DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi. Approved by AICTE &amp; ISO 9001:2015 Certified)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560111)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SIXTH SEMESTER DATA ANALYTICS LABORATORY MANUAL

SUBJECT CODE: 21CSL66

ACADEMIC YEAR: 2023-24

PREPARED BY,

PROF. AMITH PRADHAAN
ASSISTANT PROFESSOR, DEPT. OF CSE,
DSCE, BENGALURU

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute Affiliated to VTU, Belagavi. Approved by AICTE &amp; ISO 9001:2015 Certified)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560111)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION AND MISSION OF THE DEPARTMENT

VISION

To provide a vibrant learning environment in computer science and engineering with focus on industry needs and research, for the students to be successful global professionals contributing to the society.

MISSION

* To adopt a contemporary teaching learning process with emphasis on hands on and collaborative learning.

* To facilitate skill development through additional training and encourage student forums for enhanced learning.

* To collaborate with industry partners and professional societies and make the students industry ready.

* To encourage innovation through multidisciplinary research and development activities.

* To inculcate human values and ethics to groom the students to be responsible citizens.

CODE OF CONDUCT IN THE LAB

Do's

Students shall

- Come prepared for the program to be developed in the Laboratory.
- Report any broken plugs or exposed electrical wires to your faculty/laboratory technician immediately.
- Turn off the machine once you have finished using it.
- Maintain silence while working in the lab.
- Keep the Computer lab premises clean and tidy.
- Place backpacks under the table or computer counters.
- Treat fellow users of the laboratory, and all equipment within the laboratory, with the appropriate level of care and respect.

Don'ts

Students shall not

- Talk on cell phones in the lab.
- Eat or drink in the laboratory.
- Touch, connect or disconnect any plug or cable without the faculty/laboratory technician's permission.
- Install or download any software or modify or delete any system files on any lab computers.
- Read or modify other users' files.
- Meddle with other user's files.
- Leave their personal belongings unattended.

NOTE: We are not responsible for any theft.

COURSE OBJECTIVES

1. Develop proficiency in data management techniques for storing and sorting data.
2. Create MapReduce programs for diverse data processing tasks.
3. Analyze big data using linear models and spark.
4. Leverage HIVE for advanced data querying and analysis

COURSE OUTCOMES

At the end of the course, student will be able to:

CO1	**Demonstrate file management in Hadoop.**
CO2	**Process big data using MapReduce.**
CO3	**Develop competency in leveraging relational data stores within the Hadoop.**
CO4	**Demonstrate data analysis with spark and utilize MLlib for implementing Regression models.**

LIST OF EXPERIMENTS

Experiment No	Contents of the Experiment PART-A	CO's
1	FILE MANAGEMENT IN HADOOP Implement the following file management tasks in Hadoop, a) Adding Files and directories. b) Retrieving files. c) Deleting files. d) Create a new file in HDFS environment. e) List files in HDFS. f) Upload and download files in HDFS as well other properties copy file, move files and remove file operations in HDFS.	CO1
2	Implement word count / frequency program using MapReduce.	CO2
3	Implement an MR program that processes a weather dataset.	CO2
4	Perform data transfer operations using Sqoop in a Hadoop cluster setup.	CO3
5	Perform targeted data analysis using Apache Hive on the provided dataset, a) To find the movie with the highest average rating. b) Identify the most active users based on the number of ratings submitted. c) Discover movies with the highest number of positive ratings. d) Find the top genres ranked by their average rating.	CO3
6	Utilize Apache Spark to achieve the following tasks for the given dataset, a) Find the movie with the lowest average rating with RDD. b) Identify users who have rated the most movies. c) Explore the distribution of ratings over time. d)Find the highest-rated movies with a minimum number of ratings.	CO4
7	Analyze customer engagement data from an e-commerce company offering both mobile app and website platforms. Using linear regression, determine which platform the company should prioritize for improvement efforts.	CO4

Experiment No	Contents of the Experiment PART-B MINI PROJECT	CO's
1	a) Build a mini project utilizing Apache Spark to analyze a large dataset, focusing on a specific domain. OR b) Design and execute a mini project focused on building a regression model using PySpark.	CO4

1. File Management in Hadoop

Implement the following file management tasks in Hadoop,

a) Adding Files and directories.

b) Retrieving files.

c) Deleting files.

d) Create a new file in HDFS environment.

e) List files in HDFS.

f) Upload and download files in HDFS as well other properties copy file, move files and remove file operations in HDFS.

