', 'tequila', 'teriyaki', 'terminator',
'tex', 'thai', 'thalassery', 'thali', 'thandai', 'thatte', 'thepla', 'thick', 'thickshake',
'thin', 'thread', 'thukpa', 'thupka', 'tibetan', 'tikka', 'tikki', 'tiramisu', 'tiranga',
'tirupathi', 'toast', 'toffee', 'tofu', 'tokri', 'tom', 'tomato', 'tonkatsu', 'tori', 'tortilla',
'treasure', 'trio', 'triple', 'trouble', 'truffle', 'tuk', 'tukda', 'tuna', 'tundey', 'turkish', '
twister', 'uddin', 'udon', 'ulavacharu', 'ulvacharu', 'upma', 'urundai', 'uttapam', 'vada',
'vadai', 'vanilla', 'vanjiram', 'varuval', 'veg', 'vegetable', 'vegetables', 'vegetarian',
'veggie', 'vellarikka', 'velvet', 'vepudu', 'verdure', 'vietnamese', 'vindaloo', 'virgin',
'volcano', 'waffle', 'waffles', 'wai', 'walnut', 'warqi', 'wasabi', 'water', 'watermelon', 'way',
'wedge', 'wedges', 'wet', 'wheat', 'whisky', 'white', 'whole', 'whopper', 'wild', 'wine', 'wings',
'wit', 'wok', 'wonder', 'wonton', 'wood', 'woodfire', 'wrap', 'yakhni', 'yaki', 'yakisoba',
'yakitori', 'yellow', 'yoghurt', 'yogurt', 'york', 'yum', 'zafrani', 'zaraja', 'zealand',
'zinger']
```

In [ ]:

In [ ]:

```
vectorizer_cuisines = CountVectorizer()
vectorizer_cuisines.fit(df['cuisines'].values)
print(vectorizer_cuisines.get_feature_names())

X_tr_cuisines_ohe = vectorizer_cuisines.transform(X_train['cuisines'].values)
X_cv_cuisines_ohe = vectorizer_cuisines.transform(X_cv['cuisines'].values)
X_test_cuisines_ohe = vectorizer_cuisines.transform(X_test['cuisines'].values)
```

```
['afghan', 'afghani', 'african', 'american', 'andhra', 'arabian', 'asian', 'assamese',
'australian', 'awadhi', 'bakery', 'bar', 'bbq', 'belgian', 'bengali', 'beverages', 'bihari',
'biryani', 'bohri', 'british', 'bubble', 'burger', 'burmese', 'cafe', 'cantonese', 'charcoal', 'ch
ettinad', 'chicken', 'chinese', 'coffee', 'continental', 'cream', 'desserts', 'dogs', 'drinks', 'e
astern', 'european', 'fast', 'finger', 'food', 'french', 'german', 'goan', 'greek', 'grill', 'guja
rati', 'healthy', 'hot', 'hyderabadi', 'ice', 'indian', 'indonesian', 'iranian', 'italian',
'japanese', 'jewish', 'juices', 'kashmiri', 'kebab', 'kerala', 'konkan', 'korean', 'lankan', 'leba
nese', 'lucknowi', 'maharashtrian', 'malaysian', 'malwani', 'mangalorean', 'meats',
'mediterranean', 'mex', 'mexican', 'middle', 'mithai', 'modern', 'momos', 'mongolian', 'mughlai',
'naga', 'nepalese', 'north', 'only', 'oriya', 'paan', 'pan', 'parsi', 'pizza', 'portuguese', 'raja
sthani', 'raw', 'roast', 'rolls', 'russian', 'salad', 'sandwich', 'seafood', 'sindhi',
'singaporean', 'south', 'spanish', 'sri', 'steak', 'street', 'sushi', 'tamil', 'tea', 'tex', 'thai
', 'tibetan', 'turkish', 'vegan', 'vietnamese', 'wraps']
```

In [ ]:

