CSP-554-BIG DATA TECHNOLOGIES

PROJECT REPORT

Location Based Restaurants Recommendation System

NOVEMBER 14, 2023

MANOGNA VADLAMUDI A20551908

ILLINOIS INSTITUTE OF TECHNOLOGY

PROF. PANCHAL J JAWAHAR

SECTION-I: ABSTRACT

TOPIC: Location Based Restaurants Recommendation System

ABSTRACT:

Technology has become an important part of our lives. Reviews play a key role for users in decision making. People are sharing their experiences on products or services in the form of reviews, which enables other users for their decision making. It is hard for the user to go through each review, and it might mislead the user to use a wrong service or a product if he/she goes by the review given by a user with opposite taste. Therefore, we came up with a system which recommends the user depending on his needs.

PROBLEM STATEMENT:

People are sharing their experiences on products or services in the form of reviews, whichenables other users for their decision making. It is hard for the user to go through each review, andit might mislead the user to use a wrong service or a product, if he/she goes by the review given by a user with opposite taste. Therefore, we came up with a system which recommends the user depending on his needs. In the project only the positive part of the User Review was considered for sentiment analysis, which did not yield accurate results.

PROPOSED SOLUTION:

Our work proposed a new approach to recommend user better restaurants based on analyzing daily reviews. This helps the user to choose the better restaurant on the same day. we are able to extend working on the same by implementing sentimental analysis on negative User Review, which helps us to acquire much better results.

PROJECT GOALS:

- Recommend users better restaurants.
- Analyze daily reviews.
- Sentiment Analysis on positive and negative User Reviews.
- Predict restaurant ratings.

BIG DATA TECHNOLOGIES: Apache Kafka, Apache Spark, Elasticsearch

OTHER TECHNOLOGIES: PySpark, Scala, Apache Zookeeper, Python

SECTION-II: OVERVIEW

II.I : INTRODUCTION:

These days reviews play vital role in every individual's life. It helps in choosing the best product or service. They have enhanced the probability of buying the right product, going to a good movie, eating at the best restaurant etc. But sometimes they could mislead a user in making a bad choice if he/she has opposite tastes when compared to the user who has written the review. So, to overcome this problem we have developed a system, which helps the user in choosing the right restaurant according to his/her requirements.

Our work comprises of two recommendation systems in which, one uses the Static data and other uses the real time streaming data. Both of these systems are discussed in the later sections of this paper. To withstand the high competition, restaurants try to improve their quality to satisfy their customers. We processed tips data on daily basis through streaming and recommended top restaurants to the user.

Apache Zookeeper, Apache Kafka servers are used to stream our static data, which are discussed in the later sections of this paper. Inverted index have been created in Elasticsearch and visualized using Kibana. Our systems were experimented on Yelp dataset and the results, future works are also discussed in the later sections. Java, Scala and Python were used for accomplishingthe project.

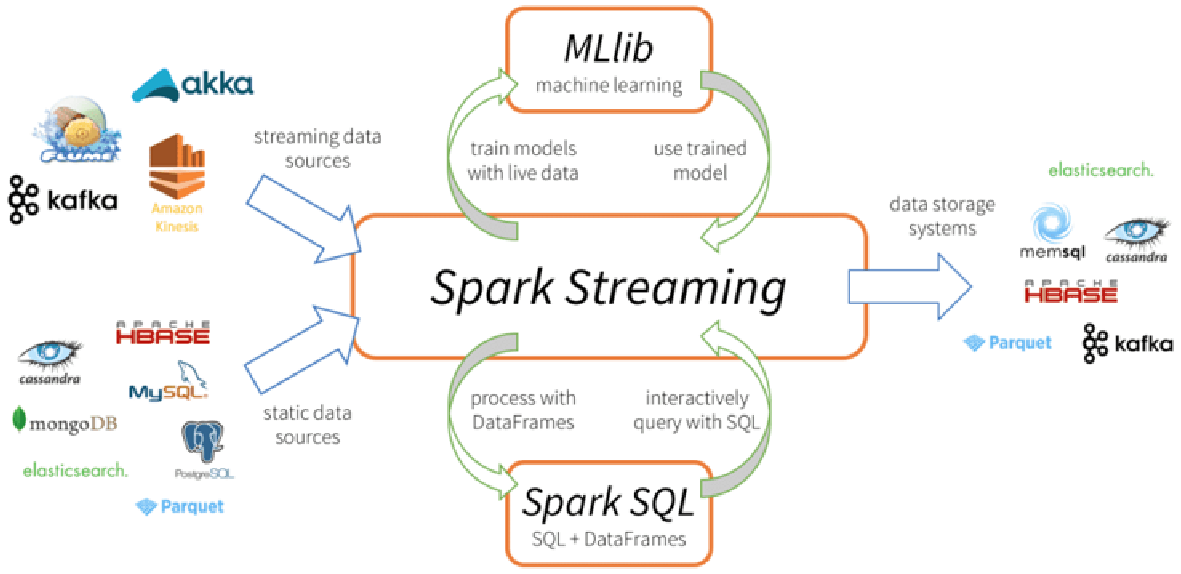
II.II : Apache-Spark:

Apache Spark provides programmers with an application programming interface centeredon a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

The availability of RDDs facilitates the implementation of both iterative algorithms, that visit their dataset multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications (compared to Apache Hadoop, a popular MapReduce implementation) may be reduced by several orders of magnitude.Among the class of iterative algorithms are the training algorithms for machine learning systems,which formed the initial impetus for developing Apache Spark.

II.III : Spark Streaming:

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaminganalytics. It ingests data in mini-batches and performs RDD transformations on those mini- batches of data. This design enables the same set of application code written for batch analyticsto be used in streaming analytics, thus facilitating easy implementation of lambda architecture. However, this convenience comes with the penalty of latency equal to the mini-batch duration. Other streaming data engines that process event by event rather than in mini- batches include Storm and the streaming component of Flink. Spark Streaming has support built-in to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets.