File management in Hadoop is primarily done through the Hadoop Distributed File System (HDFS), which is designed to store large data sets reliably, and to stream those data sets at high bandwidth to user applications. To manage files in HDFS, you use the Hadoop File System shell (hadoop fs) or the HDFS shell (hdfs dfs), which provides a variety of commands for managing files and directories.

Below are the steps and commands for performing the listed file management tasks in Hadoop:

a) Adding Files and directories.

- **Creating a Directory**: To create a directory in HDFS, use the **mkdir** command.

```
hdfs dfs -mkdir /user/hadoop/mydirectory
```

- **Adding Files to HDFS**: To add a file from your local file system to HDFS, use the **put** command.

```
hdfs dfs -put localfile.txt /user/hadoop/mydirectory
```

- **Upload a directory to HDFS**:

```
hdfs dfs -put localdirectory /path/in/hdfs
```

b) Retrieving Files.

- To retrieve files from HDFS to your local file system, use the **get** command:

```
hdfs dfs -get /path/in/hdfs/localfile.txt /path/in/local
```

c) Deleting files.

- **Deleting a File**: To delete a file from HDFS, use the **rm** command.

```
hdfs dfs -rm /user/hadoop/mydirectory/localfile.txt
```

- **Deleting a Directory**: To delete a directory and its contents, use the **rm -r** command.

```
hdfs dfs -rm -r /user/hadoop/mydirectory
```

d) Create a new file in HDFS environment.

- Creating a new empty file in HDFS is not directly supported by a command. However, you can create an empty file locally and then upload it to HDFS.

```
touch local_empty_file.txt
hdfs dfs -put local_empty_file.txt /path/in/hdfs/emptyfile.txt
```

e) List files in HDFS.

- To list files and directories in HDFS, use the **ls** command:

```
hdfs dfs -ls /path/in/hdfs
```

f) Upload and download files in HDFS as well other properties copy file, move files and remove file operations in HDFS.

- **Upload Files**: hdfs dfs -put localfile.txt /path/in/hdfs
- **Download Files**: hdfs dfs -get /path/in/hdfs/localfile.txt /path/in/local
- **Copy Files**: hdfs dfs -cp /path/in/hdfs/sourcefile.txt /path/in/hdfs/destination/
- **Move Files**: hdfs dfs -mv /path/in/hdfs/sourcefile.txt /path/in/hdfs/destination/
- **Remove Files**: hdfs dfs -rm /path/in/hdfs/localfile.txt

2. Implement Word Count / Frequency Program Using MapReduce.

Steps to be followed:

- Step-1: Open **Eclipse** → then select **File** → **New** → **Java Project** → Name it **WordCount** → then **Finish**.
- Step-2: Create Three Java Classes into the project.

  File → **New** → **Class**

  Name them **WCDriver** (having the main function), **WCMapper** and **WCReducer**.
- Step-3: You have to include two Reference Libraries, Right Click on **Project** → then select **Build Path** → Click on **Configure Build Path** → **Add External JARs** (Share → Hadoop). In this add JARs of Client, Common, HDFS, MapReduce and YARN → Click on **Apply and Close**.
- Step-4: **Mapper Code** which should be copied and pasted into the **WCMapper** Java Class file.

```
// Importing libraries

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;
```

```
public class WCMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>
{
  // Map function
 public void map(LongWritable key, Text value, OutputCollector<Text,
 IntWritable> output, Reporter rep) throws IOException
  {
     String line = value.toString();
     // Splitting the line on spaces
     for (String word : line.split(" "))
     {
        if (word.length() > 0)
        {
           output.collect(new Text(word), new IntWritable(1));
        }
     }
  }
}
```

- Step-5: **Reducer Code** which should be copied and pasted into the **WCReducer** Java Class file.

```
// Importing libraries
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable>
{
// Reduce function
public void reduce(Text key, Iterator<IntWritable> value,  OutputCollector<Text,
IntWritable> output, Reporter rep) throws IOException
{
        int count = 0;
        // Counting the frequency of each words
        while (value.hasNext())
        {
                IntWritable i = value.next();
                count += i.get();
        }
        output.collect(key, new IntWritable(count));
}
}
```

