

ZOMATO RECOMMENDATION SYSTEM USING PYSPARK

PRESENTATED BY:

ABHINEET RAJ (20MIA1146)

AYUSH MADURWAR (20MIA1009)

TANMAY TIWARI (20MIA1097)



Table Of Content

- **ABSTRACT**
- **INTRODUCTION**
- **LITERATURE REVIEW**
- **DATASET**
- **IMPLEMENTATION**
- **CONCLUSION**
- **FUTURE WORK**

ABSTRACT

The Zomato Restaurant Data Analysis and Recommendation System is a promising initiative that uses data-driven insights to offer insightful content and improve the user experience. It seeks to use data from the website Kaggle to collect data on restaurants, menus, ratings, reviews, and other pertinent information. The data is collected through descriptive statistics, exploratory data analysis, and predictive modelling methods. Restaurant recommendations are made based on user preferences and similarity-based clustering of restaurants is done using predictive modelling techniques. Collaborative filtering and content-based filtering techniques form the foundation of the recommendation system, which involves recommending restaurants to users based on their prior actions. The project has the potential to revolutionise the restaurant business by providing stakeholders with insightful information and improving the dining experience for customers.



INTRODUCTION

The Zomato Restaurant Data Analysis and Recommendation System aims to use machine learning and data analytics to provide tailored restaurant recommendations based on users' preferences. It uses Pyspark, an open-source data processing framework, to process the enormous amounts of restaurant data needed for analysis. Data pre-processing involves cleaning and transforming the data, while exploratory data analysis involves visualising the data and finding correlations between variables. The system has the potential to improve users' dining experiences and give stakeholders in the restaurant industry insightful data.



LITERATURE SURVEY

Data analysis and recommendation systems are essential tools for the restaurant industry, such as PySpark, which use data analytics to improve operations, menu, and pricing. These systems are used by online food delivery services such as Uber Eats and Grubhub to offer personalised recommendations. The two primary categories of recommendation systems are collaborative filtering and content-based filtering. Pyspark is an example of a successful recommendation system, as it allows customers to enjoy more individualised dining experiences.

DATASET DESCRIPTION

The collected data has been stored in the Comma Separated Value file Zomato.csv. It consists of 18 rows and 9 lakhs columns. It contains this information:

- Ø URL: It contains the URL of the restaurant in the Zomato website
- Ø Address: It contains address of the restaurant.
- Ø Name: It displays the name of the restaurant.
- Ø Online order: It shows whether online ordering is available in the restaurant or not
- Ø Book table: It shows whether table book option available or not
- Ø Rate: it contains the overall rating of the restaurant out of 5.
- Ø Votes: It contains total number of rating for the restaurant.
- Ø Phone: It contains phone number of restaurant
- Ø Location: It contains the neighbourhood in which restaurant is located
- Ø Rest type: It displays restaurant types

METHODOLOGY



PHASE I

DATASET LOADING



PHASE 2

EDA OF DATASET



PHASE 3

RECOMMENDATION SYSTEM
BUILDING

First of all, we imported all the required libraries which helped in EDA and building recommendation system. AND We load the dataset from gdrive folder.

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import r2_score
```

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

Mounted at /gdrive

data= spark.read.csv('/gdrive/My Drive/Big Data Framework/zomato.csv', inferSchema=True, header=True)
```


Online orders

IMPLEMENTATION

CONT.

