Rating Prediction of Zomato Restaurants

In [1]:

```
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 [2]:

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Data Preprocessing

In [104]:

```
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', "d
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]   Unzipping corpora/stopwords.zip.
```

In [166]:

```
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
```

We have to define 2 functions -

1. **func_1 - It should take in raw data as input and returns the prediction for the input.**
2. **func_2 - It should take in raw data as input and retunrs the performance metrics value ie. mean squared error .**

Function - 1

In [162]:

```
def func_1(param):
  a=votes_scalar.transform(param['votes'].values.reshape(-1, 1))
  b=cost_scalar.transform(param['approx_cost(for two people)'].values.reshape(-1, 1))
  c=vectorizer_location.transform(param['location'])
  d=vectorizer_dish_liked.transform(param['dish_liked'])
  e=number_of_cuisines_scalar.transform(param['number_of_cuisines'].values.reshape(-1, 1))
  f=vectorizer_book_table.transform(param['book_table'])
  cleaned_reviews = filterised_text(param['preprocessed_reviews'].astype(str).values)
  param['cleaned_reviews']  = cleaned_reviews
  param=param.drop(['preprocessed_reviews'],axis=1)
  g=vectorizer_text.transform(param['cleaned_reviews'])
  h=vectorizer_rest_type.transform(param['rest_type'])
  i= vectorizer_online_order.transform(param['online_order'])
  j=vectorizer_cuisines.transform(param['cuisines'])
  k=vectorizer_listed_in_tp.transform(param['listed_in(type)'])
  l=vectorizer_listed_in_ct.transform(param['listed_in(city)'])
  m=dish_liked_scalar.transform(param['mean_dish_liked'].values.reshape(-1, 1))
```

```
  n=mean_cuisines_scalar.transform(param['mean_cuisines'].values.reshape(-1, 1))
  o=number_of_liked_dishes_scalar.transform(param['number_of_liked_dishes'].values.reshape(-1, 1))
  p=Facilities_offered_scalar.transform(param['Facilities_offered'].values.reshape(-1, 1))
  final=hstack((a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p))
  x=rfm.predict(final)
  return x
```

In [150]:

```
query_point=pd.DataFrame({'online_order':['Yes','No'],'book_table':['Yes','No'],'votes':[705,455],
'location':['Mysore Road','Banashankari'],'rest_type':['Casual Dining','Cafe'],'cuisines':['North
Indian South Indian Andhra Chinese','Cafe Chinese Continental Italian'],'approx_cost(for two peopl
e)':[800.0,400.0],'listed_in(type)':['Buffet','Cafes'],'listed_in(city)':
['Banashankari','Banashankari'],'dish_liked':['Masala Dosa','Masala Dosa'],'number_of_cuisines':[4
,4],'number_of_liked_dishes':[1,7],'Facilities_offered':[0,1],'mean_dish_liked':
[3.69,3.55],'mean_cuisines':[3.60,3.55],'preprocessed_reviews':['hi this is kishan','this food is
good']})
```

In [163]:

```
func_1(query_point)
```

```
100%|██████████| 2/2 [00:00<00:00, 336.47it/s]
```

Out[163]:

```
array([3.621, 3.65 ])
```

Function - 2

In [165]:

```
def func_2(param,label):
  a=votes_scalar.transform(param['votes'].values.reshape(-1, 1))
  b=cost_scalar.transform(param['approx_cost(for two people)'].values.reshape(-1, 1))
  c=vectorizer_location.transform(param['location'])
  d=vectorizer_dish_liked.transform(param['dish_liked'])
  e=number_of_cuisines_scalar.transform(param['number_of_cuisines'].values.reshape(-1, 1))
  f=vectorizer_book_table.transform(param['book_table'])
  cleaned_reviews = filterised_text(param['preprocessed_reviews'].astype(str).values)
  param['cleaned_reviews']  = cleaned_reviews
  param=param.drop(['preprocessed_reviews'],axis=1)
  g=vectorizer_text.transform(param['cleaned_reviews'])
  h=vectorizer_rest_type.transform(param['rest_type'])
  i= vectorizer_online_order.transform(param['online_order'])
  j=vectorizer_cuisines.transform(param['cuisines'])
  k=vectorizer_listed_in_tp.transform(param['listed_in(type)'])
  l=vectorizer_listed_in_ct.transform(param['listed_in(city)'])
  m=dish_liked_scalar.transform(param['mean_dish_liked'].values.reshape(-1, 1))
  n=mean_cuisines_scalar.transform(param['mean_cuisines'].values.reshape(-1, 1))
  o=number_of_liked_dishes_scalar.transform(param['number_of_liked_dishes'].values.reshape(-1, 1))
  p=Facilities_offered_scalar.transform(param['Facilities_offered'].values.reshape(-1, 1))


  final=hstack((a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p))
  k = rfm.predict(final)
  x = mean_squared_error(label, k)
  return x
```

In [159]:

```
from sklearn.metrics import mean_squared_error
func_2(query_point,[3.7,3.4])
```

Out[159]:

```
0.03437050000000058
```