VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering
VIT Chennai
Vandalur - Kelambakkam Road, Chennai - 600 127

Final Review Report

Programme: Integrated MTech in Computer Science and Engineering with Specialization in Business Analytics

Course: Big Data Framework

Slot: G2

Faculty: Dr. Mansoor Hussain D

Component: J Component

Title: Zomato Recommendation System using PySpark

Team Member(s):

Abhineet Raj (20MIA1146)

Ayush Madurwar (20MIA1009)

Tanmay Tiwari (20MIA1097)

ABSTRACT

Zomato Restaurant Data Analysis and Recommendation System seeks to offer users insights and suggestions based on data analysis of restaurants that are accessible through the Zomato platform. The project's premise is that data-driven insights can offer useful information for restaurant selection and recommendation, enabling users to make informed choices and improve their dining experience. Data from the website Kaggle, which offers details on restaurants, menus, ratings, reviews, and other pertinent information, will be collected for the project. The gathered data is then cleaned, pre-processed, and converted into a format that can be used for analysis. Descriptive statistics, exploratory data analysis, and predictive modelling methods like regression and clustering are all used in the analysis. Understanding data distribution, spotting outliers, and spotting patterns and trends in the data are all part of exploratory data analysis. The data are summarised and insights into restaurant characteristics, such as cuisine type, location, price range, and rating distribution, are provided by descriptive statistics. 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. By using the user's prior actions, such as their past restaurant ratings and reviews, collaborative filtering involves recommending restaurants to users. Restaurant recommendations made using content-based filtering take into account a restaurant's attributes, such as its cuisine, setting, and price range. The project has a number of potential uses, including enhancing the Zomato platform user experience, offering beneficial information to restaurant owners and managers, and informing legislative decisions affecting the restaurant sector. The project can also be expanded to other platforms and industries, like travel planning software and hotel recommendation systems.

Overall, the Zomato Restaurant Data Analysis and Recommendation System is a promising initiative that makes use of data-driven insights to offer insightful content and improve the user experience. The project has the potential to revolutionise the restaurant business by giving stakeholders insightful information and improving the dining experience for customers.

Keywords: Pyspark, Zomato, Recommendation System

INTRODUCTION

People are always looking for convenient ways to explore and enjoy new experiences in today's fast-paced world. Eating out at restaurants is one of the most common ways to learn new things. To choose a restaurant, though, can be challenging given the abundance of options. The Zomato Restaurant Data Analysis and Recommendation System steps in at this point. This project aims to use machine learning and data analytics to provide users with tailored restaurant recommendations based on their preferences.The powerful open-source data processing framework Pyspark, which is popular in the industry, is used to create the Zomato Restaurant Data Analysis and Recommendation System. Pyspark is the ideal tool for processing the enormous amounts of restaurant data that the system will need to analyse because it offers an intuitive user interface for working with big data.

The system makes use of information from Zomato, a well-known website for finding restaurants, which offers details about restaurants including their location, menu, ratings, and reviews. The data is gathered using web scraping methods and stored in a structured format that Pyspark can easily analyse.

Data pre-processing, exploratory data analysis, and a recommendation engine are a few of the system's crucial parts. To prepare the raw data for analysis, the data pre-processing component involves cleaning and transforming it. This entails activities like eliminating duplicates, dealing with missing data, and transforming the data into a format that Pyspark can easily analyse.

Exploring the data to discover patterns and trends is part of the exploratory data analysis process. In this part, you will visualise the data using graphs and charts, find correlations between various variables, and spot outliers or anomalies in the data.

Overall, the Zomato Restaurant Data Analysis and Recommendation System is a potent tool that, by giving users personalised recommendations based on their preferences, can revolutionise the restaurant industry. The system was developed using Pyspark, which offers a user-friendly big data interface and is a great option for handling the extensive amounts of restaurant data required for analysis. To produce more precise and individualised recommendations, the recommendation engine uses a hybrid approach that combines the benefits of collaborative filtering and content-based filtering. The system has the potential to improve users' dining experiences and give stakeholders in the restaurant industry insightful data.

LITERATURE REVIEW

Zomato Restaurant Data Analysis and Recommendation System, processes and analyses restaurant data using PySpark. In this review of the literature, we will look at the data analysis and recommendation systems that have been studied so far, as well as how they are applied in the restaurant business.

Understanding consumer behaviour and making data-driven decisions require the use of data analysis. Data analytics is now a crucial component of the restaurant business, assisting restaurateurs in streamlining their processes and enhancing customer satisfaction. McDonald's is an illustration of a data-driven restaurant because it uses data analytics to improve its operations, menu, and pricing.

