TASK 2

In this project, the aim is to explore and analyse a dataset extracted from Steam, named steam-200k.csv, which contains details of games purchased and played by users along with the corresponding behavior and playtime. The dataset consists of four columns:

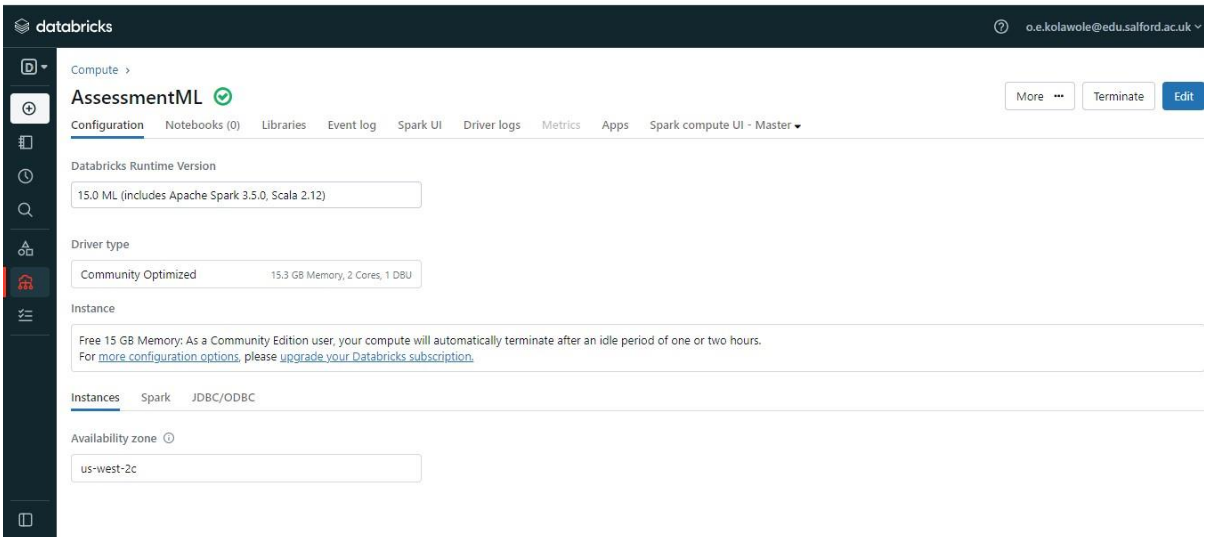
- A unique identifier for each member
- The name of the game they purchased or played
- Details of the member behaviour, either ‘purchase’ or ‘play’. Because a game has to be purchased before it can be played, hence, there will be two entries for the same game/member combination in some instances
- This column is set to 1 for rows where the behavior is ‘purchase’. For rows where the behavior is ‘play’ the value in the fourth column corresponds to the number of hours of play

The objective is to utilize this dataset to build a collaborative filtering recommender system using Apache Spark's MLlib. Collaborative filtering is a popular technique for making personalized recommendations by analyzing user behavior and similarities between users or items. By training a collaborative filtering model on the provided data, we aim to generate accurate recommendations for users based on their purchase and play behavior. By leveraging the power of collaborative filtering and Apache Spark's MLlib, we aim to provide valuable recommendations to Steam users, enhancing their gaming experience and driving engagement on the platform.

To accomplish this task, The following steps were followed:

SETUP

An ML compute as opposed to a Standard was used for this task. It has every library needed to run machine learning algorithms. The created compute is named “AssesmentML”



AssessmentML

Databricks Runtime Version

15.0 ML (includes Apache Spark 3.5.0, Scala 2.12)

Driver type

Community Optimized 15.3 GB Memory, 2 Cores, 1 DBU

Instance

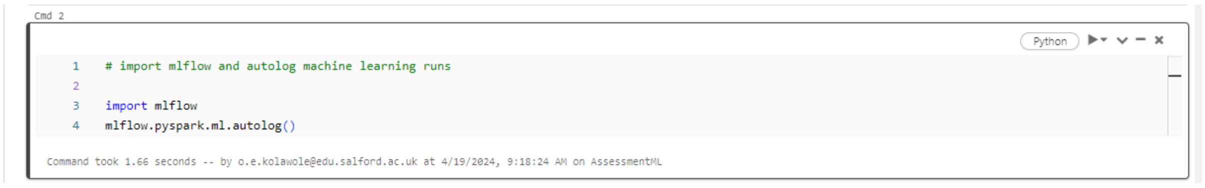
Free 15 GB Memory: As a Community Edition user, your compute will automatically terminate after an idle period of one or two hours.
For more configuration options, please upgrade your Databricks subscription.

Availability zone

us-west-2c

Fig 3.4 Compute setup.

The imported mlflow and autolog machine learning will automatically log important parameters, metrics, and models trained during the PySpark machine learning.



```
# import mlflow and autolog machine learning runs

import mlflow
mlflow.pyspark.ml.autolog()
```

```
Command took 1.66 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML
```

Fig 3.5 ML flow.

LOADING DATA AND EXPLORATORY ANALYSIS

Loading the steam-200k.csv dataset into a Spark DataFrame and conducting initial exploratory analysis of the data to gain insights into the distribution of purchases and play behavior, the most popular games, and other relevant patterns.

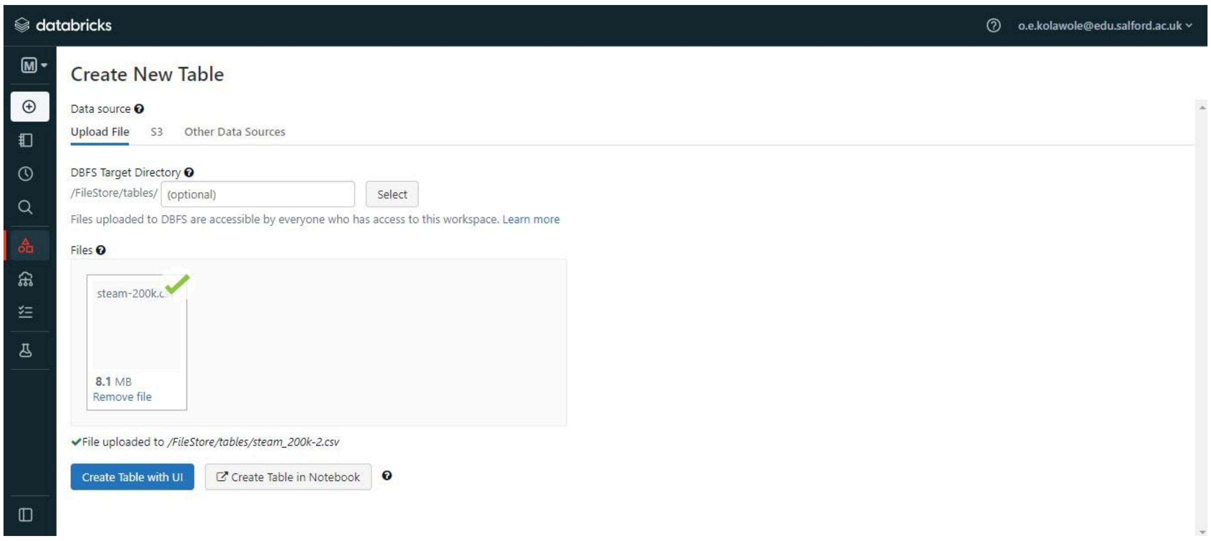
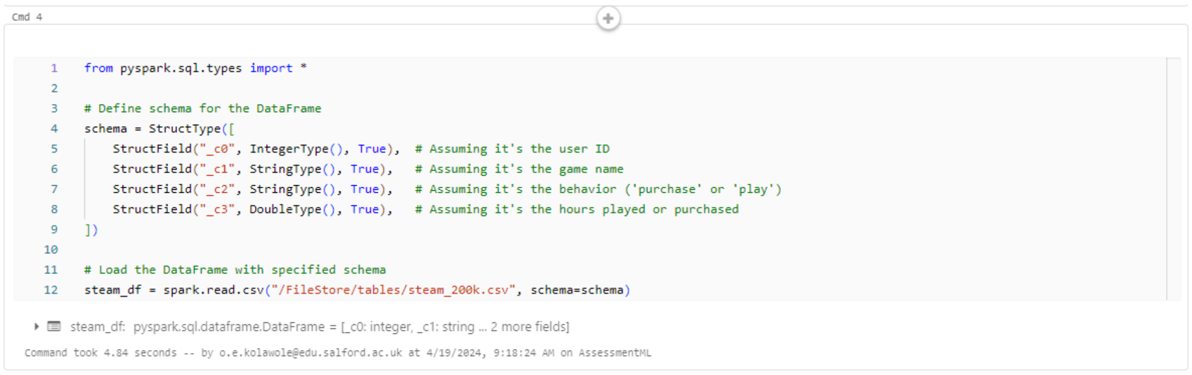


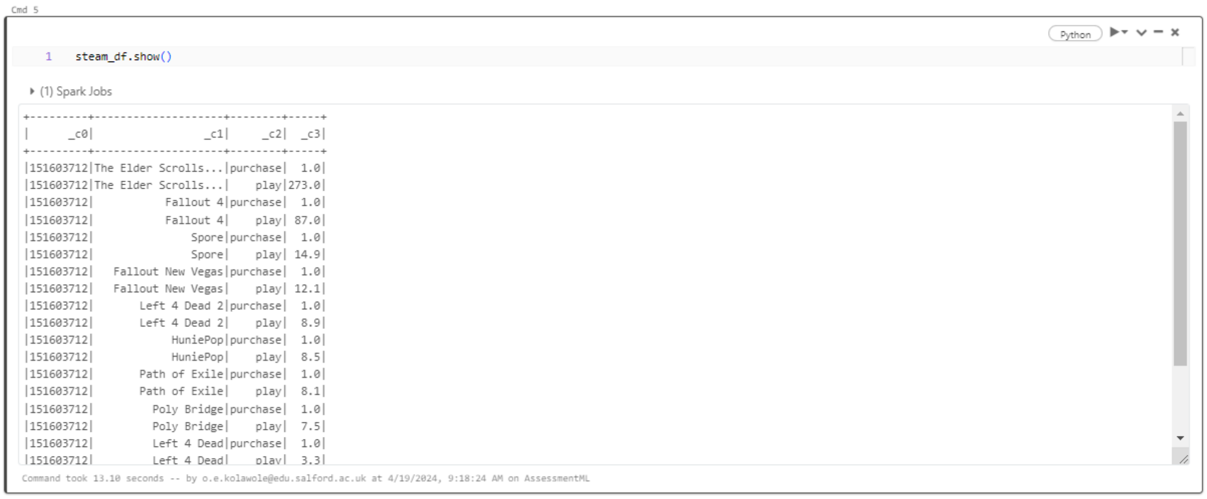
Fig 3.6 Uploading the dataset



```
Cmd 4
from pyspark.sql.types import *

# Define schema for the DataFrame
schema = StructType([
    StructField("_c0", IntegerType(), True),  # Assuming it's the user ID
    StructField("_c1", StringType(), True),   # Assuming it's the game name
    StructField("_c2", StringType(), True),   # Assuming it's the behavior ('purchase' or 'play')
    StructField("_c3", DoubleType(), True),   # Assuming it's the hours played or purchased
])

# Load the DataFrame with specified schema
steam_df = spark.read.csv("/FileStore/tables/steam_200k.csv", schema=schema)

steam_df:  pyspark.sql.dataframe.DataFrame = [_c0: integer, _c1: string ... 2 more fields]
Command took 4.84 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML
```

Fig 3.7 Schema and data loading

The DataFrame steam_df is created with the specified schema, providing a structured representation of the data. The data is read as a csv file into the databricks environment.


