1.1 DESCRIPTION OF SET UP

In this task, I am required to analyze and train a model for a game recommendation system using the steam_200k data set which contains data about games, users and purchase/play implicit feedback of the users to an associated game.

I will be making use of the Databricks platform to perform analysis and also apply a machine learning algorithm using pyspark's ALS algorithms for recommendation systems.

Our goal is to develop a recommendation system for the games which will be used to recommend games to users using a collaborative filtering approach, by training our models using ALS, using the Game play time as an implicit user feedback of games played.

PREPARATION OF ENVIROMENT (COMPUTE CLUSTER AND NOTEBOOK)

In order to complete this task on Databricks, we need to create a machine learning compute cluster which will be responsible for providing computational power and a notebook which will be the IDE where our codes will be written.

1. CREATING A CLUSTER

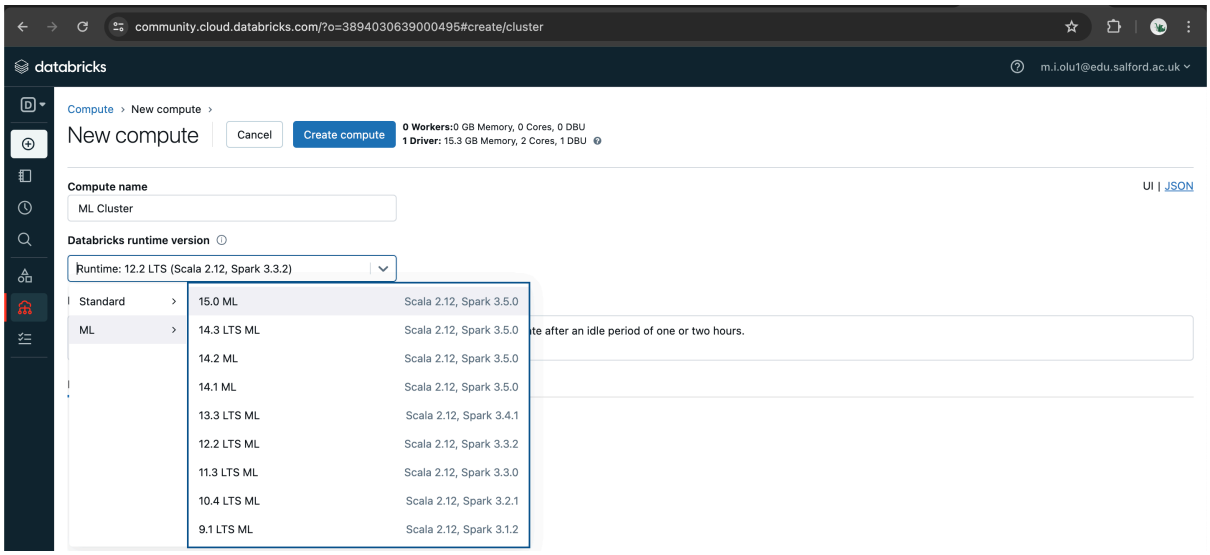


Figure 2.1.1

In figure 2.1.1, I created a machine learning cluster by selecting the ML type when creating the cluster.

In figure 2.1.2 below, I created a notebook to write the codes after creating a cluster in the previous figure 2.1.1.

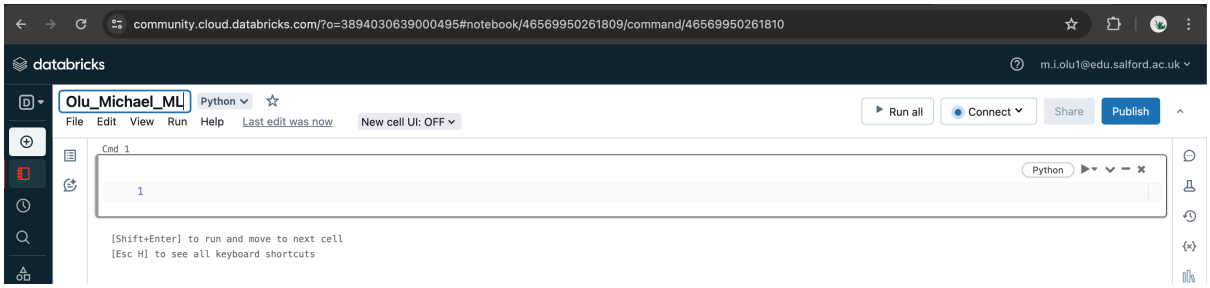


Figure 2.1.2

I created named my notebook Olu_Michael_ML in the figure above where all my codes were written.

2.1 DATA LOADING INTO DATAFRAME

CSV FILE UPLOAD

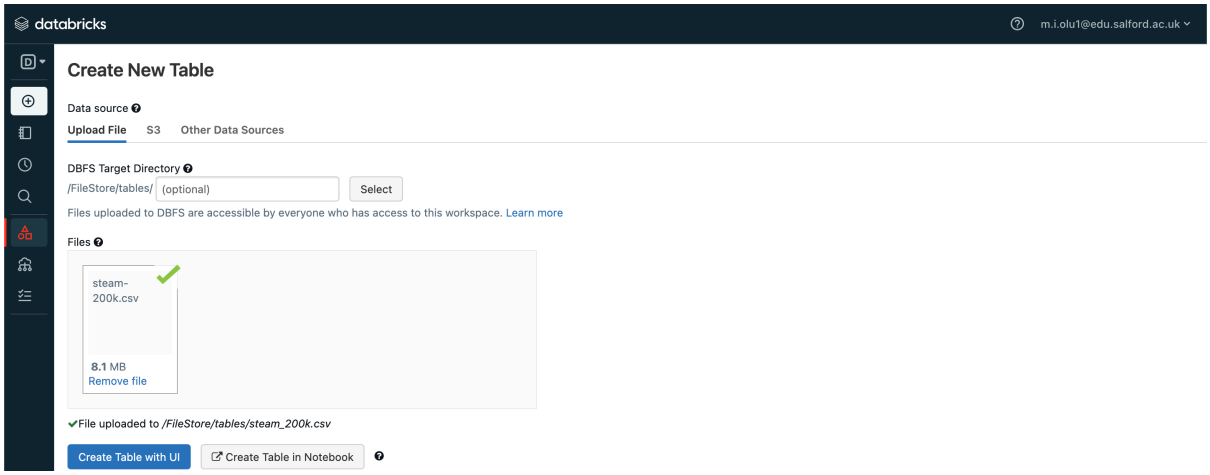


Figure 2.2.1

After our file `steam_200k.csv` has been uploaded, it can be found in the dbfs, the dataframe can now be created from this file.

DATA LOADING INTO PYSPARK DATAFRAME

Before creating the dataframe and running some analysis, I imported the MLflow library and enable autologging.

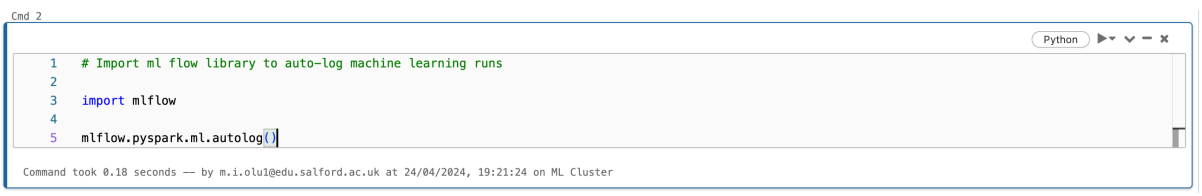


Figure 2.2.2

After initiating autologging, I viewed the contents of the file using dbutils as shown in the figure below.