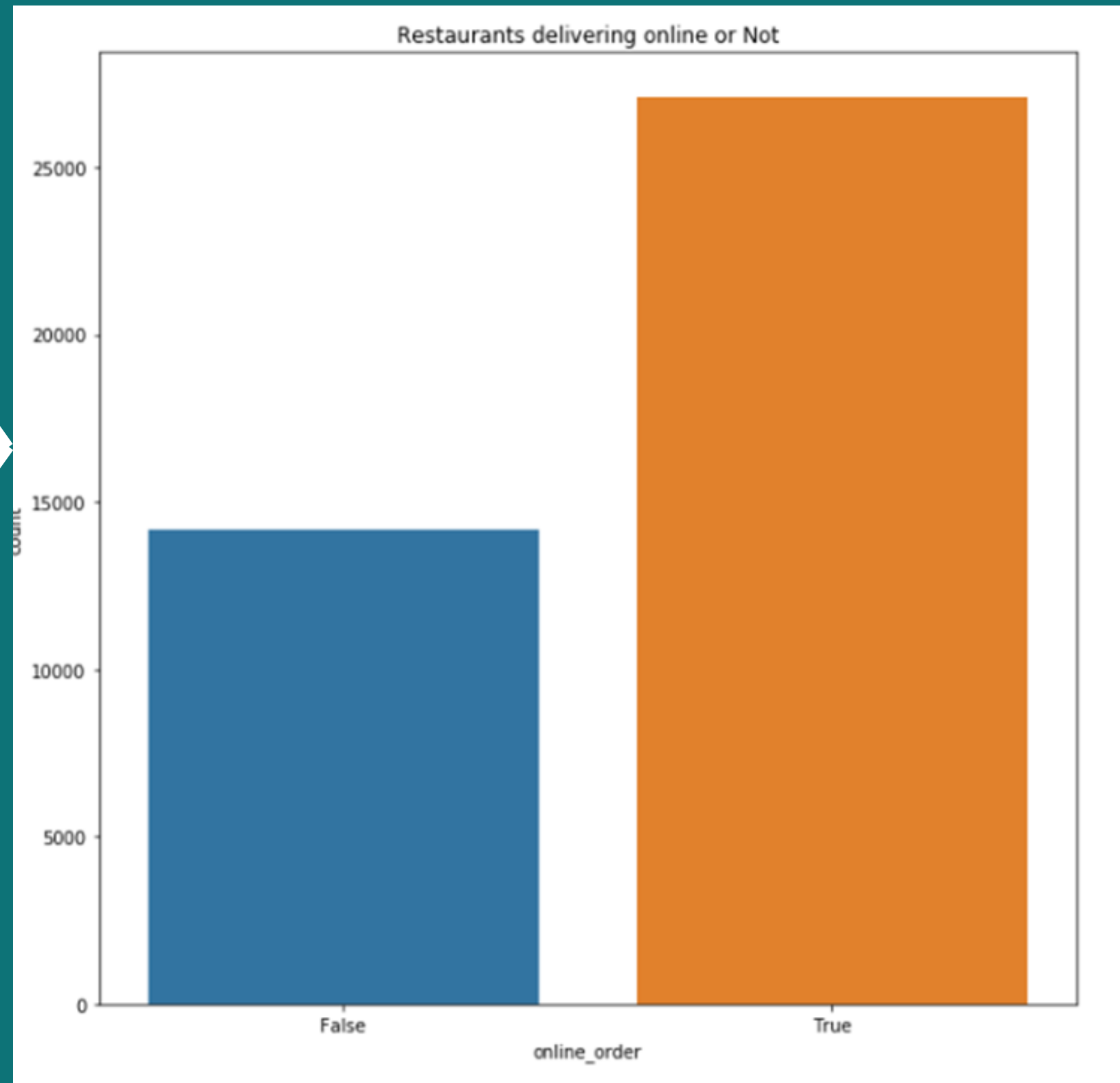


Table Booking Facility

IMPLEMENTATION

CONT.