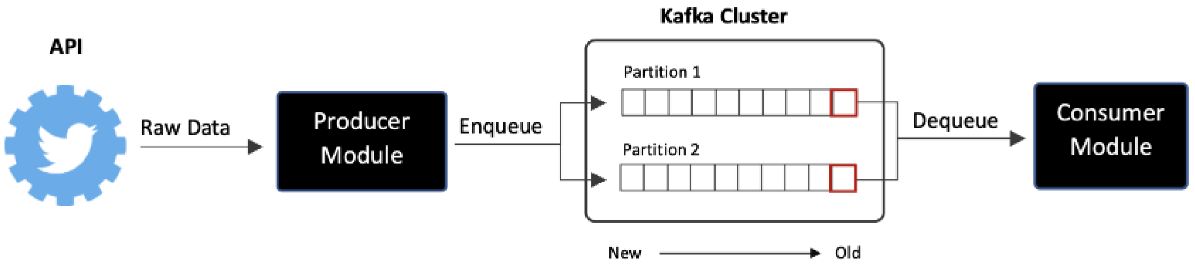
II.IV : Elasticsearch:

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant- capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. Official clients are available in Java, .NET (C#), Python, Groovy and many other languages. Elasticsearch is the most popular enterprise search engine followed by Apache Solr, also based on Lucene.

Elasticsearch is developed alongside a data-collection and log-parsing engine called Logstash, and an analytics and visualization platform called Kibana. The three products are designed for use as an integrated solution, referred to as the "ELK stack".

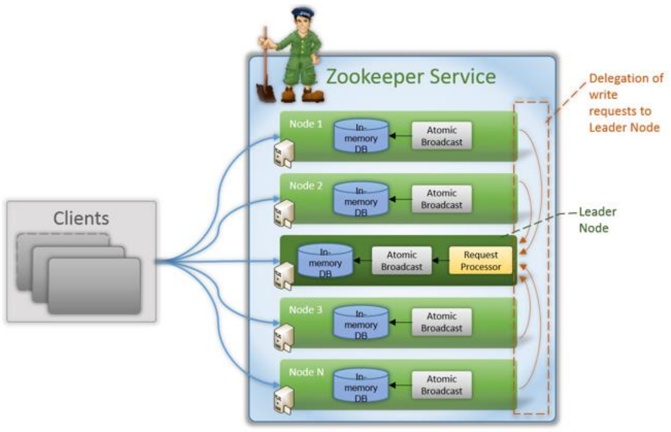
II.V :Apache Kafka:

Apache Kafka is an open-source stream processing platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Its storage layer is essentiallya "massively scalable pub/sub message queue architected as a distributed transaction log," making it highly valuable for enterprise infrastructures to process streaming data. Additionally, Kafka connects to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library.



II.VI : Apache Zookeeper:

Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented thereis a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them,which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.



SECTION-III: DESIGN

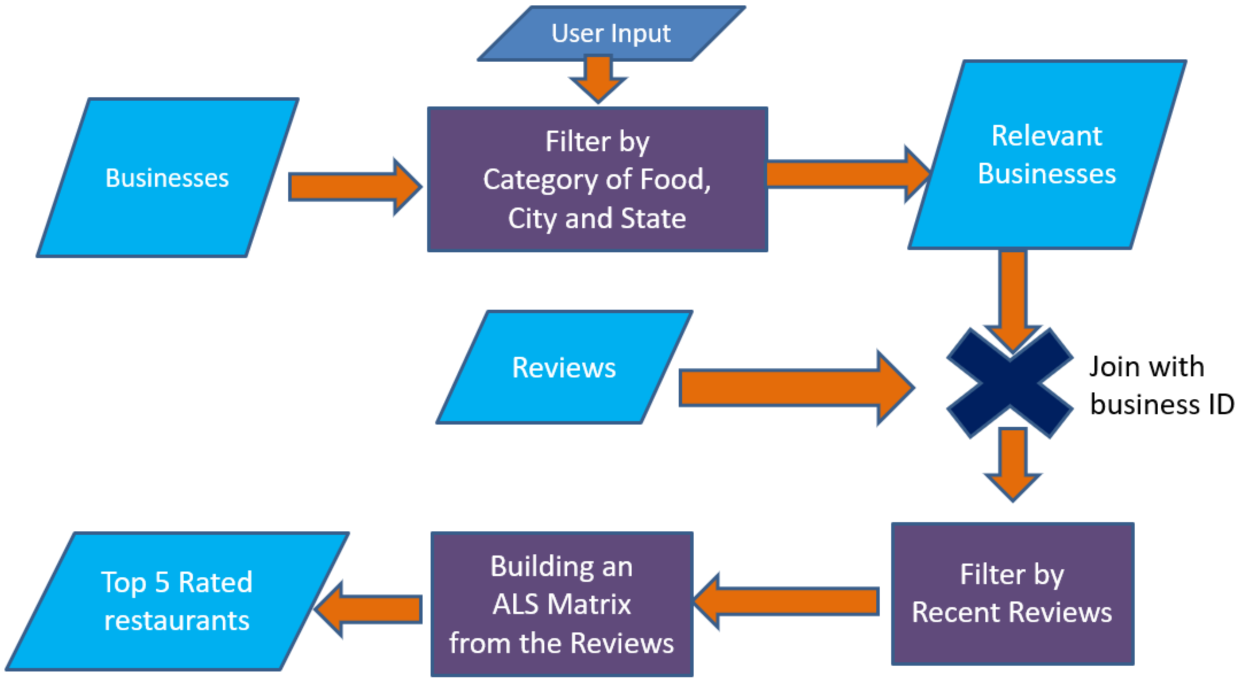
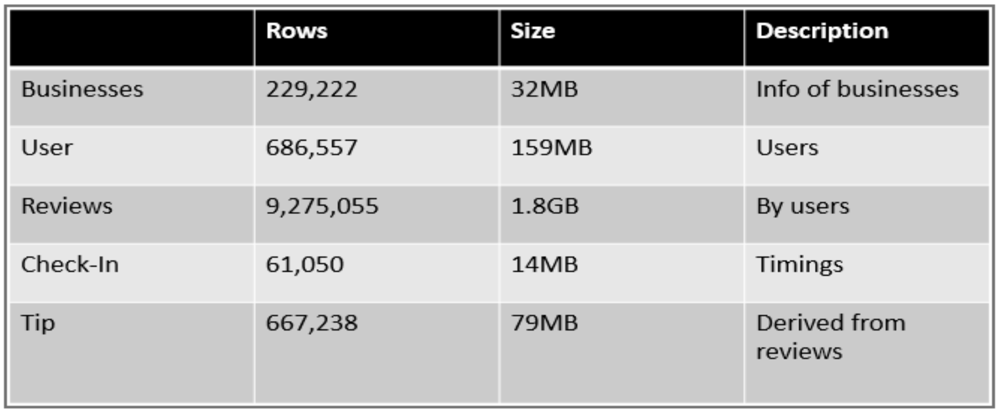