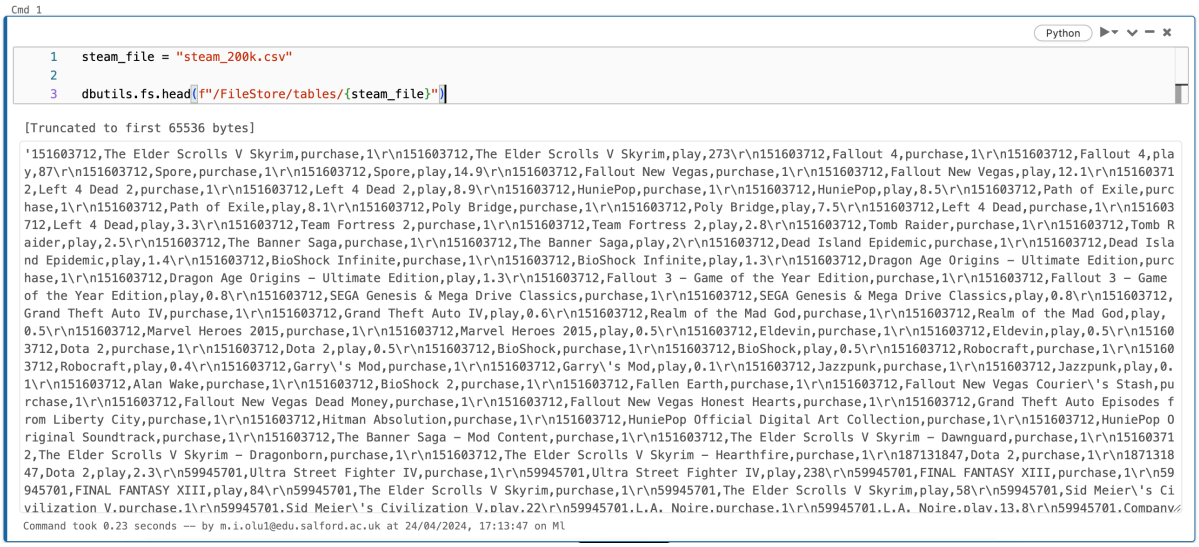


Figure 2.2.3

After viewing file contents and seeing its structure and delimiter, I created the dataframe using pyspark as shown in the figure 2.2.4.

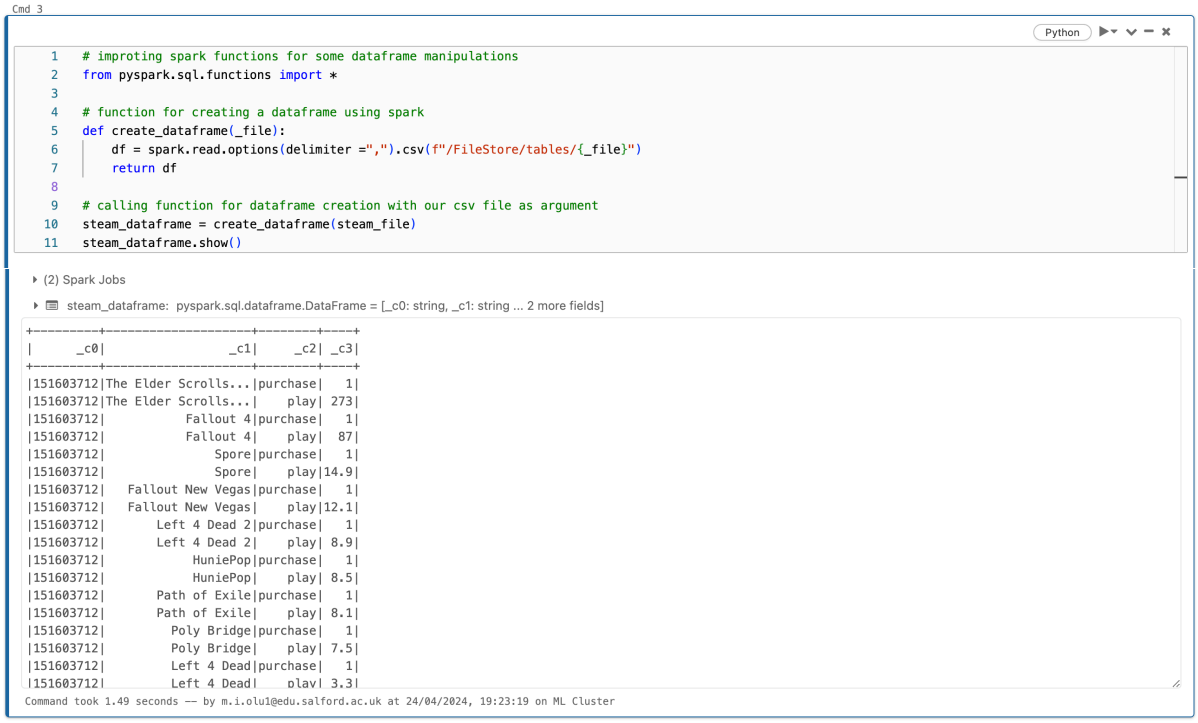


Figure 2.2.4

The dataframe was created and the schema was inferred automatically. But I have to alter this schema and change the column names to more suitable names for each column.

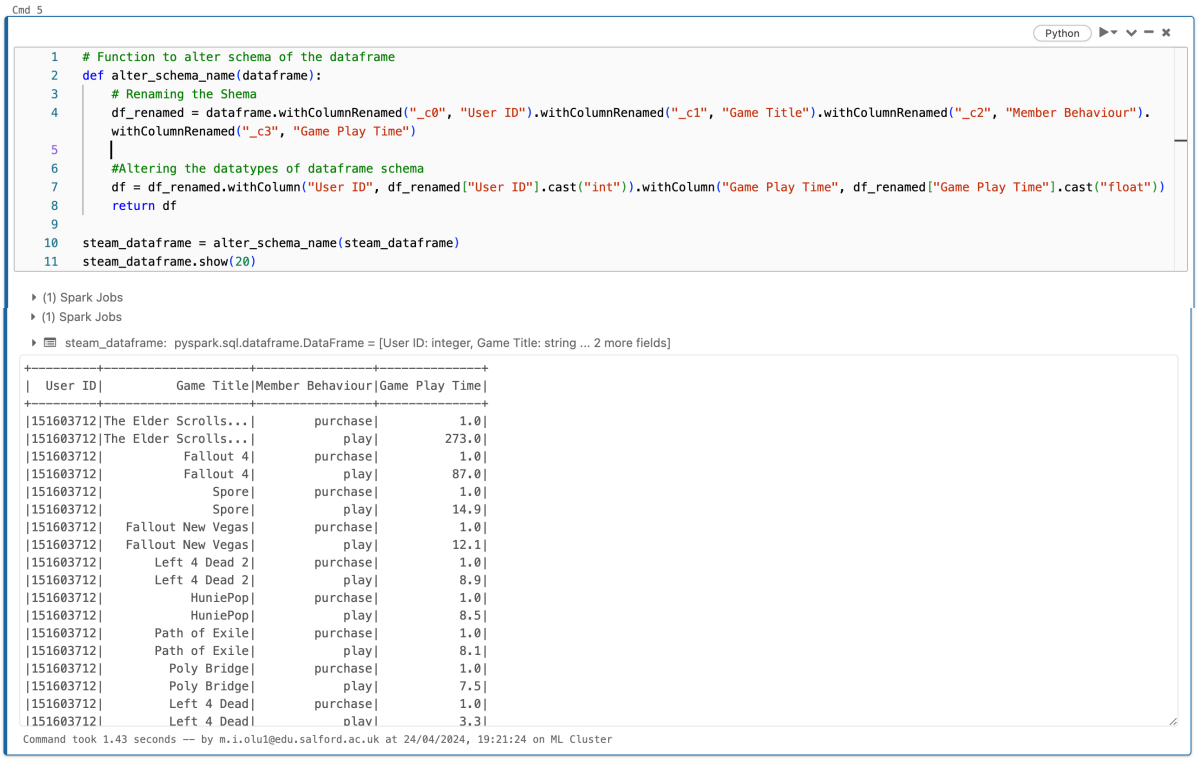


Figure 2.2.5

Figure 2.2.5 shows the schema which have been renamed using the withColumnRenamed methods to alter the dataframe schema, and also withColumn method to change the datatypes some columns.

2.2 EXPLORATORY ANALYSIS AND VISUALIZATIONS

Before training the recommendation system, I did some initial exploration of the dataset.