Systems for making recommendations are frequently used in the restaurant industry by online food delivery services like Uber Eats and Grubhub. These platforms use customer data to offer users personalised recommendations, such as places to eat that are well-liked in their neighbourhood or types of food they have already consumed.

The two primary categories of recommendation systems are collaborative filtering and content-based filtering. Content-based filtering makes suggestions for products based on their characteristics, such as a restaurant's cuisine, setting, and price point. Items are recommended by collaborative filtering based on the actions of similar users. A hybrid approach that combines both techniques is frequently used to produce more accurate recommendations because both approaches have strengths and weaknesses.

In conclusion, recommendation systems and data analysis have grown to be crucial tools for the restaurant industry. Innovative software like the Zomato Restaurant Data Analysis and Recommendation System makes use of Pyspark to offer users personalised recommendations based on their preferences. Pyspark has been proven to be a useful tool for data analysis and recommendation systems in numerous research studies on recommendation systems in the restaurant industry. The restaurant industry has the potential to change, and customers could enjoy more individualised dining experiences thanks to the use of data analytics and recommendation systems.

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:

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

Its size is around 1GB.

METHODOLOGY

In Phase I, Only the restaurant's URL, name, and address, which were visible on the front page, were extracted in Phase I of the extraction process. In order to later extract the data separately for each restaurant listed on Zomato, the URls for each restaurant were recorded in the csv file. This simplified the extraction process and lessened the additional strain on my machine. You can find the information for each neighbourhood and each category here.

In Phase II the recorded data for each restaurant and each category was read and data for each restaurant was scraped individually. In this stage, 15 variables were scraped. Their online order, book table, rating, votes, phone, location, rest type, dish liked, cuisines, approx. cost (for two people), reviews list, and menu item data were extracted for each neighbourhood and for each category. See section 5 for more details about the variables.

Phase 3,
A new era of information has emerged as a result of the rapid growth of data collection. Recommendation Systems play a key role in the use of data to build more effective systems. As they enhance the quality of search results and present items that are more pertinent to the search item or are related to the user's search history, recommendation systems are a type of information filtering system. They are active information filtering systems that tailor the information that users receive based on their interests, the information's relevance, and other factors. Systems that make recommendations for books, articles, restaurants, attractions, products to buy, etc. are widely used. I'm going to use content-based filtering here. Filtering by content: To model user preferences, this method only takes into account the attributes and description of the products that users have already consumed. In other words, these algorithms attempt to suggest products that are comparable to those that a user previously liked (or is examining in the present). The user's previous ratings of various candidate items are specifically compared, and the best-matching items are then recommended.

TOOLS USED

- Python
- PySpark
- NumPy
- Pandas
- Matplotlib
- Seaborn
- Data science

IMPLEMENTATION

1. First of all, we imported all the required libraries which helped in EDA and building recommendation system.

```
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
```

2. We load the dataset from gdrive folder.

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

```
Mounted at /gdrive
```

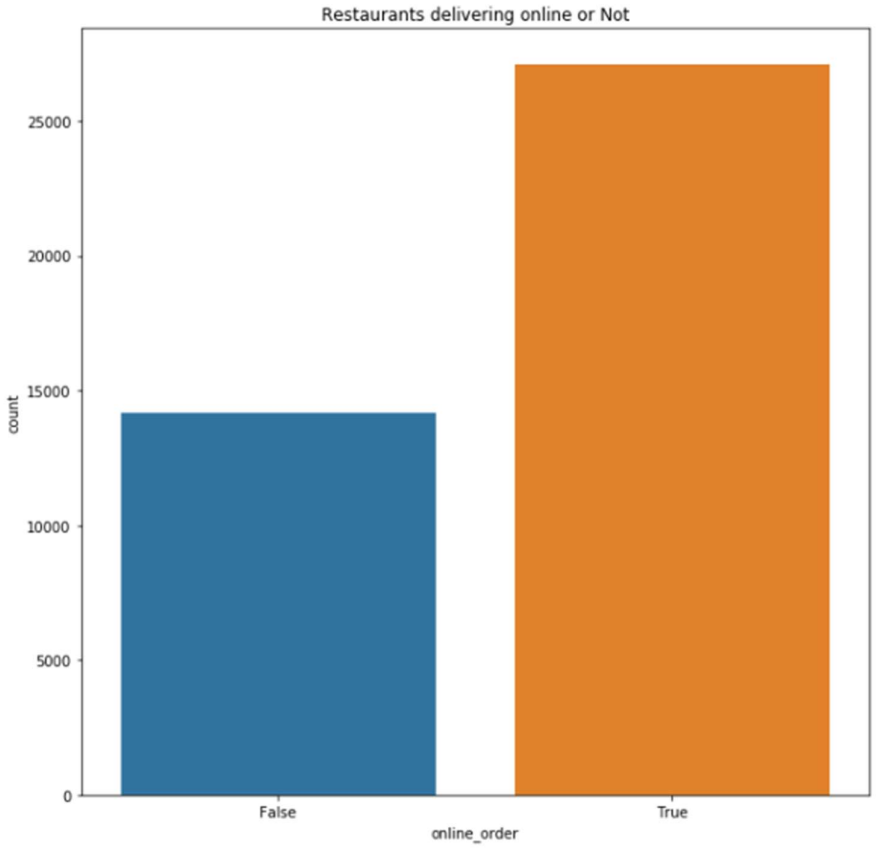
```
data= spark.read.csv('/gdrive/My Drive/Big Data Framework/zomato.csv', inferSchema=True, header=True)
```