```
vectorizer_listed_in_tp = CountVectorizer()
vectorizer_listed_in_tp.fit(df['listed_in(type)'].values)
print(vectorizer_listed_in_tp.get_feature_names())


X_tr_listed_in_tp_ohe = vectorizer_listed_in_tp.transform(X_train['listed_in(type)'].values)
X_cv_listed_in_tp_ohe = vectorizer_listed_in_tp.transform(X_cv['listed_in(type)'].values)
X_test_listed_in_tp_ohe = vectorizer_listed_in_tp.transform(X_test['listed_in(type)'].values)
```

```
['and', 'bars', 'buffet', 'cafes', 'delivery', 'desserts', 'dine', 'drinks', 'nightlife', 'out', '
pubs']
```

In [ ]:

```
vectorizer_listed_in_ct = CountVectorizer()
vectorizer_listed_in_ct.fit(df['listed_in(city)'].values)
print(vectorizer_listed_in_ct.get_feature_names())


X_tr_listed_in_ct_ohe = vectorizer_listed_in_ct.transform(X_train['listed_in(city)'].values)
X_cv_listed_in_ct_ohe = vectorizer_listed_in_ct.transform(X_cv['listed_in(city)'].values)
X_test_listed_in_ct_ohe = vectorizer_listed_in_ct.transform(X_test['listed_in(city)'].values)
```

```
['4th', '5th', '6th', '7th', 'airport', 'banashankari', 'bannerghatta', 'basavanagudi', 'bel', 'be
llandur', 'block', 'brigade', 'brookefield', 'btm', 'church', 'city', 'electronic', 'frazer', 'hsr
', 'indiranagar', 'jayanagar', 'jp', 'kalyan', 'kammanahalli', 'koramangala', 'lavelle',
'malleshwaram', 'marathahalli', 'mg', 'nagar', 'new', 'old', 'rajajinagar', 'residency', 'road', '
sarjapur', 'street', 'town', 'whitefield']
```

Vectorizing Numerical Features

In [ ]:

```
from sklearn.preprocessing import Normalizer

cost_scalar = Normalizer()
cost_scalar.fit(X_train['approx_cost(for two people)'].values.reshape(-1,1)) # finding the mean
and standard deviation of this data
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
cost_standardized_train = cost_scalar.transform(X_train['approx_cost(for two
people)'].values.reshape(-1, 1))
cost_standardized_cv = cost_scalar.transform(X_cv['approx_cost(for two people)'].values.reshape(-1
, 1))
cost_standardized_test = cost_scalar.transform(X_test['approx_cost(for two
```

```
cost_standardized_test = cost_scalar.transform(X_test['approx_cost(for two
people)'].values.reshape(-1, 1))
```

In [ ]:

```
number_of_cuisines_scalar = Normalizer()
number_of_cuisines_scalar.fit(X_train['number_of_cuisines'].values.reshape(-1,1)) # finding the
mean and standard deviation of this data
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
number_of_cuisines_standardized_train =
number_of_cuisines_scalar.transform(X_train['number_of_cuisines'].values.reshape(-1, 1))
number_of_cuisines_standardized_cv = number_of_cuisines_scalar.transform(X_cv['number_of_cuisines'
].values.reshape(-1, 1))
number_of_cuisines_standardized_test =
number_of_cuisines_scalar.transform(X_test['number_of_cuisines'].values.reshape(-1, 1))
```

In [ ]:

```
number_of_liked_dishes_scalar = Normalizer()
number_of_liked_dishes_scalar.fit(X_train['number_of_liked_dishes'].values.reshape(-1,1)) # finding
the mean and standard deviation of this data
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
number_of_liked_dishes_standardized_train =
number_of_liked_dishes_scalar.transform(X_train['number_of_liked_dishes'].values.reshape(-1, 1))
number_of_liked_dishes_standardized_cv =
number_of_liked_dishes_scalar.transform(X_cv['number_of_liked_dishes'].values.reshape(-1, 1))
number_of_liked_dishes_standardized_test =
number_of_liked_dishes_scalar.transform(X_test['number_of_liked_dishes'].values.reshape(-1, 1))
```

In [ ]:

```
from sklearn.preprocessing import Normalizer

votes_scalar = Normalizer()
votes_scalar.fit(X_train['votes'].values.reshape(-1,1))

votes_standardized_train = votes_scalar.transform(X_train['votes'].values.reshape(-1, 1))
votes_standardized_cv = votes_scalar.transform(X_cv['votes'].values.reshape(-1, 1))
votes_standardized_test = votes_scalar.transform(X_test['votes'].values.reshape(-1, 1))
```

In [ ]:

```
from sklearn.preprocessing import Normalizer

Facilities_offered_scalar = Normalizer()
Facilities_offered_scalar.fit(X_train['Facilities_offered'].values.reshape(-1,1))

Facilities_offered_standardized_train =
Facilities_offered_scalar.transform(X_train['Facilities_offered'].values.reshape(-1, 1))
Facilities_offered_standardized_cv = Facilities_offered_scalar.transform(X_cv['Facilities_offered'
].values.reshape(-1, 1))
Facilities_offered_standardized_test =
Facilities_offered_scalar.transform(X_test['Facilities_offered'].values.reshape(-1, 1))
```

In [ ]:

```
mean_cuisines_scalar = Normalizer()
mean_cuisines_scalar.fit(X_train['mean_cuisines'].values.reshape(-1,1))

mean_cuisines_standardized_train = mean_cuisines_scalar.transform(X_train['mean_cuisines'].values.
reshape(-1, 1))
mean_of_cuisines_standardized_cv = mean_cuisines_scalar.transform(X_cv['mean_cuisines'].values.res
hape(-1, 1))
mean_of_cuisines_standardized_test = mean_cuisines_scalar.transform(X_test['mean_cuisines'].values
.reshape(-1, 1))
```

In [ ]:

```
dish_liked_scalar = Normalizer()
dish_liked_scalar.fit(X_train['mean_dish_liked'].values.reshape(-1,1))
dish_liked_standardized_train =
dish_liked_scalar.transform(X_train['mean_dish_liked'].values.reshape(-1, 1))
dish_liked_standardized_cv = dish_liked_scalar.transform(X_cv['mean_dish_liked'].values.reshape(-1
, 1))
dish_liked_standardized_test =
dish_liked_scalar.transform(X_test['mean_dish_liked'].values.reshape(-1, 1))
```

Vectorizing Text Features

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_text = TfidfVectorizer()
vectorizer_text.fit(X_train['preprocessed_reviews'].values)


X_tr_preprocessed_reviews_tfidf = vectorizer_text.transform(X_train['preprocessed_reviews'].values
)
X_cv_preprocessed_reviews_tfidf = vectorizer_text.transform(X_cv['preprocessed_reviews'].values)
X_test_preprocessed_reviews_tfidf =
vectorizer_text.transform(X_test['preprocessed_reviews'].values)
```

In [ ]:

```
X_tr_total
=hstack((X_tr_order_ohe,X_tr_location_ohe,X_tr_book_table_ohe,X_tr_rest_type_ohe,X_tr_dish_liked_oh
e,X_tr_cuisines_ohe,X_tr_listed_in_tp_ohe,X_tr_listed_in_ct_ohe,cost_standardized_train,number_of_c
uisines_standardized_train,number_of_liked_dishes_standardized_train,votes_standardized_train,Faci
lities_offered_standardized_train,mean_cuisines_standardized_train,dish_liked_standardized_train,X
_tr_preprocessed_reviews_tfidf)).tocsr()
X_test_total
=hstack((X_test_order_ohe,X_test_location_ohe,X_test_book_table_ohe,X_test_rest_type_ohe,X_test_dis
h_liked_ohe,X_test_cuisines_ohe,X_test_listed_in_tp_ohe,X_test_listed_in_ct_ohe,cost_standardized_t
est,number_of_cuisines_standardized_test,number_of_liked_dishes_standardized_test,votes_standardize
d_test,Facilities_offered_standardized_test,mean_of_cuisines_standardized_test,dish_liked_standardi
zed_test,X_test_preprocessed_reviews_tfidf)).tocsr()
```