1. Number of unique records in the dataset

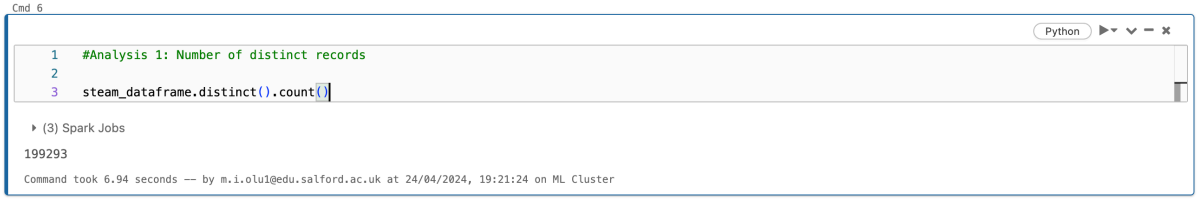


Figure 2.2.6

The result in figure 2.2.6 shows that there are 199,293 unique records in the dataset.

2. Number of unique users

Figure 2.2.7

Counting the User ID distinctly shows that there are 12,393 different users in the dataset.

3. Number of unique games

Figure 2.2.8

Counting the Game ID distinctly shows that there are 5,155 different games in the dataset.

4. Top 10 most popularly played games

Cmd 9

```
1  # #Analysis 4: Top 10 Most popularly played games
2
3  steam_dataframe.select("Game Title", "Member Behaviour", "Game Play Time").filter(steam_dataframe["Member Behaviour"] == "play").groupBy("Game
   Title").count().orderBy('count', ascending=False).limit(10).display()
```

(2) Spark Jobs

Table | Visualization 1

	Game Title	count
1	Dota 2	4841
2	Team Fortress 2	2323
3	Counter-Strike Global Offensive	1377
4	Unturned	1069
5	Left 4 Dead 2	801
6	Counter-Strike Source	715

10 rows | 3.81 seconds runtime — Refreshed 3 minutes ago

Command took 3.81 seconds -- by m.i.olu1@edu.salford.ac.uk at 25/04/2024, 16:21:36 on ML Cluster 2

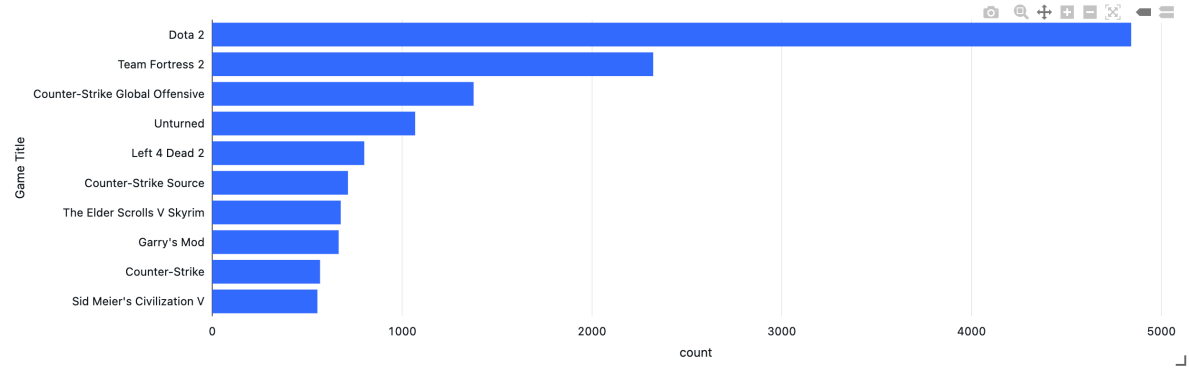
Figure 2.2.9

Figure 2.2.9 shows the 10 most popularly played games based on the number of people who has played the game in descending order with `Dota 2` as the most popular and `Sid Meier's Civilization V` as the 10th most popular. Figure 2.2.10 shows a visualization of this result.

```
3  steam_dataframe.select("Game Title", "Member Behaviour", "Game Play Time").filter(steam_dataframe["Member Behaviour"] == "play").groupBy("Game
   Title").count().orderBy('count', ascending=False).limit(10).display()
```

(2) Spark Jobs

Table | Visualization 1



Edit Visualization | 10 rows — Refreshed 8 minutes ago

Command took 3.81 seconds -- by m.i.olu1@edu.salford.ac.uk at 25/04/2024, 16:21:36 on ML Cluster 2

Figure 2.2.10

5. Top 10 games with the highest cumulative play time in hours

Figure 2.2.11

Figure 2.2.11 displays the top 10 most popular games based on cumulative play time, with Dota 2 having the highest cumulative playtime and Left 4 Dead 2 being the 10th highest, with the results displayed in descending order and visualized in Figure 2.2.12.

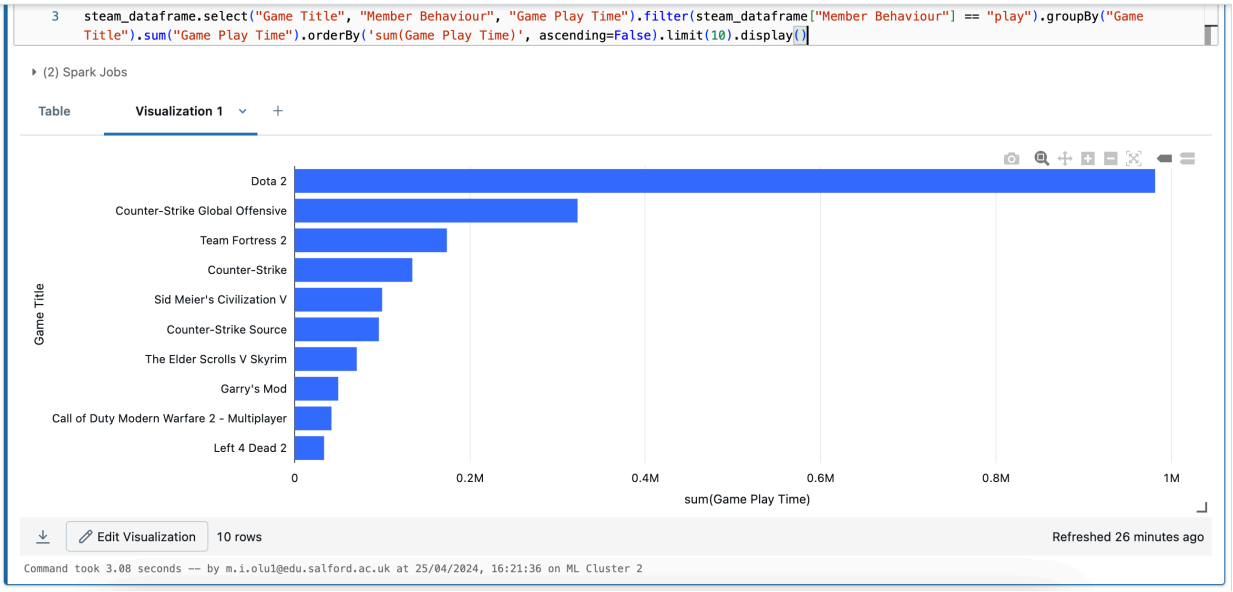


Figure 2.2.12

6. Top 10 most popularly purchased games.

Cmd 11

```
# Analysis 6: Top 10 Most popularly purchased games

steam_dataframe.select("Game Title", "Member Behaviour", "Game Play Time").filter(steam_dataframe["Member Behaviour"] == "purchase").groupBy("Game Title").count().orderBy('count', ascending=False).limit(10).display()
```

(2) Spark Jobs

Table | Visualization 1

	Game Title	count
1	Dota 2	4841
2	Team Fortress 2	2323
3	Unturned	1563
4	Counter-Strike Global Offensive	1412
5	Half-Life 2 Lost Coast	981
6	Counter-Strike Source	978

10 rows | 1.90 seconds runtime — Refreshed 1 hour ago

Command took 1.90 seconds -- by m.i.olu1@edu.salford.ac.uk at 25/04/2024, 16:21:36 on ML Cluster 2

Figure 2.2.13

Figure 2.2.13 shows the 10 most popularly purchased games in descending order. The result shows Dota 2 as the most purchased game with 4841 purchases and `Half-Life 2 Deathmatch` as the 10th with 823 purchases. Figure 2.2.14 shows a visualization of the result.