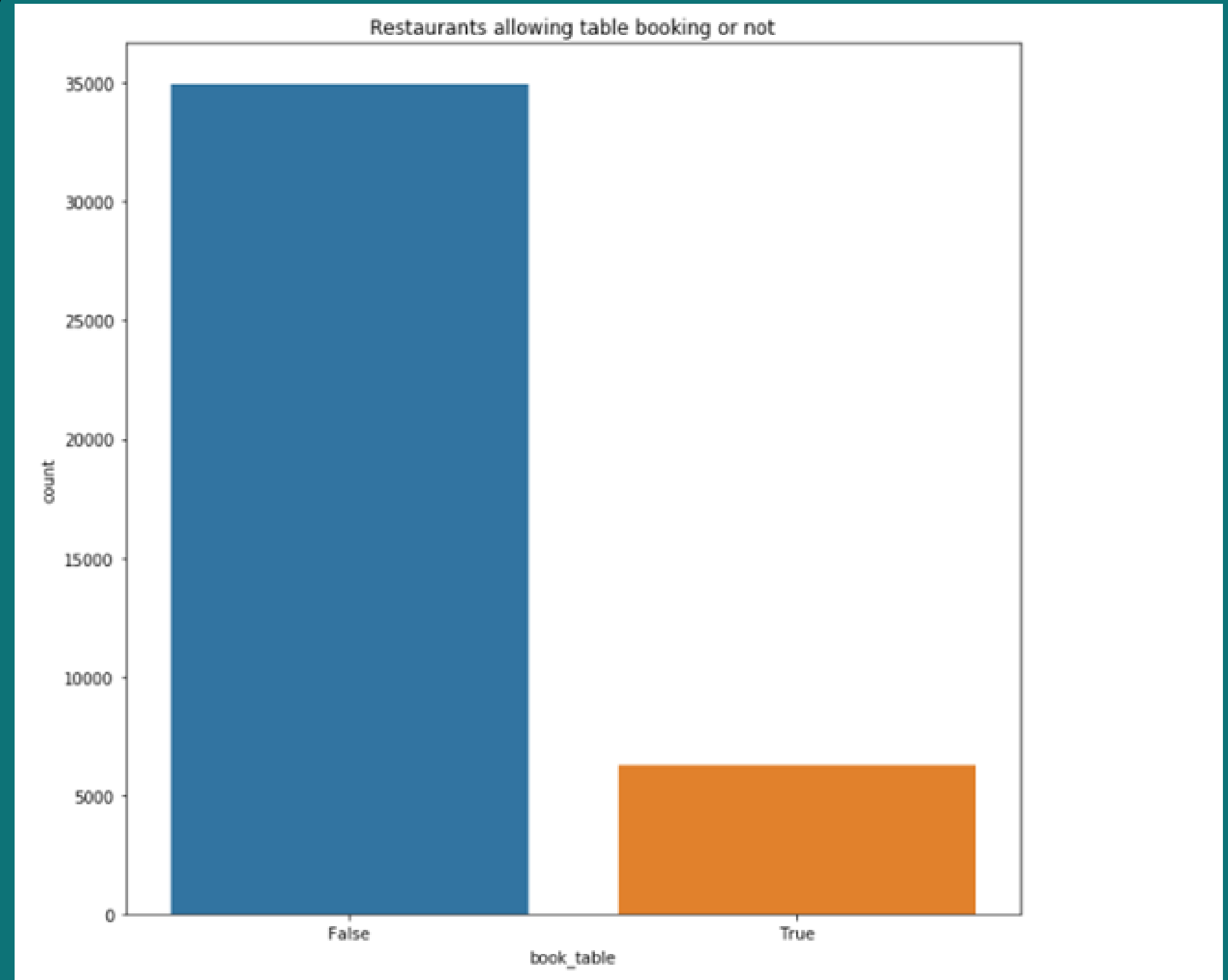
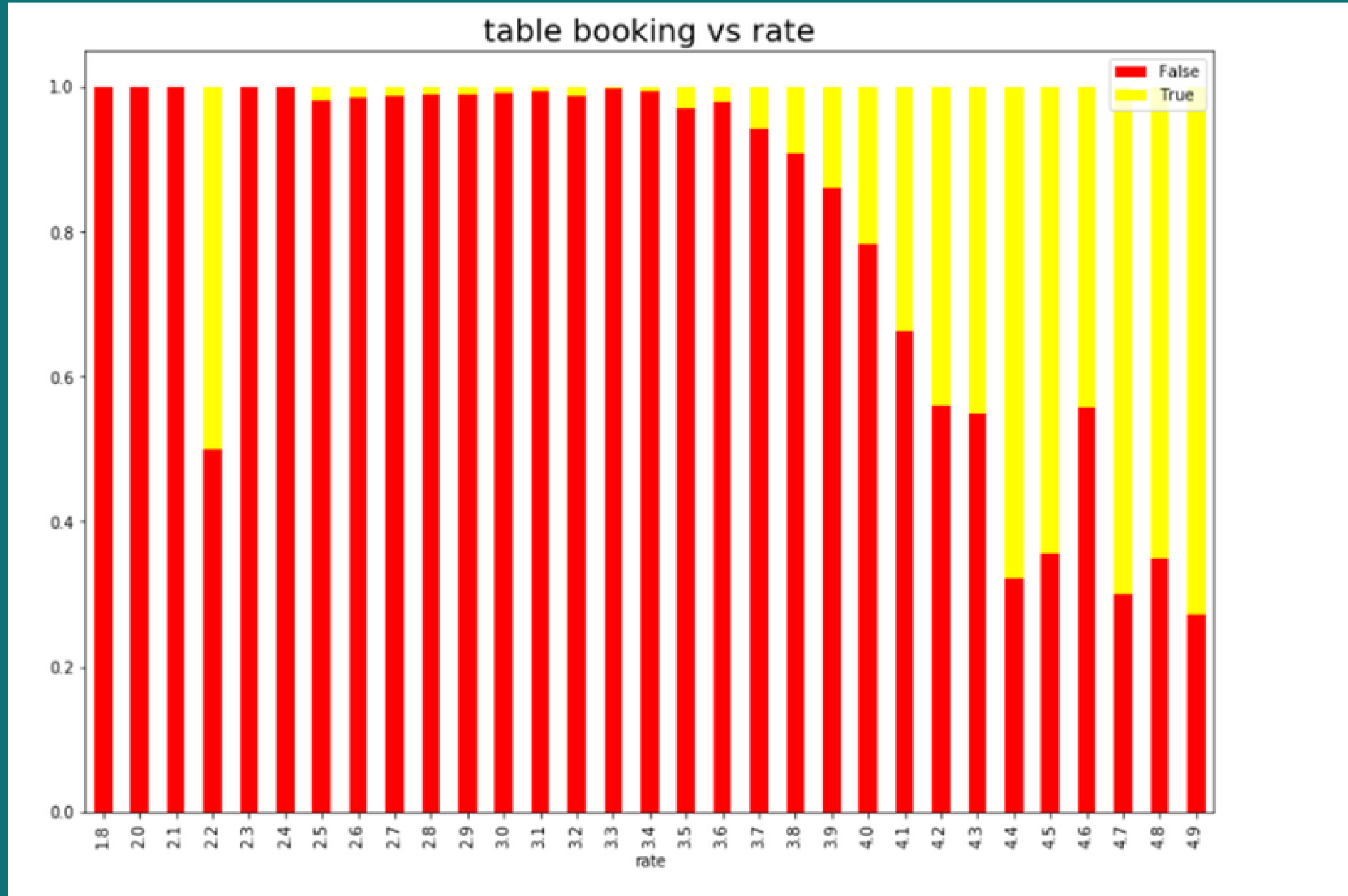