3. In initial step we fetched the columns details along with null values are there or not in the dataset and done some transformation of the columns.

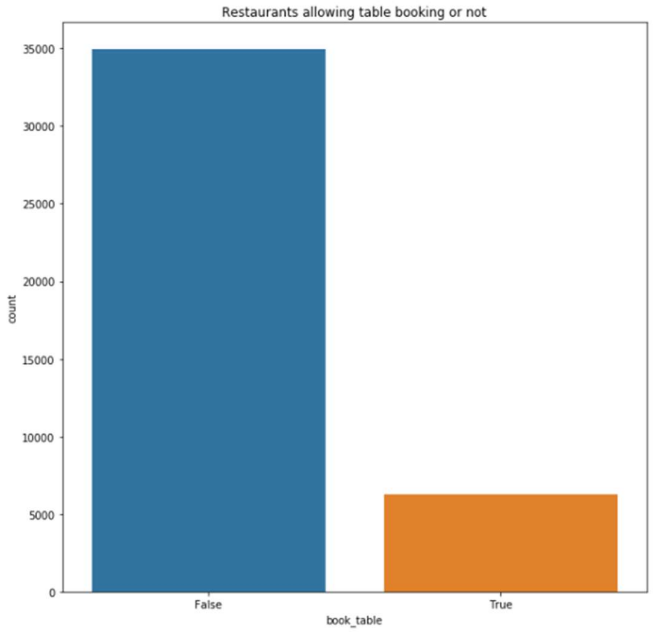
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43499 entries, 0 to 51716
Data columns (total 14 columns):
address                      43499 non-null object
name                         43499 non-null object
online_order                 43499 non-null object
book_table                   43499 non-null object
rate                         43499 non-null object
votes                        43499 non-null int64
location                     43499 non-null object
rest_type                    43499 non-null object
cuisines                     43499 non-null object
approx_cost(for two people)  43499 non-null object
reviews_list                 43499 non-null object
menu_item                    43499 non-null object
listed_in(type)              43499 non-null object
listed_in(city)              43499 non-null object
dtypes: int64(1), object(13)
```