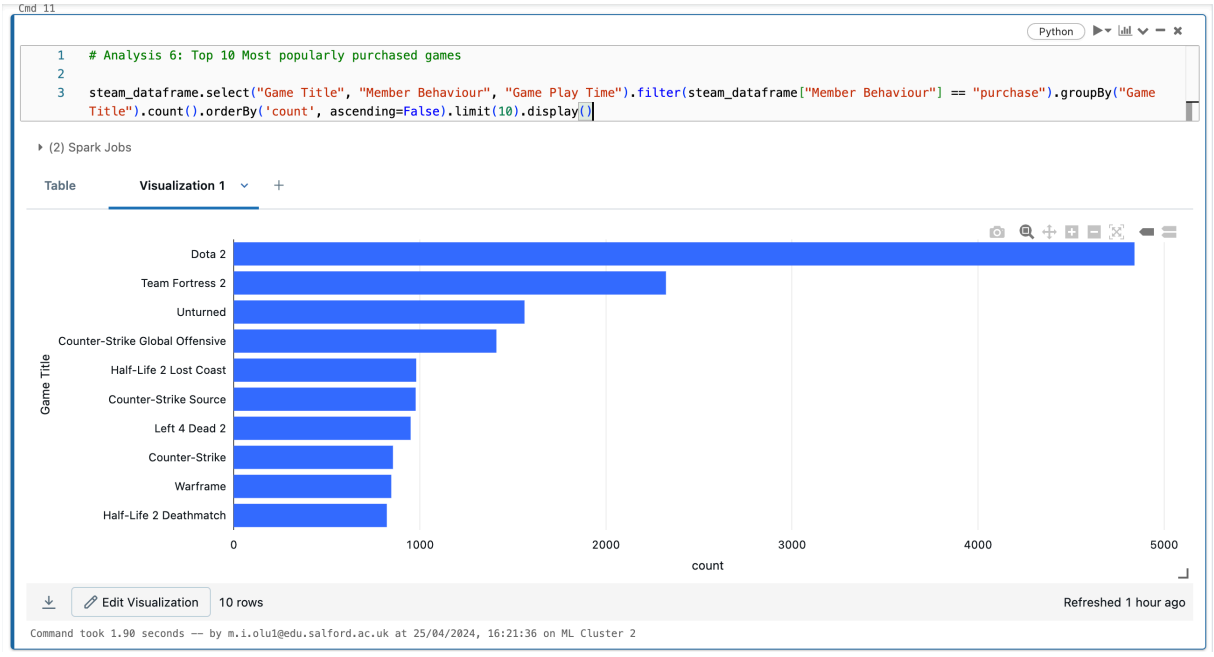


Figure 2.2.14

3 DATA PREPARATION AND PRE-PROCESSING

To train and test a model with ALS, a unique integer ID is required for the item column (Games) and a pre-determined ratio of test and training datasets.

Our original dataset did not have a column for unique IDs for the Games, so I generated Game IDs in a new column and attach this column to our dataset before we split the dataset into the training set and the test sets.

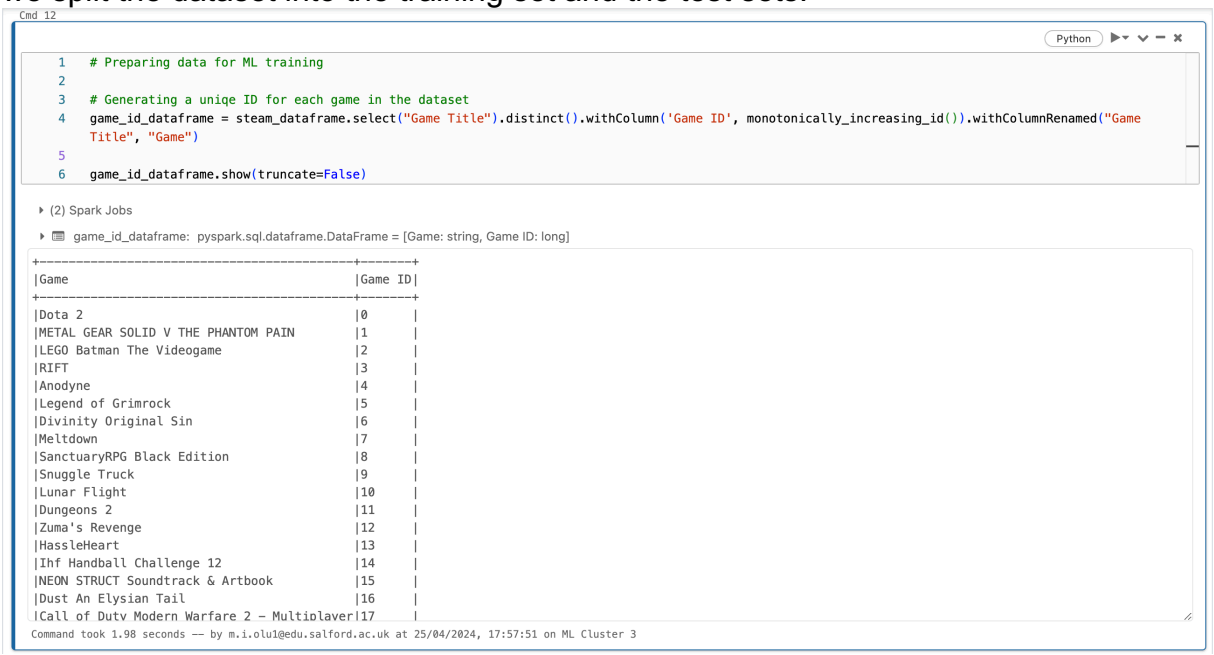


Figure 2.3.1

The unique ID for each game was generated by extracting the Game Title column and applying the spark function `monotonically_increasing_id()`, which assigns a unique serial number to each row of the distinct game dataset.

I joined this new dataframe to the main steam_dataframe with the condition where the game title in the main dataframe is equal to the game title in the game_id_dataframe.

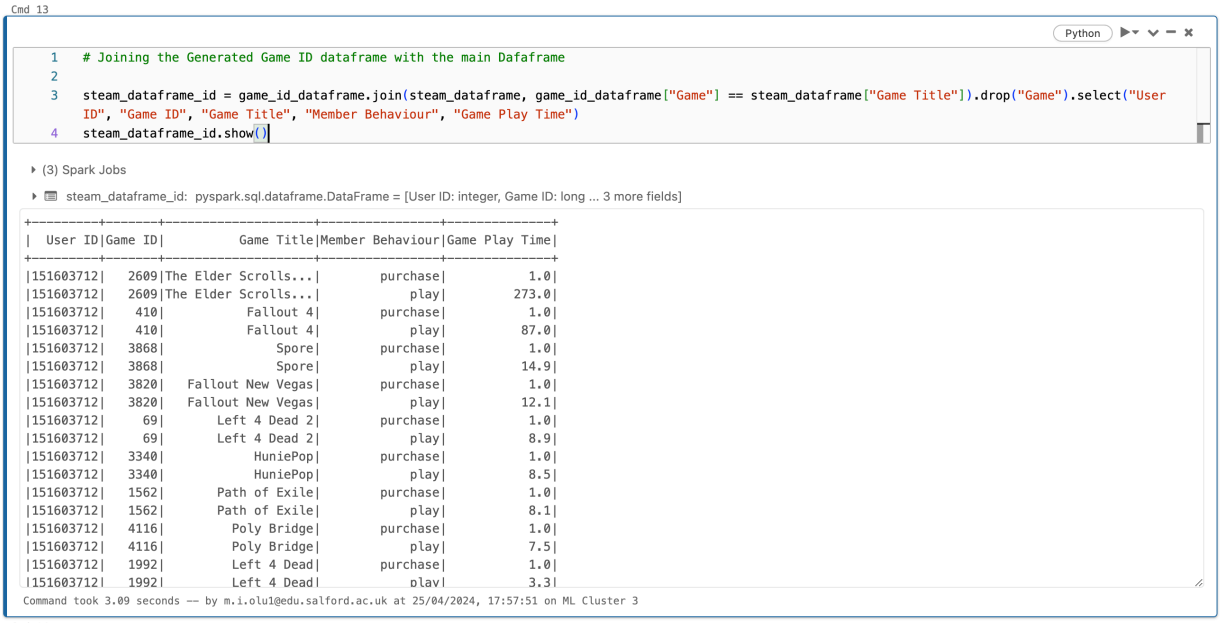


Figure 2.3.2

In the above image, the dataframe has been joined successfully, the next step is splitting the dataframe into the training and the test set.