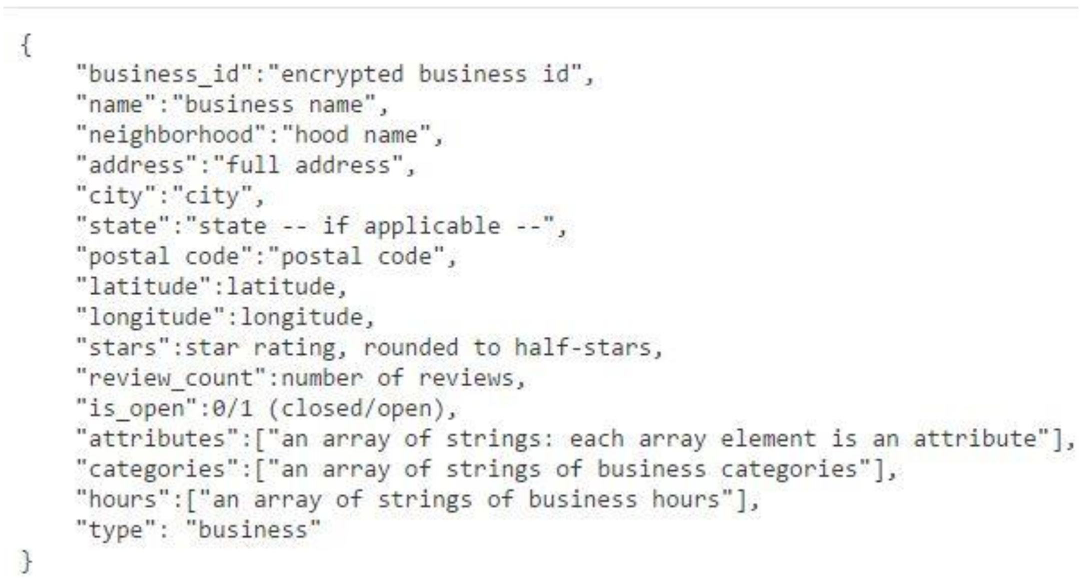
Fig: Design flow of data

The data used in the project is from open-source data sets named “YELP”. Yelp provides several datasets that contain information about businesses, reviews, and user interactions. These datasets are intended for research and educational purposes. The specific details and availability of Yelp datasets may change over time, so it's recommended to check Yelp's official website or contact Yelp directly for the most up-to-date information.



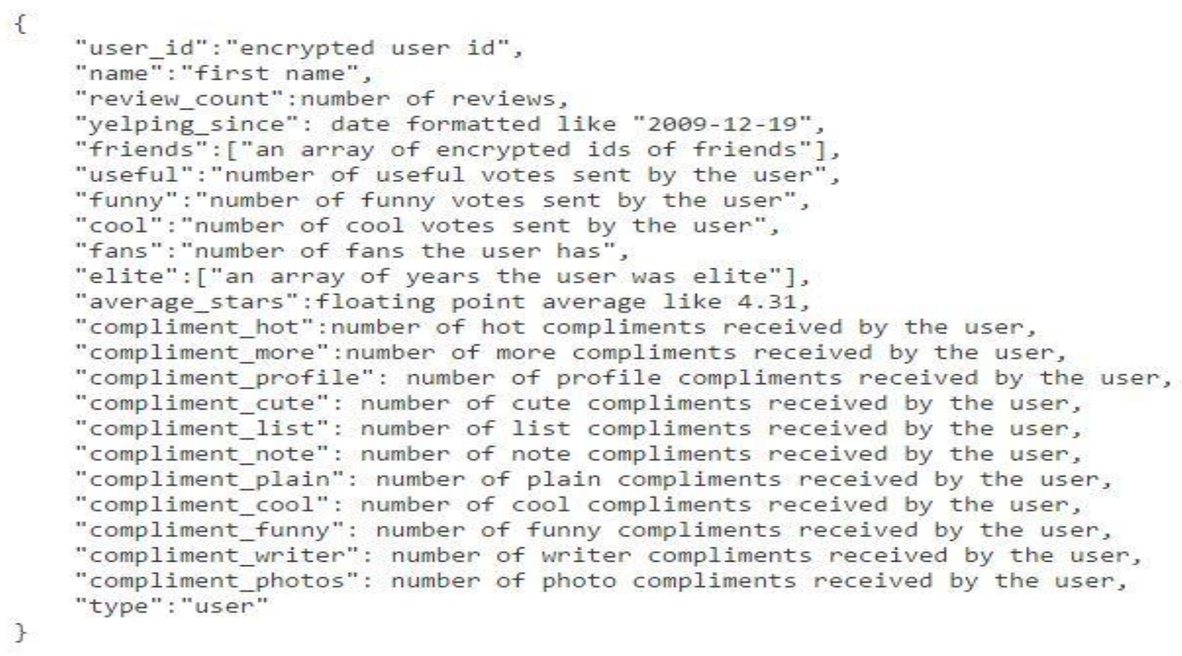
	Rows	Size	Description
Businesses	229,222	32MB	Info of businesses
User	686,557	159MB	Users
Reviews	9,275,055	1.8GB	By users
Check-In	61,050	14MB	Timings
Tip	667,238	79MB	Derived from reviews

Business Data: Details about businesses, including their names, addresses, categories, ratings, and other attributes.