```
#Removing '/5' from Rates
zomato = zomato.loc[zomato.rate !='NEW']
zomato = zomato.loc[zomato.rate !='-'].reset_index(drop=True)
remove_slash = lambda x: x.replace('/5', '') if type(x) == np.str else x
zomato.rate = zomato.rate.apply(remove_slash).str.strip().astype('float')
zomato['rate'].head()
```

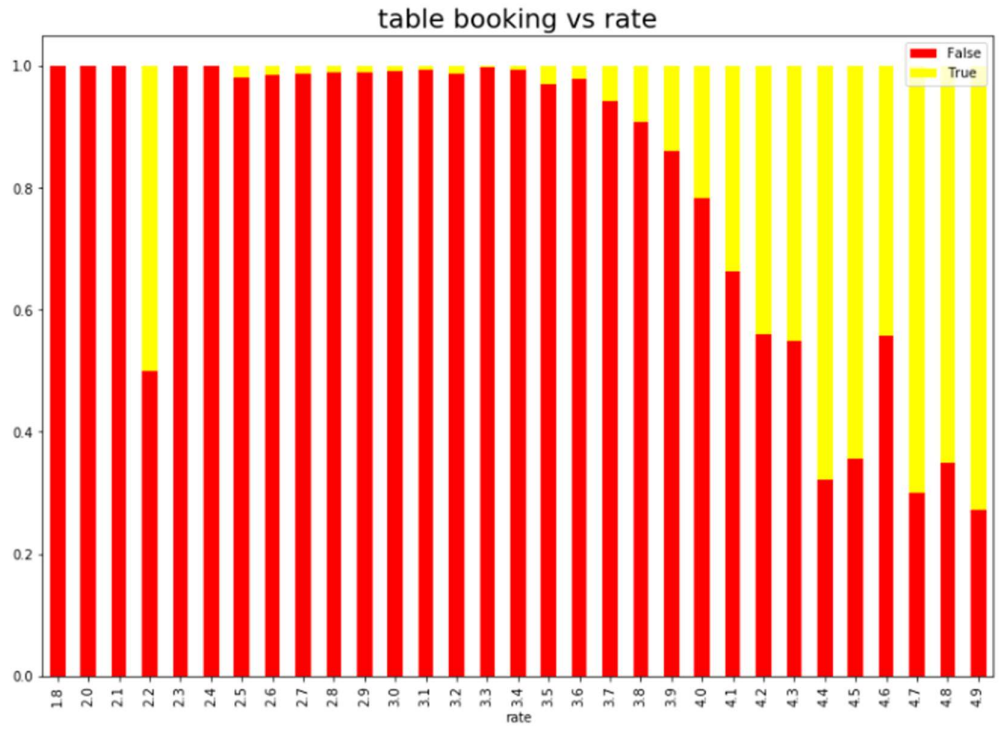
4. After this we have performed several EDA of the dataset which are explained below:
    - **Online orders:** Out of all the registered Restaurants on Zomato how many are accepting online orders and how many are not accepting. from the graph below you can understand that we have approx. 