- Step-6: **Driver Code** which should be copied and pasted into the **WCDriver** Java Class file.

```
// Importing libraries
import java.io.IOException;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool
{
public int run(String args[]) throws IOException
{
        if (args.length < 2)
        {
                System.out.println("Please give valid inputs");
                return -1;
        }

        JobConf conf = new JobConf(WCDriver.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        conf.setMapperClass(WCMapper.class);
        conf.setReducerClass(WCReducer.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        JobClient.runJob(conf);
        return 0;
}
// Main Method
public static void main(String args[]) throws Exception
{
        int exitCode = ToolRunner.run(new WCDriver(), args);
        System.out.println(exitCode);
}
}
```

- Step-7: Now you have to make a jar file.
  Right Click on **Project → Click on Export → Select export destination as Jar File → Name the jar File** (WordCount.jar) → **Click on next** → at last **Click on Finish**.

- Step-8: Open the terminal and change the directory to the workspace.
  You can do this by using "cd workspace/" command.
  Now, Create a text file (**WCFile.txt**) and move it to HDFS.
  For that open terminal and write the below code (remember you should be in the same directory as jar file you have created just now),

```
cat WCFile.text
```

- Step-9: Now, run the below command to copy the file input file into the HDFS,

```
hadoop fs -put WCFile.txt WCFile.txt
```

- Step-10: Now to run the jar file, execute the below code,

```
hadoop jar wordcount.jar WCDriver WCFile.txt WCOutput
```

- Step-11: After Executing the code, you can see the result in WCOutput file or by writing following command on terminal,

```
hadoop fs -cat WCOutput/part-00000
```

3. Implement An Mr Program That Processes A Weather Dataset.

Steps to be followed:

- Step-1: We can download the dataset from this Link, For various cities in different years. choose the year of your choice and select any one of the data text-file for analysing.

  We can get information about data from README.txt file available on the NCEI website.

- Step-2: Make a project in Eclipse with below steps:
  - First Open **Eclipse** → then select **File** → **New** → **Java Project** → Name it **MyProject** → then select **use an execution environment** → choose **JavaSE-1.8** then **next** → **Finish**.
  - In this Project Create Java class with name **MyMaxMin** → then click **Finish**.
  - Copy the below source code to this **MyMaxMin** java class.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class maxtemperature {
        public static void main(String[] args) throws Exception {
                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "maxtemperature");
                job.setJarByClass(maxtemperature.class);
                // TODO: specify a mapper
                job.setMapperClass(MaxTempMapper.class);
                // TODO: specify a reducer
                job.setReducerClass(MaxTempReducer.class);
                // TODO: specify output types
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
                // TODO: specify input and output DIRECTORIES (not files)
                FileInputFormat.setInputPaths(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                if (!job.waitForCompletion(true))
                        return;
        }
}
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Mapper;
public class MaxTempMapper extends Mapper<LongWritable, Text, Text,
IntWritable > {

public void map(LongWritable key, Text value, Context context)
```

```
throws IOException, InterruptedException {
 String line=value.toString();
 String year=line.substring(15,19);
 int airtemp;
 if(line.charAt(87)== '+')
 {
 airtemp=Integer.parseInt(line.substring(88,92));
 }
 else
        airtemp=Integer.parseInt(line.substring(87,92));
        String q=line.substring(92,93);
        if(airtemp!=9999&&q.matches("[01459]"))
        {
                context.write(new Text(year),new IntWritable(airtemp));
        }
 }
 }