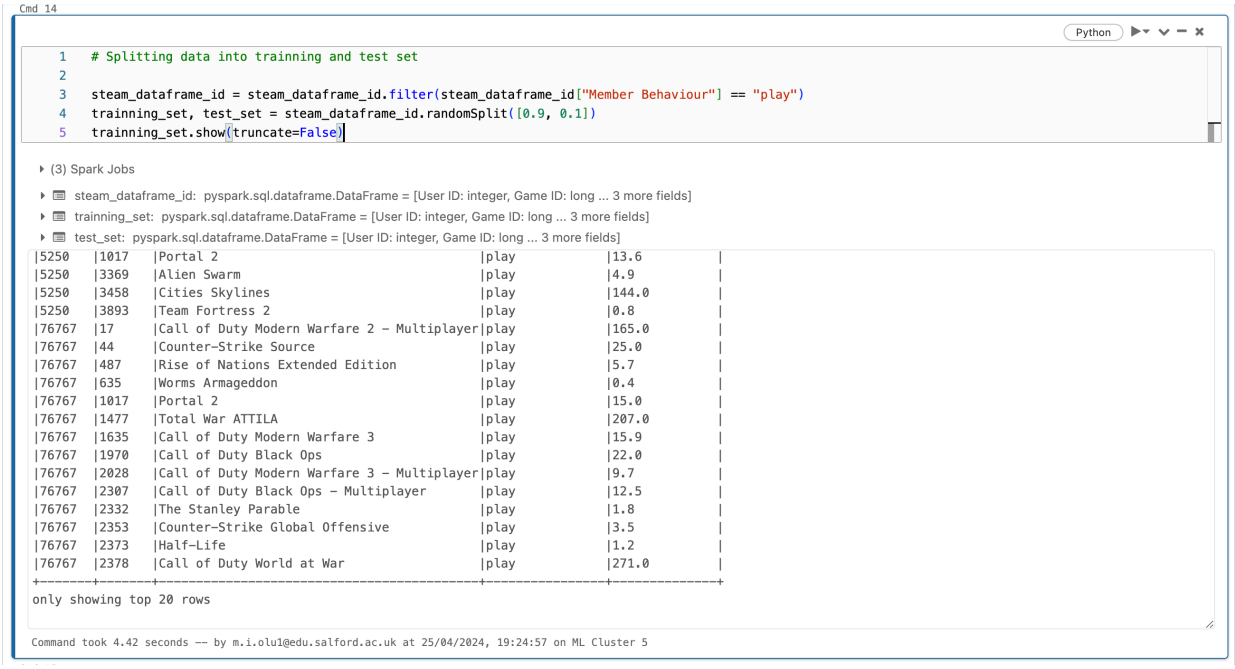
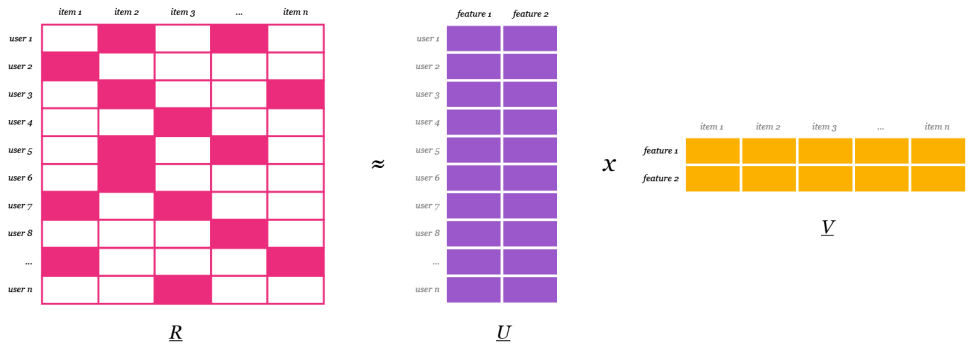


Figure 2.3.3

The dataset is divided into a training and test set with a 0.9:0.1 ratio. The dataset is filtered for rows with implicit user behaviour, 'play', to use as implicit user feedback for training the model. The corresponding values (Play time) will be used for the ratingCol parameter during instantiation.

4. MODEL TRAINING USING ALTERNATING LEAST FACTORS ALGORITHM



The ALS algorithm basically tries to get an estimation of a factorized representation of our original data matrix R with each iteration of the optimization process.

We have our users, items, and some kind of feedback data (in this case, `Game play time`) in our original u x i matrix R. Next, we would like to figure out how to split that into two matrices: one with items and hidden features of size f x i, and another with users and hidden features of size u x f. We've assigned weights to each user/item's relationship to each characteristic in U and V. We compute U and V in such a way that the product of them as nearly resembles R as possible: R ≈ U x V. (Victor, 2017).

I made use of the ALS estimator in the MLib library which uses this factorization and optimization process to train our model. First I imported the ALS class and also the RegressionEvaluator class because we are going to evaluate our model.

I created my own class object which contains various attributes and methods which I will be using to create an object for training, testing and evaluating my model going forward.

The next figure shows where I created the class with the name `GameRecommederModel`. I created this class along with some attributes and methods.

```python
# Training a model using Alternating least squares(ALS) with the trainning set
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Creating a class for recommender system trainning
class GameRecommenderModel:
    def __init__(self, rank, maxIter, regParam):
        # Setting ALS object parameters
        self.rank = rank
        self.maxIter = maxIter
        self.regParam = regParam
        self.userCol = "User ID"
        self.ratingCol = "Game Play Time"
        self.itemCol = "Game ID"
        self.testSet = test_set
        self.trainningSet = trainning_set

        # ALS object instantiator
        self.alsObject = ALS(rank=self.rank, maxIter=self.maxIter, regParam=self.regParam, userCol=self.userCol, ratingCol=self.ratingCol,
        itemCol=self.itemCol, coldStartStrategy="drop", nonnegative=True)
        # RegressionEvaluator object instantiator
        self.evaluator = RegressionEvaluator(metricName="rmse", labelCol=self.ratingCol, predictionCol="prediction")

    def trainModel(self):  # Model fitting method
        return self.alsObject.fit(self.trainningSet)

    def getTestSetPredictions(self, model): # Method to get predictions of test
        return model.transform(self.testSet)

    def getRmse(self, predictions): # Method to get RMSE of the model
        return self.evaluator.evaluate(predictions)
```

Command took 0.05 seconds -- by m.i.olu1@edu.salford.ac.uk at 26/04/2024, 18:43:44 on ML Cluster 7

Figure 2.3.4

The class carries an ALS object and a RegressionEvaluator object inside the init method, and also a, it also carries a trainModel method which will be used to train the model by fitting to the training set, a getTestSetPredictions which will be used to generate predictions for the test set, and finally a getRmse method for evaluating our model.

I made use of classes to create my ALS and Evaluator objects because of code reusability. So that each time I want to train a new model, I can simply create the objects from the class and easily access various characteristics (such as rmse and test predictions dataset) of a model all from one object.