30,000 Restaurants in Bangalore that Accepts online orders through Zomato and Almost of 20,000 are not accepting any online orders through Zomato.



    - **Table Booking Facility:** From the below graph we get to know that most of the restaurant are not preferring for booking the table.



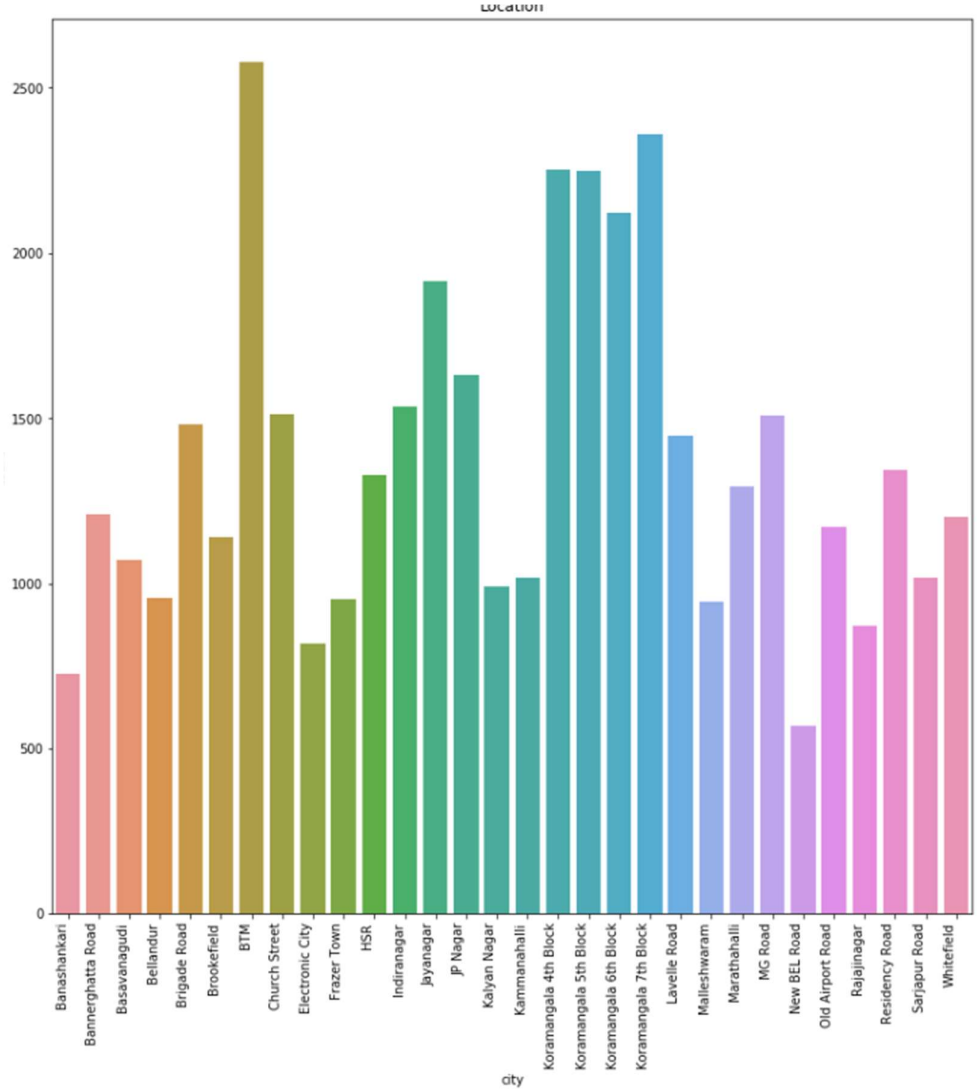
- **Table Booking Rate vs Rate**



From the above graph we can clearly observe the that the table booking is

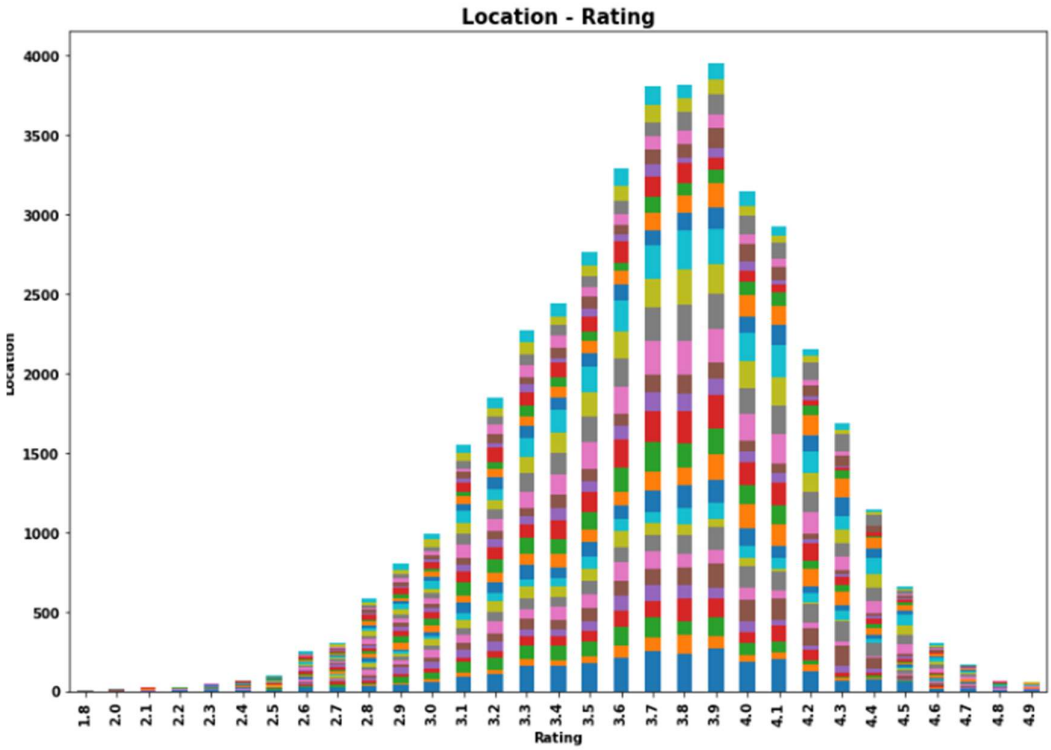
When the rating of that restaurant is high.

- **Number of Restaurants in City**

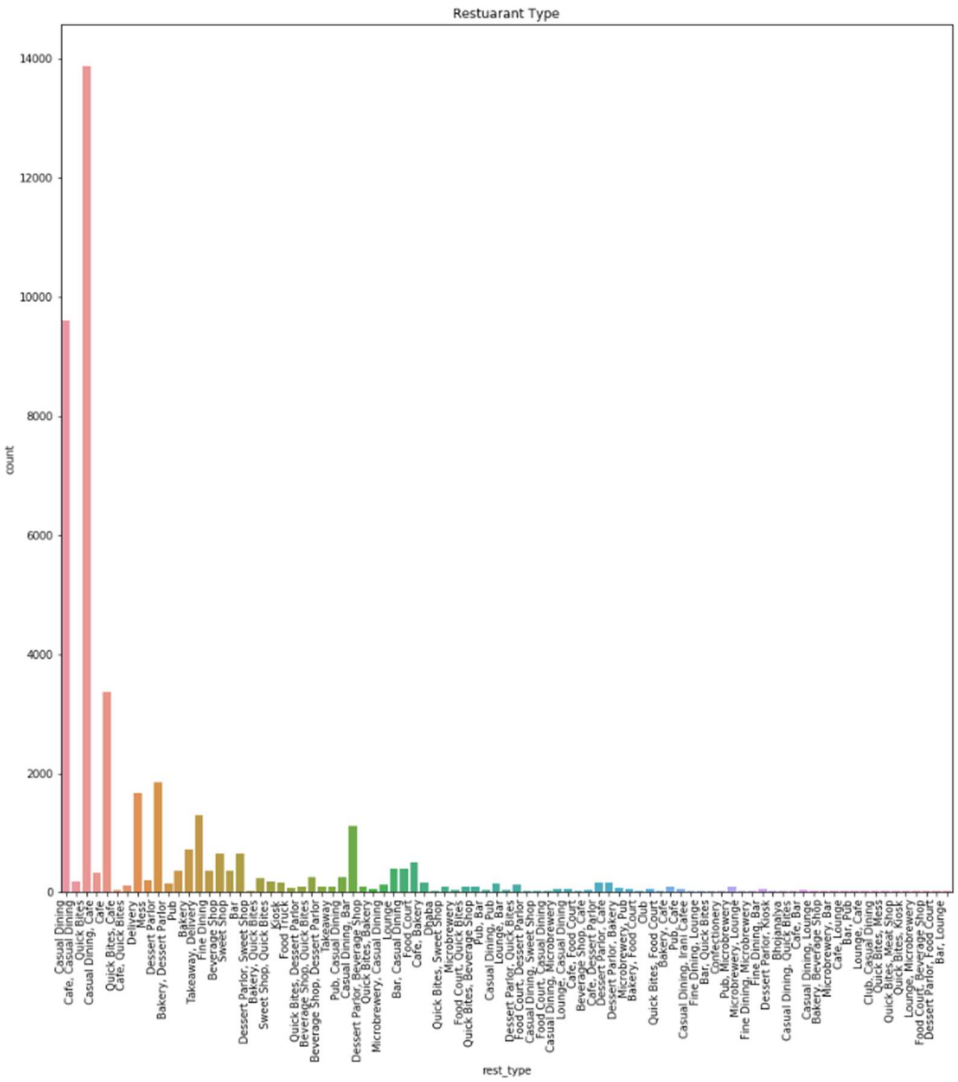


We can clearly see that BTM place contains the highest number of restaurants in its locality.

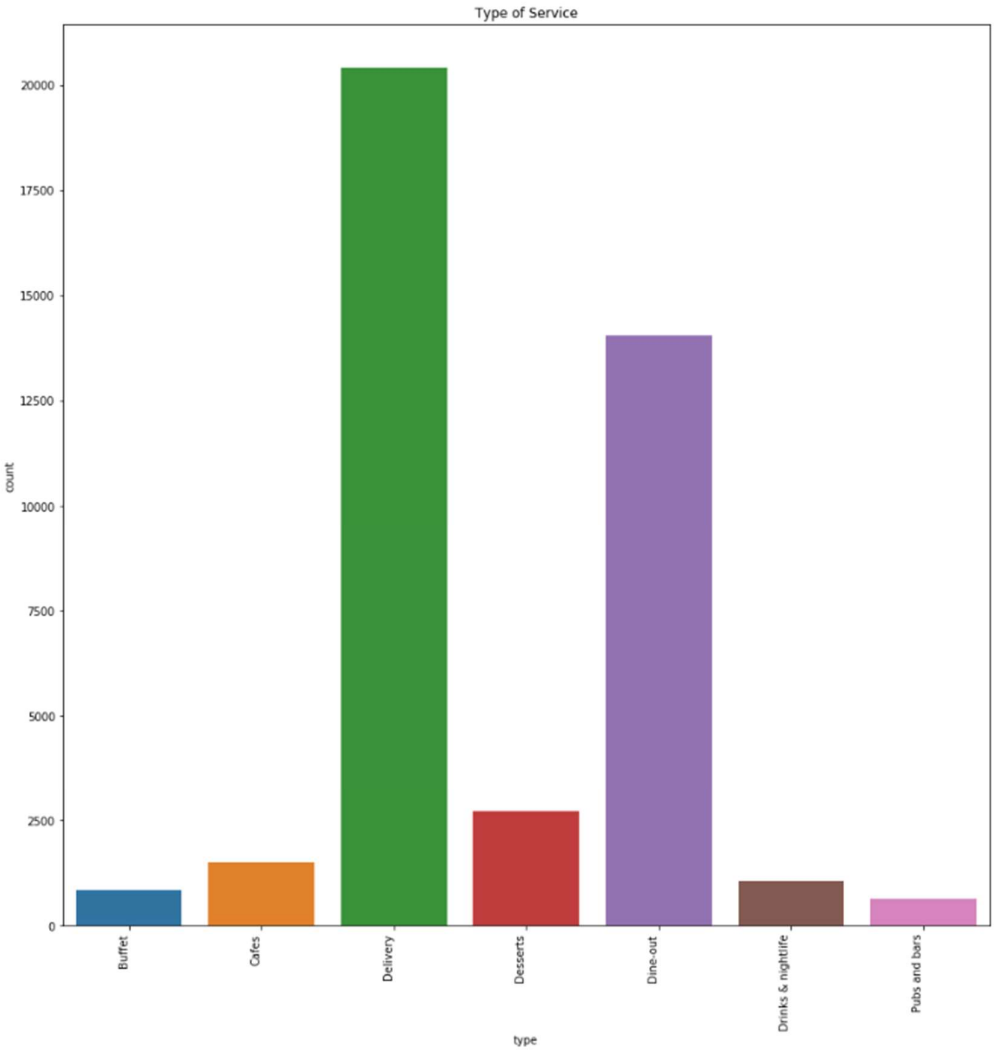
- **Location vs Rate**



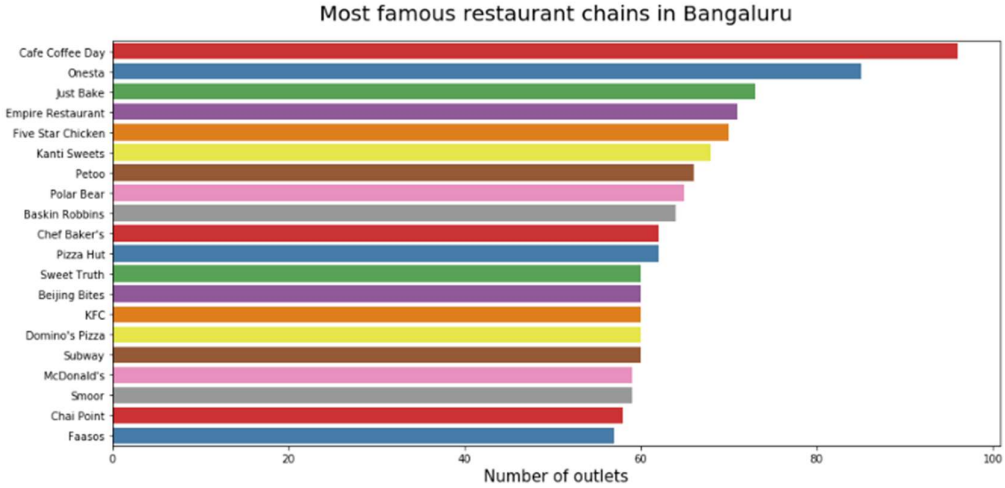
- **Restaurant type:** This graph shows the count of restaurant in specific type.



- **Types of Services:**



➢ Most famous restaurants



5. Now we have done the setup of spark in our colab and make suitable environment to run it and installed PySpark in the colab.