Table Booking Rate vs Rate

IMPLEMENTATION

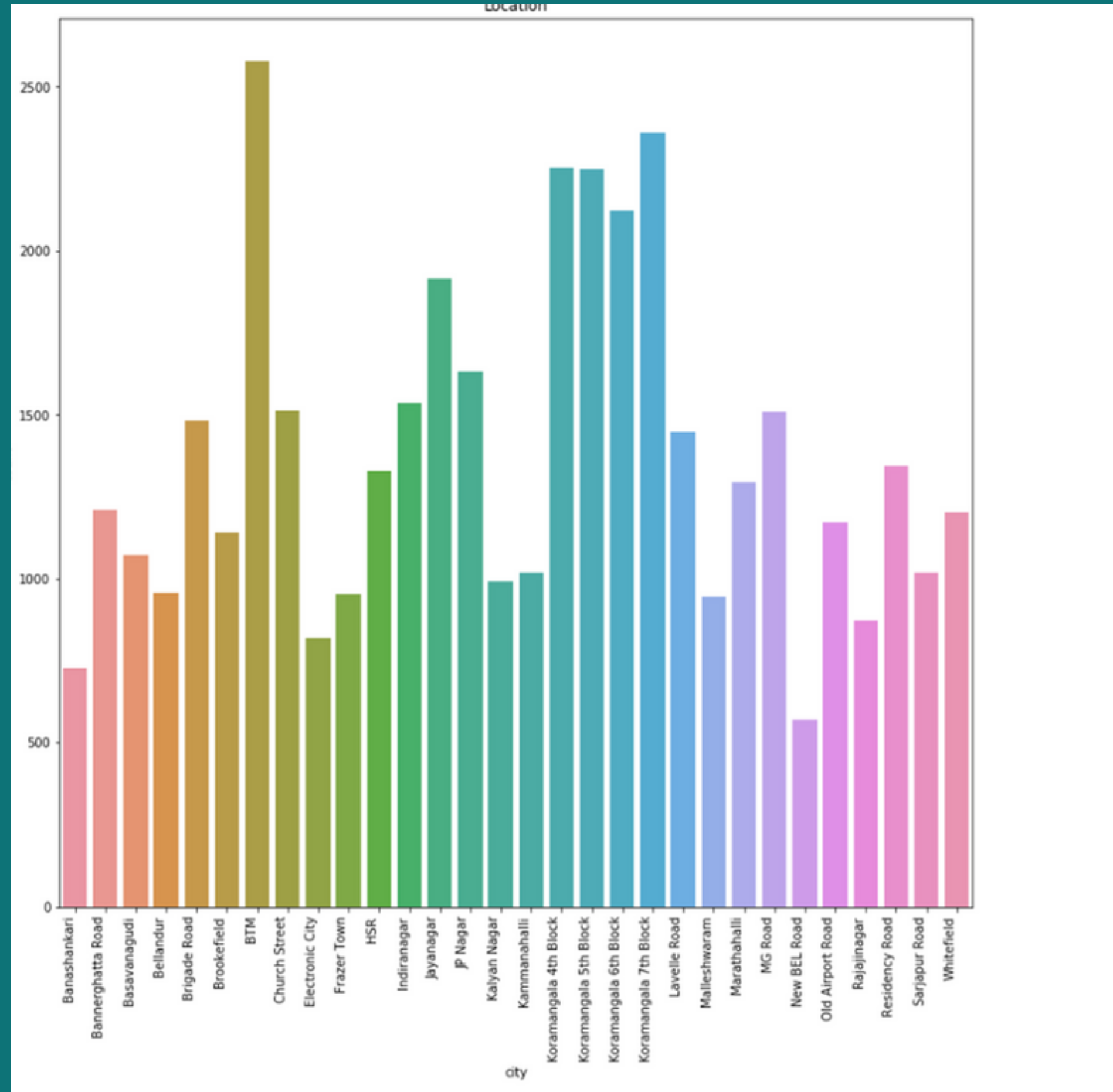
COND.