Cmd 5

```
steam_df.show()
```

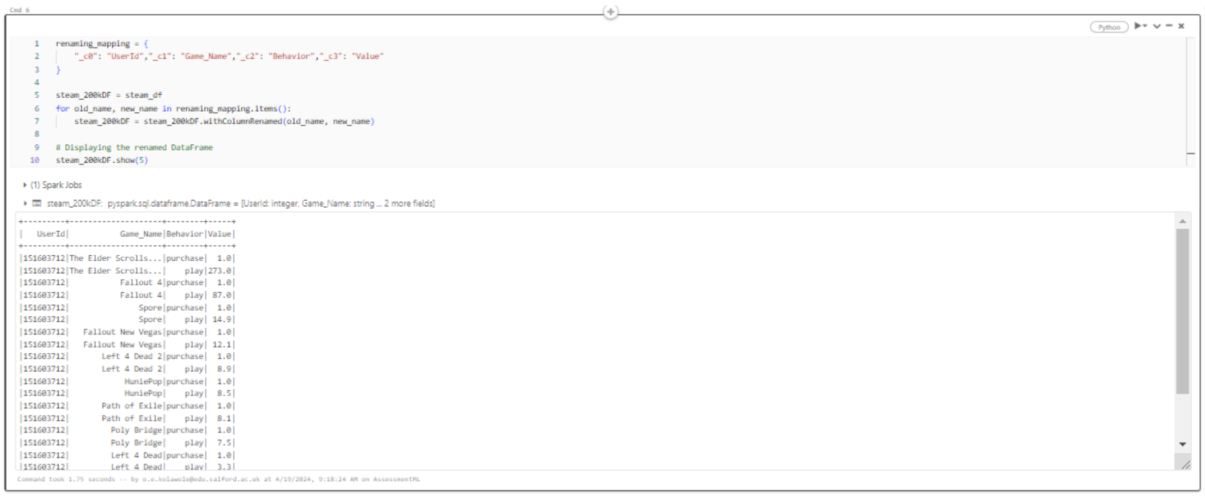
▸ (1) Spark Jobs

```
+---------+--------------------+--------+-----+
|      _c0|                 _c1|     _c2|  _c3|
+---------+--------------------+--------+-----+
|151603712|The Elder Scrolls...|purchase|  1.0|
|151603712|The Elder Scrolls...|    play|273.0|
|151603712|           Fallout 4|purchase|  1.0|
|151603712|           Fallout 4|    play| 87.0|
|151603712|               Spore|purchase|  1.0|
|151603712|               Spore|    play| 14.9|
|151603712|   Fallout New Vegas|purchase|  1.0|
|151603712|   Fallout New Vegas|    play| 12.1|
|151603712|       Left 4 Dead 2|purchase|  1.0|
|151603712|       Left 4 Dead 2|    play|  8.9|
|151603712|            HuniePop|purchase|  1.0|
|151603712|            HuniePop|    play|  8.5|
|151603712|       Path of Exile|purchase|  1.0|
|151603712|       Path of Exile|    play|  8.1|
|151603712|         Poly Bridge|purchase|  1.0|
|151603712|         Poly Bridge|    play|  7.5|
|151603712|         Left 4 Dead|purchase|  1.0|
|151603712|         Left 4 Dead|    play|  3.3|
```

```
Command took 13.10 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML
```

Fig 3.8 Displaying the data frame

The show() was used to display the first 20 rows of the steam_df to have a better understanding of the data


Cmd 6

```
renaming_mapping = {
    "_c0": "UserId","_c1": "Game_Name","_c2": "Behavior","_c3": "Value"
}

steam_200kDF = steam_df
for old_name, new_name in renaming_mapping.items():
    steam_200kDF = steam_200kDF.withColumnRenamed(old_name, new_name)

# Displaying the renamed DataFrame
steam_200kDF.show(5)
```

▸ (1) Spark Jobs

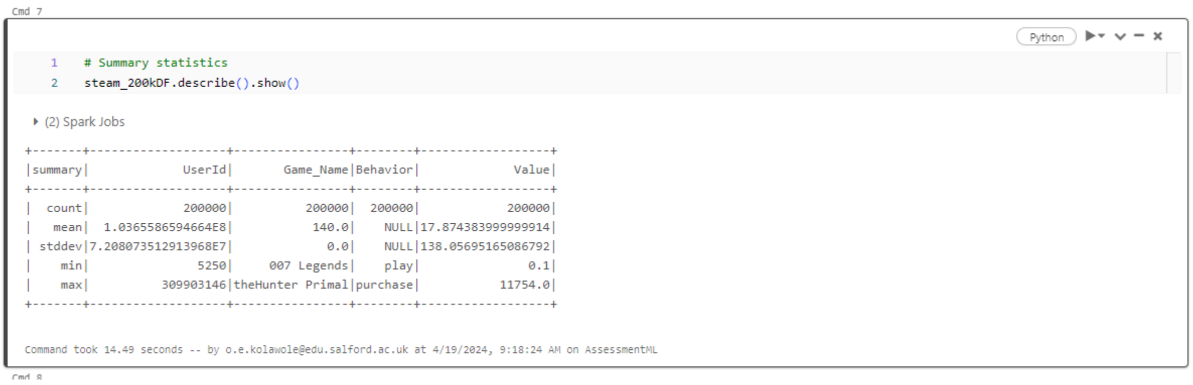
▸ steam_200kDF: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 2 more fields]

```
+---------+--------------------+--------+-----+
|   UserId|           Game_Name|Behavior|Value|
+---------+--------------------+--------+-----+
|151603712|The Elder Scrolls...|purchase|  1.0|
|151603712|The Elder Scrolls...|    play|273.0|
|151603712|           Fallout 4|purchase|  1.0|
|151603712|           Fallout 4|    play| 87.0|
|151603712|               Spore|purchase|  1.0|
|151603712|               Spore|    play| 14.9|
|151603712|   Fallout New Vegas|purchase|  1.0|
|151603712|   Fallout New Vegas|    play| 12.1|
|151603712|       Left 4 Dead 2|purchase|  1.0|
|151603712|       Left 4 Dead 2|    play|  8.9|
|151603712|            HuniePop|purchase|  1.0|
|151603712|            HuniePop|    play|  8.5|
|151603712|       Path of Exile|purchase|  1.0|
|151603712|       Path of Exile|    play|  8.1|
|151603712|         Poly Bridge|purchase|  1.0|
|151603712|         Poly Bridge|    play|  7.5|
|151603712|         Left 4 Dead|purchase|  1.0|
|151603712|         Left 4 Dead|    play|  3.3|
```

```
Command took 1.75 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML
```

Fig 3.9 Column renaming

The column name of the data frame is renamed using the renaming mapping method and the new header is UserId, Game_Name, Behavior and Value giving each column better contextual understanding.



Cmd 7

```
# Summary statistics
steam_200kDF.describe().show()
```

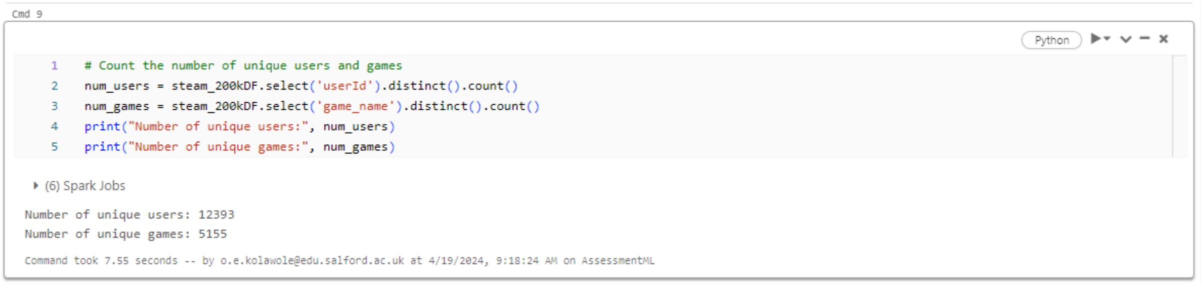
(2) Spark Jobs

```
+-------+------------------+---------------+--------+------------------+
|summary|            UserId|      Game_Name|Behavior|             Value|
+-------+------------------+---------------+--------+------------------+
|  count|            200000|         200000|  200000|            200000|
|   mean|  1.03655865946648|          140.0|    NULL|17.874383999999914|
| stddev|7.208073512913968E7|            0.0|    NULL|138.05695165086792|
|    min|              5250|    007 Legends|    play|               0.1|
|    max|         309903146|theHunter Primal|purchase|           11754.0|
+-------+------------------+---------------+--------+------------------+
```

Command took 14.49 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML

Fig 4.0 Descriptive Statistics

The descriptive statistics that provide insight into the distribution, central tendency, and variability of the Steam Dataset are a crucial component of the EDA analysis carried out for the task. A summary function was used to produce the mean and the median, which show the fundamental or important value of the findings.



Cmd 9

```
# Count the number of unique users and games
num_users = steam_200kDF.select('userId').distinct().count()
num_games = steam_200kDF.select('game_name').distinct().count()
print("Number of unique users:", num_users)
print("Number of unique games:", num_games)
```

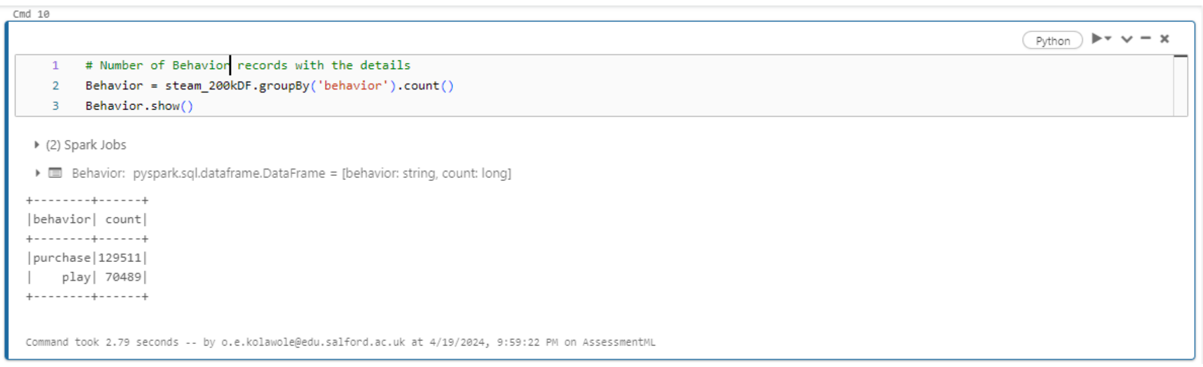
(6) Spark Jobs

```
Number of unique users: 12393
Number of unique games: 5155
```

Command took 7.55 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML

Fig 4.1 Unique features

Counting the number of distinct users and games in the Data Frame helps in tracking performance metrics.



Cmd 10

```
# Number of Behavior records with the details
Behavior = steam_200kDF.groupBy('behavior').count()
Behavior.show()
```

(2) Spark Jobs

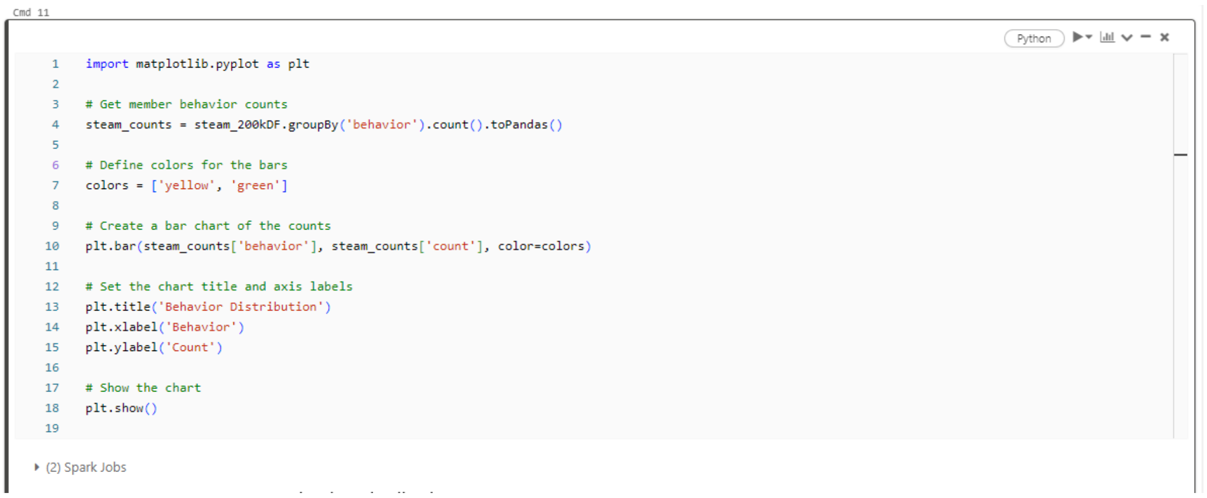
Behavior: pyspark.sql.dataframe.DataFrame = [behavior: string, count: long]

```
+--------+------+
|behavior| count|
+--------+------+
|purchase|129511|
|    play| 70489|
+--------+------+
```

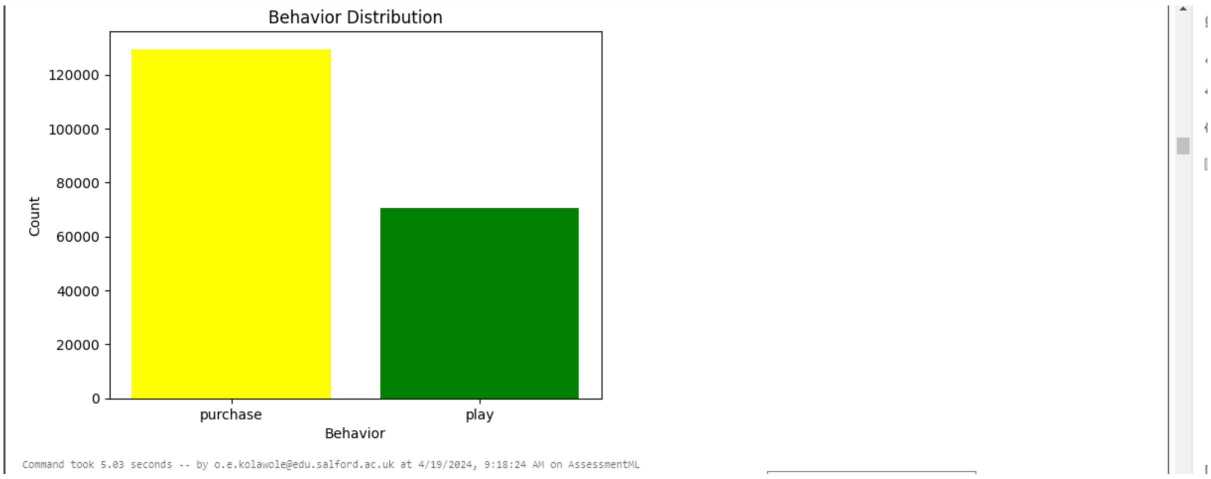
Command took 2.79 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Fig 4.2 Behavior of the members

By displaying the number of records for every behavior category in your dataset, this code provides insight into the distribution of behaviors within it. It gives a better understanding of the relative prevalence and frequency of various behaviors within the dataset. It shows the number of games played and purchased. The above code is then visualised using pyplot giving the below bar chart.



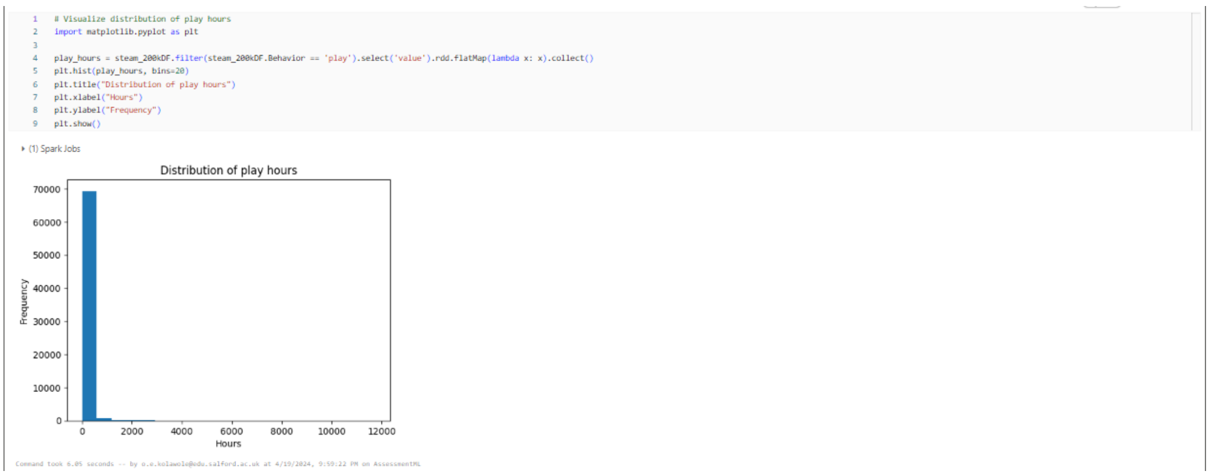
```
Cmd 11
import matplotlib.pyplot as plt

# Get member behavior counts
steam_counts = steam_200kDF.groupBy('behavior').count().toPandas()

# Define colors for the bars
colors = ['yellow', 'green']

# Create a bar chart of the counts
plt.bar(steam_counts['behavior'], steam_counts['count'], color=colors)

# Set the chart title and axis labels
plt.title('Behavior Distribution')
plt.xlabel('Behavior')
plt.ylabel('Count')

# Show the chart
plt.show()

▸ (2) Spark Jobs
```



```
Command took 5.03 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:18:24 AM on AssessmentML
```

Fig 4.3 Behavior of the members

The code creates a bar chart of behavior distribution using Matplotlib, grouping DataFrame steam_200kDF by 'behavior' column and counting occurrences. It converts result to Pandas DataFrame, defines colors, creates bar chart with x-axis and y-axis, sets title, x-label, and y-label, and displays.



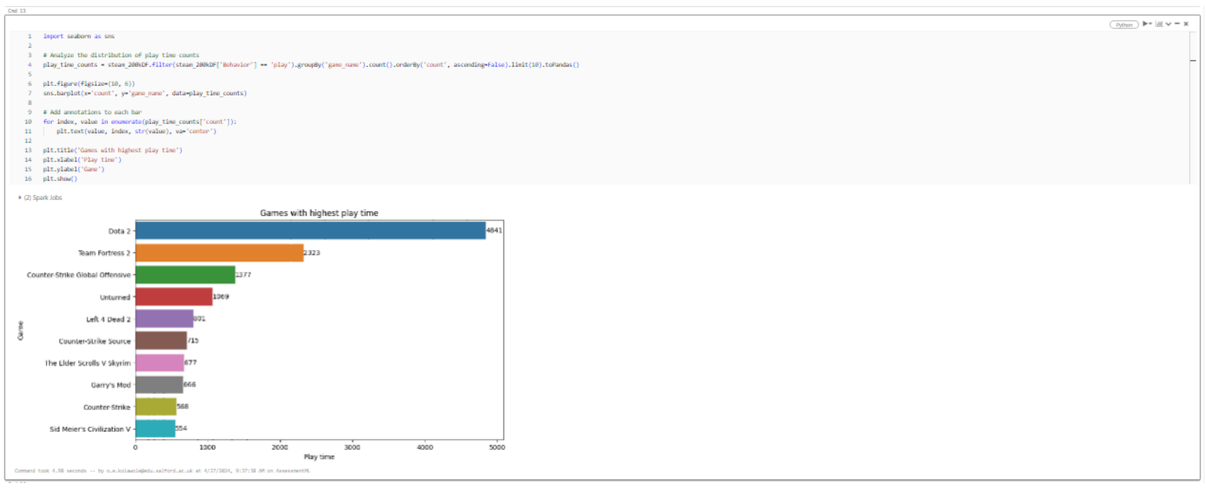
```python
# Visualize distribution of play hours
import matplotlib.pyplot as plt

play_hours = steam_200kDF.filter(steam_200kDF.Behavior == 'play').select('value').rdd.flatMap(lambda x: x).collect()
plt.hist(play_hours, bins=20)
plt.title("Distribution of play hours")
plt.xlabel("Hours")
plt.ylabel("Frequency")
plt.show()
```

▸ (1) Spark Jobs

```
Command took 6.05 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 4.4 Distribution of play hours in the play behavior of the members

The code creates a Matplotlib histogram to display the distribution of play hours. It filters the DataFrame `steam_200kDF`, selects the 'value' column, flattens the RDD structure, collects values into a Python list, creates a histogram with 20 bins, sets titles, labels, and displays the histogram. The x-axis represents hours and the y-axis represents hour frequency.



```python
import seaborn as sns

# Analyze the distribution of play time counts
play_time_counts = steam_200kDF.filter(steam_200kDF['Behavior'] == 'play').groupBy('game_name').count().orderBy('count', ascending=False).limit(10).toPandas()

plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='game_name', data=play_time_counts)

# Add annotations to each bar
for index, value in enumerate(play_time_counts['count']):
    plt.text(value, index, str(value), va='center')

plt.title('Games with highest play time')
plt.xlabel('Play time')
plt.ylabel('Game')
plt.show()
```

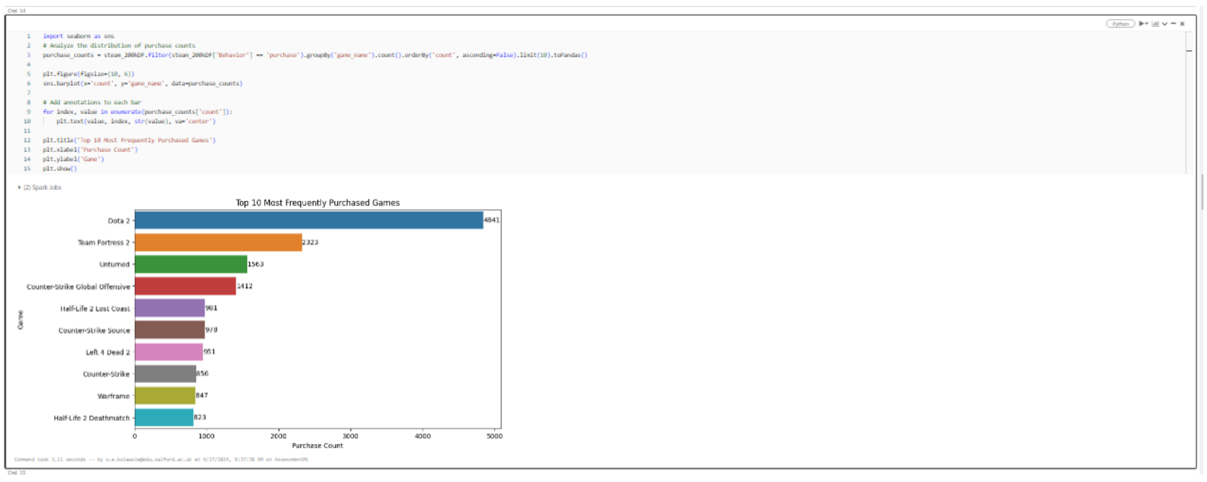
▸ (2) Spark Jobs

```
Command took 4.98 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/27/2024, 9:37:38 AM on AssessmentML
```

Fig 4.5 Games with the highest play time

The code creates a bar plot using Seaborn to analyze the distribution of play time counts for the top 10 games. It filters the DataFrame `steam_200kDF`, groups it by 'game_name', and counts the occurrences of each game. The results are converted to a Pandas DataFrame, and a bar plot is created with 'count' as the x-axis and 'game_name' as the y-axis.

This gives additional description of the data, giving an insight of the behavior of the member. The highest play game is Dota 2 of almost 5000 hours.


Cmd 14

```python
import seaborn as sns
# Analyze the distribution of purchase counts
purchase_counts = steam_200kDF.filter(steam_200kDF['Behavior'] == 'purchase').groupBy('game_name').count().orderBy('count', ascending=False).limit(10).toPandas()

plt.figure(figsize=(10, 6))
sns.barplot(x='count', y='game_name', data=purchase_counts)

# Add annotations to each bar
for index, value in enumerate(purchase_counts['count']):
    plt.text(value, index, str(value), va='center')

plt.title('Top 10 Most Frequently Purchased Games')
plt.xlabel('Purchase Count')
plt.ylabel('Game')
plt.show()
```

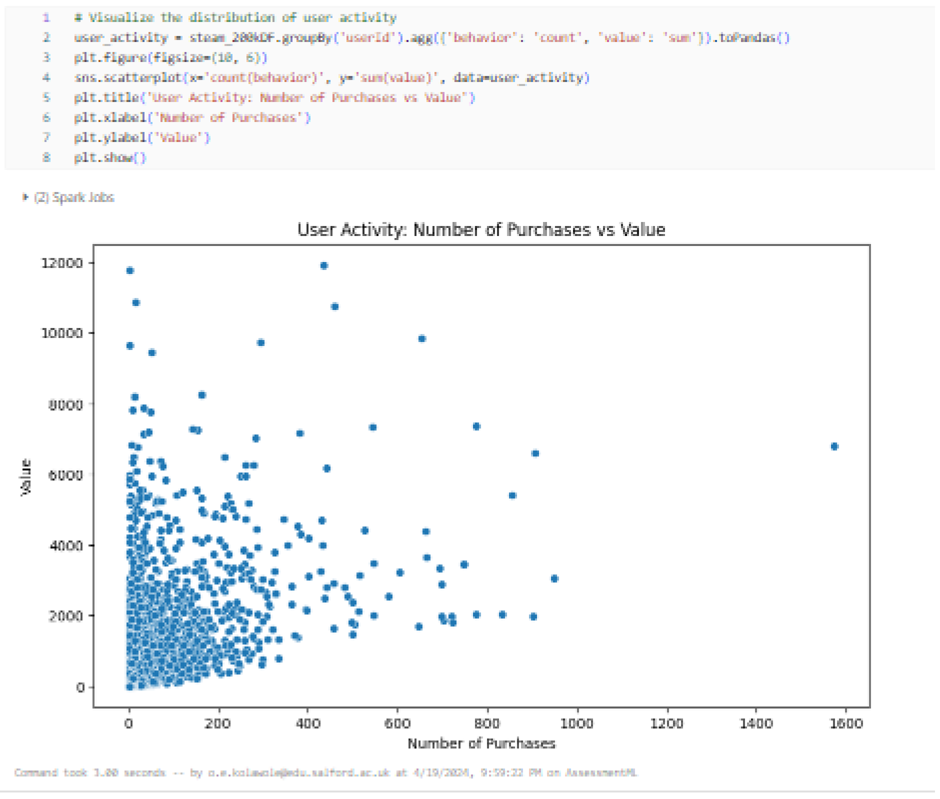
▸ (2) Spark Jobs

```
Command took 3.11 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/27/2024, 9:37:38 AM on AssessmentML
```

Cmd 15

Fig 4.6 Frequently purchased games

The code uses Seaborn and Matplotlib to visualize the distribution of purchase counts for the top 10 most frequently purchased games. It filters the DataFrame `steam_200kDF`, groups it by 'game_name', and counts the occurrences of each game. The results are converted to a Pandas DataFrame, and a horizontal bar plot is created with 'count' as the x-axis and 'game_name' as the y-axis.

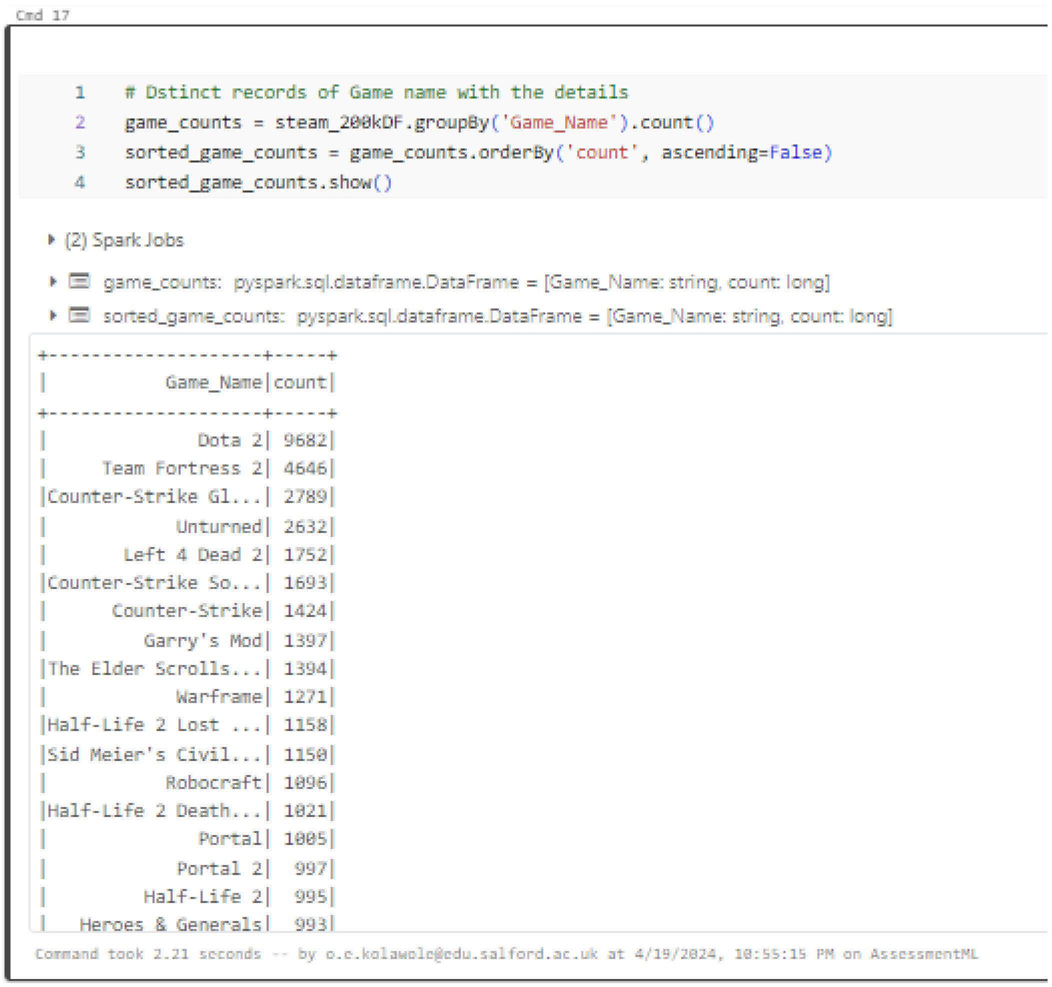

```python
# Visualize the distribution of user activity
user_activity = steam_200kDF.groupBy('userId').agg({'behavior': 'count', 'value': 'sum'}).toPandas()
plt.figure(figsize=(10, 6))
sns.scatterplot(x='count(behavior)', y='sum(value)', data=user_activity)
plt.title('User Activity: Number of Purchases vs Value')
plt.xlabel('Number of Purchases')
plt.ylabel('Value')
plt.show()
```

▸ (2) Spark Jobs

```
Command took 1.00 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 4.7 User activity

The code for visualizing user activity distribution using Seaborn and Matplotlib is well-structured. It groups the DataFrame `steam_200kDF` by 'userId', aggregates the count of 'behavior' and the sum of 'value' for each user, converts the result to a Pandas DataFrame, creates a scatter plot, sets title, x-label, and y-label, and displays the plot.