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;
public class MaxTempReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException
{
        int maxvalue=Integer.MIN_VALUE;
        for (IntWritable value : values)
{
maxvalue=Math.max(maxvalue, value.get());
}
        context.write(key, new IntWritable(maxvalue));
}
}
```

- Now we need to add external jar for the packages that we have import. Download the jar package Hadoop Common and Hadoop MapReduce Core according to the Hadoop version.
- Now we add these external jars to our **MyProject**.

  Right Click on **MyProject** → then select **Build Path** → Click on **Configure Build Path** and select **Add External jars….** and add jars from its download location then click → **Apply and Close**.
- Now export the project as jar file.

  Right-click on **MyProject** choose **Export..** and go to **Java** → **JAR file** click → **Next** and choose your export destination then click → **Next**. choose Main Class as **MyMaxMin** by clicking → **Browse** and then click→**Finish** →**Ok**.

- Step-4: Start the Hadoop daemons.

  start-dfs.sh

  start-yarn.sh

- Step-5: Move the dataset to Hadoop HDFS.
  hdfs dfs -put /file_path /destination
  In below command / shows the root directory of our HDFS,
  hdfs dfs -put /home/…./……./datasetname.txt /
  hdfs dfs -ls /

- Step-6: Now Run your Jar File with below command and produce the output in MyOutput File.
  hadoop jar /jar_file_location /dataset_location_in_HDFS /output-file_name
  hadoop jar /…./…./…./Project.jar /datasetname.txt /MyOutput
- Step-7: Now Move to localhost:50070/, under utilities select Browse the file system and download **part-r-00000** in **/MyOutput** directory to see result.
- Step-8: See the result in downloaded file.

4. Perform data transfer operations using Sqoop in a Hadoop cluster setup.

Prerequisites:

a) **Hadoop Cluster**: Ensure your Hadoop cluster is up and running.

b) **Sqoop Installation**: Sqoop should be installed and configured in your Hadoop cluster.

c) **Database Access**: Access to the source database (e.g., MySQL, PostgreSQL) with the required JDBC drivers installed on the Sqoop nodes.

d) **JDBC Driver**: Ensure the JDBC driver for your database is available in the Sqoop lib directory.

- Step-1: Connect to the MySQL Database

```
cp mysql-connector-java-*.jar $SQOOP_HOME/lib/
```

- Step-2: Create the Sqoop Import and Export Command

```
sqoop import \
  --connect
jdbc:mysql://<MYSQL_HOST>:<MYSQL_PORT>/<DATABASE_NAME> \
  --username <USERNAME> \
  --password <PASSWORD> \
  --table <TABLE_NAME> \
  --target-dir <HDFS_TARGET_DIR> \
  --num-mappers 1
```

```
sqoop export \
  --connect jdbc:mysql://<MYSQL_HOST>:<MYSQL_PORT>/<DATABASE_NAME> \
  --username <USERNAME> \
  --password <PASSWORD> \
  --table <TABLE_NAME> \
  --export-dir <HDFS_EXPORT_DIR> \
  --input-fields-terminated-by ',' \
  --num-mappers 1
```

- Step-3: Execute the Sqoop Command

  Run the Sqoop import command in your Hadoop cluster terminal.

- Step-4: Verify the Imported and Exported Data

  ```
  hadoop fs -ls <HDFS_TARGET_DIR>
  SELECT * FROM <TABLE_NAME>;
  ```

5. Perform targeted data analysis using Apache Hive on the provided dataset,

a) To find the movie with the highest average rating.

b) Identify the most active users based on the number of ratings submitted.

c) Discover movies with the highest number of positive ratings.

d) Find the top genres ranked by their average rating.

Steps to be followed:

Prerequisites:

1. **Apache Hive Setup**: Ensure you have Apache Hive installed and configured.
2. **Dataset**: Load the provided dataset into Hive tables. Assume we have two tables: **movies** and **ratings**.
   - **movies** table has columns: **movie_id**, **title**, **genres**.
   - **ratings** table has columns: **user_id**, **movie_id**, **rating**.

How to transfer dataset from windows to VM?

Step-1: Go to VM → Network → Adapter 1 → Attached to: Select Bridge adapter → Name: Intel® Ethernet Connection → Advanced → Promiscuous mode: Allow all → OK

Step-2: Run Hadoop

Step-3: At the top right an option is available with double headed arrow marks → Enable connection → Connect the information (IP address will be found here).

Step-4: Go to terminal in Hadoop → mkdir windows → cd windows → ls

Step-5: Go to windows and check if the dataset is available in either **documents** or **pictures**, if not put it in the said folders.

Step-6: Go to CMD → cd pictures/documents → scp movies.csv hadoop@172.25.4.67:/home/hadoop/windows

It will ask for password: **hadoop**

Step-7: Go to home/hadoop/windows/movies.csv and check for the dataset which will be moved from windows to Hadoop.

Step-8: Go to terminal in VM and execute the following commands,