Number of Restaurants in City

IMPLEMENTATION

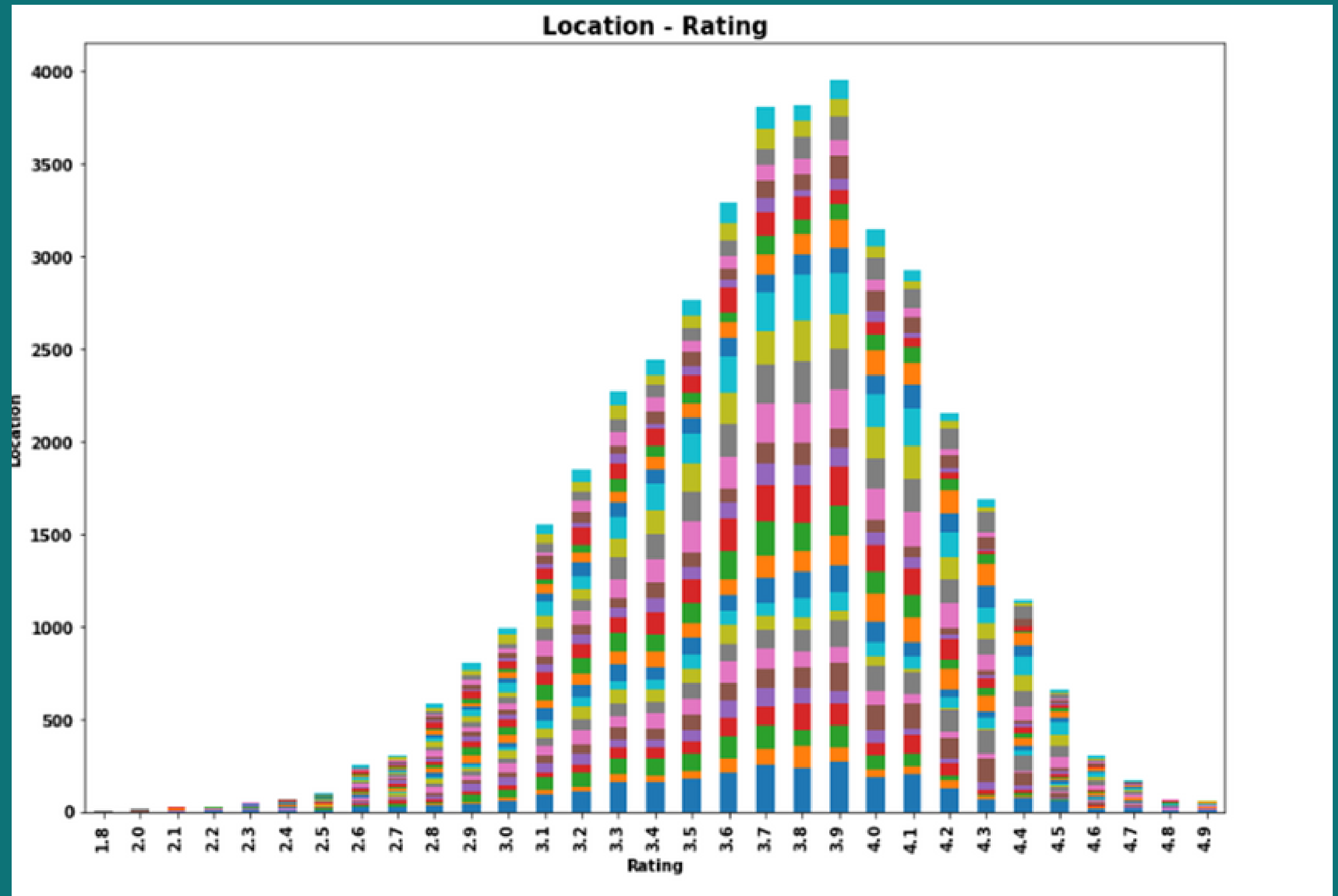
CONT.D.