Cmd 17

```python
# Dstinct records of Game name with the details
game_counts = steam_200kDF.groupBy('Game_Name').count()
sorted_game_counts = game_counts.orderBy('count', ascending=False)
sorted_game_counts.show()
```

```
(2) Spark Jobs
game_counts:  pyspark.sql.dataframe.DataFrame = [Game_Name: string, count: long]
sorted_game_counts:  pyspark.sql.dataframe.DataFrame = [Game_Name: string, count: long]
```

```
+--------------------+-----+
|           Game_Name|count|
+--------------------+-----+
|              Dota 2| 9682|
|     Team Fortress 2| 4646|
|Counter-Strike Gl...| 2789|
|            Unturned| 2632|
|       Left 4 Dead 2| 1752|
|Counter-Strike So...| 1693|
|      Counter-Strike| 1424|
|         Garry's Mod| 1397|
|The Elder Scrolls...| 1394|
|            Warframe| 1271|
|Half-Life 2 Lost ...| 1158|
|Sid Meier's Civil...| 1150|
|          Robocraft| 1096|
|Half-Life 2 Death...| 1021|
|              Portal| 1005|
|            Portal 2|  997|
|         Half-Life 2|  995|
|   Heroes & Generals|  993|
```

```
Command took 2.21 seconds -- by o.c.kolawole@edu.salford.ac.uk at 4/19/2024, 10:55:15 PM on AssessmentML
```

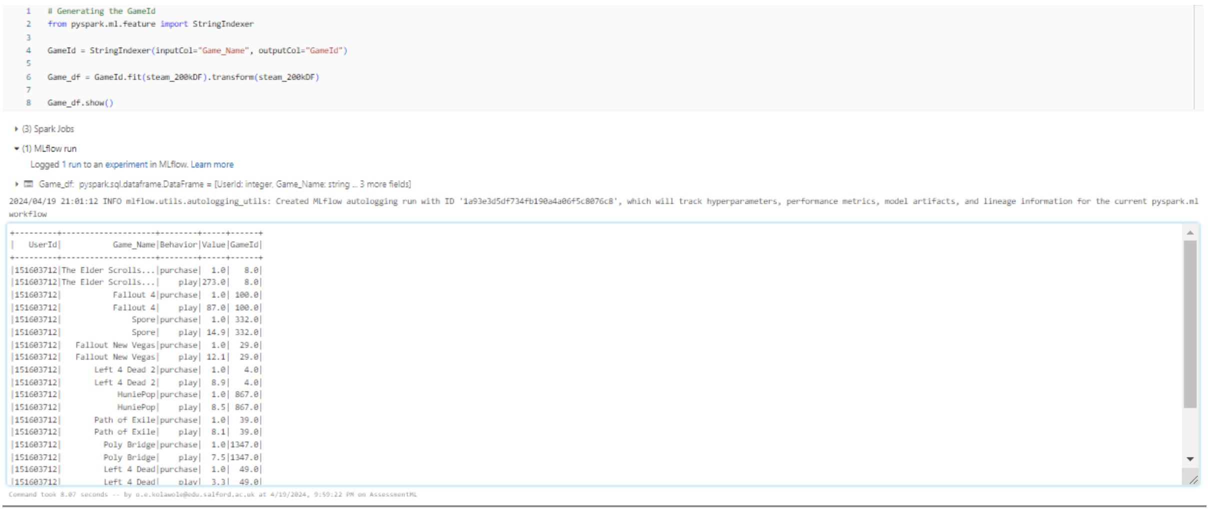
Fig 4.8 Game Count

The code correctly groups the DataFrame `steam_200kDF` by the 'Game_Name' column and counts the occurrences of each game. It then sorts the resulting DataFrame `game_counts` by the count column in descending order and displays the distinct records of game names and their respective counts. The sorted DataFrame is then displayed.

The game names along with their respective count was been generated to determine the users interest.

DATA PREPARATION AND PREPROCESSING

We will preprocess the data to prepare it for training the recommender system. This may involve converting the data into the required format, splitting it into training and test sets, and generating unique integer IDs for the games.



```
# Generating the GameId
from pyspark.ml.feature import StringIndexer

GameId = StringIndexer(inputCol="Game_Name", outputCol="GameId")

Game_df = GameId.fit(steam_200kDF).transform(steam_200kDF)

Game_df.show()
```

(3) Spark Jobs
(1) MLflow run
Logged 1 run to an experiment in MLflow. Learn more
Game_df: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 3 more fields]

```
2024/04/19 21:01:12 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '1a93e3d5df734fb190a4a06f5c8076c8', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
```

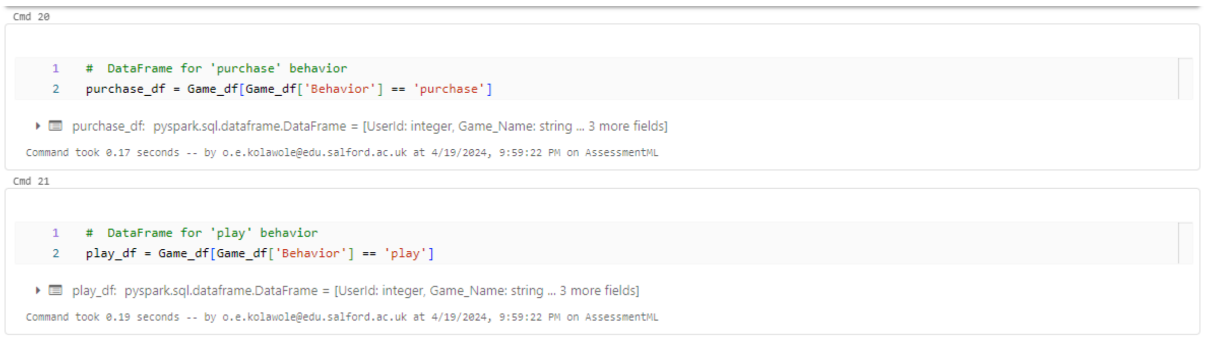
```
+---------+-------------------+--------+-----+------+
|   UserId|          Game_Name|Behavior|Value|GameId|
+---------+-------------------+--------+-----+------+
|151603712|The Elder Scrolls...|purchase|  1.0|   8.0|
|151603712|The Elder Scrolls...|    play|273.0|   8.0|
|151603712|          Fallout 4|purchase|  1.0| 100.0|
|151603712|          Fallout 4|    play| 87.0| 100.0|
|151603712|              Spore|purchase|  1.0| 332.0|
|151603712|              Spore|    play| 14.9| 332.0|
|151603712|  Fallout New Vegas|purchase|  1.0|  29.0|
|151603712|  Fallout New Vegas|    play| 12.1|  29.0|
|151603712|      Left 4 Dead 2|purchase|  1.0|   4.0|
|151603712|      Left 4 Dead 2|    play|  8.9|   4.0|
|151603712|           HuniePop|purchase|  1.0| 867.0|
|151603712|           HuniePop|    play|  8.5| 867.0|
|151603712|      Path of Exile|purchase|  1.0|  39.0|
|151603712|      Path of Exile|    play|  8.1|  39.0|
|151603712|        Poly Bridge|purchase|  1.0|1347.0|
|151603712|        Poly Bridge|    play|  7.5|1347.0|
|151603712|        Left 4 Dead|purchase|  1.0|  49.0|
|151603712|        Left 4 Dead|    play|  3.3|  49.0|
```

Command took 8.07 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Fig 4.9 Generating GameId

To create a distinct numerical identifier (GameId) for every distinct game name in the Data Frame steam_200kDF, StringIndexer from PySpark's MLlib was used. It displays the data frame with the new GameId column.

This process assigns a unique numerical identifier (GameId) to each distinct game name in the DataFrame, which can be useful for machine learning tasks that require numerical inputs.



Cmd 20

```
#  DataFrame for 'purchase' behavior
purchase_df = Game_df[Game_df['Behavior'] == 'purchase']
```

purchase_df: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 3 more fields]

Command took 0.17 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Cmd 21

```
#  DataFrame for 'play' behavior
play_df = Game_df[Game_df['Behavior'] == 'play']
```

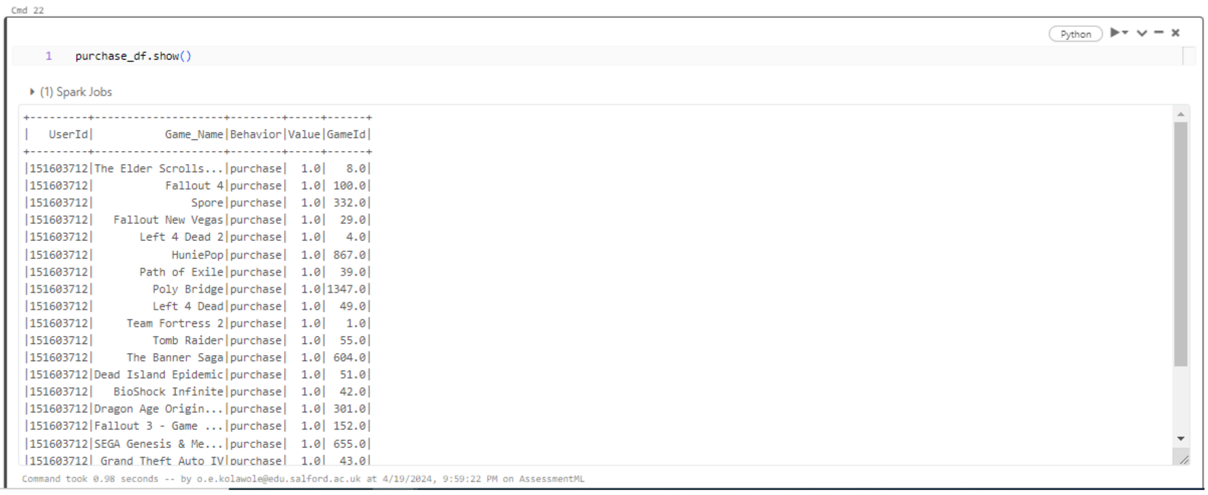
play_df: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 3 more fields]

Command took 0.19 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Fig 5.0 Generating the distinct behavior data frame

The data frame is split into two based on the behavior. The filter() method keeps the rows needed in each newly created data frame based on the rows needed (purchase_df and play_df).

We would be using the purchase data frame for the analysis and the value for the purchase data frame is 1.



Cmd 22

```
purchase_df.show()
```

(1) Spark Jobs

```
+---------+--------------------+--------+-----+------+
|   UserId|           Game_Name|Behavior|Value|GameId|
+---------+--------------------+--------+-----+------+
|151603712|The Elder Scrolls...|purchase|  1.0|   8.0|
|151603712|           Fallout 4|purchase|  1.0| 100.0|
|151603712|               Spore|purchase|  1.0| 332.0|
|151603712|   Fallout New Vegas|purchase|  1.0|  29.0|
|151603712|       Left 4 Dead 2|purchase|  1.0|   4.0|
|151603712|            HuniePop|purchase|  1.0| 867.0|
|151603712|       Path of Exile|purchase|  1.0|  39.0|
|151603712|         Poly Bridge|purchase|  1.0|1347.0|
|151603712|         Left 4 Dead|purchase|  1.0|  49.0|
|151603712|     Team Fortress 2|purchase|  1.0|   1.0|
|151603712|         Tomb Raider|purchase|  1.0|  55.0|
|151603712|     The Banner Saga|purchase|  1.0| 604.0|
|151603712|Dead Island Epidemic|purchase|  1.0|  51.0|
|151603712|   BioShock Infinite|purchase|  1.0|  42.0|
|151603712|Dragon Age Origin...|purchase|  1.0| 301.0|
|151603712|Fallout 3 - Game ...|purchase|  1.0| 152.0|
|151603712|SEGA Genesis & Me...|purchase|  1.0| 655.0|
|151603712| Grand Theft Auto IV|purchase|  1.0|  43.0|
```

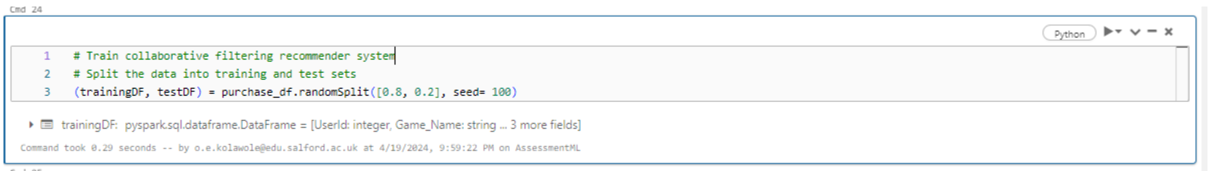
Command took 0.98 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Fig 5.1 Displaying the Purchase_df

The newly created purchase_df is viewed.

MODEL TRAINING, EVALUATION AND HYPERPARAMETER

The purchase_df DataFrame is split into training and test sets for training a collaborative filtering recommender system. The randomSplit() method is used to randomly split the data into two sets based on the provided weights which is 80% training and 20% testing and the seed parameter ensures reproducibility by specifying a seed for the random number generator.



Cmd 24

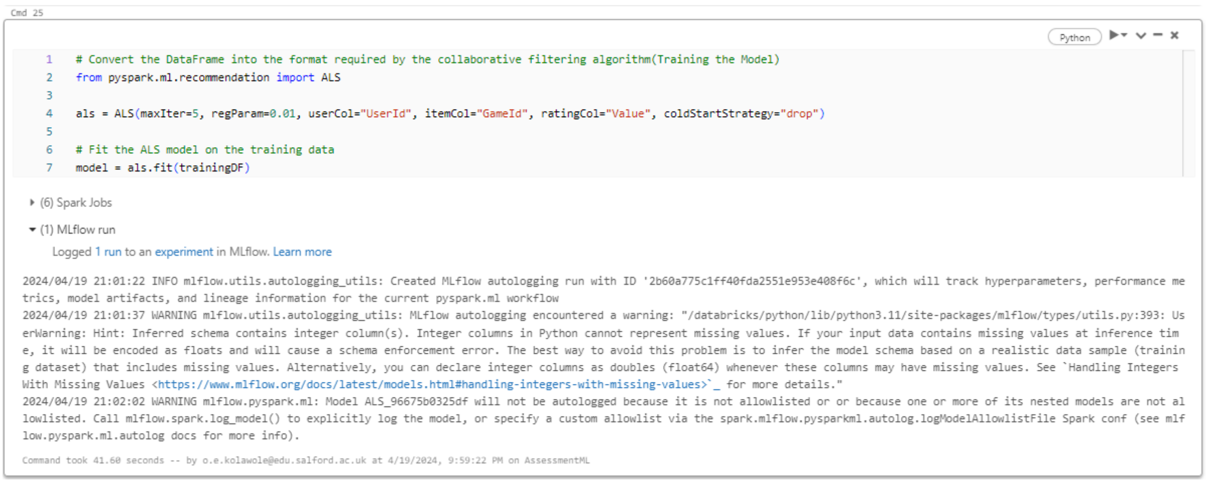
```
# Train collaborative filtering recommender system
# Split the data into training and test sets
(trainingDF, testDF) = purchase_df.randomSplit([0.8, 0.2], seed= 100)
```

trainingDF: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 3 more fields]

Command took 0.29 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML

Fig 5.2 Splitting the purchase_df

We will use MLlib's Alternating Least Squares (ALS) algorithm to train a collaborative filtering model on the prepared data which helps to assess its accuracy and effectiveness.

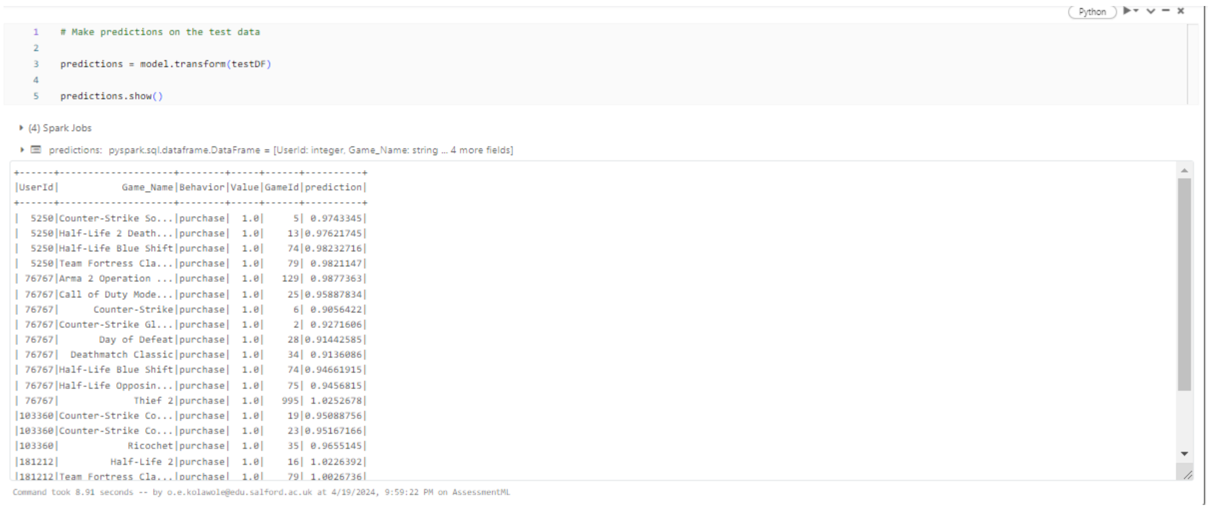


```python
# Convert the DataFrame into the format required by the collaborative filtering algorithm(Training the Model)
from pyspark.ml.recommendation import ALS

als = ALS(maxIter=5, regParam=0.01, userCol="UserId", itemCol="GameId", ratingCol="Value", coldStartStrategy="drop")

# Fit the ALS model on the training data
model = als.fit(trainingDF)
```

```
▸ (6) Spark Jobs
▾ (1) MLflow run
    Logged 1 run to an experiment in MLflow. Learn more

2024/04/19 21:01:22 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '2b60a775c1ff40fda2551e953e408f6c', which will track hyperparameters, performance me
trics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/04/19 21:01:37 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/databricks/python/lib/python3.11/site-packages/mlflow/types/utils.py:393: Us
erWarning: Hint: Inferred schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference tim
e, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (trainin
g dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers
With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details."
2024/04/19 21:02:02 WARNING mlflow.pyspark.ml: Model ALS_96675b0325df will not be autologged because it is not allowlisted or or because one or more of its nested models are not al
lowlisted. Call mlflow.spark.log_model() to explicitly log the model, or specify a custom allowlist via the spark.mlflow.pysparkml.autolog.logModelAllowlistFile Spark conf (see mlf
low.pyspark.ml.autolog docs for more info).

Command took 41.60 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 5.3 Alternating Least Squares (ALS) algorithm training

Inspecting the model's predictions on the test data is essential for evaluating the model's performance and understanding how well it generalizes to unseen data. This allows us to examine the model's predictions alongside the original test data. The predicted value can be compared to the purchase value so we can determine the difference between the values.



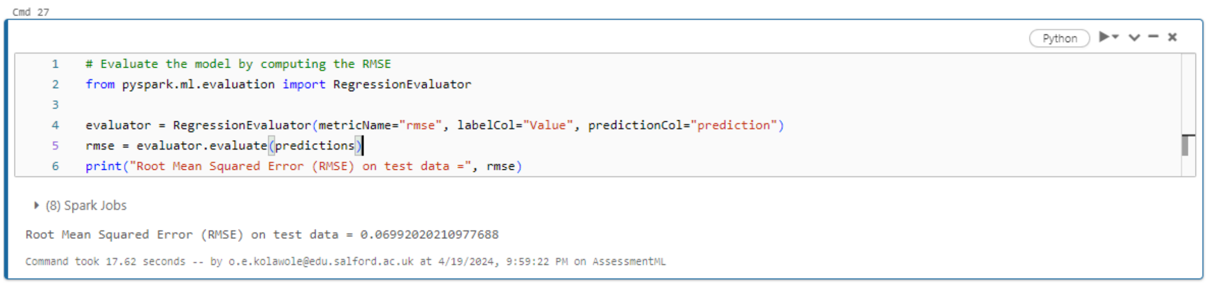
```python
# Make predictions on the test data

predictions = model.transform(testDF)

predictions.show()
```

```
▸ (4) Spark Jobs
▸ predictions:  pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 4 more fields]

+------+--------------------+--------+-----+------+----------+
|UserId|           Game_Name|Behavior|Value|GameId|prediction|
+------+--------------------+--------+-----+------+----------+
|  5250|Counter-Strike So...|purchase|  1.0|     5| 0.9743345|
|  5250|Half-Life 2 Death...|purchase|  1.0|    13|0.97621745|
|  5250|Half-Life Blue Shift|purchase|  1.0|    74|0.98232716|
|  5250|Team Fortress Cla...|purchase|  1.0|    79| 0.9821147|
| 76767|Arma 2 Operation ...|purchase|  1.0|   129| 0.9877363|
| 76767|Call of Duty Mode...|purchase|  1.0|    25|0.95887834|
| 76767|      Counter-Strike|purchase|  1.0|     6| 0.9056422|
| 76767|Counter-Strike Gl...|purchase|  1.0|     2| 0.9271606|
| 76767|       Day of Defeat|purchase|  1.0|    28|0.91442585|
| 76767|  Deathmatch Classic|purchase|  1.0|    34| 0.9136086|
| 76767|Half-Life Blue Shift|purchase|  1.0|    74|0.94661915|
| 76767|Half-Life Opposin...|purchase|  1.0|    75| 0.9456815|
| 76767|             Thief 2|purchase|  1.0|   995| 1.0252678|
|103360|Counter-Strike Co...|purchase|  1.0|    19|0.95088756|
|103360|Counter-Strike Co...|purchase|  1.0|    23|0.95167166|
|103360|            Ricochet|purchase|  1.0|    35| 0.9655145|
|181212|          Half-Life 2|purchase|  1.0|    16| 1.0226392|
|181212|Team Fortress Cla...|purchase|  1.0|    79| 1.0026736|
Command took 8.91 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 5.4 Prediction of test data

The code uses PySpark's RegressionEvaluator to evaluate a regression model, importing it from pyspark.ml.evaluation. The RegressionEvaluator object is instantiated with metricName set to "rmse", labelCol set to "Value", and predictionCol set to "prediction". The evaluate method computes the RMSE on predictions.
To calculate the RMSE, the actual values (Value column) are compared to the predicted values (prediction column).

The average difference between the actual and predicted values is indicated by the RMSE value that you will receive in the output. Better model performance is indicated by lower RMSE values.



Cmd 27

```python
# Evaluate the model by computing the RMSE
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="Value", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data =", rmse)
```

(8) Spark Jobs

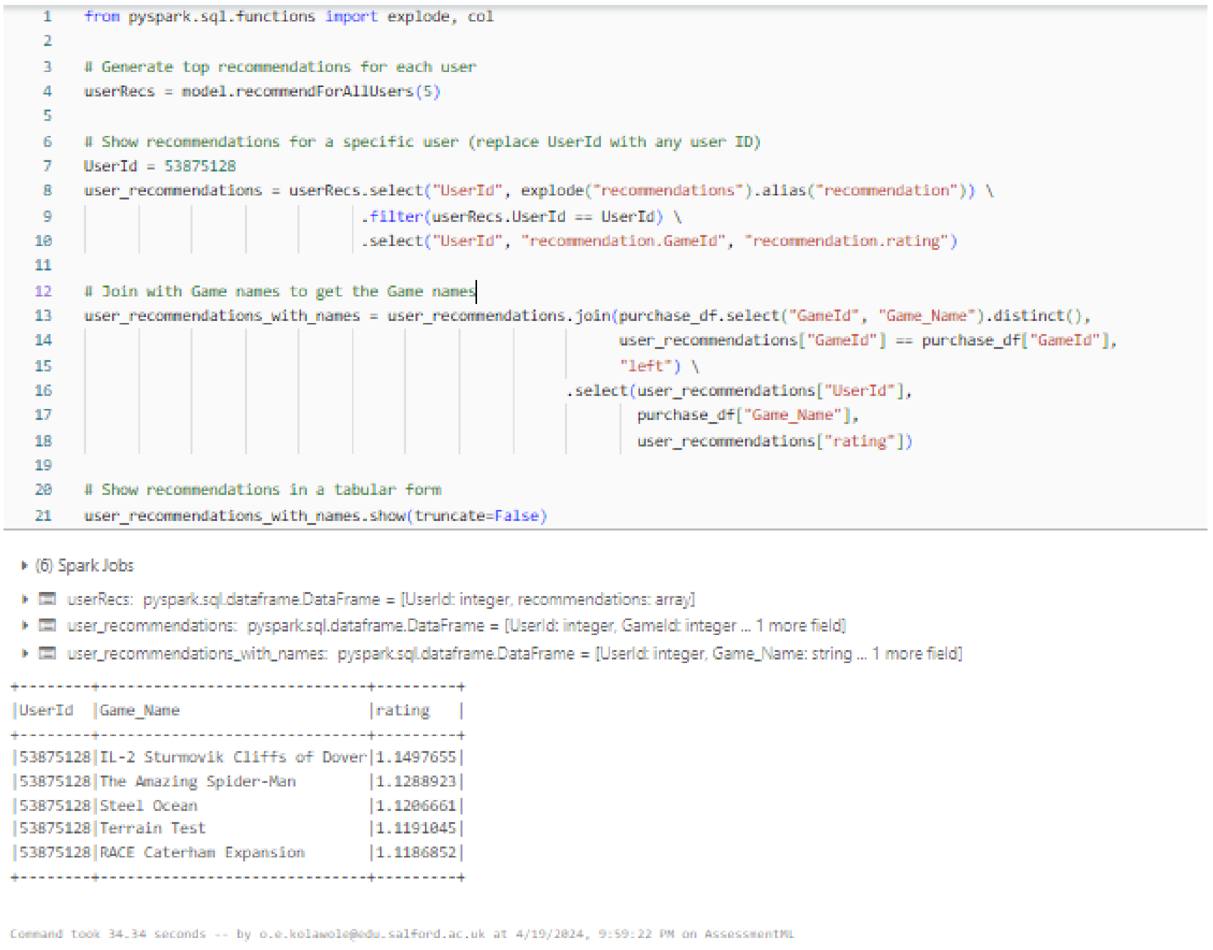
```
Root Mean Squared Error (RMSE) on test data = 0.06992020210977688
Command took 17.62 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 5.5 Model Evaluation

A lower RMSE value indicates that the model's predictions are closer to the actual values, suggesting better performance. The RMSE of 0.0699 indicates that, on average, the model's predictions are off by approximately 0.0699 units from the actual values. This means that the model is performing well on the test data.

Generating top recommendations for each user using the ALS model and displaying recommendations for a specific user identified by their UserId. It retrieves the top 5 recommendations for the specified user and joins them with the Data Frame containing game names (purchase_df) to display the recommendations with their corresponding game names.

The result for the UserId 53875128 is displayed below showing the game name and rating given by the ALS model. Higher ratings indicate higher predicted preference or likelihood of the user enjoying the game.



```python
from pyspark.sql.functions import explode, col

# Generate top recommendations for each user
userRecs = model.recommendForAllUsers(5)

# Show recommendations for a specific user (replace UserId with any user ID)
UserId = 53875128
user_recommendations = userRecs.select("UserId", explode("recommendations").alias("recommendation")) \
                              .filter(userRecs.UserId == UserId) \
                              .select("UserId", "recommendation.GameId", "recommendation.rating")

# Join with Game names to get the Game names
user_recommendations_with_names = user_recommendations.join(purchase_df.select("GameId", "Game_Name").distinct(),
                                                            user_recommendations["GameId"] == purchase_df["GameId"],
                                                            "left") \
                                                      .select(user_recommendations["UserId"],
                                                              purchase_df["Game_Name"],
                                                              user_recommendations["rating"])

# Show recommendations in a tabular form
user_recommendations_with_names.show(truncate=False)
```

```
▸ (6) Spark Jobs
▸ userRecs:  pyspark.sql.dataframe.DataFrame = [UserId: integer, recommendations: array]
▸ user_recommendations:  pyspark.sql.dataframe.DataFrame = [UserId: integer, GameId: integer ... 1 more field]
▸ user_recommendations_with_names:  pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 1 more field]

+--------+----------------------------+---------+
|UserId  |Game_Name                   |rating   |
+--------+----------------------------+---------+
|53875128|IL-2 Sturmovik Cliffs of Dover|1.1497655|
|53875128|The Amazing Spider-Man      |1.1288923|
|53875128|Steel Ocean                 |1.1206661|
|53875128|Terrain Test                |1.1191045|
|53875128|RACE Caterham Expansion     |1.1186852|
+--------+----------------------------+---------+

Command took 34.34 seconds -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 5.6 Model Recommendations

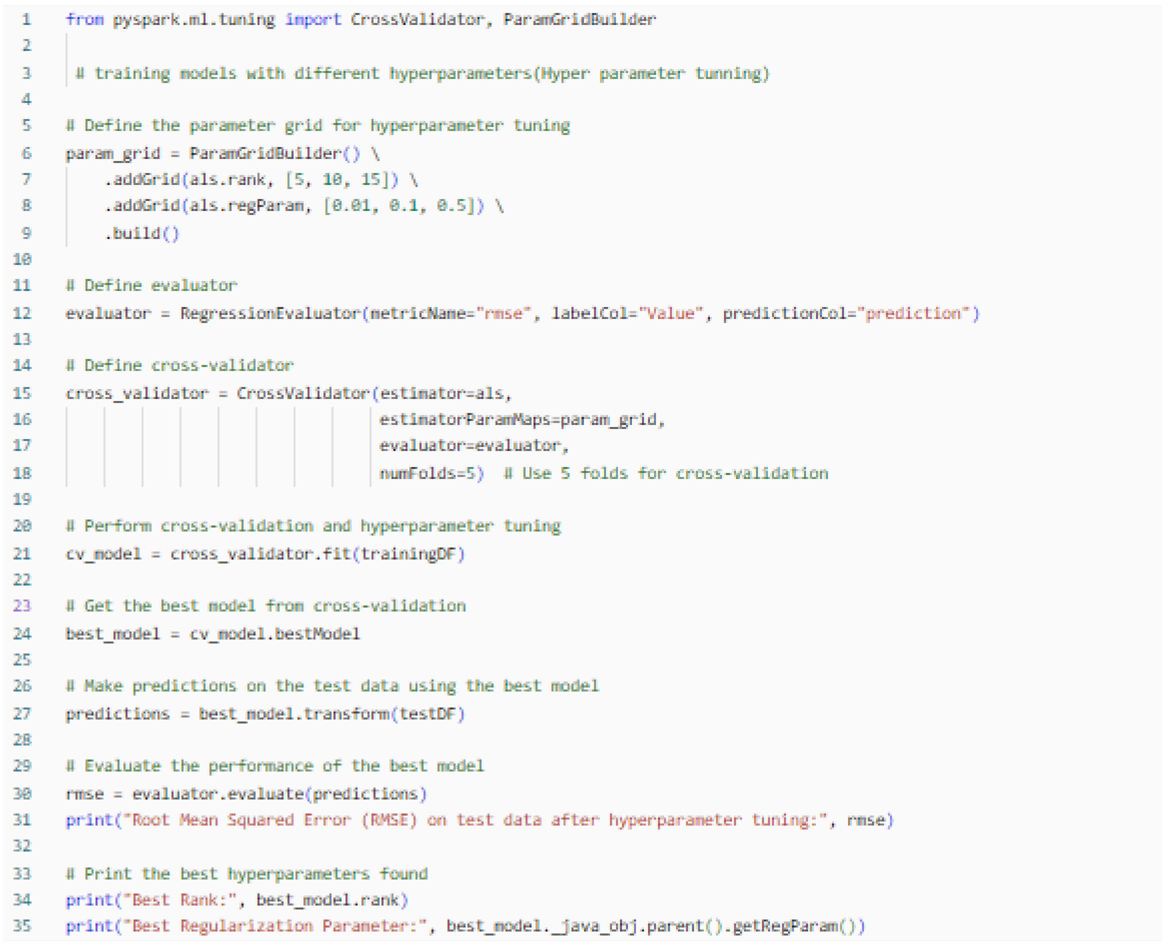
Hyperparameter Tuning

The code uses PySpark to perform hyperparameter tuning for the ALS model using cross-validation. It defines a parameter grid, a regression evaluator, and a cross-validator. The ALS estimator, grid, evaluator, and number of folds are set to 5. Cross-validation and hyperparameter tuning are performed using the `fit()` method. The best model is obtained, predictions are made on test data, and performance is evaluated based on RMSE. The best hyperparameters are printed out.

This ensures that the best-performing model is selected and its performance is evaluated using a reliable evaluation metric (RMSE).

It selects the best-performing model based on cross-validation and evaluates the performance and displays the rmse, and best rank and regularization parameter found during the hyperparameter tuning.

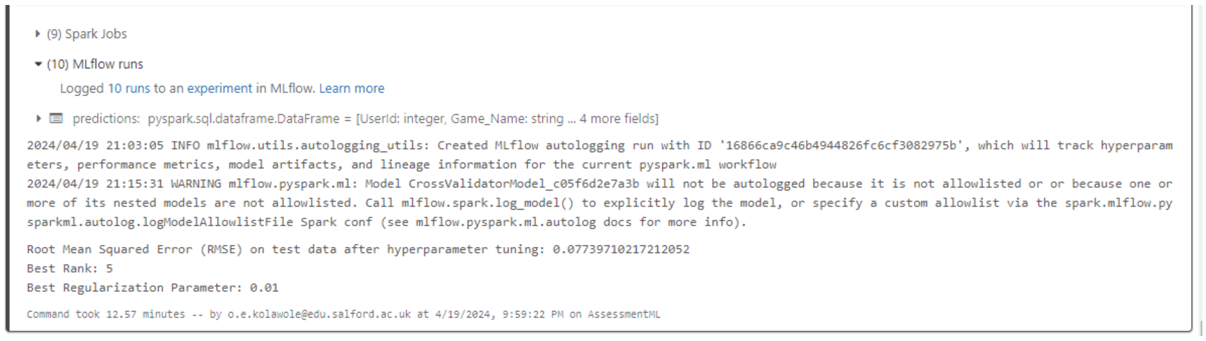
It's a crucial step in building accurate and effective recommendation systems.



```python
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

 # training models with different hyperparameters(Hyper parameter tunning)

# Define the parameter grid for hyperparameter tuning
param_grid = ParamGridBuilder() \
    .addGrid(als.rank, [5, 10, 15]) \
    .addGrid(als.regParam, [0.01, 0.1, 0.5]) \
    .build()

# Define evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="Value", predictionCol="prediction")

# Define cross-validator
cross_validator = CrossValidator(estimator=als,
                                 estimatorParamMaps=param_grid,
                                 evaluator=evaluator,
                                 numFolds=5)  # Use 5 folds for cross-validation

# Perform cross-validation and hyperparameter tuning
cv_model = cross_validator.fit(trainingDF)

# Get the best model from cross-validation
best_model = cv_model.bestModel

# Make predictions on the test data using the best model
predictions = best_model.transform(testDF)

# Evaluate the performance of the best model
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data after hyperparameter tuning:", rmse)

# Print the best hyperparameters found
print("Best Rank:", best_model.rank)
print("Best Regularization Parameter:", best_model._java_obj.parent().getRegParam())
```

(9) Spark Jobs

(10) MLflow runs



(9) Spark Jobs

(10) MLflow runs

Logged 10 runs to an experiment in MLflow. Learn more

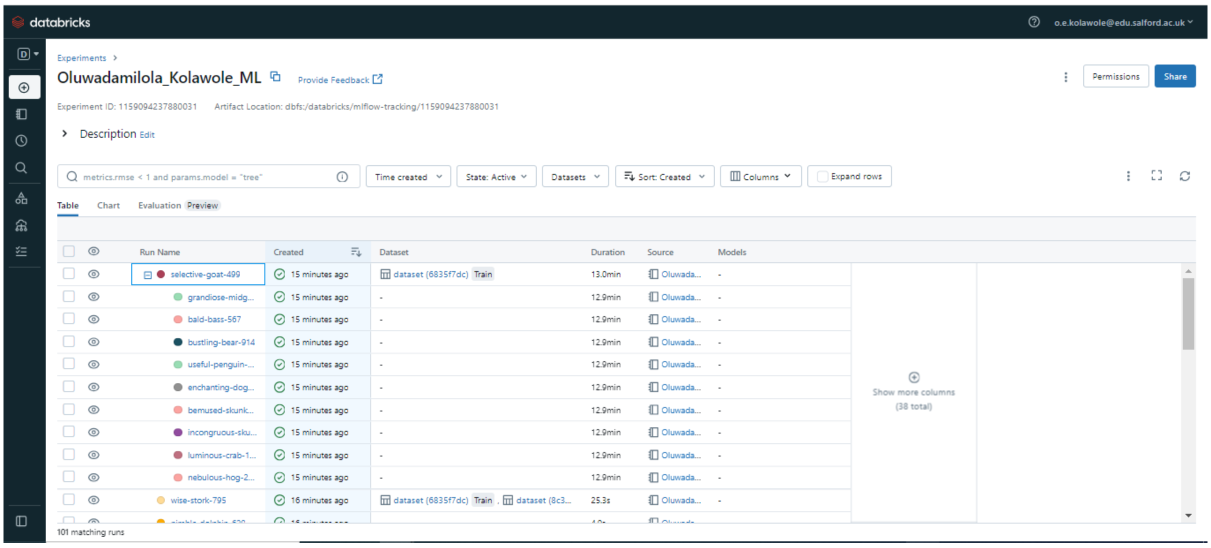
predictions: pyspark.sql.dataframe.DataFrame = [UserId: integer, Game_Name: string ... 4 more fields]

```
2024/04/19 21:03:05 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '16866ca9c46b4944826fc6cf3082975b', which will track hyperparam
eters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/04/19 21:15:31 WARNING mlflow.pyspark.ml: Model CrossValidatorModel_c05f6d2e7a3b will not be autologged because it is not allowlisted or or because one or
more of its nested models are not allowlisted. Call mlflow.spark.log_model() to explicitly log the model, or specify a custom allowlist via the spark.mlflow.py
sparkml.autolog.logModelAllowlistFile Spark conf (see mlflow.pyspark.ml.autolog docs for more info).

Root Mean Squared Error (RMSE) on test data after hyperparameter tuning: 0.07739710217212052
Best Rank: 5
Best Regularization Parameter: 0.01

Command took 12.57 minutes -- by o.e.kolawole@edu.salford.ac.uk at 4/19/2024, 9:59:22 PM on AssessmentML
```

Fig 5.7 Hyper parameter tuning

A lower RMSE indicates better performance, so an RMSE of 0.0774 suggests that the model's predictions are, on average, approximately 0.0774 units away from the actual values. This value could be considered satisfactory depending on the context of your specific problem and the range of the target variable.

Summarily, the tuned ALS model with a rank of 5 and a regularization parameter of 0.01 performs well based on the relatively low RMSE of 0.0774 on the test data. The high level of accuracy shows that the model can accurately predict user preferences or behaviors because its predictions are typically close to the actual values.



Oluwadamilola_Kolawole_ML

Experiment ID: 1159094237880031   Artifact Location: dbfs:/databricks/mlflow-tracking/1159094237880031

metrics.rmse < 1 and params.model = "tree"

Run Name	Created	Dataset	Duration	Source	Models
selective-goat-499	15 minutes ago	dataset (6835f7dc) Train	13.0min	Oluwada...	-
grandiose-midg...	15 minutes ago	-	12.9min	Oluwada...	-
bald-bass-567	15 minutes ago	-	12.9min	Oluwada...	-
bustling-bear-914	15 minutes ago	-	12.9min	Oluwada...	-
useful-penguin-...	15 minutes ago	-	12.9min	Oluwada...	-
enchanting-dog...	15 minutes ago	-	12.9min	Oluwada...	-
bemused-skunk...	15 minutes ago	-	12.9min	Oluwada...	-
incongruous-sku...	15 minutes ago	-	12.9min	Oluwada...	-
luminous-crab-1...	15 minutes ago	-	12.9min	Oluwada...	-
nebulous-hog-2...	15 minutes ago	-	12.9min	Oluwada...	-
wise-stork-795	16 minutes ago	dataset (6835f7dc) Train , dataset (8c3...	25.3s	Oluwada...	-

101 matching runs

Fig 5.8: ML Experiment runs

DISCUSSION

When the ALS model's hyperparameters are tuned to achieve a low RMSE, it shows that the model is doing a good job of predicting user preferences. By providing specific and relevant recommendations, this precise recommendation system can increase user satisfaction and engagement in real-world applications.

When a platform recognizes its users' preferences and recommends content or products that suit their interests, users are more likely to interact with it. You can provide users with a more customized experience by utilizing the tuned ALS model, which could result in higher conversion rates, longer user retention periods, and happier users.

Furthermore, a precise recommendation system can support other corporate goals like raising revenue, boosting metrics for user engagement, and improving the user experience in general. As a result, implementing the tuned ALS model in practical applications could greatly benefit your platform or company.