```
> startCDH

> ls

>cp /home/hadoop/windows/movies.csv   /home/hadoop/

>ls

>hadoop fs -put movies.csv movies1.csv (Give any name of your choice)

>hadoop fs -ls

>hadoop fs -ls /user

>hadoop fs -ls /user/hive (Check for warehouse)

>hive

hive> create table if not exists movies(
 id int,
 name string,
 genre string)
  row format delimited
 fields terminated by ',' ;

hive> select * from movies;

hive> load data local inpath 'movies.csv' into table  movies;

Copying data from file:/home/hadoop/movie.csv

Copying file: file:/home/hadoop/movie.csv

Loading data to table default.movies
```

```
hive> select * from movies;

hive> select id from movies;

hive> select name from movies;

hive> show tables;

hive> drop table movies;
```

a. To find the movie with the highest average rating.

```
select movie_id, avg(rating) as avg_rating
from ratings
group by movie_id
order by avg_rating desc
limit 1;
```

b. Identify the most active users based on the number of ratings submitted.

```
select user_id, count(*) as num_ratings
from ratings
group by user_id
order by num_ratings desc
limit 10; -- you can adjust the limit as needed
```

c. Discover movies with the highest number of positive ratings.

assuming a positive rating is 4 or 5:

```
select movie_id, count(*) as positive_ratings
from ratings
where rating >= 4
group by movie_id
order by positive_ratings desc
limit 10; -- you can adjust the limit as needed
```

d. Find the top genres ranked by their average rating.

this step requires joining the **movies** and **ratings** tables,

```
-- explode genres and calculate average ratings for each genre
with exploded_genres as (
  select movie_id, explode(split(genres, '[|]')) as genre
  from movies
),
genre_ratings as (
  select g.genre, avg(r.rating) as avg_rating
  from exploded_genres g
  join ratings r on g.movie_id = r.movie_id
  group by g.genre
)
select genre, avg_rating
from genre_ratings
order by avg_rating desc;
```

6. Utilize Apache Spark to achieve the following tasks for the given dataset,

a) Find the movie with the lowest average rating with RDD.

b) Identify users who have rated the most movies.

c) Explore the distribution of ratings over time.

d)Find the highest-rated movies with a minimum number of ratings.

Steps to be followed:

Prerequisites:

- **Apache Spark Setup**: Ensure you have Apache Spark installed and configured. (If not go to terminal and install pyspark using pip install pyspark)
- **Dataset**: Load the dataset.
    - **movies** dataset has columns: **movie_id**, **title**, **genres**.
    - **ratings** dataset has columns: **user_id**, **movie_id**, **rating**.
- **Loading Data**: First, load the data into Spark RDDs or DataFrames.

```
from pyspark.sql import SparkSession

# Create RDDs

#movies_rdd = movies_df.rdd

#ratings_rdd = ratings_df.rdd

# Initialize Spark session

spark = SparkSession.builder.appName("MovieRatingsAnalysis").getOrCreate()

# Load datasets

movies_df = spark.read.csv("/Users/amithpradhaan/Desktop/ml-latest-small/movies.csv",
header=True, inferSchema=True)

ratings_df = spark.read.csv("/Users/amithpradhaan/Desktop/ml-latest-small/ratings.csv",
header=True, inferSchema=True)

# Create RDDs

movies_rdd = movies_df.rdd

ratings_rdd = ratings_df.rdd
```

a) Find the Movie with the Lowest Average Rating Using RDD.

```
# Compute average ratings

avg_ratings_rdd = ratings_rdd.map(lambda x: (x['movieId'], (x['rating'], 1))) \

    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1])) \

    .mapValues(lambda x: x[0] / x[1])

# Find the movie with the lowest average rating

lowest_avg_rating = avg_ratings_rdd.sortBy(lambda x: x[1]).first()

print(f"Movie with the lowest average rating: {lowest_avg_rating}")
```

b) Identify Users Who Have Rated the Most Movies.