Location vs Rate

IMPLEMENTATION

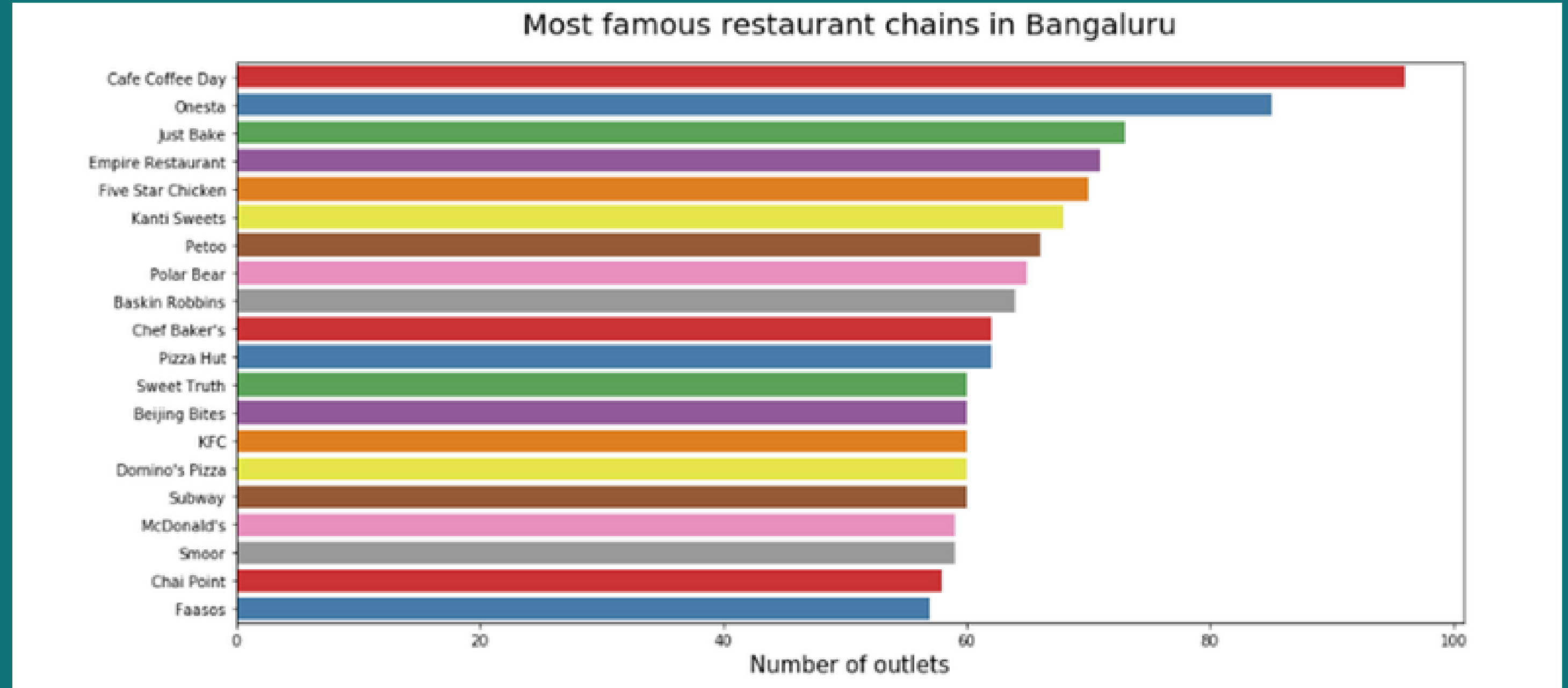
CONT.



Most famous restaurants

IMPLEMENTATION

CONT.D.



SETUP PYSPARK

IMPLEMENTATION

CONT.D.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://dlcdn.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz
!tar -zxvf spark-3.3.2-bin-hadoop3.tgz
```

```
!pip install pyspark

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.3.2.tar.gz (281.4 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 5.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.5
  Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 KB 22.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.3.2-py2.py3-none-any.whl size=281824028 sha256=76652a3f491488c7404051da517552e34fb26885fbed0ca9a8640007f1098255
  Stored in directory: /root/.cache/pip/wheels/6c/e3/9b/0525ce8a69478916513509d43693511463c6468db0de237c86
Successfully built pyspark
Installing collected packages: py4j, pyspark
  Attempting uninstall: py4j
    Found existing installation: py4j 0.10.9.7
    Uninstalling py4j-0.10.9.7:
      Successfully uninstalled py4j-0.10.9.7
  Successfully installed py4j-0.10.9.5 pyspark-3.3.2
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.3.2-bin-hadoop3"
```

```
import pyspark
spark = pyspark.sql.Session.builder.appName("zomato-recommendation").getOrCreate()
sc = spark.sparkContext
```

We have deleted the unnecessary columns and again done the pre-processing steps such as removing null values and renaming columns according to our use.

IMPLEMENTATION

CONT D.

```
In [73]: #Deleting Unnnecessary Columns
from pyspark.sql.functions import col
zomato=data.drop("url", "dish_liked", "phone")
```

```
In [74]: from pyspark.sql.functions import count, when
count_all = zomato.count()
count_distinct = zomato.distinct().count()
count_duplicates = count_all - count_distinct
print(count_duplicates)
zomato = zomato.dropDuplicates()
```

25720

```
In [75]: from pyspark.sql.functions import col, count
null_counts = zomato.agg(*[count(col(c)).alias(c) for c in zomato.columns])
null_counts.show()
zomato = zomato.dropna(how='any')
zomato.printSchema()
```

```
# rename the columns of the DataFrame
zomato = zomato.withColumnRenamed("approx_cost(for two people)", "cost") \
               .withColumnRenamed("listed_in(type)", "type") \
               .withColumnRenamed("listed_in(city)", "city")
```

```
# print the column names of the DataFrame
print(zomato.columns)
```

```
In [78]: from pyspark.sql.functions import regexp_replace
zomato = zomato.withColumn("cost", col("cost").cast("string"))
zomato = zomato.withColumn("cost", regexp_replace(col("cost"), ",", "."))
zomato = zomato.withColumn("cost", col("cost").cast("float"))
zomato.printSchema()
```

We have found the mean rating of the restaurant and stored them in a new column named as 'Mean Rating' in the dataset.

IMPLEMENTATION

CONT D.

```
In [83]: from pyspark.sql.functions import mean
          from pyspark.sql.window import Window
          from pyspark.sql.functions import col

          window = Window.partitionBy("name")
          zomato = zomato.withColumn("Mean Rating", mean(col("rate")).over(window))
```

```
In [86]: from pyspark.ml.linalg import Vectors
          from pyspark.sql.functions import udf
          from pyspark.sql.types import DoubleType
          from pyspark.ml.linalg import VectorUDT
          to_vector = udf(lambda x: Vectors.dense(x), VectorUDT())

          zomato = zomato.withColumn("Mean Rating", to_vector(col("Mean Rating")))
```

MinMaxScaler object is created with input column as "Mean Rating" and output column as "Scaled Mean Rating". The minimum and maximum values for scaling are set to 1 and 5, respectively and fitting and transforming the data is done and the values are round off to two decimal places

```
In [91]: from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.linalg import Vectors
from pyspark.sql.functions import col, udf
from pyspark.sql.types import DoubleType

vector_to_double = udf(lambda x: float(x[0]), DoubleType())
scaler = MinMaxScaler(inputCol="Mean Rating", outputCol="Scaled Mean Rating", min=1, max=5)
scalerModel = scaler.fit(zomato)
zomato = scalerModel.transform(zomato)
zomato = zomato.withColumn("Scaled Mean Rating", vector_to_double(col("Scaled Mean Rating")))
zomato = zomato.withColumn("Scaled Mean Rating", F.round(col("Scaled Mean Rating"), 2))
zomato.sample(0.1).show(3)
```


Removing URLs and punctuations

IMPLEMENTATION

CONT.D.

```
In [100... import re
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)

remove_urls_udf = udf(remove_urls, StringType())

zomato = zomato.withColumn("reviews_list", remove_urls_udf(zomato["reviews_list"]))

In [101... zomato.select('reviews_list', 'cuisines').sample(False, 0.05)

Out[101]: DataFrame[reviews_list: string, cuisines: string]

In [104... from pyspark.ml.feature import CountVectorizer
from pyspark.sql.functions import udf, lit, array
from pyspark.sql.types import ArrayType, StructType, StructField, StringType, IntegerType

def get_top_words(column, top_nu_of_words, nu_of_word):
    vec = CountVectorizer(ngram_range=nu_of_word, stopWords='english')
    bag_of_words = vec.fit(column).transform(column)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:top_nu_of_words]

get_top_words_udf = udf(get_top_words, ArrayType(StructType([
    StructField("word", StringType(), True),
    StructField("frequency", IntegerType(), True)
])))

zomato = zomato.withColumn("top_words", get_top_words_udf(zomato["reviews_list"], lit(10), array(lit(1), lit(2)))))
```

```
from pyspark.sql.functions import col

indices = df_percent.select(col("index")).rdd.flatMap(lambda x: x).collect()

from pyspark.ml.feature import HashingTF, IDF, Tokenizer

tokenizer = Tokenizer(inputCol="reviews_list", outputCol="words")
wordsData = tokenizer.transform(df_percent)

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
tfidf_matrix = idfModel.transform(featurizedData).select("features")

from pyspark.ml.feature import Normalizer

normalizer = Normalizer(inputCol="features", outputCol="norm")
data = normalizer.transform(tfidf_matrix)

cosine_similarities = data.rdd.cartesian(data.rdd).map(lambda x: (x[0][0], x[1][0], float(x[0][1].dot(x[1][1])))).toDF(["id1", "id2", "similarity"])
```

Recommending system function building

```
[161_ from pyspark.sql.functions import col

def recommend(name, cosine_similarities=cosine_similarities):
    recommend_restaurant = []
    idx = indices.index(name)
    score_series = pd.Series(cosine_similarities[idx]).sort_values(ascending=False)

    top30_indexes = list(score_series.iloc[0:31].index)

    for each in top30_indexes:
        recommend_restaurant.append(list(df_percent.index)[each])

    df_new = spark.createDataFrame([], ['cuisines', 'Mean Rating', 'cost'])

    for each in recommend_restaurant:
        df_new = df_new.union(df_percent.select(['cuisines', 'Mean Rating', 'cost']).where(col('name') == each).sample(False, 1.0))

    df_new = df_new.dropDuplicates(['cuisines', 'Mean Rating', 'cost'])
    df_new = df_new.orderBy(col('Mean Rating').desc()).limit(10)

    print('TOP %s RESTAURANTS LIKE %s WITH SIMILAR REVIEWS: ' % (str(df_new.count()), name))

    return df_new
```


```
In [ ]: recommend('Pai Vihar')
```


results

Index	
Gokul Kuteera	North Indian, Chinese, South Indian
Cinnamon	North Indian, Chinese, Biryani
Pazzito Kitchen	North Indian
Kakaji	North Indian
Mayura Sagar	Chinese, North Indian, South Indian
Melange - Hotel Ekao	North Indian, Chinese, Continental, Mangalorean
Punjabi Tasty Khana	North Indian, Chinese, Biryani
Sri Lakshmi Dhaba	North Indian, Chinese
Punjabi Dawat	North Indian, Chinese

CONCLUSION

Popular online food delivery service Zomato offers a huge database of restaurant information, menus, and customer reviews. Zomato can develop an intelligent recommender system that can provide users with personalised recommendations based on their preferences and history using this enormous amount of data. In this project, we analysed the data and created a recommender system for Zomato using Pyspark, a well-known big data processing framework. Cleaning and pre-processing the Zomato data as part of the project's first phase made it ready for analysis.



FUTURE WORK

The Zomato data analysis and recommendation system using PySpark has a number of potential areas for further research. By incorporating more sophisticated machine learning techniques like deep learning and reinforcement learning, the current recommendation system can be made better. These methods can be used to extract from the data more intricate patterns and relationships, producing recommendations that are more precise.



TEAM MEMBER

- 1.ABHINEET RAJ (20MIA1146)**
- 2.AYUSH MADURWAR (20MIA1097)**
- 3.TANMAY TIWARI (20MIA1009)**

The image features a solid teal background. In the center is a white hexagon with a thick teal border. The words "THANK YOU" are written in a bold, dark grey, sans-serif font, centered within the hexagon. The text is arranged in two lines: "THANK" on the top line and "YOU" on the bottom line. There are also some grey geometric shapes in the corners of the image, specifically triangles pointing towards the center.

**THANK
YOU**