In [ ]:

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

Hyperparameter Tuning for Random Forest

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor
```

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor()
parameters = {'n_estimators': [1, 5, 10, 50, 100], 'max_depth': [5, 10, 20, 75, 100]}
mdl = RandomizedSearchCV(rf, param_distributions =parameters, cv=10)
mdl.fit(X_tr_total,y_train)
```

Out[ ]:

```
RandomizedSearchCV(cv=10, error_score=nan,
```

```
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   max_samples=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_score=False,
                                                   random_state=None, verbose=0,
                                                   warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [5, 10, 20, 75, 100],
                                        'n_estimators': [1, 5, 10, 50, 100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [ ]:

```
print(mdl.best_estimator_)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=75, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Applying Random Forest Algorithm

In [ ]:

```
rfm= RandomForestRegressor(n_estimators=100, criterion='mse', max_depth=None, min_samples_split=2,
min_samples_leaf=1)
rfm.fit(X_tr_total,y_train)
```

Out[ ]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [ ]:

```
import pickle
filename = 'finalized_model.sav'
pickle.dump(rfm, open(filename, 'wb'))
```

In [ ]:

```
from sklearn.metrics import mean_squared_error
y_pred_lr = rfm.predict(X_test_total)
mean_squared_error(y_test, y_pred_lr)
```

Out[ ]:

```
0.027927709527412244
```

Hyperparameter tuning for Decision tree

Hyperparameter tuning for Decsion tree

In [ ]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
dt1 = DecisionTreeRegressor()
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 45, 75,
100]}
mdl = RandomizedSearchCV(dt1, parameters)
mdl.fit(X_tr_total,y_train)
```

Out[ ]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=DecisionTreeRegressor(ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features=None,
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   presort='deprecated',
                                                   random_state=None,
                                                   splitter='best'),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'max_depth': [1, 5, 10, 50, 100, 500,
                                                      1000],
                                        'min_samples_split': [5, 10, 20, 45, 75,
                                                              100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [ ]:

```
mdl.best_params_
```

Out[ ]:

```
{'max_depth': 50, 'min_samples_split': 100}
```

In [ ]:

```
mdl.best_estimator_
```

Out[ ]:

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=50,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=100,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

Applying Descision tree regressor

In [ ]:

```
from sklearn.tree import DecisionTreeRegressor
mdl= DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=50,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=100,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
mdl.fit(X_tr_total,y_train)
```

Out[ ]:

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=50,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=100,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [ ]:

```
from sklearn.metrics import mean_squared_error
y_pred_lr = mdl.predict(X_test_total)
mean_squared_error(y_test, y_pred_lr)
```

Out[ ]:

```
0.06577735390392918
```

Hyperparameter tuning for xgboost

In [ ]:

```
import xgboost as xgb
model_d = xgb.XGBRegressor()
param_grid = {"n_estimators":[25, 50, 100, 200, 250, 500], "max_depth": [3, 5, 7, 9, 11, 13]}
d = RandomizedSearchCV(model_d, param_distributions = param_grid, n_iter=10, cv=10,
return_train_score=True)
d.fit(X_tr_total, y_train)
print(d.best_estimator_)
```

```
[20:57:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[20:57:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[20:58:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[20:58:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[20:59:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:00:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:00:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:01:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:01:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:02:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:02:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:02:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:20] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:03:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
```