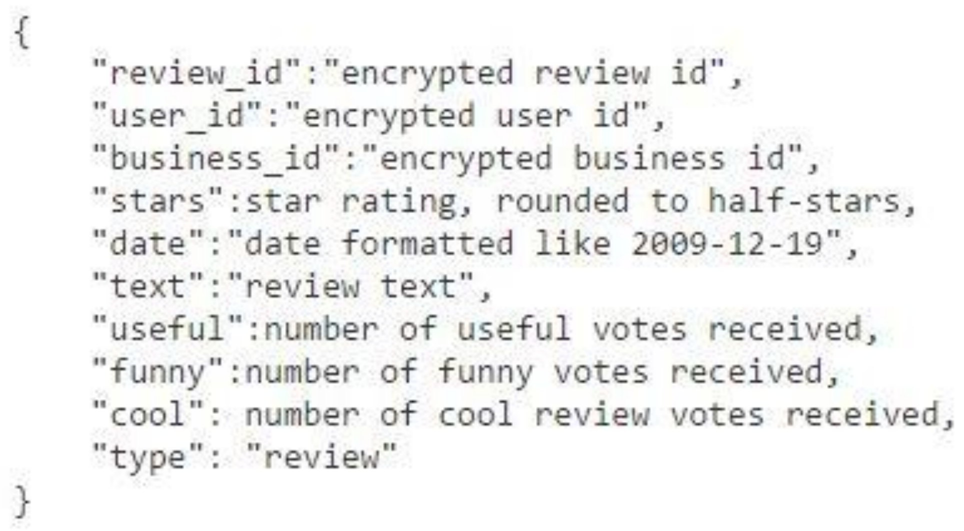
```
{
    "business_id":"encrypted business id",
    "name":"business name",
    "neighborhood":"hood name",
    "address":"full address",
    "city":"city",
    "state":"state -- if applicable --",
    "postal code":"postal code",
    "latitude":latitude,
    "longitude":longitude,
    "stars":star rating, rounded to half-stars,
    "review_count":number of reviews,
    "is_open":0/1 (closed/open),
    "attributes":["an array of strings: each array element is an attribute"],
    "categories":["an array of strings of business categories"],
    "hours":["an array of strings of business hours"],
    "type": "business"
}
```

User Data: Information about users, including user IDs, names, and review counts.



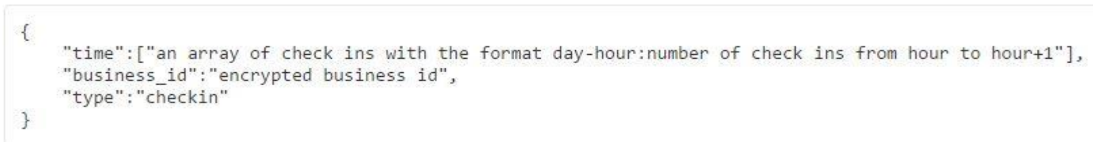
```
{
    "user_id":"encrypted user id",
    "name":"first name",
    "review_count":number of reviews,
    "yelping_since": date formatted like "2009-12-19",
    "friends":["an array of encrypted ids of friends"],
    "useful":"number of useful votes sent by the user",
    "funny":"number of funny votes sent by the user",
    "cool":"number of cool votes sent by the user",
    "fans":"number of fans the user has",
    "elite":["an array of years the user was elite"],
    "average_stars":floating point average like 4.31,
    "compliment_hot":number of hot compliments received by the user,
    "compliment_more":number of more compliments received by the user,
    "compliment_profile": number of profile compliments received by the user,
    "compliment_cute": number of cute compliments received by the user,
    "compliment_list": number of list compliments received by the user,
    "compliment_note": number of note compliments received by the user,
    "compliment_plain": number of plain compliments received by the user,
    "compliment_cool": number of cool compliments received by the user,
    "compliment_funny": number of funny compliments received by the user,
    "compliment_writer": number of writer compliments received by the user,
    "compliment_photos": number of photo compliments received by the user,
    "type":"user"
}
```

Review Data: Reviews written by users, including the text of the reviews, the ratings given, and the date of the review.



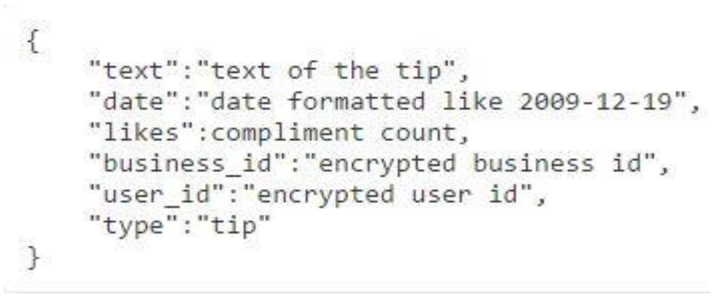
```
{
    "review_id":"encrypted review id",
    "user_id":"encrypted user id",
    "business_id":"encrypted business id",
    "stars":star rating, rounded to half-stars,
    "date":"date formatted like 2009-12-19",
    "text":"review text",
    "useful":number of useful votes received,
    "funny":number of funny votes received,
    "cool": number of cool review votes received,
    "type": "review"
}
```

Check-in Data: Data related to user check-ins at various businesses.



```
{
    "time":["an array of check ins with the format day-hour:number of check ins from hour to hour+1"],
    "business_id":"encrypted business id",
    "type":"checkin"
}
```

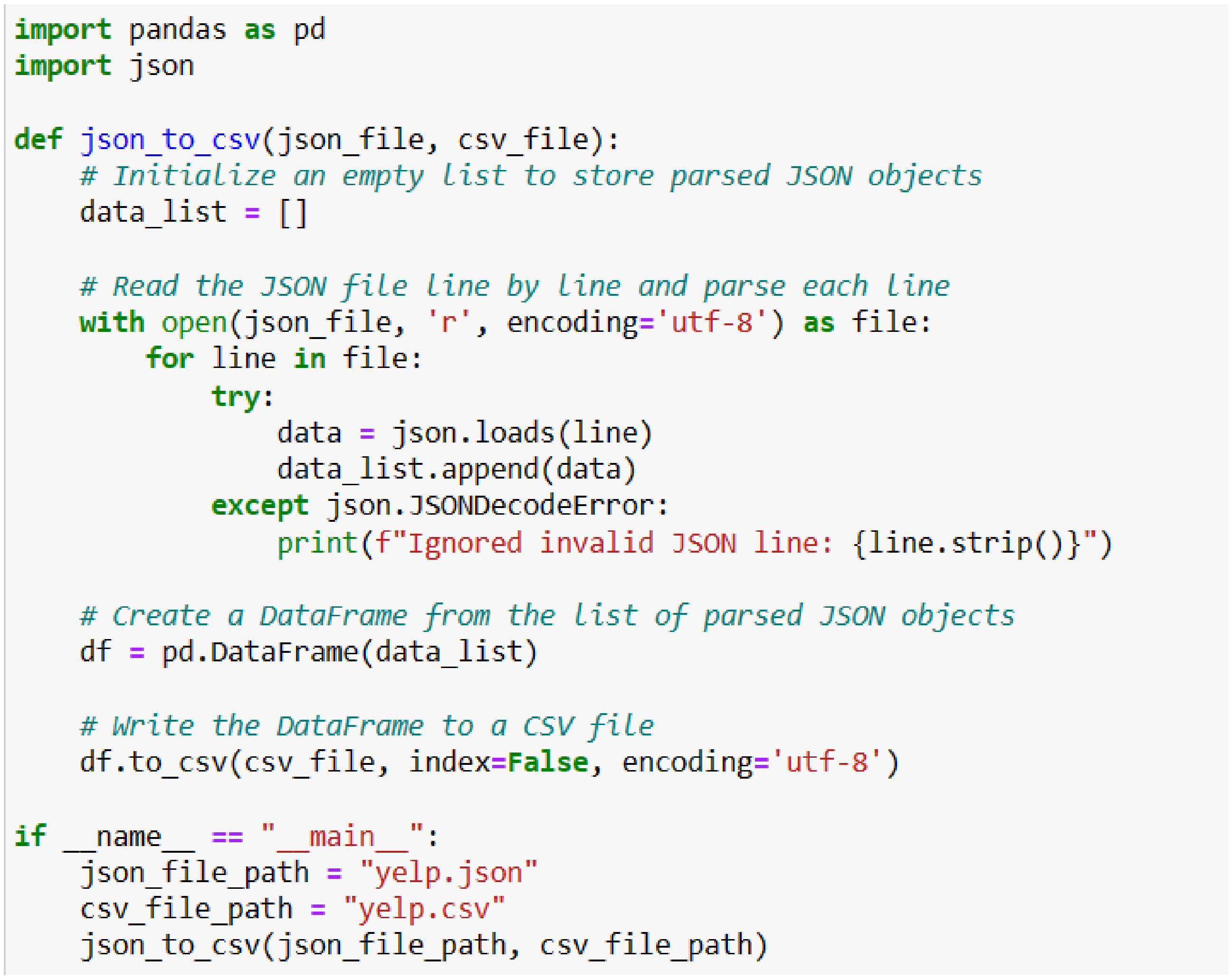
Tip Data: Tips given by users about specific businesses.



```
{
    "text":"text of the tip",
    "date":"date formatted like 2009-12-19",
    "likes":compliment count,
    "business_id":"encrypted business id",
    "user_id":"encrypted user id",
    "type":"tip"
}
```

SECTION-IV: IMPLEMENTATION

This project can be implemented using the combination of datasets available and big data technologies. The dataset is available to download on online. Datasets are available in json format, these are converted into csv format for our project. This can be achieved by running the following python code in Jupyter notebook.



```python
import pandas as pd
import json

def json_to_csv(json_file, csv_file):
    # Initialize an empty list to store parsed JSON objects
    data_list = []

    # Read the JSON file line by line and parse each line
    with open(json_file, 'r', encoding='utf-8') as file:
        for line in file:
            try:
                data = json.loads(line)
                data_list.append(data)
            except json.JSONDecodeError:
                print(f"Ignored invalid JSON line: {line.strip()}")

    # Create a DataFrame from the list of parsed JSON objects
    df = pd.DataFrame(data_list)

    # Write the DataFrame to a CSV file
    df.to_csv(csv_file, index=False, encoding='utf-8')

if __name__ == "__main__":
    json_file_path = "yelp.json"
    csv_file_path = "yelp.csv"
    json_to_csv(json_file_path, csv_file_path)
```

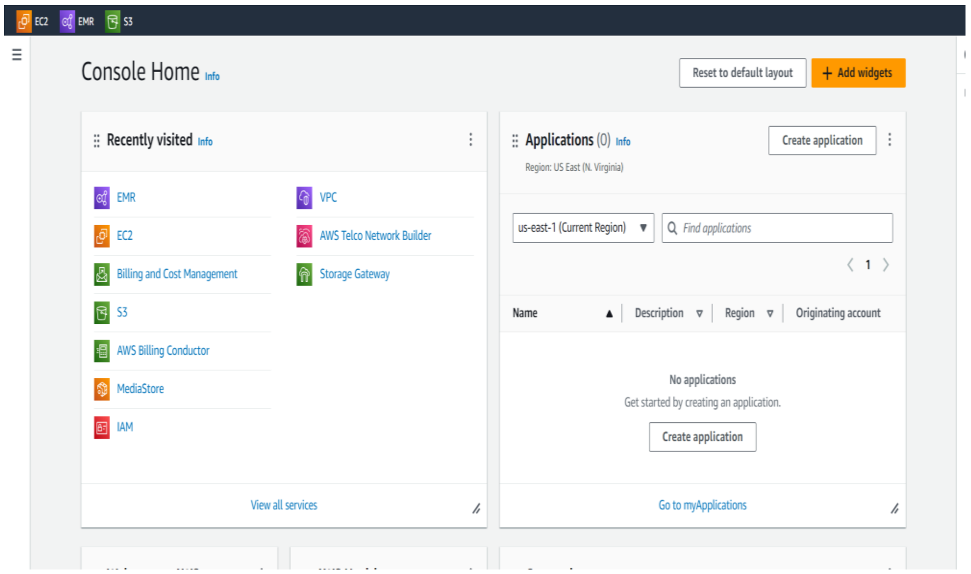
STEPS:

1. Convert the dataset from .json to .csv
2. Performed the sentimental analysis on review dataset.
3. Take the input from the user,

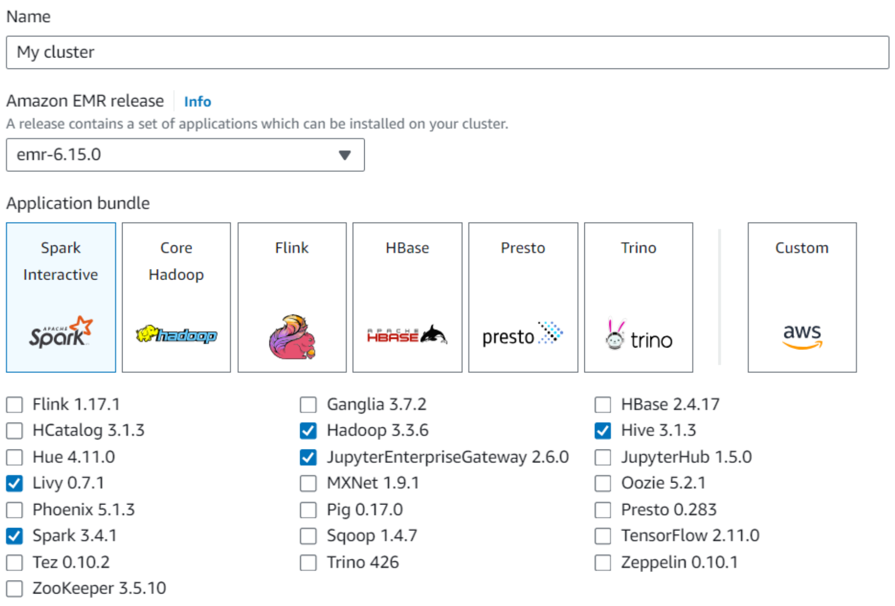
    Input: City, State, Category of food
4. Based on the input, filter the business data file
5. Find out the relevant restaurants based on the input by user, let's call them Target Restaurants
6. Now, get the data from review file which contains the ratings or reviews given to the restaurants.
7. Join Target Restaurants with reviews
8. Build an ALS Matrix comprising of users who have reviewed the target restaurants. This data is used as a training data
9. Test data: current user, list of restaurants
10. Test and predict ratings in the required food category for the current user
11. Return top 5 restaurants based on the average rating

To execute the project, we have to use Spark(Scala):

- Log-in to AWS console.



- Creating a cluster with Spark Interactive Application bundle.



Name

My cluster

Amazon EMR release | Info

A release contains a set of applications which can be installed on your cluster.

emr-6.15.0

Application bundle

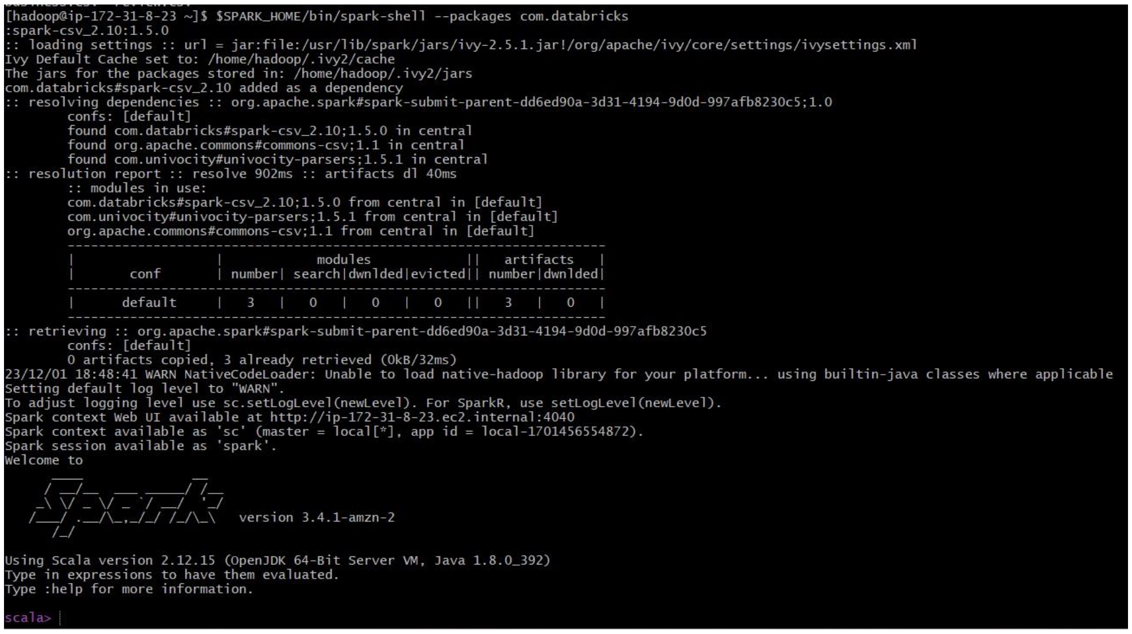
☐ Flink 1.17.1	☐ Ganglia 3.7.2	☐ HBase 2.4.17
☐ HCatalog 3.1.3	☑ Hadoop 3.3.6	☑ Hive 3.1.3
☐ Hue 4.11.0	☑ JupyterEnterpriseGateway 2.6.0	☐ JupyterHub 1.5.0
☑ Livy 0.7.1	☐ MXNet 1.9.1	☐ Oozie 5.2.1
☐ Phoenix 5.1.3	☐ Pig 0.17.0	☐ Presto 0.283
☑ Spark 3.4.1	☐ Sqoop 1.4.7	☐ TensorFlow 2.11.0
☐ Tez 0.10.2	☐ Trino 426	☐ Zeppelin 0.10.1
☐ ZooKeeper 3.5.10		

- After creating a spark cluster establish the connection with the virtual machine. This can be done using Putty or Git Bash, we have used Git Bash for execution.

Establish a connection with the server, once connection is established, in order to run Scala required for the project - run the following line,

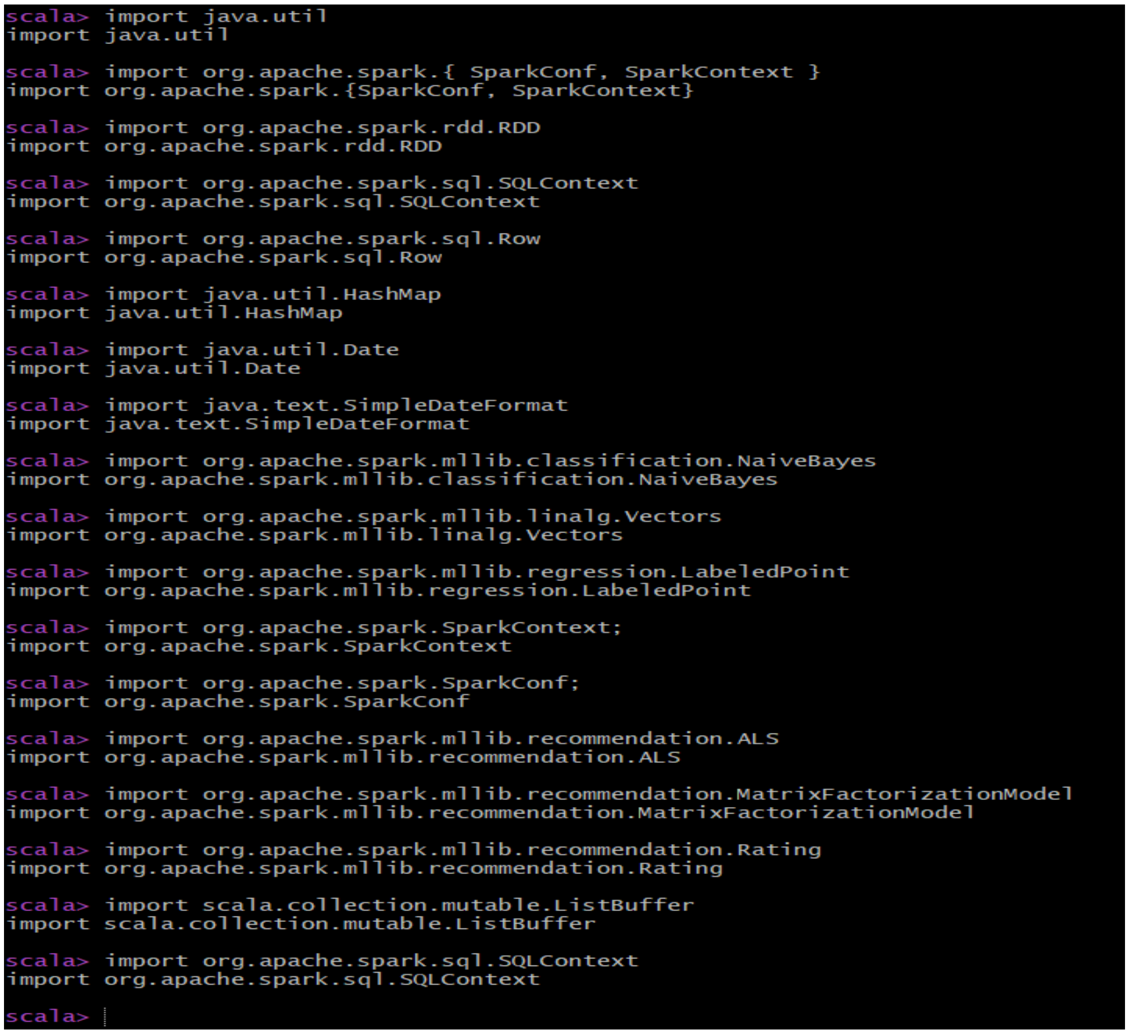
$SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.10:1.5.0

The terminal will now be connected to scala where we can execute our project.



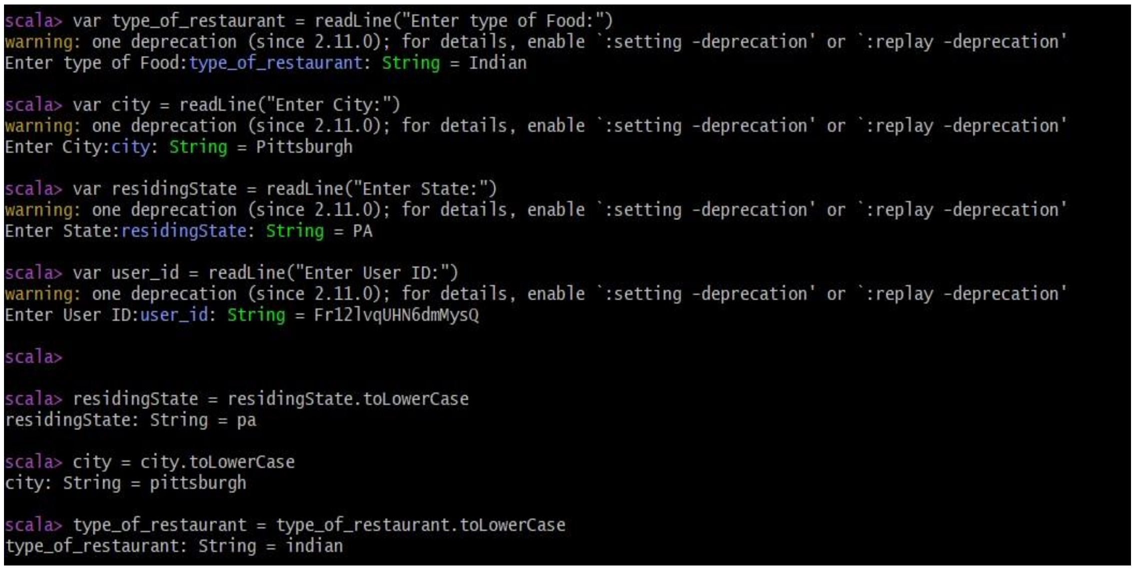
```
[hadoop@ip-172-31-8-23 ~]$ $SPARK_HOME/bin/spark-shell --packages com.databricks
:spark-csv_2.10:1.5.0
:: loading settings :: url = jar:file:/usr/lib/spark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /home/hadoop/.ivy2/cache
The jars for the packages stored in: /home/hadoop/.ivy2/jars
com.databricks#spark-csv_2.10 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-dd6ed90a-3d31-4194-9d0d-997afb8230c5;1.0
        confs: [default]
        found com.databricks#spark-csv_2.10;1.5.0 in central
        found org.apache.commons#commons-csv;1.1 in central
        found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 902ms :: artifacts dl 40ms
        :: modules in use:
        com.databricks#spark-csv_2.10;1.5.0 from central in [default]
        com.univocity#univocity-parsers;1.5.1 from central in [default]
        org.apache.commons#commons-csv;1.1 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   3   |   0   |   0   |   0   ||   3   |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-dd6ed90a-3d31-4194-9d0d-997afb8230c5
        confs: [default]
        0 artifacts copied, 3 already retrieved (0kB/32ms)
23/12/01 18:48:41 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://ip-172-31-8-23.ec2.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1701456554872).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.1-amzn-2
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 1.8.0_392)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

The following import statements are used in our process,