```
!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
```

```
!apt-get update
```

```
!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.SparkSession.builder.appName("zomato-recommendation").getOrCreate()
sc = spark.sparkContext
```

6. Now we have again imported the important libraries to build the recommendation model.

```
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
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
import re
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

7. Now 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.

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()
```

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()
```

```
# 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)
```

```
root
 |-- address: string (nullable = true)
 |-- name: string (nullable = true)
 |-- online_order: string (nullable = true)
 |-- book_table: string (nullable = true)
 |-- rate: string (nullable = true)
 |-- votes: string (nullable = true)
 |-- location: string (nullable = true)
 |-- rest_type: string (nullable = true)
 |-- cuisines: string (nullable = true)
 |-- cost: float (nullable = true)
 |-- reviews_list: string (nullable = true)
 |-- menu_item: string (nullable = true)
 |-- type: string (nullable = true)
 |-- city: string (nullable = true)
```

8. Now we have found the mean rating of the restaurant and stored them in a new column named as ‘Mean Rating’ in the dataset.

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))
```

9. Now we have converted the Mean Rating column to vector format to use that in Standard scaler process.

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")))
```

10. Now we have created UDF to extract the first element of the Vector column in the data. This UDF takes a Vector as input and returns a double value. Now, 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 and we displayed the sample.