CREATING THE ALS MODEL INSTANCE

Cmd 20

```python
rank, maxIter, regParam = (15, 15, 0.2) # Defining values for ALS estimator hyperparameters

gameModel = GameRecommenderModel(rank, maxIter, regParam) # Creating object (ALS instance) with hyper parameters
trainnedGameModel = gameModel.trainModel() # Calling method for trainning model
```

▸ (9) Spark Jobs

▾ (1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

```
2024/05/02 04:05:40 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID 'f6e62f4dc9b94f7e8af627421047830b', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/05/02 04:05:54 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/databricks/python/lib/python3.11/site-packages/mlflow/types/utils.py:393: UserWarning: Hint: Inferred schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference time, 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 (training 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."
2024/05/02 04:07:10 WARNING mlflow.pyspark.ml: Model ALS_c52659ad873c 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.pysparkml.autolog.logModelAllowlistFile Spark conf (see mlflow.pyspark.ml.autolog docs for more info).
```

Command took 1.55 minutes -- by m.i.olu1@edu.salford.ac.uk at 02/05/2024, 04:51:44 on My Cluster 7

Figure 2.3.5

The ALS model instance was created using initial values of 15, 15, and 0.2 for the first experiment run, and trained using the trainModel() method from the object

created. The output indicates that the training session has been logged in the experiment UI.

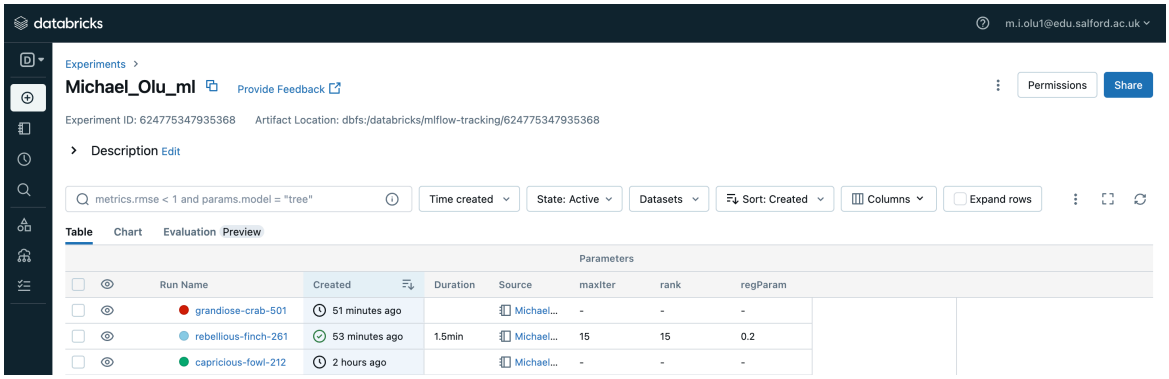


Figure 2.3.6

We will use this UI as tool to evaluate and compare our model, trained with different hyper parameters for subsequent runs towards a goal to obtain the best model from our dataset.

VIEWING PREDICTIONS FOR TEST SET

Cmd 17

```
#  Calling the getTestSetPredictions method to view predictions made from trainned model
predictions = gameModel.getTestSetPredictions(trainnedGameModel)
predictions.display()
```

(2) Spark Jobs

predictions: pyspark.sql.dataframe.DataFrame = [User ID: integer, Game ID: integer ... 4 more fields]

Table

	User ID	Game ID	Game Title	Member Behaviour	Game Play Time	prediction
1	76767	1017	Portal 2	play	15	16.148693
2	76767	3917	Banished	play	24	28.640377
3	298950	653	Dragon Age Origins - Ultimate Edition	play	0.2	3.4520493
4	298950	962	Sid Meier's Civilization V	play	135	166.31955
5	298950	1329	Shadow Warrior	play	0.2	7.2455206
6	298950	1918	The Wolf Among Us	play	8.2	3.8144841

6,209 rows | 15.02 seconds runtime

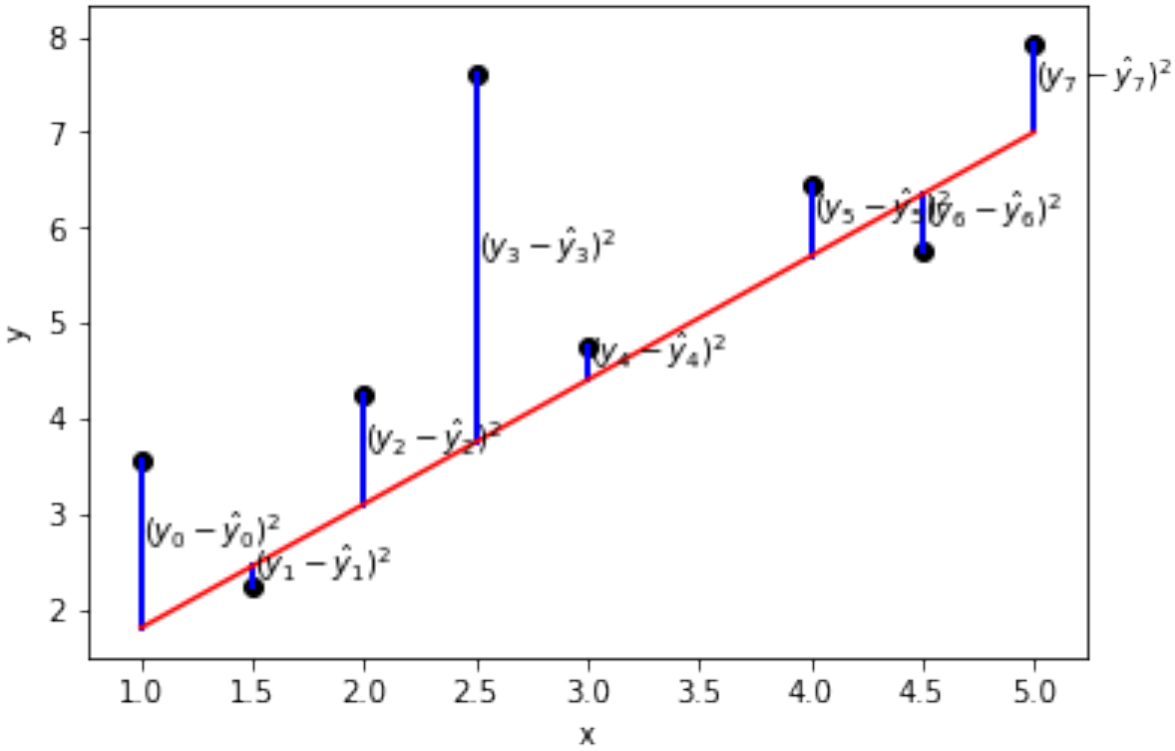
Refreshed 15 hours ago

Command took 15.02 seconds -- by m.i.olu1@edu.salford.ac.uk at 26/04/2024, 18:05:41 on ML Cluster 7

Figure 2.3.7

Figure 2.3.7 illustrates the model's predictions vs actual values. To assess the model's performance, we evaluate its RMSE and train more models with different hyper parameters. This process, known as hyper parameter tuning, aims to identify the model with the best performance by generating the lowest RMSE.

The RMSE (root mean squared error) is the root value of the average of the squared sum of differences between the predictions and the actual values.



According to the figure above, the root mean squared error can be calculated by the equation:

$$\textbf{RMSE} = \sqrt{\frac{(y_0 - \hat{y_0})^2 + (y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2 + \cdots + (y_n - \hat{y_n})^2}{n}}$$

Where y_n is the actual implicit feedback value, $\hat{y_n}$ is the predicted value and n is the total number of items in the dataset.

The RMSE of the predictions in the dataset in figure 2.3.8 is computed by the RegressionEvaluator object and shown below.

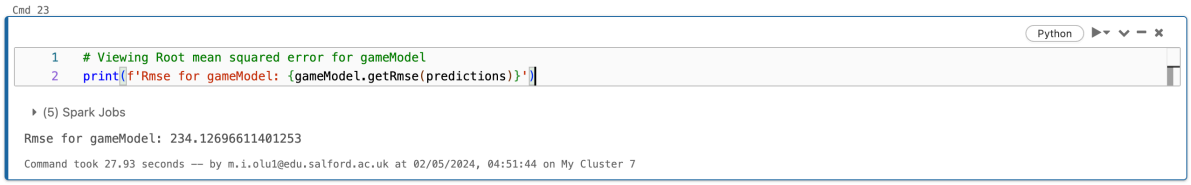


Figure 2.3.8

The result above shows the rmse of our first model as ~234.126966

HYPER PARAMETER TUNING

The hyper parameter tuning for my models was done using the `ParamGridBuilder` library from the pyspark ml library. This tool uses a pre-defined range of hyper parameter values and performs multiple runs with different combinations to achieve the best performing models.