```
scala> import java.util
import java.util

scala> import org.apache.spark.{ SparkConf, SparkContext }
import org.apache.spark.{SparkConf, SparkContext}

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> import java.util.HashMap
import java.util.HashMap

scala> import java.util.Date
import java.util.Date

scala> import java.text.SimpleDateFormat
import java.text.SimpleDateFormat

scala> import org.apache.spark.mllib.classification.NaiveBayes
import org.apache.spark.mllib.classification.NaiveBayes

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.SparkContext;
import org.apache.spark.SparkContext

scala> import org.apache.spark.SparkConf;
import org.apache.spark.SparkConf

scala> import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.ALS

scala> import org.apache.spark.mllib.recommendation.MatrixFactorizationModel
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel

scala> import org.apache.spark.mllib.recommendation.Rating
import org.apache.spark.mllib.recommendation.Rating

scala> import scala.collection.mutable.ListBuffer
import scala.collection.mutable.ListBuffer

scala> import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.SQLContext

scala> 
```

It takes input from the user as follows:



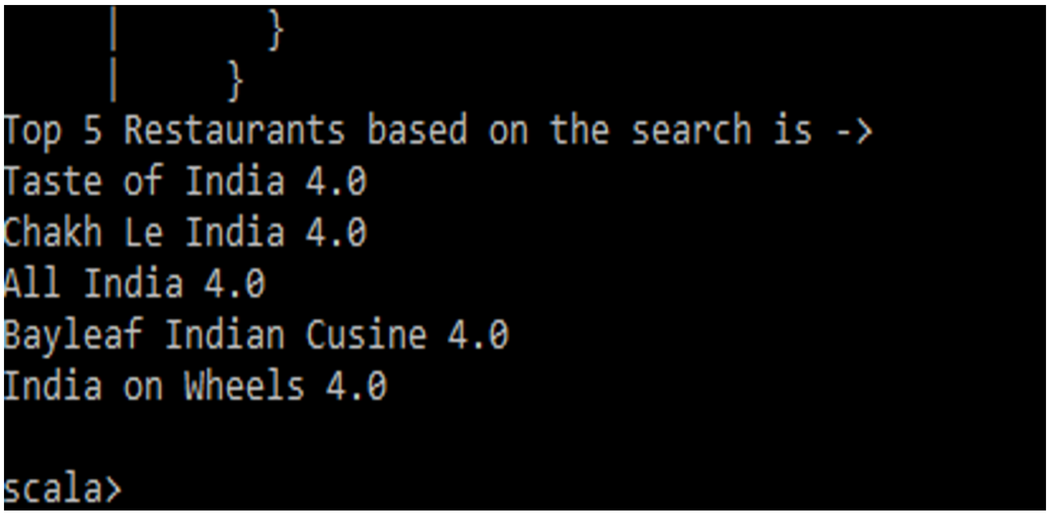
```
scala> var type_of_restaurant = readLine("Enter type of Food:")
warning: one deprecation (since 2.11.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
Enter type of Food:type_of_restaurant: String = Indian

scala> var city = readLine("Enter City:")
warning: one deprecation (since 2.11.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
Enter City:city: String = Pittsburgh

scala> var residingState = readLine("Enter State:")
warning: one deprecation (since 2.11.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
Enter State:residingState: String = PA

scala> var user_id = readLine("Enter User ID:")
warning: one deprecation (since 2.11.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
Enter User ID:user_id: String = Fr12lvqUHN6dmMysQ

scala>

scala> residingState = residingState.toLowerCase
residingState: String = pa

scala> city = city.toLowerCase
city: String = pittsburgh

scala> type_of_restaurant = type_of_restaurant.toLowerCase
type_of_restaurant: String = indian
```

SECTION-V: CONCLUSION

RESULTS:

The project has successfully run and the following is the user input and the generated output for the given input of user choice.

OUTPUT:



```
     |       }
     |     }
Top 5 Restaurants based on the search is ->
Taste of India 4.0
Chakh Le India 4.0
All India 4.0
Bayleaf Indian Cusine 4.0
India on Wheels 4.0

scala>
```

FUTURE SCOPE

The proposed enhancement to the project is done to only the static part due to time constraints, dynamic part can also be enhanced by sentiment analysis and negative part of user reviews for a more GUI friendly way of representation.

SECTION-VI: BIBLIOGRAPHY

[1] https://github.com/patilankita79/Location-based-Restaurants-Recommendation-System/tree/master/BigDataProject

[2] https://developer.confluent.io/what-is-apache-kafka/

[3] https://haocai1992.github.io/data/science/2022/01/13/build-recommendation-system-using-scala-spark-and-hadoop.html

[4] https://docs.aws.amazon.com/lex/latest/dg/sentiment-analysis.html

[5] https://docs.aws.amazon.com/lex/latest/dg/sentiment-analysis.html

[6] https://pypi.org/project/zookeeper/

[7] https://www.yelp.com/dataset