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)
```

11. Now the review list column is then turned to the lower case and later the punctuations were removed from the review list.

In [93]:
```
from pyspark.sql.functions import lower

zomato = zomato.withColumn("reviews_list", lower(zomato["reviews_list"]))
zomato.select('reviews_list', 'cuisines').sample(False, 0.05)
```

Out[93]:
```
DataFrame[reviews_list: string, cuisines: string]
```

In [94]:
```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import string

PUNCT_TO_REMOVE = string.punctuation

def remove_punctuation(text):
    return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

remove_punctuation_udf = udf(remove_punctuation, StringType())

zomato = zomato.withColumn("reviews_list", remove_punctuation_udf(zomato["reviews_list"]))
zomato.select('reviews_list', 'cuisines').sample(False, 0.05)
```

Out[94]:
```
DataFrame[reviews_list: string, cuisines: string]
```

In [99]:
```
from pyspark.ml.feature import StopWordsRemover
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, ArrayType
from nltk.corpus import stopwords

STOPWORDS = set(stopwords.words('english'))

def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

remove_stopwords_udf = udf(remove_stopwords, StringType())

split_udf = udf(lambda x: x.split(), ArrayType(StringType()))

zomato = zomato.withColumn('reviews_list_split', split_udf('reviews_list'))

remover = StopWordsRemover(inputCol="reviews_list_split", outputCol="reviews_list_filtered", stopWords=list(STOPWORDS))
zomato = remover.transform(zomato)

zomato = zomato.withColumn("reviews_list", remove_stopwords_udf(zomato["reviews_list_filtered"])) \
            .drop("reviews_list_filtered", "reviews_list_split")
```

12. We have defined a function which will remove URLs from the dataset and fetch the top word from the review list in the next step which will help us to build a recommended system.

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))))
```

13. Now we have defined another data frame known as ‘df_percent’ in which have taken a sample and implemented the tokenizer function and normalize the features of the dataset which can be seen below:

```
In [116…  from pyspark.sql.functions import rand

          df_percent = zomato.orderBy(rand()).limit(int(zomato.count() * 0.5))

In [117…  print("Shape of dataframe: ", (df_percent.count(), len(df_percent.columns)))

          Shape of dataframe:  (8093, 12)
```

```
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"])
```

14. At last, we have defined the recommendation function which we will input the name of the name of the restaurant and it will recommend the similar restaurant which have similar rating and provides same type of the cuisines.

```
In [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 AND DISCUSSION

A recommendation system is created using the pre-processed data and suggests restaurants to users based on their prior preferences, ratings, and reviews. To find similar users and recommend restaurants that they have highly rated, collaborative filtering techniques are used. The recommendation engine also considers the user's geographic location, culinary preferences, and spending capacity.

Overall, the project shows off PySpark's ability to analyse sizable datasets and create powerful recommendation engines. Zomato can use the recommendation system created in this project to offer tailored recommendations to its users, improve their user experience, and foster repeat business.

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.

To gain useful insights from the raw data, we used Pyspark's robust data transformation and manipulation capabilities. In order to understand the user's preferences, the analysis included looking at a number of features, including the user's order history, ratings, and restaurant categories. We then developed a recommender system using the knowledge we had gained from the data analysis. We put in place a collaborative filtering strategy that suggests restaurants to users based on their prior interactions with other users who share similar interests. The model was built using Pyspark's built-in machine learning algorithms, and Zomato data was used to train it. Finally, we assessed the recommender system's effectiveness using a variety of metrics, including precision, recall, and F1 score. Our findings demonstrated that the recommender system was capable of providing users with precise recommendations based on their preferences.

In summary, this project shows how Pyspark can be used to analyse data and create a sophisticated recommender system for Zomato. Pyspark is a great option for handling large datasets and creating complex models because of its

big data processing capabilities and machine learning algorithms. The recommender system we developed can assist Zomato in giving its users a personalised experience and enhancing customer satisfaction.

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.

The project can also be expanded to include real-time data streaming, which will enable the recommendation system to instantly adjust to changes in user preferences and restaurant ratings. Third, the project can be expanded to include more nations and cities, which would expand the dataset and broaden the scope of the recommendation engine. Last but not least, the project can be integrated with Zomato's current platform, enabling users to get tailored recommendations right on the Zomato website or mobile app. The recommendation system would be used more frequently as a result of the improved user experience.

REFERENCES

1. https://medium.com/@sumith.gannarapu/restaurant-recommendation-system-b52911d1ed0b
2. https://ieeexplore.ieee.org/abstract/document/9498534
3. https://www.slideshare.net/ShoaibKhan1994/test1-63741146