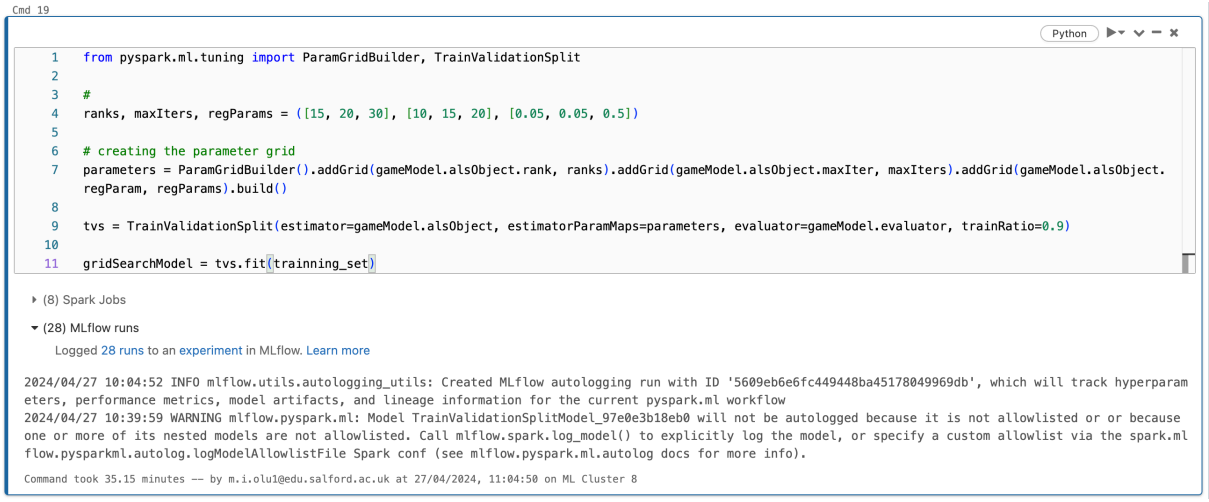


Figure 2.3.9

I defined each of the parameters as a list of values: 15, 20 and 30 for the rank, 10, 15 and 20 for the maxIters and 0.005, 0.05 and 0.5 for the regParams.

Figure 2.3.10 shows the predictions made from the best model after running multiple times with the ParamGridBuilder.

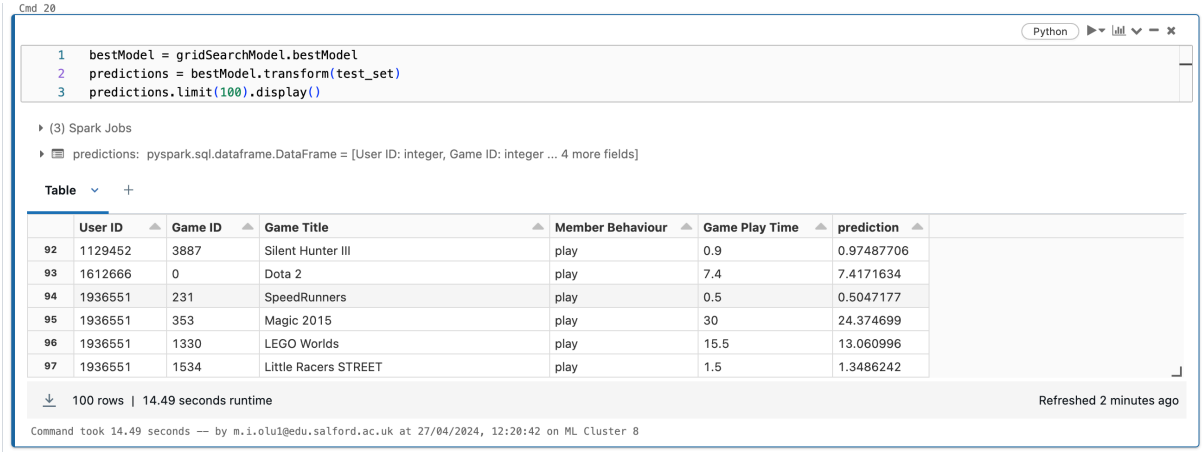


Figure 2.3.10

The result shows that the model has improved its predictions, having the predictions closer to the Game Play Time.

We can also use some attributes of the bestModel to extract the hyper parameters which was used to train that model.

Figure 2.3.11

The figure above shows the parameters has been extracted and assigned to new variables and printed out to the console. It gave 30 for the Rank value, 0.5 for the regParam and 20 for the maxIter.

We can now use these hyper parameter values to train our own model and check for the rmse as shown below.

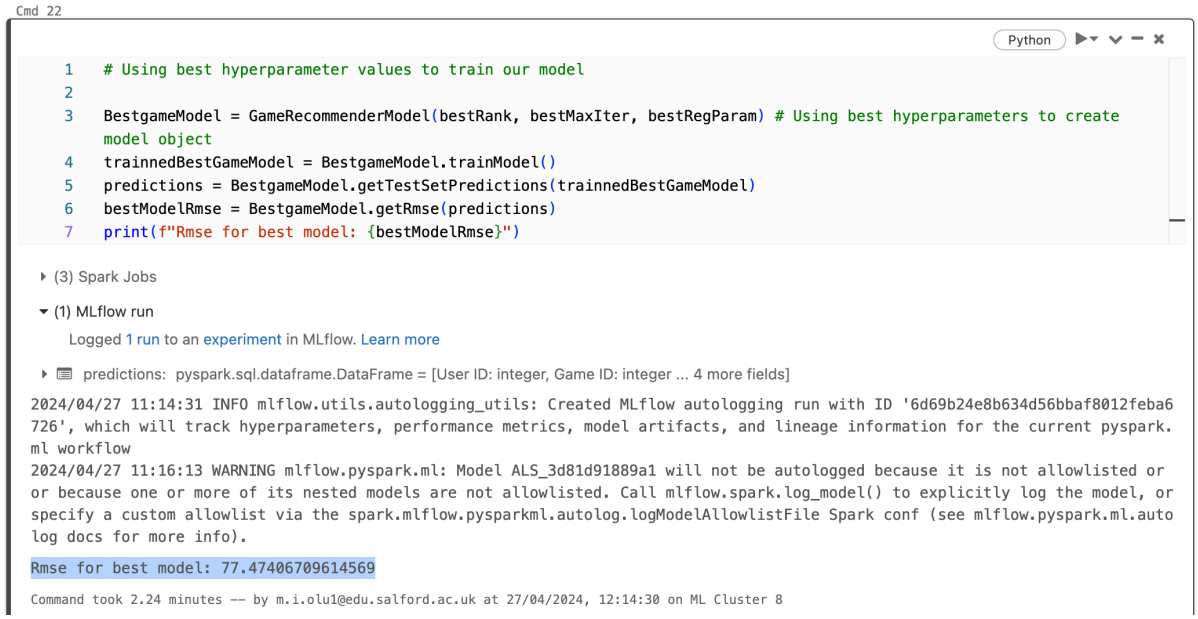


Figure 2.3.12

Getting the rmse of the Best model shows the rmse as 77.474, which is more than 3 times better than the first model which was ran with random hyper parameter values.

The multiple runs made by the ParamGridBuilder were logged and we can view this from the experiment UI. We can also compare different hyper parameters with their corresponding rmse values.