```
# Compute number of ratings per user

user_ratings_count = ratings_rdd.map(lambda x: (x['userId'], 1)) \

    .reduceByKey(lambda x, y: x + y) \

    .sortBy(lambda x: x[1], ascending=False)

# Get top users

top_users = user_ratings_count.take(10)

print(f"Top users by number of ratings: {top_users}")
```

c) Explore the Distribution of Ratings Over Time.

```
from pyspark.sql.functions import from_unixtime, year, month

# Convert timestamp to date and extract year and month

ratings_df = ratings_df.withColumn("year", year(from_unixtime(ratings_df['timestamp']))) \

                    .withColumn("month", month(from_unixtime(ratings_df['timestamp'])))

# Group by year and month to get rating counts

ratings_over_time = ratings_df.groupBy("year", "month").count().orderBy("year", "month")
```

```
# Show distribution

ratings_over_time.show()
```

d) Find the Highest-Rated Movies with a Minimum Number of Ratings.

Set a minimum number of ratings, ex: 100.

```
# Compute average ratings and count ratings per movie

movie_ratings_stats = ratings_rdd.map(lambda x: (x['movieId'], (x['rating'], 1))) \

    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1])) \

    .mapValues(lambda x: (x[0] / x[1], x[1]))  # (avg_rating, count)

# Filter movies with a minimum number of ratings

min_ratings = 100

qualified_movies = movie_ratings_stats.filter(lambda x: x[1][1] >= min_ratings)

# Find the highest-rated movies

highest_rated_movies = qualified_movies.sortBy(lambda x: x[1][0], ascending=False).take(10)

print(f"Highest-rated movies with at least {min_ratings} ratings: {highest_rated_movies}")
```

<u>EXPECTED OUTPUTS:</u>

a) **Lowest Average Rating Movie**:

- Movie with the lowest average rating: (movieId, averageRating)

b) **Top Users by Number of Ratings**:

- Top users by number of ratings: [(userId1, count1), (userId2, count2), ...]

c) **Distribution of Ratings Over Time**:

```
+----+-----+-----+
|year|month|count|
+----+-----+-----+
|1996|    3|   58|
|1996|    4|  165|
|1996|    5|  832|
|1996|    6|  882|
|1996|    7|  490|
|1996|    8| 1010|
|1996|    9|  384|
|1996|   10|  935|
|1996|   11|  978|
|1996|   12|  306|
|1997|    1|  250|
|1997|    2|  323|
|1997|    3|  398|
|1997|    4|  219|
|1997|    5|  303|
|1997|    6|   84|
|1997|    7|   70|
|1997|    9|  236|
|1997|   10|    1|
|1997|   11|    2|
+----+-----+-----+
```

d) Highest-Rated Movies with Minimum Number of Ratings:

- Highest-rated movies with at least 100 ratings: [(movieId1, (averageRating1, count1)), (movieId2, (averageRating2, count2)), ...]

7. Analyze customer engagement data from an e-commerce company offering both mobile app and website platforms. Using linear regression, determine which platform the company should prioritize for improvement efforts.

Prerequisites:

- **Apache Spark Setup**: Ensure you have Apache Spark installed and configured. (If not go to terminal and install pyspark using pip install pyspark)
- **Dataset**: The dataset that we make use of is cruise_ship_info
  The dataset contains 159 instances with 9 features.
  The Description of dataset is as below:
  - Ship Name
  - Cruise Line
  - Age (as of 2013)
  - Tonnage (1000s of tons)
  - passengers (100s)
  - Length (100s of feet)
  - Cabins (100s)
  - Passenger Density
  - Crew (100s)

Program:

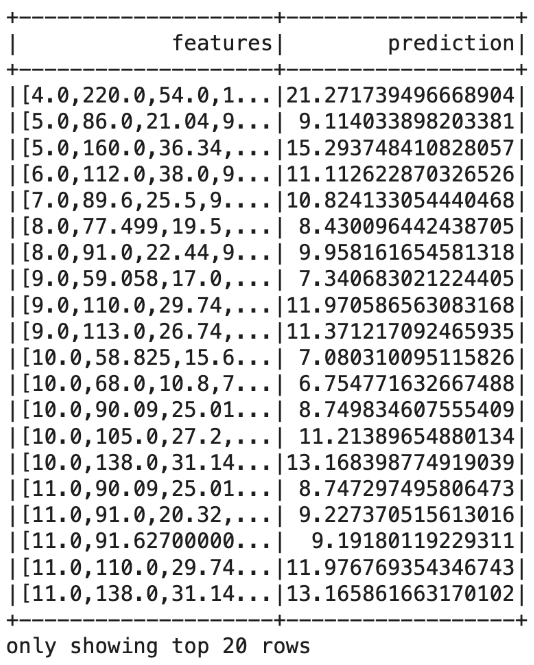
```
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('housing_price_model').getOrCreate()
#create spark dataframe of input csv file
df=spark.read.csv('/content/drive/MyDrive/cruise_ship_info.csv',inferSchema=True,header=True)
df.show(10)
#prints structure of dataframe along with datatype
df.printSchema()
#In our predictive model, below are the columns
df.columns
from pyspark.ml.feature import StringIndexer
indexer=StringIndexer(inputCol='Cruise_line',outputCol='cruise_cat')
indexed=indexer.fit(df).transform(df)
#above code will convert string to numeric feature and create a new dataframe
#new dataframe contains a new feature 'cruise_cat' and can be used further
#feature cruise_cat is now vectorized and can be used to fed to model
for item in indexed.head(5):
```

```
print(item)
print('\n')

 from pyspark.ml.linalg import Vectors
 from pyspark.ml.feature import VectorAssembler
 #creating vectors from features
 #Apache MLlib takes input if vector form
 assembler=VectorAssembler(inputCols=['Age',
  'Tonnage',
  'passengers',
  'length',
  'cabins',
  'passenger_density',
  'cruise_cat'],outputCol='features')
 output=assembler.transform(indexed)
 output.select('features','crew').show(5)
 #final data consist of features and label which is crew.
 final_data=output.select('features','crew')
 #splitting data into train and test
 train_data,test_data=final_data.randomSplit([0.7,0.3])
 train_data.describe().show()
 test_data.describe().show()
 #import LinearRegression library
 from pyspark.ml.regression import LinearRegression
 #creating an object of class LinearRegression
 #object takes features and label as input arguments
 ship_lr=LinearRegression(featuresCol='features',labelCol='crew')
 #pass train_data to train model
 trained_ship_model=ship_lr.fit(train_data)
 #evaluating model trained for Rsquared error
 ship_results=trained_ship_model.evaluate(train_data)
 print('Rsquared Error :',ship_results.r2)
 #testing Model on unlabeled data
 #create unlabeled data from test_data
 unlabeled_data=test_data.select('features')
```

```
unlabeled_data.show(5)
predictions=trained_ship_model.transform(unlabeled_data)
predictions.show()
```

EXPECTED OUTPUT:



```
+--------------------+------------------+
|            features|        prediction|
+--------------------+------------------+
|[4.0,220.0,54.0,1...|21.271739496668904|
|[5.0,86.0,21.04,9...| 9.114033898203381|
|[5.0,160.0,36.34,...|15.293748410828057|
|[6.0,112.0,38.0,9...|11.112622870326526|
|[7.0,89.6,25.5,9....|10.824133054440468|
|[8.0,77.499,19.5,...| 8.430096442438705|
|[8.0,91.0,22.44,9...| 9.958161654581318|
|[9.0,59.058,17.0,...| 7.340683021224405|
|[9.0,110.0,29.74,...|11.970586563083168|
|[9.0,113.0,26.74,...|11.371217092465935|
|[10.0,58.825,15.6...| 7.080310095115826|
|[10.0,68.0,10.8,7...| 6.754771632667488|
|[10.0,90.09,25.01...| 8.749834607555409|
|[10.0,105.0,27.2,...| 11.21389654880134|
|[10.0,138.0,31.14...|13.168398774919039|
|[11.0,90.09,25.01...| 8.747297495806473|
|[11.0,91.0,20.32,...| 9.227370515613016|
|[11.0,91.62700000...|  9.19180119229311|
|[11.0,110.0,29.74...|11.976769354346743|
|[11.0,138.0,31.14...|13.165861663170102|
+--------------------+------------------+
only showing top 20 rows
```