```
n favor of reg:squarederror.
[21:03:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:04:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:04:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:05:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:05:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:06:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:06:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:06:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:07:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:07:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:28] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:08:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:09:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:10:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:10:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:10:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
```

```
n favor of reg:squarederror.
[21:10:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:10:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:11:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:11:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:11:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:12:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:12:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:12:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:13:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:13:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:14:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:15:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:16:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:17:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:18:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:19:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:20:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:21:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:22:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:23:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:24:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
```

```
n favor of reg:squarederror.
[21:25:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:25:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:25:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
[21:25:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=7, min_child_weight=1, missing=None, n_estimators=250,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Applying xgboost

In [ ]:

```
from xgboost.sklearn import XGBRegressor
c=XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=7, min_child_weight=1, missing=None, n_estimators=250,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
c.fit(X_tr_total,y_train)
y_pred_lr = c.predict(X_test_total)
mean_squared_error(y_test, y_pred_lr)
```

```
[15:52:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated i
n favor of reg:squarederror.
```

Out[ ]:

```
0.038084723971161166
```

Hyperparameter tuning for Linear regression

In [ ]:

```
 from sklearn.linear_model import SGDRegressor
model_d = SGDRegressor(loss='squared_loss')
param_grid = {"alpha":[0.001, 0.01,0.1,1,10,100,1000]}
d = RandomizedSearchCV(model_d, param_distributions = param_grid, n_iter=10, cv=10,
return_train_score=True)
d.fit(X_tr_total, y_train)
print(d.best_estimator_)
```

```
SGDRegressor(alpha=0.001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

Applying linear regression

In [ ]:

```
from sklearn.linear_model import SGDRegressor
x = SGDRegressor(alpha=0.001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
```

```
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [ ]:

```
x.fit(X_tr_total, y_train)
```

Out[ ]:

```
SGDRegressor(alpha=0.001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [ ]:

```
from sklearn.metrics import mean_squared_error
y_pred_lr = x.predict(X_test_total)
mean_squared_error(y_test, y_pred_lr)
```

Out[ ]:

```
0.08059014915125161
```

Deep Learning MODELS

In [ ]:

```
X=k.copy()
Y=y
x_train, x_test, y_train, y_test = train_test_split(np.asarray(X), np.asarray(Y), test_size=0.33, s
huffle= True)
```

In [ ]:

```
import keras
num_classes = 10

input_shape = (10,)

y_train_binary = keras.utils.to_categorical(y_train, num_classes)
y_test_binary = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.reshape(34650, 10,1)
x_test = x_test.reshape(17067, 10,1)
```

CNN-LSTM model

In [ ]:

```
from keras.layers import Input, Dense, LSTM, MaxPooling1D, Conv1D,Dropout
from keras.models import Model
#k = Dropout(0.5)(pool1)

input_layer = Input(shape=(10, 1))
conv2 = Conv1D(filters=64,
               kernel_size=3,
               strides=1,
               activation='relu')(input_layer)
dropout1 = Dropout(0.25)(conv2)
pool1 = MaxPooling1D(pool_size=1)(dropout1)
dropout1 = Dropout(0.5)(pool1)
lstm1 = LSTM(64,return_sequences=True)(dropout1)
conv3 = Conv1D(filters=128,
               kernel_size=8,
               strides=1,
               activation='relu')(lstm1)
dropout2 = Dropout(0.25)(conv3)
```

```
dropout2 = Dropout(0.25)(conv3)
lstm2 = LSTM(128,return_sequences=False)(dropout2)
output_layer = Dense(1, activation='sigmoid')(lstm2)
model = Model(inputs=input_layer, outputs=output_layer)

model.summary()
```

```
Model: "model_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_40 (InputLayer)        (None, 10, 1)             0
_________________________________________________________________
conv1d_88 (Conv1D)           (None, 8, 64)             256
_________________________________________________________________
dropout_61 (Dropout)         (None, 8, 64)             0
_________________________________________________________________
max_pooling1d_79 (MaxPooling (None, 8, 64)             0
_________________________________________________________________
dropout_62 (Dropout)         (None, 8, 64)             0
_________________________________________________________________
lstm_31 (LSTM)               (None, 8, 64)             33024
_________________________________________________________________
conv1d_89 (Conv1D)           (None, 1, 128)            65664
_________________________________________________________________
dropout_63 (Dropout)         (None, 1, 128)            0
_________________________________________________________________
lstm_32 (LSTM)               (None, 128)               131584
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 129
=================================================================
Total params: 230,657
Trainable params: 230,657
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(loss='mse',optimizer='adam')
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          validation_data=(x_test, y_test), verbose=1)
```

```
Train on 34650 samples, validate on 17067 samples
Epoch 1/10
34650/34650 [==============================] - 10s 287us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 2/10
34650/34650 [==============================] - 9s 252us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 3/10
34650/34650 [==============================] - 9s 252us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 4/10
34650/34650 [==============================] - 9s 253us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 5/10
34650/34650 [==============================] - 9s 251us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 6/10
34650/34650 [==============================] - 9s 253us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 7/10
34650/34650 [==============================] - 9s 251us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 8/10
34650/34650 [==============================] - 9s 254us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 9/10
34650/34650 [==============================] - 9s 258us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 10/10
34650/34650 [==============================] - 9s 254us/step - loss: 7.2646 - val_loss: 7.2830
```

Out[ ]:

```
<keras.callbacks.callbacks.History at 0x7efecb305fd0>
```

In [ ]:

```
score = model.evaluate(x_test, y_test)
score
```

```
17067/17067 [==============================] - 1s 76us/step
```

Out[ ]:

```
7.2829501636838385
```

In [ ]:

```python
model.save_weights("model_1_weights.h5")
```

CNN Model

In [ ]:

```python
from keras.layers import Input, Dense, LSTM, MaxPooling1D, Conv1D,Dropout, GlobalMaxPool1D
input_layer = Input(shape=(10, 1))
conv2 = Conv1D(filters=64,
               kernel_size=3,
               strides=1,
               activation='relu')(input_layer)
pool1 = MaxPooling1D(pool_size=1)(conv2)
drop1 = Dropout(0.5)(pool1)
pool2 = MaxPooling1D(pool_size=1)(drop1)
conv3 = Conv1D(filters=64,
               kernel_size=3,
               strides=1,
               activation='relu')(pool2)
drop2 = Dropout(0.5)(conv3)
conv4 = Conv1D(filters=64,
               kernel_size=3,
               strides=1,
               activation='relu')(drop2)
pool3 = MaxPooling1D(pool_size=1)(conv4)
conv5 = Conv1D(filters=64,
               kernel_size=3,
               strides=1,
               activation='relu')(pool3)
x = GlobalMaxPool1D()(conv5)
output_layer = Dense(1, activation='sigmoid')(x)
model_2 = Model(inputs=input_layer, outputs=output_layer)

model_2.compile(loss='mse',optimizer='adam')
model_2.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          validation_data=(x_test, y_test), verbose=1)
```

```
Train on 34650 samples, validate on 17067 samples
Epoch 1/10
34650/34650 [==============================] - 4s 103us/step - loss: 7.2887 - val_loss: 7.2830
Epoch 2/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 3/10
34650/34650 [==============================] - 3s 95us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 4/10
34650/34650 [==============================] - 3s 97us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 5/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 6/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 7/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 8/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 9/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 10/10
34650/34650 [==============================] - 3s 96us/step - loss: 7.2646 - val_loss: 7.2830
```

Out[ ]:

```
<keras.callbacks.callbacks.History at 0x7efcca464e60>
```

<keras.callbacks.callbacks.History at 0x7efeca4f4ef0>

In [ ]:

```
model_2.evaluate(x_train,y_train)
```

```
34650/34650 [==============================] - 1s 38us/step
```

Out[ ]:

```
7.264551804750341
```

In [ ]:

```
model_2.save_weights('model_2_weights.h5')
```

LSTM Model

In [ ]:

```
from keras.layers import Input, Dense, LSTM, MaxPooling1D, Conv1D,Dropout, GlobalMaxPool1D
input_layer = Input(shape=(10, 1))
lstm1 = LSTM(64,return_sequences=True)(input_layer)
pool1 = MaxPooling1D(pool_size=1)(lstm1)
drop1 = Dropout(0.5)(pool1)
pool2 = MaxPooling1D(pool_size=1)(drop1)
lstm2 = LSTM(128,return_sequences=True)(pool2)
drop2 = Dropout(0.5)(lstm2)
lstm3 = LSTM(64,return_sequences=True)(drop2)
pool3 = MaxPooling1D(pool_size=1)(lstm3)
lstm4 = LSTM(64,return_sequences=True)(pool3)
x = GlobalMaxPool1D()(lstm4)
output_layer = Dense(1, activation='sigmoid')(x)
model_3 = Model(inputs=input_layer, outputs=output_layer)

model_3.compile(loss='mse',optimizer='adam')
model_3.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          validation_data=(x_test, y_test), verbose=1)
```

```
Train on 34650 samples, validate on 17067 samples
Epoch 1/10
34650/34650 [==============================] - 31s 901us/step - loss: 7.4128 - val_loss: 7.2932
Epoch 2/10
34650/34650 [==============================] - 30s 857us/step - loss: 7.2707 - val_loss: 7.2865
Epoch 3/10
34650/34650 [==============================] - 30s 856us/step - loss: 7.2665 - val_loss: 7.2836
Epoch 4/10
34650/34650 [==============================] - 29s 850us/step - loss: 7.2650 - val_loss: 7.2831
Epoch 5/10
34650/34650 [==============================] - 29s 846us/step - loss: 7.2647 - val_loss: 7.2830
Epoch 6/10
34650/34650 [==============================] - 29s 848us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 7/10
34650/34650 [==============================] - 30s 865us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 8/10
34650/34650 [==============================] - 29s 851us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 9/10
34650/34650 [==============================] - 29s 847us/step - loss: 7.2646 - val_loss: 7.2830
Epoch 10/10
34650/34650 [==============================] - 29s 843us/step - loss: 7.2646 - val_loss: 7.2830
```

Out[ ]:

<keras.callbacks.callbacks.History at 0x7efeca1a5c18>

In [ ]:

```
model_3.evaluate(x_test,y_test)
```

```
17067/17067 [==============================] - 4s 256us/step
```

Out[ ]:

```
7.282953401102768
```

In [ ]:

```
model_3.save_weights('model_3_weights.h5')
```

Conclusion

In [ ]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Mean Squared Loss']
ptable.add_row(["Random forest Regressor","0.02"])
ptable.add_row(["Decision tree Regressor","0.06"])
ptable.add_row(["CNN-LSTM","0.07"])
ptable.add_row(["CNN Model","0.07"])
ptable.add_row(["CNN-LSTM","0.07"])
ptable.add_row(["XGboost Regressor","0.03"])
ptable.add_row(["Linear Regressor","0.08"])
print(ptable)
```

```
+-------------------------+-------------------+
|          Model          | Mean Squared Loss |
+-------------------------+-------------------+
| Random forest Regressor |        0.02       |
| Decision tree Regressor |        0.06       |
|         CNN-LSTM        |        0.07       |
|        CNN Model        |        0.07       |
|         CNN-LSTM        |        0.07       |
|    XGboost Regressor    |        0.03       |
|     Linear Regressor    |        0.08       |
+-------------------------+-------------------+
```

Random Forest Regressor Model is giving the Best MSE Value of 0.02