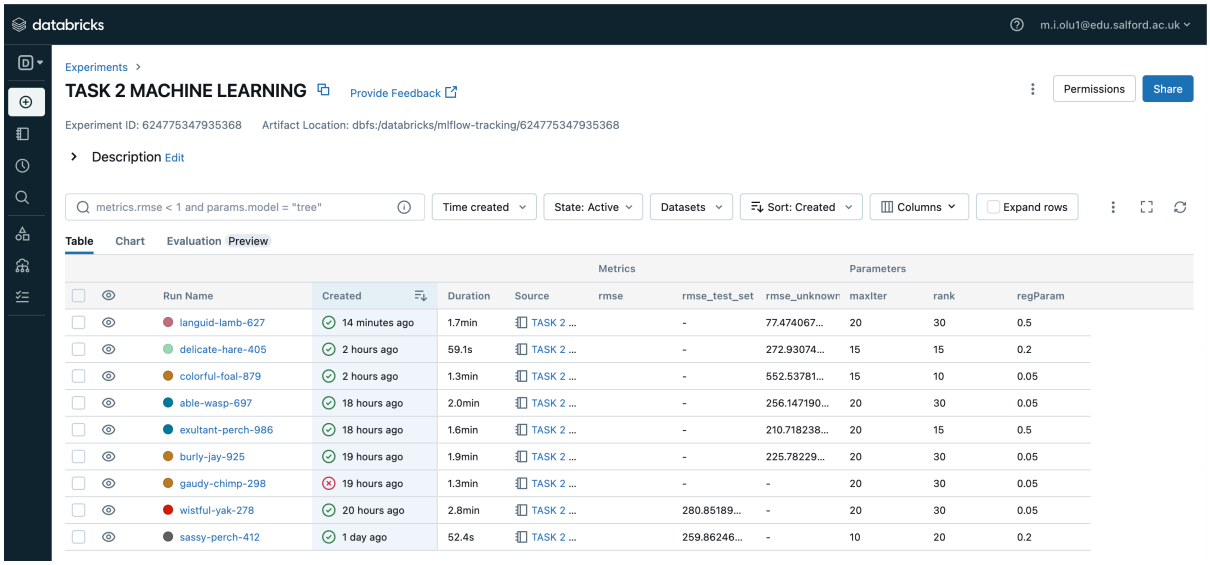


Figure 2.3.13

Each row in the table corresponds to each model ran. It shows that the gridSearchModel ran multiple times using different maxIter, rank and regParam values. The lowest rmse appears to be the model with 77.47067, with its corresponding hyper parameters as shown, which are same values which we extracted from the bestModel in the notebook.

RECOMMENDING GAMES FOR USERS

The model can be used to create a recommendation system using the recommendForAllUsers method on our best model. The parameter 10 is passed to extract the top 10 game recommendations based on predicted implicit behaviour and game play time for each user in the dataset, labelled as 'rating' in each dictionary.

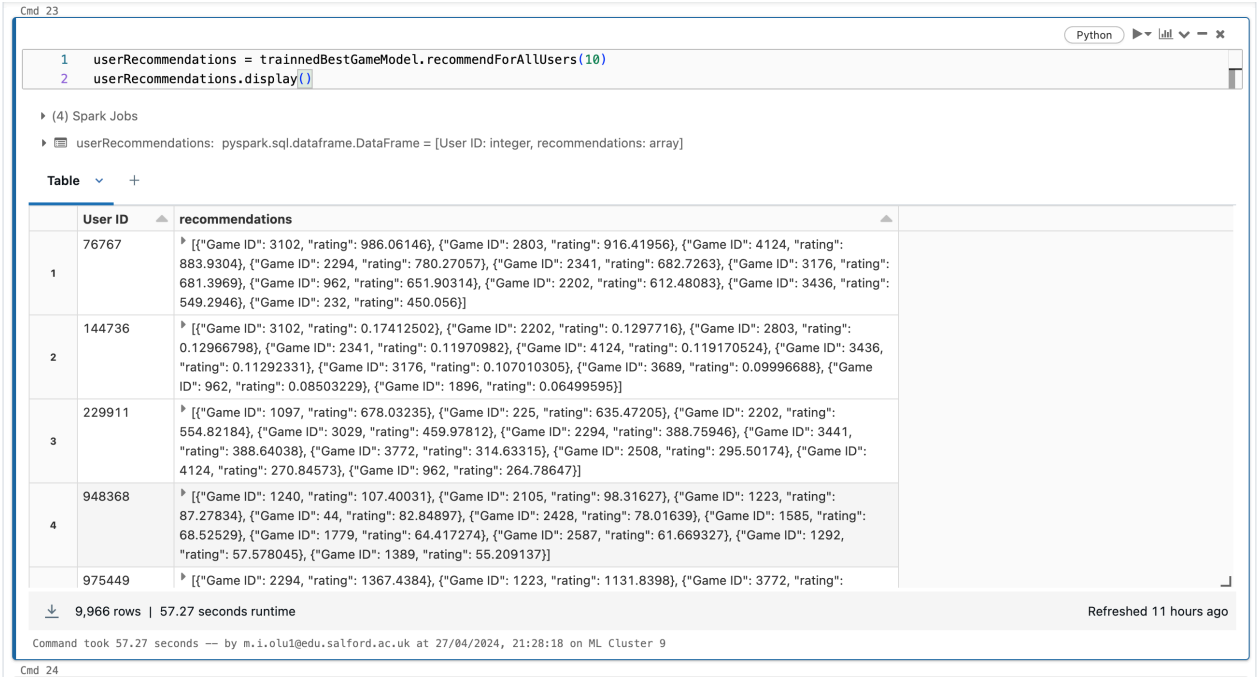


Figure 2.3.13

I wrote a query to extract the recommendations for a particular user using their User ID and also matched the Game ID to the game Titles by joining the extracted user row from the recommendation table to the game_id_dataframe which I created earlier on.

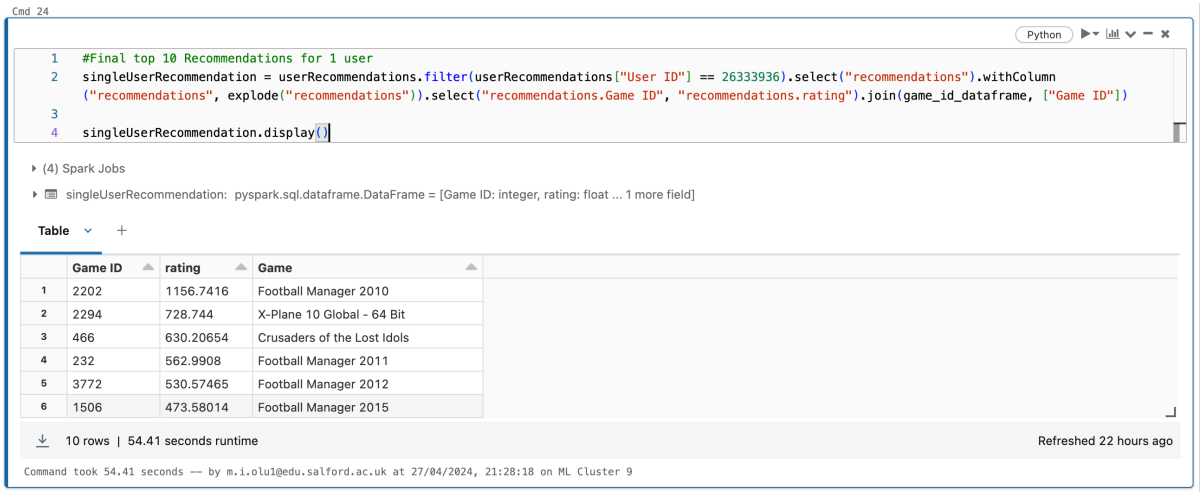


Figure 2.3.14

From the results in figure 2.3.14 above, we can see that same types of games were recommended for user `26333936`. It can be deuced that this user top choice of games is Football Manager for various years.

5. DISCUSSION OF RESULTS.

The model's initial training yielded an RMSE of ~234.126966, with hyper parameters of 15 for rank, 15 for maxIter, and 0.2 for regParam. After multiple experiments using ParamGridBuilder, the best RMSE was ~77.5, with parameters of 30, 20, and 0.5 for rank, maxIter, and regParam. This best model was used to create recommendations for all users, and further filtered to check the recommendations for one particular user and revealed this user's top preference to be the `Football Manager` game.

Bibliography

Victor. (2017, August 23). *ALS Implicit Collaborative Filtering*. Retrieved from medium: https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe