

Assessment

Big Data Tools and Techniques

M.Sc Data Science



University of
Salford
MANCHESTER

**Task 1: Clinical Trial Data Analysis in
Databricks with DataFrame, RDD, and Spark
SQL Approaches**

**Task 2: Building a Collaborative Filtering
Recommender System for Users based on
'Purchase' and 'Play' Behavior**

Name : Remya Vellakkadaparambu Karthikeyan

Student ID: 00715519

Date: 2nd May 2024

Contents

Task 1: Clinical Trial Data Analysis in Databricks with DataFrame, RDD, and Spark SQL Approaches	5
1.1 Introduction.....	5
1.2 Set up Requirements	6
1.3 Building Rerunnable code.....	7
1.4 Unzipping Data inside Databricks.....	8
1.5 Data cleaning and preparation.....	9
1.6 The number of studies in the dataset (checking distinct studies explicitly) (All codes run a function in reusable notebook)	15
1.7 List all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type	21
1.8 The top 5 conditions (from Conditions) with their frequencies.....	25
1.9 Find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.	34
1.10 Plot number of completed studies for each month in 2023	40
1.11 Additional Analyses and Visualization in Dataframe Approach	47
1.12 Additional Analyses and Visualization in RDD Approach	53
1.13 Additional Analyses and Visualization in Spark SQL Approach	57
1.14 Visualization of Clinical Trial 2023 in Power BI	61
1.15 Analyses of historical dataset Clinicaltrial_2020 and Clinicaltrial_2021 ..	63
1.15.1 The number of studies in the dataset, checking distinct studies explicitly.....	64
1.15.2 List all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type.	64
1.15.3 The top 5 conditions (from Conditions) with their frequencies.....	66
1.15.4 Find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored. ..	67
1.15.5 Plot number of completed studies for each month in 2023. Include visualization as well as a table of all the values.	67

1.16 Additional Analysis and Visualization for historical dataset (codes - cmd 49 to 53 in the RDD notebook)	70
1.17 Conclusion.....	71
Task 2: Building a Collaborative Filtering Recommender System for Users based on ' <i>Purchase</i> ' and ' <i>Play</i> ' Behavior.....	73
2.1 Introduction	73
2.2 Set up Requirements	74
2.3 Data Management.....	76
2.3 Data Preprocessing and Preparation Before Training.....	77
2.3.1 Data Pre-processing	77
2.3.2 Data Preparation before Training.....	78
2.4 Exploratory Data Analysis on Dataframes with and without Game IDs.....	80
2.5 Visualization on Dataframes using Python Matplotlib, and Plotly	85
2.6 Visualization on Dataframes using Power BI	89
2.7 Collaborative Filtering Recommender System based on User Behavior ' <i>Purchase</i> '	93
2.8 Collaborative Filtering Recommender System based on ' <i>Play</i> ' User Behavior.....	101
2.9 Collaborative Filtering Recommender System based on User Behavior ' <i>Purchase</i> ' and ' <i>Play</i> '	108
2.10 Selection of Hyperparameters in Recommender System based on User Behavior.....	115
2.10.1 Selection of Hyperparameters in Recommender System based on ' <i>Purchase</i> ' User Behavior.....	116
2.10.2 Selection of Hyperparameters in Recommender System based on ' <i>Play</i> ' User Behavior	117
2.10.3 Selection of Hyperparameters in Recommender System based on ' <i>Purchase</i> ' and ' <i>Play</i> ' User Behavior	117
2.11 Model Training and Evaluation.....	118
2.11.1 Model training and Evaluation of Recommender System based on ' <i>Purchase</i> ' Behavior	118

2.11.2 Model training and Evaluation of Recommender System based on ' <i>Play</i> ' Behavior	120
2.11.3 Model training and Evaluation of Recommender System based on ' <i>Purchase</i> ' and ' <i>Play</i> ' Behavior.....	121
2.12 MLflow Experiment Tracking	122
2.13 Recommender System based on ' <i>Purchase</i> ' Behavior -Prediction Visualization	123
2.14 Recommender System based on ' <i>Play</i> ' Behavior -Prediction Visualization	125
2.15 Recommender System based on ' <i>Purchase</i> ' and ' <i>Play</i> ' Behavior - Prediction Visualization	126
2.16 Results and Discussion	128
2.17 Conclusions.....	128

N.B: some visualization deleted from codes for ease of exporting the files

Task 1: Clinical Trial Data Analysis in Databricks with DataFrame, RDD, and Spark SQL Approaches

1.1 Introduction

Clinical trials are an essential part of the medical and pharmaceutical research process, offering valuable insights into the effectiveness, safety, and side effects of treatments and interventions. Analysing clinical trial data can provide important information for researchers, healthcare professionals, and policymakers.

The **main objective** of the task is to analyse clinical trial data using three different approaches within Databricks: and PySpark DataFrame, PySpark RDD and Spark SQL. Each approach offers distinct capabilities for data manipulation, analysis, and insights. The order of approach is chosen as follows.

- 1) **PySpark DataFrame** - High-level API, facilitating efficient and straightforward data analysis.
- 2) **PySpark RDD** - a low-level API that allows for flexible data manipulation and processing.
- 3) **Spark SQL** - provides a way to interact with data using familiar SQL queries.

The most significant questions related to clinical trials, such as the number of distinct studies in the dataset, study types and frequencies, common conditions and sponsors, and the monthly completion rate of studies etc along with additional analyses and visualizations are addressed in this report. For visualization, Python's matplotlib, plotly and Power BI are utilized to communicate the insights efficiently.

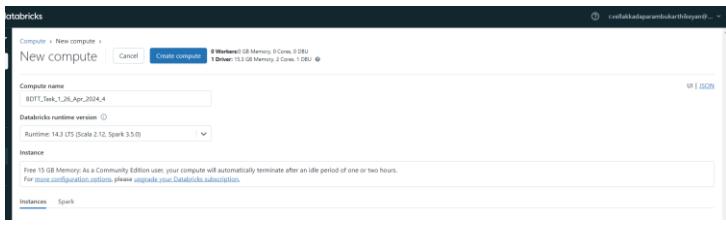
1.2 Set up Requirements

The following points and figures give the setup requirements for the task implementation in Databricks.

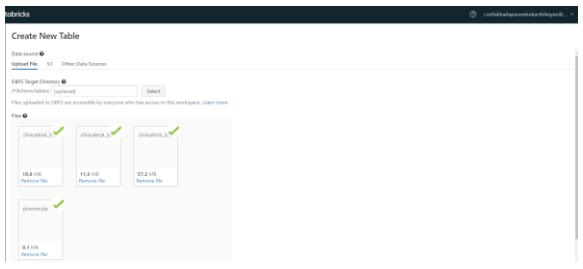
After logging into the Databricks workspace perform the following:

- Create a new cluster or use an existing cluster with necessary resources (CPU, memory etc) and with appropriate runtime version for the type of work you want to perform.
- Upload or import your data to the Databricks environment.
- Create a notebook where you will write your code.

Create a new cluster



Upload or import data



Create a notebook



Figure 1.1 Set up requirements

1.3 Building Rerunnable code

Figure 1.2 code snippet sets up a reusable framework for working with clinical trial data from the year 2023 in a Databricks notebook. By parameterizing the file root and dynamically extracting the trial year, the code makes it rerunnable and reusable. The code uses Databricks utilities (**'dbutils'**) to manage the movement of files between (DBFS) to the local /tmp directory on the driver node. This setup allows the notebook to rerun smoothly by providing a fresh start each time, minimizing file handling errors, and ensuring that the correct data is in place for the analysis to proceed. Similar procedure done for `pharma.zip`, `clinicaltrial_2020.zip` and `clinicaltrial_2021.zip` files. The instructions for smooth running of the notebooks are included as markdown text in the start of each file.

```
Remya_Vellakkadaparambu_Karthikeyan_df Python v
File Edit View Run Help Last edit was 5 minutes ago New cell Ut OFF ▾
Run all Unknown Share Publish ▾
Cmd 1
Upload clinicaltrial_2023.zip and pharma.zip to DBFS File system and then run the codes
Cmd 2
1 #Defining reusable codes for Clinical Trial 2023
2 fileroot = "clinicaltrial_2023"
Command took 0.07 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:32 AM on BDTT_Task_1_25_Apr_2024_2
Cmd 3
1 #Extracting the year of clinical trial
2 fileName = fileroot.split("_")
3 trial_Year = fileName[1]
4 trial_Year
'2023'
Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:32 AM on BDTT_Task_1_25_Apr_2024_2
Cmd 4
1 #Copying the file to /tmp directory on driver node
2 dbutils.fs.cp("/fileStore/tables/" + fileroot + ".zip", "file:///tmp/")
True

```



```
Remya_Vellakkadaparambu_Karthikeyan_df Python v
File Edit View Run Help Last edit was 5 minutes ago New cell Ut OFF ▾
Run all Unknown Share Publish ▾
1 #Checking the fileroot.zip in DBFS directory
2 dbutils.fs.ls("/fileStore/tables/")
[FileInfo(path="dbfs:/FileStore/tables/Occupancy_Detection_Data.csv", name='Occupancy_Detection_Data.csv', size=50968, modificationTime=1709126397000),
 FileInfo(path="dbfs:/FileStore/tables/15021_2021_2.csv", name='15021_2021_2.csv', size=497239, modificationTime=1710956905000),
 FileInfo(path="dbfs:/FileStore/tables/account-models/", name='account-models/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/accounts/", name='accounts/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/accounts-1.zip", name='accounts-1.zip', size=529792, modificationTime=1706709316000),
 FileInfo(path="dbfs:/FileStore/tables/accounts-2.zip", name='accounts-2.zip', size=529792, modificationTime=1706709374000),
 FileInfo(path="dbfs:/FileStore/tables/activations/", name='activations/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/activations/activation1.zip", name='activation1.zip', size=8411369, modificationTime=1706706901000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2020/", name='clinicaltrial_2020/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2020-1.zip", name='clinicaltrial_2020-1.zip', size=10599182, modificationTime=1713878811000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2020-2.zip", name='clinicaltrial_2020-2.zip', size=10599182, modificationTime=1713883665000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2021/", name='clinicaltrial_2021/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2021-1.zip", name='clinicaltrial_2021-1.zip', size=1508457, modificationTime=171387858000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2021-2.zip", name='clinicaltrial_2021-2.zip', size=1508457, modificationTime=171387844000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2021-3.zip", name='clinicaltrial_2021-3.zip', size=1508457, modificationTime=1713879991000),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2023/", name='clinicaltrial_2023/', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/clinicaltrial_2023-1.zip", name='clinicaltrial_2023-1.zip', size=57166668, modificationTime=1714087810000),
 FileInfo(path="dbfs:/FileStore/tables/ethnic.csv", name='ethnic.csv', size=0, modificationTime=0),
 FileInfo(path="dbfs:/FileStore/tables/flood.csv", name='flood.csv', size=128984, modificationTime=1707921250000),
 FileInfo(path="dbfs:/FileStore/tables/flood.zip", name='flood.zip', size=52053, modificationTime=1707920914000),
Command took 0.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:33 AM on BDTT_Task_1_25_Apr_2024_2

```

```

Cmd 6
1 #Checking the fileroot.zip in local tmp directory
2 dbutils.fs.ls("/file:/tmp/")

FileInfo(path='file:/tmp/systemd-private-77be7a91095a46e48d056cbfa784384a-systemd-logind.service-mk21cc', name='systemd-private-77be7a91095a46e48d056cbfa784384a-systemd-logind.serv
ice-mk21cc', size=4096, modificationTime=1714080566628),
FileInfo(path='file:/tmp/chaffeur-env.sh', name='chaffeur-env.sh', size=156, modificationTime=1714080571984),
FileInfo(path='file:/tmp/chaffeur-daemon.pid', name='chaffeur-daemon.pid', size=4, modificationTime=1714080574152),
FileInfo(path='file:/tmp/custom-spark.conf', name='custom-spark.conf', size=783, modificationTime=1714080572048),
FileInfo(path='file:/tmp/X11-unix', name='X11-unix', size=4096, modificationTime=1714080566576),
FileInfo(path='file:/tmp/driver-env.sh', name='driver-env.sh', size=3459, modificationTime=1714080597348),
FileInfo(path='file:/tmp/.font-unix', name='.font-unix', size=4096, modificationTime=1714080566576),
FileInfo(path='file:/tmp/.java_pid457', name='java_pid457', size=0, modificationTime=1714080663436),
FileInfo(path='file:/tmp/tmp.snlantkpp', name='tmp.snlantkpp', size=0, modificationTime=1714080574116),
FileInfo(path='file:/tmp/python_ls_logs', name='python_ls_logs', size=4096, modificationTime=17140805133579),
FileInfo(path='file:/tmp/clinicaltrial_2023.zip', name='clinicaltrial_2023.zip', size=57106668, modificationTime=1714087353709),
FileInfo(path='file:/tmp/Rserve', name='Rserve', size=4096, modificationTime=1714080701232),
FileInfo(path='file:/tmp/systemd-private-77be7a91095a46e48d056cbfa784384a-systemd-resolved.service-bofie9', name='systemd-private-77be7a91095a46e48d056cbfa784384a-systemd-resolved.
service-bofie9', size=4096, modificationTime=1714080566576),
FileInfo(path='file:/tmp/driver-daemon-params', name='driver-daemon-params', size=19, modificationTime=1714080598956),
FileInfo(path='file:/tmp/RTmp08Fj3J3', name='RTmp08Fj3J3', size=4096, modificationTime=1714080977147),
FileInfo(path='file:/tmp/driver-daemon.pid', name='driver-daemon.pid', size=4, modificationTime=1714080599092),
FileInfo(path='file:/tmp/.X11-unix', name='.X11-unix', size=4096, modificationTime=1714080566576),
FileInfo(path='file:/tmp/pharma.zip', name='pharma.zip', size=109982, modificationTime=1714085629780)
Command took 0.09 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:33 AM on DOTT_Task_1_25_Apr_2024_2

Cmd 7
1 #Making 'fileroot' accessible by the command line
2 import os
3 os.environ['fileroot'] = fileroot

Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:33 AM on DOTT_Task_1_25_Apr_2024_2

```

Figure 1.2 Building reusable and rerunnable codes by parameterizing fileroot

1.4 Unzipping Data inside Databricks

Figure 1.3 code snippet demonstrates how to handle a zipped clinical trial data file within Databricks. Similar procedure done for pharma.zip, clinicaltrial_2020.zip and clinicaltrial_2021.zip files.

```

Unzipping the file into the /tmp directory
Cmd 9
1 %sh
2 unzip -d /tmp /tmp/$fileroot.zip
Archive: /tmp/clinicaltrial_2023.zip
inflating: /tmp/clinicaltrial_2023.csv
Command took 2.50 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/26/2024, 12:22:33 AM on DOTT_Task_1_25_Apr_2024_2
Cmd 10
1 #Moving the unzipped file to the DBFS directory
2 dbutils.fs.mv("file:/tmp/" + fileroot + ".csv", "/FileStore/tables/" + fileroot + ".csv", True )
True
Command took 17.32 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 11:13:20 PM on unknown compute
-- ...
Checking the contents of the unzipped file by listing the directory

```

Figure 1.3 Unzipping data inside Databricks

1.5 Data cleaning and preparation

The following steps are followed in data cleaning and preparation:

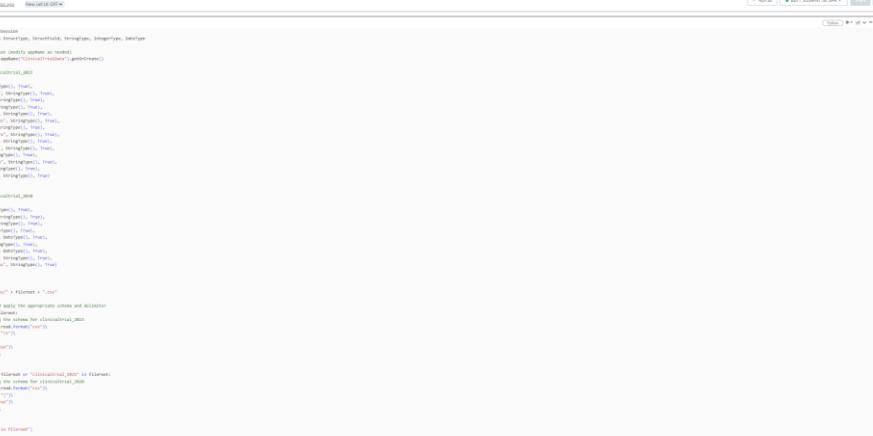
- Delimiter considerations: The delimiter is set to a tab character (`\t`), suggesting the CSV file uses tab-separated values, and the `quote` option is left empty, indicating that no quoting character is used. The `header` option is set to `true` to interpret the first line as column names, allowing for meaningful data labeling.
 - Removing Unwanted Characters: Unwanted characters are removed from the data in a specified column. A regular expression pattern is defined to match specific characters, such as leading and trailing

quotation marks. These characters are replaced with an empty string to clean the data.

- Trimming Spaces: The leading and trailing spaces are removed from the data in the specified column. This ensures that the data values are consistent and free of extraneous whitespace.

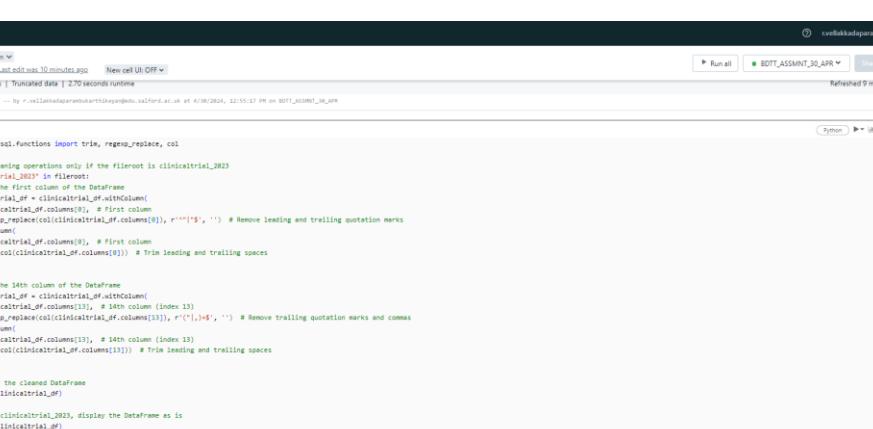
Figure 1.4, 1.5 and 1.6 shows the codes used in dataframes, RDD and SQL for data cleaning and data preparation respectively.

Delimiter considerations



```
Reusable codes_DF Python 3
File Edit View Run Help Last edit was 10 minutes ago New cell UI OFF ▾
Run all ⚡ BD7_ASSMT_30_APP Publish Refreshed 9 minutes ago
Cell 1 of 1
1 from pyspark.sql.functions import trim, regexp_replace, col
2
3 # Perform cleaning operations only if the fileroot is clinicaltrial_2023
4 if "clinicaltrial_2023" in fileroot:
5     # Clean the first column of the DataFrame
6     clinicaltrial_df = clinicaltrial_df.withColumn(
7         "clinicaltrial_df.columns[0]", trim(col("clinicaltrial_df.columns[0]")) # First column
8     ).regexp_replace("clinicaltrial_df.columns[0]", r'^["\s"]+|["\s"]+$') # Remove leading and trailing quotation marks
9     .withColumn(
10         "clinicaltrial_df.columns[0]", # First column
11         trim(col("clinicaltrial_df.columns[0]")) # Trim leading and trailing spaces
12     )
13
14     # Clean the 1st column of the DataFrame
15     clinicaltrial_df = clinicaltrial_df.withColumn(
16         "clinicaltrial_df.columns[13]", trim(col("clinicaltrial_df.columns[13]")) # 14th column (index 13)
17     ).regexp_replace("clinicaltrial_df.columns[13]", r'(["\s"],)+|,"') # Remove trailing quotation marks and commas
18     .withColumn(
19         "clinicaltrial_df.columns[13]", # 14th column (index 13)
20         trim(col("clinicaltrial_df.columns[13]")) # Trim leading and trailing spaces
21     )
22
23     # Display the cleaned DataFrame
24     display(clinicaltrial_df)
25 else:
26     # If not clinicaltrial_2023, display the DataFrame as is
27     display(clinicaltrial_df)
28
29 # (1) Spark Jobs
```

Removing Unwanted Characters



```
Reusable codes_DF Python 3
File Edit View Run Help Last edit was 10 minutes ago New cell UI OFF ▾
Run all ⚡ BD7_ASSMT_30_APP Publish Refreshed 9 minutes ago
Cell 1 of 1
1 from pyspark.sql.functions import trim, regexp_replace, col
2
3 # Perform cleaning operations only if the fileroot is clinicaltrial_2023
4 if "clinicaltrial_2023" in fileroot:
5     # Clean the first column of the DataFrame
6     clinicaltrial_df = clinicaltrial_df.withColumn(
7         "clinicaltrial_df.columns[0]", trim(col("clinicaltrial_df.columns[0]")) # First column
8     ).regexp_replace("clinicaltrial_df.columns[0]", r'^["\s"]+|["\s"]+$') # Remove leading and trailing quotation marks
9     .withColumn(
10         "clinicaltrial_df.columns[0]", # First column
11         trim(col("clinicaltrial_df.columns[0]")) # Trim leading and trailing spaces
12     )
13
14     # Clean the 1st column of the DataFrame
15     clinicaltrial_df = clinicaltrial_df.withColumn(
16         "clinicaltrial_df.columns[13]", trim(col("clinicaltrial_df.columns[13]")) # 14th column (index 13)
17     ).regexp_replace("clinicaltrial_df.columns[13]", r'(["\s"],)+|,"') # Remove trailing quotation marks and commas
18     .withColumn(
19         "clinicaltrial_df.columns[13]", # 14th column (index 13)
20         trim(col("clinicaltrial_df.columns[13]")) # Trim leading and trailing spaces
21     )
22
23     # Display the cleaned DataFrame
24     display(clinicaltrial_df)
25 else:
26     # If not clinicaltrial_2023, display the DataFrame as is
27     display(clinicaltrial_df)
28
29 # (1) Spark Jobs
```

Additional Cleaning for Question 3 The top 5 conditions (from Conditions) with their frequencies.

```

1 # Import pandas and all
2 import pandas as pd
3 from sqlalchemy import create_engine
4 from sqlalchemy import MetaData
5 from sqlalchemy import Table
6 from sqlalchemy import select, regexp_replace, replace, exclude, trim, upper, lower
7
8 # Import necessary modules
9 # Import necessary modules
10 # Import necessary modules
11 # Import necessary modules
12
13 # Create a connection to the database
14 engine = create_engine('mysql://root:@127.0.0.1:3306/test')
15
16 # Create a session
17 Session = sessionmaker(bind=engine)
18 session = Session()
19
20 # Create a DataFrame
21 df = pd.read_sql_query("SELECT * FROM conditions", engine)
22
23 # Drop rows where 'Condition' is null
24 df = df.dropna(subset=['Condition'])
25
26 # Create a new column 'Condition' with the first 5 characters of 'Condition'
27 df['Condition'] = df['Condition'].str[:5]
28
29 # Group by 'Condition' and count
30 df_grouped = df.groupby('Condition').size().reset_index(name='Count')
31
32 # Sort by 'Count' in descending order
33 df_grouped = df_grouped.sort_values(by='Count', ascending=False)
34
35 # Take the top 5 rows
36 top_5 = df_grouped.head(5)
37
38 # Print the result
39 print(top_5)

```

Code 41

```

1 display(filtered_df)

```

(1) Spark Jobs

	Id	Study Title	Acronym	Status	Conditions
1	NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Mental Health Issue E.G.
2	NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Depression
3	NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Psychosis
4	NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Personality Disorder
5	NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Substance Abuse
6	NCT05992571	Oral Ketone Monoester Supplementation and Resting-state Brain Connectivity	null	RECRUITING	Cerebrovascular Function

4,884 rows | Truncated data | 1.30 seconds runtime

Refreshed yesterday

Command took 1.30 seconds -- by r.velakkadaparambukarthikeyan@edu.salford.ac.uk at 4/25/2024, 11:51:44 PM on BDT_Task_1_25_Apr_2024_3

Code 41

```

1 display(filtered_df)

```

(1) Spark Jobs

	Interventions	Sponsor	Collab
1	BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention BEHAVIORAL: Enhanced usual care	Sangath	Hanvan
2	BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention BEHAVIORAL: Enhanced usual care	Sangath	Hanvan
3	BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention BEHAVIORAL: Enhanced usual care	Sangath	Hanvan
4	BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention BEHAVIORAL: Enhanced usual care	Sangath	Hanvan
5	BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention BEHAVIORAL: Enhanced usual care	Sangath	Hanvan
6	OTHER: Placebo DIETARY_SUPPLEMENT: β-CHB	McMaster University	Alzheir

4,884 rows | Truncated data | 1.30 seconds runtime

Refreshed yesterday

Command took 1.30 seconds -- by r.velakkadaparambukarthikeyan@edu.salford.ac.uk at 4/25/2024, 11:51:44 PM on BDT_Task_1_25_Apr_2024_2

Cmd 41

```
1 display(filtered_df)
```

▶ (1) Spark Jobs

Table +

	Collaborators	Enrollment	Funder Type	Type	Study Design
1	Harvard Medical School (HMS and HSDM) London School of Hygiene and Tropical Medicine	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Purpose: TREATMENT
2	Harvard Medical School (HMS and HSDM) London School of Hygiene and Tropical Medicine	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Purpose: TREATMENT
3	Harvard Medical School (HMS and HSDM) London School of Hygiene and Tropical Medicine	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Purpose: TREATMENT
4	Harvard Medical School (HMS and HSDM) London School of Hygiene and Tropical Medicine	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Purpose: TREATMENT
5	Harvard Medical School (HMS and HSDM) London School of Hygiene and Tropical Medicine	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Purpose: TREATMENT
6	Alzheimer's Society of Brant*, Halton and Norfolk,* Hamilton Halton	30.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: CROSSEOVER Masking: DOUBLE (PARTICIPANT, INVESTIGATOR, OUTCOMES ASSESSOR) Primary Purpose: BASIC SCIENCE

4,884 rows | Truncated data | 1.30 seconds runtime

Refreshed yesterday

Command took 1.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/23/2024, 11:51:44 PM on EOTT_Task_3_25_Apr_2024_2

Cmd 41

```
1 display(filtered_df)
```

▶ (1) Spark Jobs

Table +

	Enrollment	Funder Type	Type	Study Design	Start	Completion
1	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Masking: DOUBLE (INVESTIGATOR, "OUTCOMES_ASSESSOR) Primary Purpose: TREATMENT	2018-08-20	2019-02-28
2	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Masking: DOUBLE (INVESTIGATOR, "OUTCOMES_ASSESSOR) Primary Purpose: TREATMENT	2018-08-20	2019-02-28
3	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Masking: DOUBLE (INVESTIGATOR, "OUTCOMES_ASSESSOR) Primary Purpose: TREATMENT	2018-08-20	2019-02-28
4	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Masking: DOUBLE (INVESTIGATOR, "OUTCOMES_ASSESSOR) Primary Purpose: TREATMENT	2018-08-20	2019-02-28
5	250.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: PARALLEL Masking: DOUBLE (INVESTIGATOR, "OUTCOMES_ASSESSOR) Primary Purpose: TREATMENT	2018-08-20	2019-02-28
6	30.0	OTHER	INTERVENTIONAL	Allocation: RANDOMIZED Intervention Model: CROSSOVER Masking: TRIPLE (PARTICIPANT, INVESTIGATOR, OUTCOMES_ASSESSOR) Primary Purpose: BASIC SCIENCE	2023-10-25	2024-08

4,884 rows | Truncated data | 1.30 seconds runtime

Refreshed yesterday

Command took 1.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/23/2024, 11:51:44 PM on EOTT_Task_3_25_Apr_2024_2

Figure 1.4 Data cleaning and preparation in PySpark DataFrame approach

Delimiter considerations

```

1  from pyspark import SparkContext
2  from pyspark.sql.functions import trim, regexp_replace
3  import re
4
5  # Assuming sc is your SparkContext and the RDD is already created
6  # sc = SparkContext.getOrCreate()
7
8  # Function to clean the first field in a line
9  def clean_first_field(line):
10    # Split the line into fields using tab as the delimiter
11    fields = line.split("\t")
12
13    # Clean the first field (first column) by removing leading and trailing quotation marks and trimming spaces
14    first_field = fields[0]
15    first_field = re.sub('^\\"|\"$', "", first_field) # Remove leading and trailing quotation marks
16    first_field = first_field.strip() # Trim leading and trailing spaces
17
18    # Replace the cleaned first field in the list of fields
19    fields[0] = first_field
20
21    # Reconstruct the line by joining the fields with the tab delimiter
22    cleaned_line = "\t".join(fields)
23
24
25    return cleaned_line
26
27
28  # Function to clean the 14th field in a line
29  def clean_14th_field(line):
30    # Split the line into fields using tab as the delimiter
31    fields = line.split("\t")
32
33    # Check if the line has at least 14 fields
34    if len(fields) < 14:
35      # Return the line unchanged if there aren't enough fields
36      return line
37
38
```

Removing unwanted characters and Trimming spaces

```

16     first_field = first_field.strip() # Trim leading and trailing spaces
17     # Replace the cleaned first field in the list of fields
18     fields[0] = first_field
19
20     # Reconstruct the line by joining the fields with the tab delimiter
21     cleaned_line = '\t'.join(fields)
22
23     return cleaned_line
24
25
26     # Function to clean the 14th field in a line
27     def clean_14th_field(line):
28         # Split the line into fields using tab as the delimiter
29         fields = line.split("\t")
30
31         # Check if the line has at least 14 fields
32         if len(fields) < 14:
33             # Return the line unchanged if there aren't enough fields
34             return line
35
36         # Clean the 14th field (14th column) by removing trailing commas and quotation marks
37         # `fields[13]` represents the 14th column since indexing starts at 0
38         fields[13] = re.sub(r'[,"]+', '', fields[13]) # Remove trailing quotation marks and commas
39         fields[13] = fields[13].strip() # Trim leading and trailing spaces
40
41         # Reconstruct the line by joining the fields with the tab delimiter
42         cleaned_line = '\t'.join(fields)
43
44     return cleaned_line
45
46
# Perform operations based on the value of fileroot
def process_clinicaltrial_rdd(clinicaltrial_rdd, fileroot):
    if fileroot == "clinicaltrial_2023":
        # Clean the first field in each line
        clinicaltrial2023RDD = clinicaltrial_rdd.map(clean_first_field)

        # Clean the 14th field in each line
        clinicaltrial2023RDD = clinicaltrial2023RDD.map(clean_14th_field)

        # Discard the header from the RDD
        clinicaltrial2023RDD_no_header = clinicaltrial2023RDD.zipWithIndex().filter(lambda line_index: line_index[1] != 0).map(lambda line_index: line_index[0])

        # Display the first few cleaned lines in the RDD (for demonstration purposes)
        for line in clinicaltrial2023RDD_no_header.take(5):
            print(line)

    elif fileroot == "clinicaltrial_2020" or "clinicaltrial_2021":
        # Extract the header row from the RDD
        header = clinicaltrial_rdd.first()

        # Remove the header row from the RDD
        clinicaltrial_rdd_no_header = clinicaltrial_rdd.filter(lambda line: line != header)

        # Display the first few lines without header (for demonstration purposes)
        for line in clinicaltrial_rdd_no_header.take(5):
            print(line)

    process_clinicaltrial_rdd(clinicaltrial_rdd, fileroot)

```

Sample data after cleaning

The screenshot shows a Databricks notebook titled 'Remya_Vellakkadaparambu_Karthikeyan_rdd'. The code in the editor is the same as the one above, but it's part of a larger notebook structure. The results panel displays the cleaned data, which includes a single row of study details:

```

+ (1) Spark Jobs
[[{"x": "Study Title", "Acronym": "Status", "Conditions": "Interventions", "Sponsor": "Collaborators", "Enrollment": "Design Type", "Type": "Study Design", "Start": "Completion"}, {"NCTRB36304071": "Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India", "P": "RECRUITING", "C": "COMPLETED", "M": "Mental Health Issue (E.G., Depression, Psychosis, Personality Disorder, Substance Abuse)", "B": "BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention|BEHAVIORAL: Enhanced usual care", "S": "Sanath"}]

```

Below the results, a command summary is shown: 'Command took 1.19 seconds -- by r.vellakkadaparambu_karthikeyan@edu.salford.ac.uk at 4/27/2024, 12:52:32 AM on BDT_task1_27_Apr_1'

```

Remya_Vellakkadaparambu_Karthikeyan_rdd Python ▾
File Edit View Run Help Last edit was 4 hours ago New cell UI: OFF ▾
Run all Share Publish ▾
(I) Spark Job
Collaborators', 'Enrollment', 'Funder Type', 'ID', 'Study Design', 'Start', 'Completion', [NCT03638471], 'Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India', 'PRIDE', 'COMPLETED', 'Mental Health Issue (E.G., Depression, Psychosis, Personality Disorder, Substance Abuse)', 'BEHAVIORAL: PRIDE 'Step 1' problem-solving intervention|BEHAVIORAL: Enhanced usual care', 'Sangath', 'Harvard Medical School (HMS and HSDM)|London School of Hygiene and Tropical Medicine', '250.0', 'OTHER', 'INTERVENTIONAL', 'Allocation: RANDOMIZED|Intervention Model: PARALLEL|Masking: DOUBLE (INVESTIGATOR, OUTCOMES_ASSESSOR)|Primary Purpose: TREATMENT', '2018-08-20', '2019-02-28']]

```

Command took 1.19 seconds -- by r.vellakkadaparambu.karthikeyan@ed.ac.uk on 4/27/2024, 12:52:32 AM on BDT_task1_27_Apr_1

Figure 1.5 Data cleaning and preparation in PySpark RDD approach

```

Python
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DateType

# Create or get a Spark session (modify appName as needed)
spark = SparkSession.builder.appName("ClinicalTrialData").getOrCreate()

# Define the schema for clinicaltrial_2023
schema_2023 = StructType([
    StructField("Id", StringType(), True),
    StructField("Study_Title", StringType(), True),
    StructField("Acronym", StringType(), True),
    StructField("Status", StringType(), True),
    StructField("Interventions", StringType(), True),
    StructField("Sponsor", StringType(), True),
    StructField("Collaborators", StringType(), True),
    StructField("Enrollment", StringType(), True),
    StructField("Funder_Type", StringType(), True),
    StructField("Start", StringType(), True),
    StructField("Completion", StringType(), True),
    StructField("Start", StringType(), True),
    StructField("Completion", StringType(), True)
])

# Define the schema for clinicaltrial_2020
schema_2020 = StructType([
    StructField("Id", StringType(), True),
    StructField("Sponsor", StringType(), True),
    StructField("Status", StringType(), True),
    StructField("Start", StringType(), True),
    StructField("Interventions", StringType(), True),
    StructField("Type", StringType(), True),
    StructField("Submission", DateType(), True),
    StructField("Conditions", StringType(), True),
    StructField("Interventions", StringType(), True)
])

# Define the schema for clinicaltrial_2011
schema_2011 = StructType([
    StructField("Id", StringType(), True),
    StructField("Title", StringType(), True),
    StructField("Acronym", StringType(), True),
    StructField("Status", StringType(), True),
    StructField("Interventions", StringType(), True),
    StructField("Funder_Type", StringType(), True),
    StructField("Start", StringType(), True),
    StructField("Completion", StringType(), True),
    StructField("Start", StringType(), True),
    StructField("Completion", StringType(), True)
])

# Specify the file path
file_path = "filestore/tables/" + fileroot + ".csv"

# Determine the file type and apply the appropriate schema and delimiter
if "clinicaltrial_2023" in fileroot:
    # Read the CSV file using the schema for clinicaltrial_2023
    clinicaltrial_df = spark.read.format("csv")\
        .option("delimiter", ",")\
        .option("quote", "")\
        .option("header", "true")\
        .schema(schema_2023)\

elif "clinicaltrial_2020" in fileroot or "clinicaltrial_2011" in fileroot:
    # Read the CSV file using the schema for clinicaltrial_2020
    clinicaltrial_df = spark.read.format("csv")\
        .option("delimiter", ",")\
        .option("header", "true")\
        .schema(schema_2020)\

else:
    print("Unknown file type in fileroot")

# Display the DataFrame
display(clinicaltrial_df)

# clinicaltrial_df: pyspark.sql.dataframe.DataFrame[Id: string, Study_Title: string ... 12 more fields]

```

ID	Study Title	Acronym	Status	Conditions
1	NCT03630471 Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Mental Health Issue (E.G., Depression, Psychosis, Personality Disorder, Substance Abuse)
2	NCT05992371 Oral Ketone Monocarboxyl Supplementation and Resting-state Brain Connectivity	null	RECRUITING	Cerebrovascular Function/Cognition
3	NCT00237471 Impact of Tight Glycaemic Control in Acute Myocardial Infarction	null	TERMINATED	Myocardial Infarct/Hyperglycemia
4	NCT03820271 New Prognostic Predictive Models of Mortality of Decompensated Cirrhotic Patients Waiting for Liver Transplantation	SUPERMELD	RECRUITING	Decompensated Cirrhosis/Liver Transplantation
5	NCT06229171 inTake Care Development and Validation of an Innovative Personalized Digital Health Solution for Medication Adherence Support in Cardiovascular Prevention	inTakeCare	NOT_YET_RECRUITING	Hypertension/Treatment Adherence and Compliance/Digital Health
6	NCT02045371 Tailored Inhibitory Control Training to Reverse Et-Linked Deficits in Mid-life	REV	COMPLETED	Smoking/Alcohol Drinking/Prescription Drug Abuse/Substance-Related Disorders

The data cleaning and preparation in Spark SQL approach follows the same codes as in DataFrame approach as the clinicaltrial table is created from dataframe.

Figure 1.6 Data cleaning and preparation in Spark SQL approach

1.6 The number of studies in the dataset (checking distinct studies explicitly) (All codes run a function in reusable notebook)

Dataframe approach

```

29 # Assuming 'clinicaltrial_df' is the DataFrame you want to work with
30 # Define the column names based on the DataFrame's structure
31 # For clinicaltrial_2022, the column names are 'Id' and 'Study Title'
32 # For clinicaltrial_2020 and clinicaltrial_2021, the column names may be different
33
34 if "clinicaltrial_2022" in fileroot:
35     # Define column names for clinicaltrial_2022
36     id_col = "Id"
37     title_col = "Study Title"
38 elif "clinicaltrial_2020" in fileroot or "clinicaltrial_2021" in fileroot:
39     # Define column names for clinicaltrial_2020 and clinicaltrial_2021
40     # Assume the 'Id' column name is 'Id' and there is no 'Study Title' column
41     id_col = "Id"
42     title_col = None
43 else:
44     # Default case (modify as needed based on data)
45     id_col = "Id"
46     title_col = "Study Title"
47
48 # Call the function to calculate distinct counts
49 if title_col is not None:
50     results = count_distinct_studies(clinicaltrial_df, id_col, title_col)
51 else:
52     # If there's no 'Study Title' column, only calculate the distinct count for 'Id'
53     results = count_distinct_studies(clinicaltrial_df, id_col, id_col)
54
55 # Display the dictionary of results (optional)
56 print(results)
57
58 # (6) Spark Jobs
59
60 Number of distinct studies based on 'Id': 483422
61 Number of distinct studies based on 'Study Title': 481103
62 {'distinct_studies_id': 483422, 'distinct_studies_title': 481103}

```

Command took 23.86 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 5/1/2024, 10:25:07 PM on BDTT ASSESSMENT_1MAY

Distinct 20 studies based on 'Id'

```

17
18     # Display the first 20 distinct Ids
19     print("First 20 distinct ids in column '(id_col)':")
20     display(first_20_distinct_ids)
21
22     # Assuming 'clinicaltrial_df' is the DataFrame you want to work with
23     # Define the column name for 'Id'
24     id_col = "Id" # Assuming the column name for 'Id' is 'Id'
25
26     # Call the function to display the first 20 distinct Ids
27     display(first_20_distinct_ids(clinicaltrial_df, id_col))
28
29
30     # Spark Jobs
31
32 First 20 distinct Ids in column 'Id':
33
34 Table +
```

	Id
1	NCT06229171
2	NCT01055171
3	NCT04784871
4	NCT00416871
5	NCT03269071
6	NCT00237471
7	NCT01777771
...	...
20	...

20 rows | 2.00 seconds runtime

Distinct 20 studies based on 'Study Title'

```

15
16     return
17
18     # Get the distinct study titles
19     distinct_study_titles = df.select(title_col).distinct()
20
21     # Select the first 20 distinct study titles
22     first_20_distinct_study_titles = distinct_study_titles.limit(20)
23
24     # Display the first 20 distinct study titles
25     print("First 20 distinct study titles in column '(title_col)'")
26     display(first_20_distinct_study_titles)
27
28     # Assuming 'clinicaltrial_df' is the DataFrame you want to work with
29     # Define the column name for 'Study Title'
30     title_col = "Study Title" # Assuming the column name for 'Study Title' is 'Study Title'
31
32     # Call the function to display the first 20 distinct study titles
33     display(first_20_distinct_study_titles(clinicaltrial_df, title_col))
34
35
36     # Spark Jobs
37
38 First 20 distinct study titles in column 'Study Title':
39
40 Table +
```

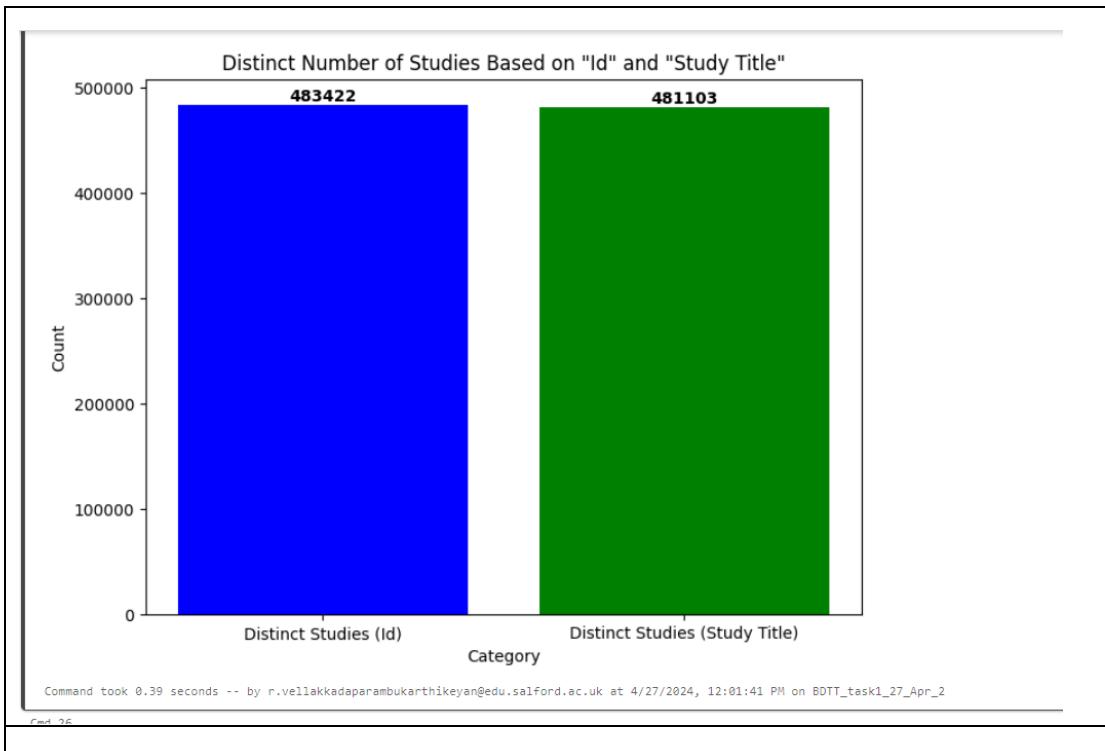
	Study Title
1	Impact of Tight Glycemic Control in Acute Myocardial Infarction
2	Neural Stem Cell Transplantation in Multiple Sclerosis Patients
3	Computerized Brief Alcohol Intervention (BBI) for Bringe Drinking High-Risk and Infected Women
4	Reversal of the Anti-platelet Effects of Trasylol
5	Oral Folate Monotherapy Compared to Folic Acid and Retsing-state Brain Connectivity
6	RCT of the Effect of Ursodeoxycholic Acid Compared to Eicosapentaenoic Acid on Obstructive Sleep Apnea
...	...
20	...

20 rows | 1.79 seconds runtime

Visualizing the answer

```

1 import matplotlib.pyplot as plt
2 import pandas as pd
3
4 # Create a DataFrame for the data
5 data = [
6     {'Category': 'Distinct Studies (Id)', 'Count': 20},
7     {'Category': 'Distinct Studies (Study Title)', 'Count': 20}
8 ]
9
10 df = pd.DataFrame(data)
11
12 # Create a bar plot
13 plt.figure(figsize=(8, 6))
14 bars = plt.bar(df['Category'], df['Count'], color=['blue', 'green'])
15
16 # Add labels and title
17 plt.xlabel('Category')
18 plt.ylabel('Count')
19 plt.title('Distinct Number of Studies Based on "Id" and "Study Title"')
20
21 # Add hover-over data: annotate bars
22 for bar in bars:
23     # Get the height of the bar (count value)
24     yval = bar.get_height()
25     # Add text annotation above the bar to display the count value
26     plt.text(bar.get_x() + bar.get_width() / 2, yval, f'{int(yval)}',
27              ha='center', va='bottom', fontsize=10, fontweight='bold')
28
29 # Display the plot
30 plt.show()
```



Main Idea : The distinct count of 'Id' and 'Study Title' are selected from the clinicaltrial_df and same is visualized.



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Remya_Vellakkadaparambu_Karthikeyan_dd
- File Menu:** File Edit View Run Help Last edit was 11 minutes ago New cell Up: OFF
- Run Button:** Run all
- Share Button:** BDTT_task1_27_Apr_2
- Python Version:** Python 3
- Code Cell (Cell 22):**

```
1 # Get the distinct Ids
2 distinct_ids_rdd = clinicaltrial2023RDD.map(lambda line: line.split('\t')[0]).distinct()
3
4 # Get the first 20 distinct Ids
5 first_20_distinct_ids = distinct_ids_rdd.take(20)
6
7 # Display the first 20 distinct Ids
8 print("First 20 distinct Ids:")
9 for id in first_20_distinct_ids:
10     print(id)
```
- Output Cell (Cell 23):**
 - (1) Spark Jobs
 - First 20 distinct Ids:
 - Id
 - NCT0177271
 - NCT03659671
 - NCT03839971
 - NCT09416871
 - NCT05913271
 - NCT04006171
 - NCT0266071
 - NCT01808171

Distinct 20 studies based on ‘Study Title’

Remya_Vellakkadaparambu_Karthikeyan_rdd Python ▾

File Edit View Run Help Last edit was 12 minutes ago New cell UI: OFF ▾

Run all BDTT_task1_27_Apr_2 Share Publish

Code 23 Python ▾ ▾ ▾ ▾ ▾

```
1 # Get the distinct study titles
2 distinct_study_titles_rdd = clinicaltrial2023R00.map(lambda line: line.split('\t')[1]).distinct()
3
4 # Get the first 20 distinct study titles
5 first_20_distinct_study_titles = distinct_study_titles_rdd.take(20)
6
7 # Display the first 20 distinct study titles
8 print("First 20 distinct study titles:")
9 for title in first_20_distinct_study_titles:
10     print(title)
11
```

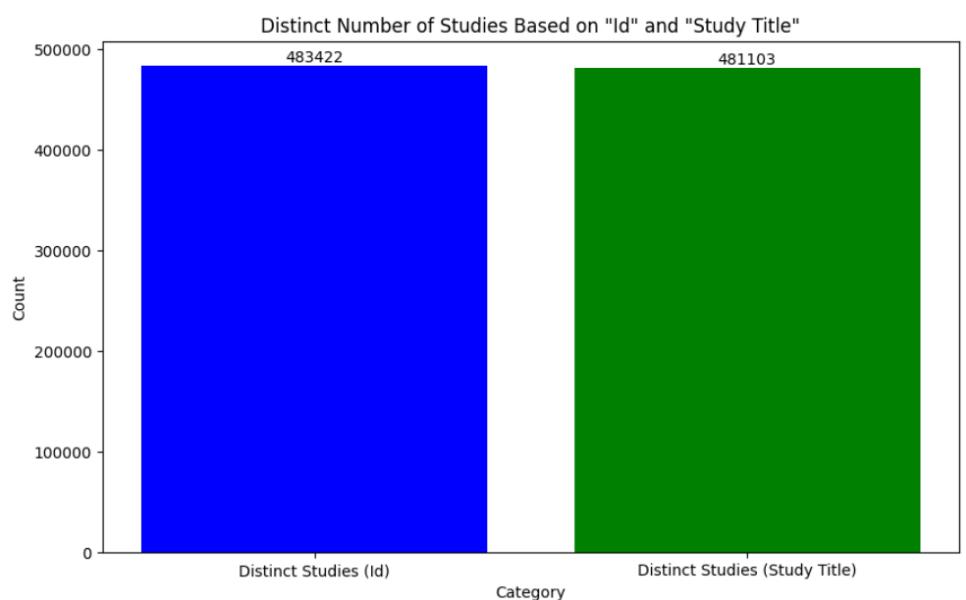
► (1) Spark jobs

First 20 distinct study titles:

Impact of Tight Glycaemic Control in Acute Myocardial Infarction
Molecular Testing for the MD Anderson Cancer Center Personalized Cancer Therapy Program
A Pilot Study Exploring the Efficacy and Safety of Topical Oxybutynin 3% Gel for Primary Focal Hyperhidrosis in Adolescents and Young Adults
Biology of Juvenile Myositis
Safety and Blood Levels of Tenoforodisoproxil Fumarate in HIV Infected Pregnant Women and Their Babies
Modulation of Gut Microbiota in Early Sepsis: A Pilot Study
Clinical Study of a Fixed Combination of Timolol-Brimonidine-Dorzolamide
Validation of the Quark RMR Calorimeter (Cosmed) Versus Deltrac II (GE Health Care Clinical Systems)
Occlusafe® Assisted MI Alone or With DEB-TACE Compared to MI With DEB-TACE in the Treatment of HCC
The Clinical Utility of Extracorporeal Shock Wave Therapy on Burns
Long-term Follow up of Patients With Long-standing Hip and Groin Pain
Study of the Cutaneous Microcirculation in Elderly People
Mass Balance Study of TS-142 in Healthy Adult Subjects.
The Effect of a Physician-Nurse Supplementary Triage Assistance Team on Emergency Department Patient Wait Times
Prostate Cancer: Genomic Progressivity
Comparison of Chemotherapy vs Radiotherapy in Prostate

Visualizing the answer

```
1 import matplotlib.pyplot as plt
2
3 # Data for the bar plot
4 data = [
5     ('Distinct Studies (Id)', distinct_studies_id),
6     ('Distinct Studies (Study Title)', distinct_studies_title)
7 ]
8
9 # Unzip the data
10 categories, counts = zip(*data)
11
12 # Create a bar plot
13 plt.figure(figsize=(10, 6))
14 bars = plt.bar(categories, counts, color=['blue', 'green'])
15
16 # Add labels and title
17 plt.xlabel('Category')
18 plt.ylabel('Count')
19 plt.title('Distinct Number of Studies Based on "Id" and "Study Title"')
20
21 # Add hover-over data: annotate bars
22 for bar in bars:
23     yval = bar.get_height()
24     plt.text(bar.get_x() + bar.get_width() / 2, yval, round(yval),
25             ha='center', va='bottom')
26
27 # Display the plot
28 plt.show()
```



Main Idea: The distinct count of 'Id' and 'Study Title' are selected from the clinicaltrial_rdd and same is visualized.

Spark SQL Approach

```
Cmd 44
1 SELECT COUNT(DISTINCT Id) AS number_of_distinct_studies
2 FROM clinicaltrial_table;
* (3) Spark Jobs
Table ▾ +
number_of_distinct_studies
1 483422
↓ 1 row | 7.74 seconds runtime
Refreshed 1 hour ago
Command took 7.74 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:31:41 AM on BDT_TASK1_SQL_1

Cmd 45
1 SELECT COUNT(DISTINCT `Study_Title`) AS number_of_distinct_study_titles
2 FROM clinicaltrial_table;
* (3) Spark Jobs
Table ▾ +
number_of_distinct_study_titles
1 481103
↓ 1 row | 7.94 seconds runtime
Refreshed now
Command took 7.94 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 1:51:42 AM on BDT_TASK1_SQL_1
```

Distinct 20 studies based on 'Id'

Cmd 46

```

1 SELECT DISTINCT Id
2 FROM clinicaltrial_table
3 LIMIT 20;

```

(2) Spark Jobs

Table +

Id
1 NCT06229171
2 NCT01055171
3 NCT04784871
4 NCT00416871
5 NCT03269071
6 NCT00237471
7 NCT01777771

20 rows | 1.52 seconds runtime

Refreshed 2 minutes ago

Command took 1.52 seconds -- by r.vellakkadaperambukarthikeyan@salford.ac.uk at 4/29/2024, 1:52:17 AM on BDIT_TASK1_SQL_3

Distinct 20 studies based on 'Study Title'

Cmd 47

```

1 SELECT DISTINCT Study_Title
2 FROM clinicaltrial_table
3 LIMIT 20;
4

```

(2) Spark Jobs

Table +

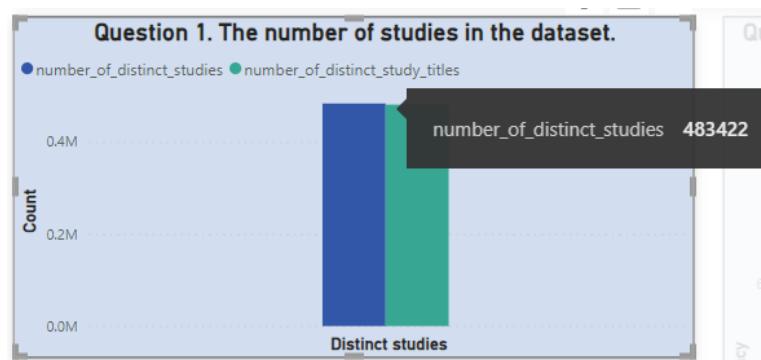
Study_Title
1 Impact of Tight Glycaemic Control in Acute Myocardial Infarction
2 Neural Stem Cell Transplantation in Multiple Sclerosis Patients
3 Computerized Brief Alcohol Intervention (B) for Binge Drinking HIV At-Risk and Infected Women
4 Reversal of the Anti-platelet Effects of Ticagrelor
5 Oral Ketone Monester Supplementation and Resting-state Brain Connectivity
6 RCT of the Effect of Uvulopalatopharyngoplasty Compared to Expectancy in Patients With Obstructive Sleep Apnea

20 rows | 1.04 seconds runtime

Refreshed 1 minute ago

Command took 1.04 seconds -- by r.vellakkadaperambukarthikeyan@salford.ac.uk at 4/29/2024, 1:53:15 AM on BDIT_TASK1_SQL_3

Visualizing the answer



Main Idea: The distinct count of 'Id' and 'Study Title' are selected from the clinicaltrial_table and same is visualized using POWER BI. As the Study Title column hinders the smooth running of general and reusable code for all versions of clinical datasets, it is removed from the notebook.

Table 1.1 Discussion of Results of Question 1

Number of distinct studies based on 'Id': There are **483,422** distinct studies using the 'Id' column.

Number of distinct studies based on 'Study Title': There are **481,103** distinct studies using the 'Study Title' column.

Insights: The result based on Study Title is slightly lower than the count based on 'Id,' suggesting that there may be duplicate study titles in the data, while each study has a unique identifier ('Id'). This shows reduced data quality and integrity.

1.7 List all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type

Dataframe approach

```
(cell 12) from pyspark.sql.functions import desc
2 from pyspark.sql import DataFrame
3
4 def group_and_count(df: DataFrame, group_col: str, order_desc: bool = True):
5     """
6         Group the DataFrame by a specified column and count the occurrences.
7
8     Args:
9         df (DataFrame): The DataFrame containing clinical trial data.
10        group_col (str): The column name to group by.
11        order_desc (bool): Whether to order the results in descending order based on the count. Defaults to True.
12
13    Returns:
14        None
15    """
16    # Group by the specified column and count the occurrences of each group
17    grouped_frequencies = df.groupby(group_col).count()
18
19    # If ordering in descending order is specified, order the result
20    if order_desc:
21        grouped_frequencies = grouped_frequencies.orderBy(desc("count"))
22
23    # Display the result
24    display(grouped_frequencies)
25
26    # Assuming "clinicaltrial_df" is the DataFrame you want to work with
27    # Define the column name to group by
```

cks

mya_Vellakkadaparambu_Karthikeyan_df Python

Edit View Run Help Last edit on 18 minutes ago New cell UI OFF

Run all BOT Assessment_1... Share Publish

```
8 # Arguments
9 #     df (DataFrame): The DataFrame containing clinical trial data.
10 #     group_col (str): The column name to group by.
11 #     order_desc (bool): Whether to order the results in descending order based on the count. Defaults to True.
12 #
13 # Returns:
14 #     None
15 #
16 #     # Group by the specified column and count the occurrences of each group
17 #     grouped_frequencies = df[group_col].count()
18 #
19 #     # If ordering in descending order is specified, order the result
20 #     if order_desc:
21 #         grouped_frequencies = grouped_frequencies.orderBy(desc("count"))
22 #
23 #     # Display the result
24 #     display(grouped_frequencies)
25 #
26 #     # Assuming "clinicalTrial_df" is the DataFrame you want to work with
27 #     # Define the column name to group by
28 #     group_col = "Type" # Example: you can change this to the column you want to group by
29 #
30 #     # Call the function to group and count the DataFrame
31 #     group_and_count(clinicalTrial_df, group_col)
32 
```

(3) Open file

Table +

Type	count
INTERVENTIONAL	371382
OBSERVATIONAL	110221
EXPANDED_ACCESS	928
null	891

± 4 rows | 1.78 seconds runtime

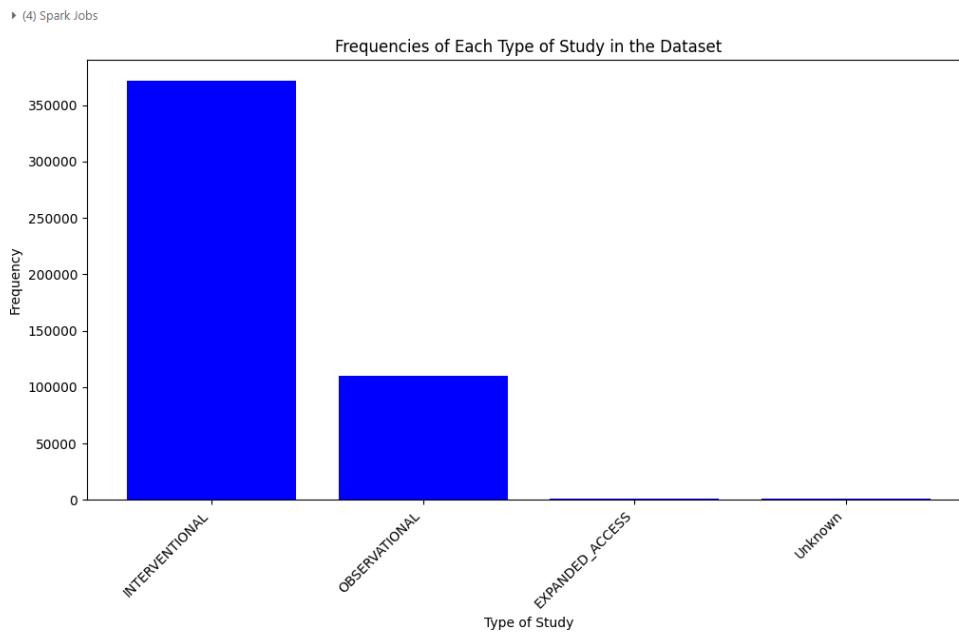
Command took 7.16 seconds -- by r.vellakkadaparambukarthikeyan@edsu.salford.ac.uk on 5/1/2024, 10:20:07 PM on BOT Assessment_1_2407

Refreshed 22 minutes ago

Visualizing the answer

```
ricks
temya_Vellakkadaparambu_Karthikeyan_df Python ▾
File Edit View Run Help Last edit was 24 minutes ago New cell UI: OFF ▾
Run all Share Publish ▾
1 import matplotlib.pyplot as plt
2
3 # Convert Spark DataFrame to Pandas DataFrame
4 type_frequencies_df = type_frequencies_ordered.toPandas()
5
6 # Check for null values and replace them with 'Unknown'
7 type_frequencies_df["Type"] = type_frequencies_df["Type"].fillna("Unknown")
8
9 # Plotting the bar chart
10 plt.figure(figsize=(12, 6))
11 plt.bar(type_frequencies_df["Type"], type_frequencies_df["count"], color='blue')
12
13 # Add labels and title
14 plt.xlabel("Type of Study")
15 plt.ylabel("Frequency")
16 plt.title("Frequencies of Each Type of Study in the Dataset")
17
18 # Rotate x-axis labels for better readability
19 plt.xticks(rotation=45, ha="right")
20
21 # Display the chart
22 plt.show()

▶ (4) Spark Jobs
```



Main Idea : The code groups the clinicaltrial_df by 'Type' column, orders the result by frequency and displays the result

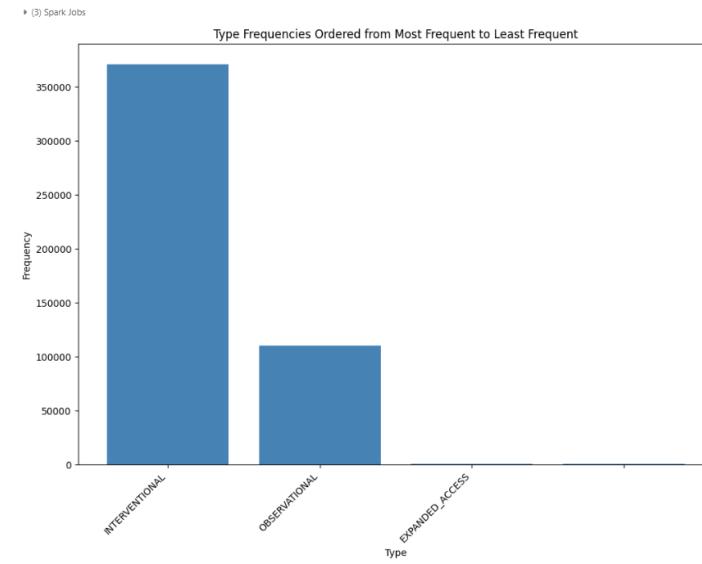
RDD Approach

Visualizing the answer

```
bricks

Remya_Vellakkadaparambu_Karthikeyan_rdd Python V
File Edit View Run Help Last edit was 30 minutes ago New cell UI OFF ▾ Run all Share Publish

1 import matplotlib.pyplot as plt
2
3 # Get the type frequencies ordered from most frequent to least frequent
4 type_counts_rdd = type_rdd.map(lambda type_i : (type_i, 1)).reduceByKey(lambda a, b: a + b)
5 type_counts_ordered_rdd = type_counts_rdd.sortBy(lambda x: -x[1])
6 results = type_counts_ordered_rdd.collect()
7
8 # Prepare data for plotting
9 type_names = []
10 type_counts = []
11 for type_, count in results:
12     type_names.append(type_)
13     type_counts.append(count)
14
15 # Plotting the type frequencies
16 plt.figure(figsize=(12, 8))
17 plt.bar(type_names, type_counts, color='steelblue')
18 plt.xlabel('Type')
19 plt.ylabel('Frequency')
20 plt.title('Type Frequencies Ordered From Most Frequent to Least Frequent')
21 plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for readability
22 plt.show()
```



Main Idea: The code groups the clinicaltrial_rdd by 'Type' column, orders the result by frequency and displays the result

Spark SQL Approach

Cmd 49

```

1 SELECT Type, COUNT(*) AS frequency
2 FROM clinicaltrial_table
3 GROUP BY Type
4 ORDER BY frequency DESC;
5

```

(2) Spark Jobs

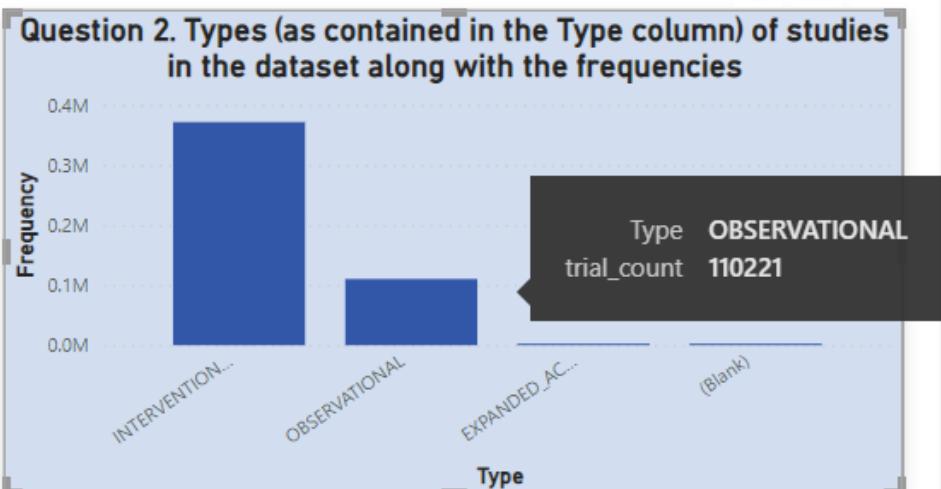
Type	frequency
INTERVENTIONAL	371382
OBSERVATIONAL	110221
EXPANDED_ACCESS	928
null	891

4 rows | 6.39 seconds runtime

Command took 6.39 seconds --- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDIT_TASK1_SQL_1

Refreshed 1 hour ago

Visualizing the answer in POWER BI



Main Idea: The code groups the clinicaltrial_table by 'Type' column, orders the result by frequency and displays the result

Table 1.2 Discussion of Results

Type	Count
INTERVENTIONAL	371382
OBSERVATIONAL	110221
EXPANDED_ACCESS	928
NULL or UNKNOWN	891

Insights: There are different clinical trial types in the dataset, which can be useful for understanding the focus and scope of research efforts in the clinical trial data. The null or unknown type shows reduced data quality and integrity.

1.8 The top 5 conditions (from Conditions) with their frequencies.

There are three considerations in this question.

- 1) Directly finding the top 5 conditions from the dataset
- 2) Splitting the conditions column in the dataset for the delimiter '|'
- 3) Splitting the conditions column, by taking the e.g conditions out of bracket

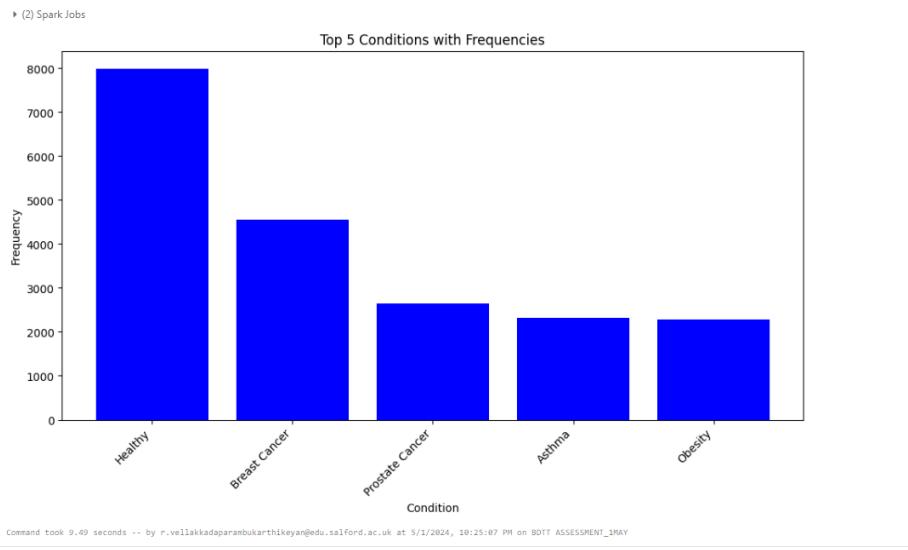
Dataframe approach

Directly finding the top 5 conditions from the dataset

```

1  from pyspark.sql import DataFrame
2  from pyspark.sql.functions import desc, col, split, explode, trim, isnull
3
4  def display_top_5_conditions(clinicaltrial_df: DataFrame, fileroot: str):
5
6      #Display the top 5 conditions and their frequencies based on the given fileroot.
7
8      Args:
9          clinicaltrial_df (DataFrame): The DataFrame containing the data to work with.
10         fileroot (str): The root of the file to determine which operations to perform.
11
12     Returns:
13         None
14     """
15
16     if fileroot == "clinicaltrial_2003":
17         # For clinicaltrial_2003, group by "Conditions" column and count occurrences
18         condition_frequencies = clinicaltrial_df.groupby("Conditions").count()
19         # Order the conditions from most frequent to least frequent and limit to the top 5
20         top_5_conditions = condition_frequencies.orderBy(desc("count")).limit(5)
21         # Display the top 5 conditions with their frequencies
22         top_5_conditions.show()
23
24     elif fileroot == ("clinicaltrial_2009", "clinicaltrial_2011"):
25         # For clinicaltrial_2009 and clinicaltrial_2011, split the "conditions" column by commas
26         conditions_df = clinicaltrial_df.select(split("conditions", ',').alias('condition_list'))
27         # Use explode to create a new DataFrame with one row per condition
28         exploded_conditions_df = conditions_df.explode("condition_list")
29         # Filter out empty strings and whitespace-only conditions
30         exploded_conditions_df = exploded_conditions_df.filter(
31             ~isnull(exploded_conditions_df["condition"]) & trim(exploded_conditions_df["condition"]) != ""
32         )
33
34         # Calculate the frequency of each condition using groupby and count
35         condition_frequency_df = exploded_conditions_df.groupby("condition").count()
36
37         # Order the conditions by frequency in descending order and limit to the top 5
38         top_5_conditions_df = condition_frequency_df.orderBy(desc("count")).limit(5)
39
40         # Display the top 5 conditions and their frequencies
41         print("Top 5 conditions and their frequencies:")
42         top_5_conditions_df.show()
43
44     display_top_5_conditions(clinicaltrial_df, fileroot)
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
527
528
529
529
530
531
532
533
534
535
536
537
537
538
539
539
540
541
542
543
544
545
546
547
547
548
549
549
550
551
552
553
554
555
556
557
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
576
577
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
614
614
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
```

Visualization



Splitting the conditions column in the dataset for the delimiter '|'

```
Cod 36
```

```
1  from pyspark.sql.functions import col, regexp_replace, split, explode, trim, expr, lower
2
3  # Step 1: Split the 'Conditions' column by pipe ('|') to create an array
4  cleaned_df = clinicaltrial120320F.withColumn(
5      "Conditions",
6      split(col("Conditions"), r"\|")
7  )
8
9  # Step 2: Explode the 'Conditions' array to separate rows for each condition
10 exploded_df = cleaned_df.withColumn(
11     "Conditions",
12     explode(col("Conditions"))
13 )
```

cleaned_df: pyspark.sql.DataFrame = [id: string, StudyTitle: string ... 12 more fields]
exploded_df: pyspark.sql.DataFrame = [id: string, StudyTitle: string ... 12 more fields]

Command took 0.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:16:30 PM on BDTP_Task1_27_Apr_2

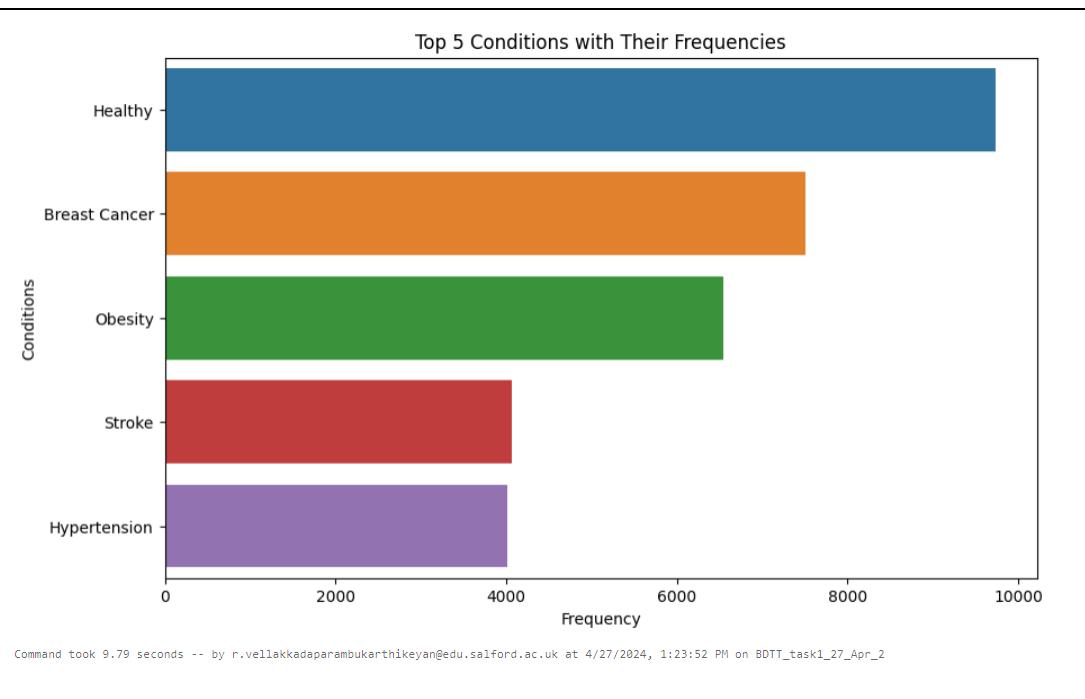
```
Cod 37
```

```
1  from pyspark.sql.functions import col
2
3  # Step 1: Group the exploded data by 'Conditions' and count occurrences
4  condition_counts = exploded_df.groupBy("Conditions").count()
5
6  # Step 2: Order the DataFrame by count in descending order
7  condition_counts_sorted = condition_counts.orderBy(col("count").desc())
8
9  # Step 3: Select the top 5 conditions
10 top_5_conditions = condition_counts_sorted.limit(5)
11
12 # Display the top 5 conditions with their frequencies
13 top_5_conditions.show()
```

condition_counts: pyspark.sql.DataFrame = [Conditions: string, count: long]
condition_counts_sorted: pyspark.sql.DataFrame = [Conditions: string, count: long]
top_5_conditions: pyspark.sql.DataFrame = [Conditions: string, count: long]

Conditions	count
Healthy	9731
Breast Cancer	7582
Obesity	6549
Stroke	4071
Hypertension	4028

Visualization



Command took 9.79 seconds -- by r.yellakkadaparambukarthikveyan@salford.ac.uk at 4/27/2024, 1:23:52 PM on BDT task1 27 Apr 2

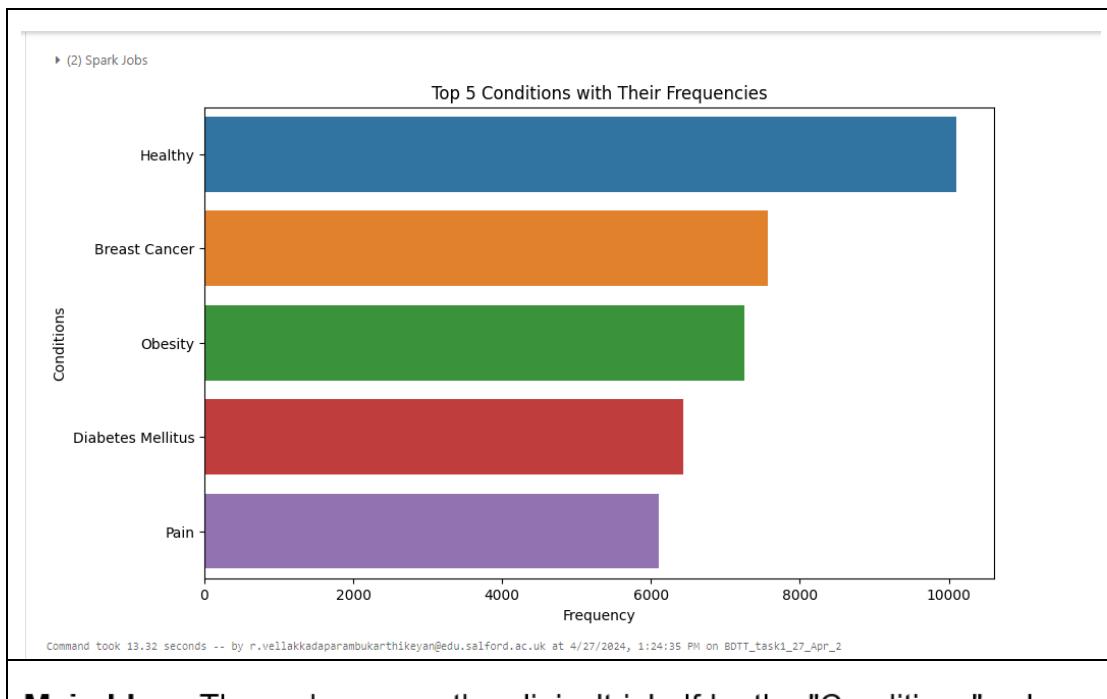
Splitting the conditions column, by taking the e.g conditions out of bracket

```
1 from pyspark.sql.functions import col, regexp_replace, split, explode, trim, expr, lower
2
3 # Step 1: Remove quotes and clean the "Conditions" column and trim leading and trailing spaces
4 cleaned_df = exploded_df.withColumn(
5     "conditions",
6     trim(regexp_replace(col("conditions"), r'''", '')))
7
8 # Step 2: Replace square brackets ('[' and ']') and parentheses '(' and ')' with no space in "Conditions" column
9 cleaned_df = cleaned_df.withColumn(
10     "conditions",
11     regexp_replace(col("conditions"), r'[\[|\]|\(|\)|\)]', ''))
12
13 # Step 3: Split the "Conditions" column by pipe ('|') to create an array
14 cleaned_df = cleaned_df.withColumn(
15     "conditions",
16     split(col("conditions"), r"\|"))
17
18 # Step 4: Explode the "Conditions" array to separate rows for each condition
19 cleaned_df = cleaned_df.withColumn(
20     "conditions",
21     explode(col("conditions")))
22
23 # Step 5: Split each condition by comma (',') and flatten the nested conditions
24 cleaned_df = cleaned_df.withColumn(
25     "conditions",
26     split(unnest("conditions"), ","))
27
28 # Step 6: Explode the array created in Step 5
29 cleaned_df = cleaned_df.withColumn(
30     "conditions",
31     explode(col("conditions")))
32
33 # Step 7: Trim leading and trailing whitespaces from the "Conditions" column
34 cleaned_df = cleaned_df.withColumn(
35     "conditions",
36     trim(col("conditions")))
37
38 # Step 8: Filter out rows with the condition "e.g." in a case-insensitive manner
39 filtered_df = cleaned_df.filter(
40     ~lower(col("conditions")).like("%e.g.%"))
41
```

```
1 from pyspark.sql.functions import col
2
3 # Step 10: Calculate the top 5 conditions with their frequencies
4 top_5_conditions = filtered_df.groupBy("Conditions").count() # Group by 'Conditions' and count occurrences
5 top_5_conditions = top_5_conditions.orderBy(col("count").desc()) # Order by count in descending order
6 top_5_conditions = top_5_conditions.limit(5) # Select the top 5 conditions
7
8 # Display the top 5 conditions with their frequencies
9 top_5_conditions.show()

+ [2] Spark Jobs
+-- top_5_conditions: pyspark.sql.DataFrameType [Conditions: string, count: long]
+-----+
| Conditions|count|
+-----+-----+
| Healthy|10802|
| Breast Cancer| 7564|
| Obesity| 7257|
| Diabetes Mellitus| 6438|
| Pain| 6105|
+-----+
```

Visualization



Main Idea : The code groups the clinicaltrial_df by the "Conditions" column, orders the count column in descending order, selects the top 5 most common conditions from the ordered data and display them.

RDD Approach

Directly finding the top 5 conditions from the dataset

The screenshot shows a Jupyter Notebook interface with the following details:

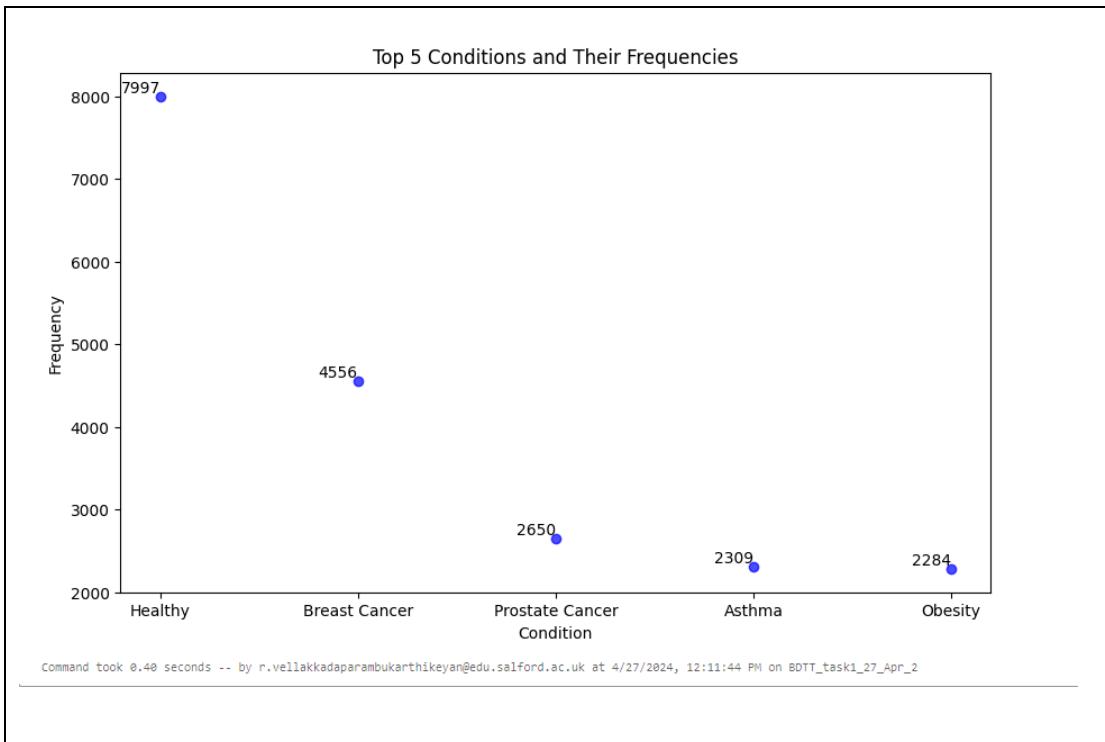
- Title Bar:** Remya_Vellakkadaparambu_Karthikeyan_rdd
- Header Buttons:** Run all, EDIT, Task1_27_Apr_2, Share, Publish
- Code Content (Python 3):**

```
1 # Define a function to extract the "Conditions" column and handle missing values
2 def extract_condition(line):
3     fields = line.split(',')
4     if len(fields) > 4: # Ensure there are at least 5 fields
5         condition = fields[4] # 'Conditions' column is at index 4
6         return condition if condition else None # Return None if the condition is empty
7     else:
8         return None # Return None if the line does not have enough fields
9
10 # Get the "Conditions" column from the RDD and filter out invalid lines
11 condition_rdd = clinicaltrial2023M0_no_header.map(extract_condition).filter(lambda x: x is not None)
12
13 # Create key-value pairs where the key is the condition and the value is 1
14 condition_counts_rdd = condition_rdd.map(lambda condition: (condition, 1))
15
16 # Reduce the RDD by key (condition) to count occurrences
17 condition_frequencies_rdd = condition_counts_rdd.reduceByKey(lambda a, b: a + b)
18
19 # Sort the results from most frequent to least frequent
20 sorted_condition_frequencies_rdd = condition_frequencies_rdd.sortBy(lambda x: -x[1])
21
22 # Take the top 5 conditions
23 top_5_conditions = sorted_condition_frequencies_rdd.take(5)
24
25 # Display the top 5 conditions with their frequencies
26 print("Top 5 conditions and their frequencies")
27 for condition, count in top_5_conditions:
28     print(f"Condition: {condition}, Count: {count}")
29
```
- Output Section:**

Top 5 conditions and their frequencies

Condition	Count
Healthy	7997
Breast Cancer	4556
Cervical Cancer	2050
Asthma	1380
Obesity	2284

Visualization



Splitting the conditions column in the dataset for the delimiter ‘|’

```

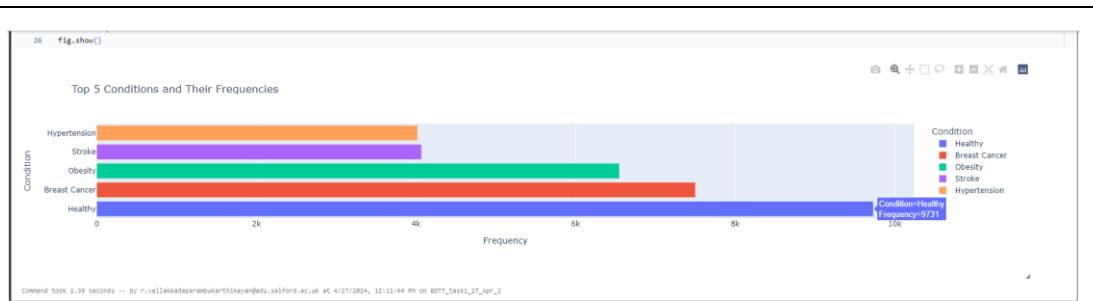
abricks
Remya_Vellakkadaparambu_Karthikeyan_rdd Python v
File Edit View Run Help Last edit was 1 hour ago New cell Use OFF ▾
cmd 33
1 # Step 1: Define a function to split the 'Conditions' column by pipe ('|')
2 def split_conditions(line):
3     # Split the line by the character to get the fields
4     fields = line.split('|')
5     # Ensure the line has at least 5 fields and get the 'Conditions' field
6     if len(fields) > 4:
7         conditions = fields[4] # 'Conditions' column is at index 4
8         # Split the 'Conditions' field by pipe ('|') and return as a list
9         return [field for field in conditions.split('|')]
10    else:
11        return []
12
13 # Step 2: Use flatMap to split and explode the 'Conditions' column
14 exploded_conditions_rdd = clinicaltrial2022RDD_no_header.flatMap(split_conditions)
15
16 # Display the results (you can also perform other operations such as counting, filtering, etc.)
17 for condition in exploded_conditions_rdd.take(20):
18     print(condition)
19

▶ (1) SparkJobs
("Cerebral Health Issue (E.G., Depression, Psychosis, Personality Disorder, Substance Abuse)",),
("Cerebrovascular Function"),
("Cognitive"),
("Hypocardial Infarct"),
("Hyperglycemia"),
("Decompensated Cirrhosis"),
("Hypertension"),
("Hypertension"),
("Treatment Adherence and Compliance"),
("Digital Health"),
("Smoking"),
("Alcohol Drinking"),
("Prescription Drug Abuse")
cmd 39
1 # Step 1: Group the exploded RDD by 'Conditions' and count occurrences
2 condition_counts_rdd = exploded_conditions_rdd.map(lambda condition: (condition, 1)).reduceByKey(lambda a, b: a + b)
3
4 # Step 2: Sort the RDD by count in descending order
5 condition_counts_sorted_rdd = condition_counts_rdd.sortBy(lambda x: -x[1])
6
7 # Step 3: Take the top 5 conditions
8 top_5_conditions_rdd = condition_counts_sorted_rdd.take(5)
9
10 # Display the top 5 conditions with their frequencies
11 print("Top 5 conditions and their frequencies:")
12 for condition, count in top_5_conditions_rdd:
13     print(f"Condition: {condition}, Count: {count}")
14

▶ (2) SparkJobs
Top 5 conditions and their frequencies:
Condition: ('Healthy'), Count: 9731
Condition: ('Breast Cancer'), Count: 7997
Condition: ('Obesity'), Count: 6549
Condition: ('Stroke'), Count: 4071
Condition: ('Hypertension'), Count: 4020
Command took 28.76 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:44 PM on BDTT_task1_27_Apr_2

```

Visualization



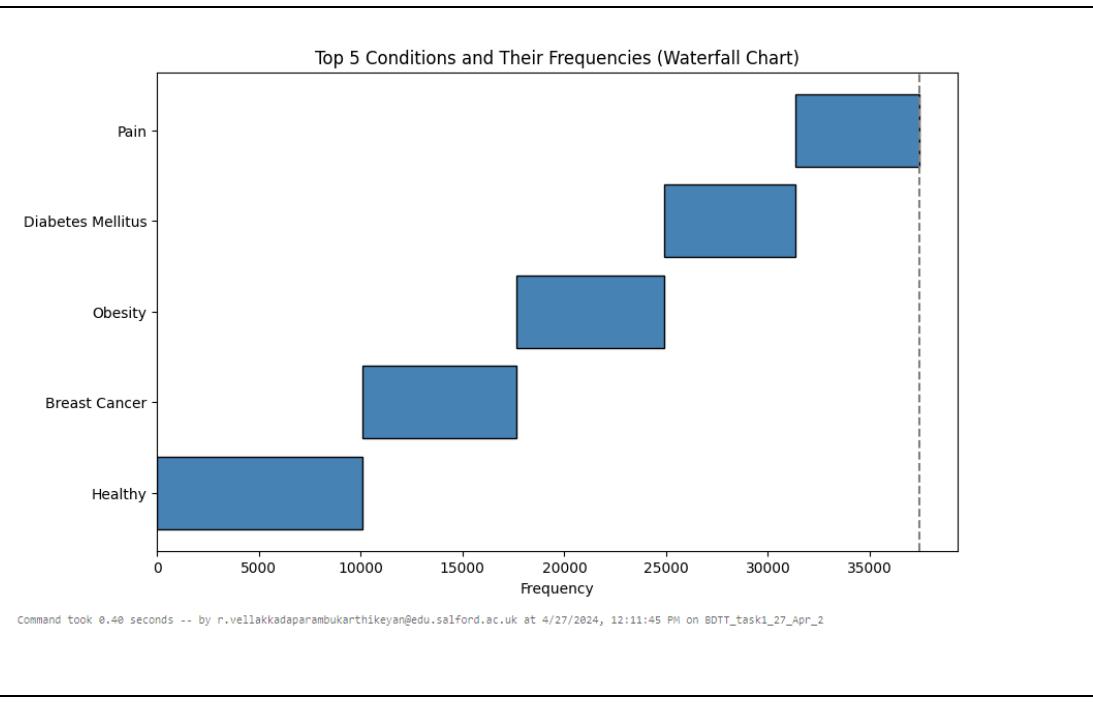
Splitting the conditions column, by taking the e.g conditions out of bracket

```
Cmd 39
1 from pyspark import SparkContext
2 from pyspark.sql import SparkSession
3 import re
4
5 # Step 1: Define a function to clean and trim the 'Conditions' field
6 def clean_conditions(line):
7     fields = line.split("\n")
8     if len(fields) > 4: # Ensure there are at least 5 fields
9         conditions = fields[4]
10        if conditions:
11            # Remove quotes and trim leading and trailing whitespaces
12            conditions = conditions.replace("'", "").strip()
13            # Replace square brackets and parentheses
14            conditions = re.sub(r'\[\w+\]\(\w+\)', '', conditions)
15            # Split the conditions by pipe ('|') and flatten nested conditions
16            conditions = re.split(r'\|', conditions)
17            # Trim leading and trailing whitespaces from each condition
18            conditions = [condition.strip() for condition in conditions]
19            # Filter out conditions that contain 'e.g.' (case-insensitive)
20            conditions = [condition for condition in conditions if 'e.g.' not in condition.lower()]
21        return conditions
22    return []
23
24 # Create an RDD from the cleaned data (assuming clinicaltrial2023RDD_no_header is the original RDD without header)
25 cleaned_conditions_rdd = clinicaltrial2023RDD_no_header.flatMap(clean_conditions)
26
27 # Now, you have a cleaned RDD of conditions after all the filtering and transformations
28
29
Command took 0.18 seconds -- by r.vellankandaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 11:11:44 PM on EDT_task1_27_Apr_2
Cmd 48
```

```
Cmd 48
1 # Step 10: Calculate the top 5 conditions with their frequencies in RDD
2 # First, map each condition to a key-value pair (condition, 1)
3 condition_counts_rdd = cleaned_conditions_rdd.map(lambda condition: (condition, 1))
4
5 # Next, reduce by key to count occurrences of each condition
6 condition_counts_rdd = condition_counts_rdd.reduceByKey(lambda a, b: a + b)
7
8 # Sort the RDD by count in descending order
9 sorted_condition_counts_rdd = condition_counts_rdd.sortBy(lambda x: -x[1])
10
11 # Take the top 5 conditions with the highest frequencies
12 top_5_conditions_rdd = sorted_condition_counts_rdd.take(5)
13
14 # Display the top 5 conditions with their frequencies
15 print("Top 5 conditions and their frequencies:")
16 for condition, count in top_5_conditions_rdd:
17     print(f"Condition: {condition}, Count: {count}")
18
19
Command took 18.42 seconds -- by r.vellankandaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 11:11:44 PM on EDT_task1_27_Apr_2

```

Visualization



Main Idea : The code groups the clinicaltrial_rdd by the "Conditions" column, orders the count column in descending order, selects the top 5 most common conditions from the ordered data and display them.

Spark SQL Approach

Directly finding the top 5 conditions from the dataset

```
Cmd 52
1 SELECT Conditions, COUNT(*) AS count
2 FROM clinicaltrial_table
3 GROUP BY Conditions
4 ORDER BY count DESC
5 LIMIT 5;
6

▶ (2) Spark Jobs
Table +
```

Conditions	count
Healthy	7997
Breast Cancer	4556
Prostate Cancer	2650
Asthma	2309
Obesity	2284

↓ 5 rows | 6.80 seconds runtime Refreshed 1 hour ago

Command took 6.80 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTT_TASK1_SQL_1

Splitting the conditions column in the dataset for the delimiter ‘|’

```
1 WITH exploded_conditions AS (
2     -- Step 1: Split the 'Conditions' column by pipe and explode
3     SELECT EXPLODE(SPLIT(Conditions, '\|')) AS Condition
4     FROM clinicaltrial_table
5 ),
6
7 condition_counts AS (
8     -- Step 2: Group the exploded data by 'Condition' and count occurrences
9     SELECT Condition, COUNT(*) AS count
10    FROM exploded_conditions
11   GROUP BY Condition
12 ),
13
14 sorted_condition_counts AS (
15     -- Step 3: Order the condition counts by count in descending order
16     SELECT Condition, count
17     FROM condition_counts
18    ORDER BY count DESC
19 )
20
21 -- Step 4: Select the top 5 conditions with their frequencies
22 SELECT *
23 FROM sorted_condition_counts
24 LIMIT 5;
25
```

Table +

Condition	count
1 Healthy	9731
2 Breast Cancer	7502
3 Obesity	6549
4 Stroke	4071
5 Hypertension	4020

 5 rows | 8.11 seconds runtime

Command took 8.11 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BD**T****D****T** TASK1_SQL_1

Splitting the conditions column, by taking the e.g conditions out of bracket

```
1 --# Cleaned_conditions #_
2 -- Step 1: Remove quotes and trim leading and trailing spaces from 'Conditions' column
3 SELECT TRIM(BOTH REPLACE(Conditions, '\"', '')) AS Condition
4 FROM cleaned_conditions
5
6 --_new_cleaned_conditions #_
7 -- Step 2: Replace square brackets ('[' and ']') and parentheses ('(' and ')') in 'Condition' column
8 SELECT REVERSE(REPLACE(condition, '[]{}()', '')) AS condition
9 FROM cleaned_conditions
10
11 --_split_conditions #_
12 -- Step 3: Split 'Condition' column by pipe ('|') and explode the array
13 SELECT EXPLODE(SPLIT(condition, '|')) AS Condition
14 FROM new_cleaned_conditions
15
16 --_split_by_comma_conditions #_
17 -- Step 4: Split each condition by comma and explode the array
18 SELECT EXPLODE(SPLIT(condition, ',')) AS condition
19 FROM split_conditions
20
21 --_trimmed_conditions #_
22 -- Step 5: Trim leading and trailing whitespace from 'Condition' column
23 SELECT TRIM(Condition) AS Condition
24 FROM split_by_comma_conditions
25
26 --_filtered_conditions #_
27 -- Step 6: Filter out rows with 'e.g.' in a case-insensitive manner
28 SELECT Condition
29 FROM trimmed_conditions
30 WHERE Condition NOT LIKE '%e.g%'
31
32 --_top_5_conditions #_
33 -- Step 7: Calculate the top 5 conditions with their frequencies
34 SELECT Condition, COUNT(*) AS Frequency
35 FROM filtered_conditions
36 GROUP BY Condition
37 ORDER BY Frequency DESC
38 LIMIT 5
39
40 -- Display the top 5 conditions with their frequencies
41 SELECT Condition, frequency
42 FROM top_5_conditions
```

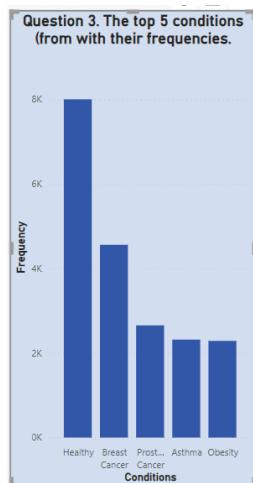
Table ▾ +

	Condition	frequency
1	Healthy	10102
2	Breast Cancer	7564
3	Obesity	7257
4	Diabetes Mellitus	6438
5	Pain	6103

↓ 5 rows | 9.40 seconds runtime

Command took 9.40 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTT_TASK1_SQL_

Visualizing the answer in POWER BI



Main Idea: The code groups the clinicaltrial_table by the "Conditions" column, orders the count column in descending order, selects the top 5 most common conditions from the ordered data and display them.

Table 1.3 Discussion of Results

Table 1 (Directly finding the top 5 conditions)		Table 2 (Splitting the conditions column in the dataset for the delimiter ' ')	
Conditions	Count	Conditions	Count
Healthy	7997	Healthy	9731
Breast Cancer	4556	Breast Cancer	7502
Prostate Cancer	2650	Obesity	6549
Asthma	2309	Stroke	4071
Obesity	2284	Hypertension	4020

Table 3 (Splitting the conditions column, by taking the e.g conditions out of bracket)	
Conditions	Count

Healthy	10102
Breast Cancer	7564
Obesity	7257
Diabetes Mellitus	6438
Pain	6103

Insights: The differences in results across the three tables highlight the importance of data preprocessing and cleaning methods in influencing the outcome of data analysis. Table 3 appears to offer the most precise and comprehensive analysis due to its handling of complex condition entries, which can lead to a more accurate representation of the top conditions in the dataset.

1.9 Find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

Dataframe approach
<pre>Out[44]: 1 # Select the Sponsor column from the Dataframe 2 sponsor2023DF = clinicaltrials2023DF.select("Sponsor") 3 4 # Get the first 5 rows as a list of Row objects 5 first_5_sponsors = sponsor2023DF.head(5) 6 7 # Print the first 5 sponsors 8 for row in first_5_sponsors: 9 print(row["Sponsor"]) 10 11 # (1) Spark jobs # (2) sponsor2023DF: pyspark.sql.dataframe.DataFrame = [Sponsor: string] Sangath McMaster University Hôpital Sainte-Justine - Hôpital de Paris Istituto Auxologico Italiano Comment took 0.58 seconds ... By rcellikakaren@earthlink.net on Apr 17/2024, 1:57:14 PM on BDT, task_1_27_Apr_2 Comment took 0.58 seconds ... By rcellikakaren@earthlink.net on Apr 17/2024, 1:57:14 PM on BDT, task_1_27_Apr_2</pre>
<pre>Out[45]: 1 pharmaDF.display() # (1) Spark jobs Table ✓ + Company → Parent_Company → Penalty_Amount → Subtraction_From_Penalty → Penalty_Amount_Adjusted_For_Eliminating_Multiple_C 1 Abbott Laboratories Abbott Laboratories \$5475.000 \$0 \$5475.000 2 Abbott Laboratories Inc. AbbVie \$1500.000.000 \$0 \$1500.000.000 3 Abbott Laboratories Inc. AbbVie \$126.500.000 \$0 \$126.500.000 4 Abbott Laboratories Puerto Rico, Inc. Abbott Laboratories \$49.045 \$0 \$49.045 5 Acclarient Inc. Johnson & Johnson \$18.000.000 \$0 \$18.000.000 6 7 968 rows 0.59 seconds runtime Comment took 0.59 seconds ... By rcellikakaren@earthlink.net on Apr 17/2024, 1:57:47 PM on BDT, task_1_27_Apr_2 Comment took 0.59 seconds ... By rcellikakaren@earthlink.net on Apr 17/2024, 1:57:47 PM on BDT, task_1_27_Apr_2</pre>

```

1  from pyspark.sql.functions import count, desc
2  # From parent company column, extracting the list of all pharmaceutical companies
3  pharmaceutical_companies = [row[0] for row in pharmaDF.select("Parent_Company").distinct().collect()]
4
5  # Using clinical trial Sponsor column, filtering out all non pharmaceutical companies
6  non_pharmaceutical_sponsors = clinicaltrial2021DF.filter(~clinicaltrial2021DF['Sponsor'].isin(pharmaceutical_companies))
7
8  # Count the number of clinical trials sponsored by each sponsor and sort in descending order
9  tenTopSponsors = non_pharmaceutical_sponsors.groupby("Sponsor") \
10   .agg(count("").alias("numberOfTrials")) \
11   .sort(desc("numberOfTrials"))
12
13 # Show the top 10 sponsors
14 tenTopSponsors.show(10, truncate = False)

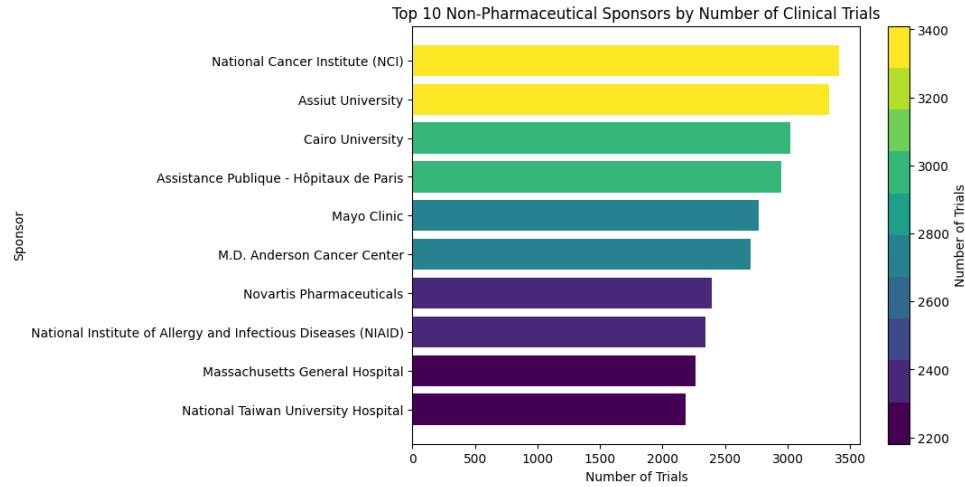
▶ (4) Spark jobs
▶ non_pharmaceutical_sponsors: pyspark.sql.DataFrame[{"id": string, "Study Title": string, ... 12 more fields}]
▶ tenTopSponsors: pyspark.sql.DataFrame[{"Sponsor": string, "numberOfTrials": long}]

+-----+-----+
|Sponsor|numberOfTrials|
+-----+-----+
|National Cancer Institute (NCI)|3410|
|Assiut University|3335|
|Cairo University|3023|
|Assistance Publique - Hôpitaux de Paris|2951|
|Mayo Clinic|2766|
|M.D. Anderson Cancer Center|2592|
|Novartis Pharmaceuticals|2393|
|National Institute of Allergy and Infectious Diseases (NIAID)|2340|
|Massachusetts General Hospital|2263|
|National Taiwan University Hospital|2181|
+-----+-----+
only showing top 10 rows

Command took 9.11 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 1:58:40 PM on BDTE_task1_27_Apr_2

```

Visualizing the answer



Command took 7.03 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 1:59:35 PM on BDTE_task1_27_Apr_2

Main Idea : The code begins by obtaining a list of all pharmaceutical companies from the `pharmaDF` DataFrame's `Parent_Company` column, filters the clinical trial dataset (`clinicaltrial_df`) to exclude pharmaceutical companies by checking if the `Sponsor` column does not contain any of the extracted pharmaceutical companies, groups the filtered DataFrame (`non_pharmaceutical_sponsors`) by the `Sponsor` column and counts the number of clinical trials each sponsor has sponsored, displays the result.

RDD Approach

```
1 # Step 1: Extract the "Sponsor" column from the RDD
2 sponsor2023RDD = clinicaltrial2023RDD_no_header.map(lambda line: line.split('t')[6])
3
4 # Step 2: Take the first 5 sponsor rows
5 first_5_sponsors_rdd = sponsor2023RDD.take(5)
6
7 # Step 3: Print the first 5 sponsors
8 print("First 5 sponsors:")
9 for sponsor in first_5_sponsors_rdd:
10     print(sponsor)
11

► (1) Spark Jobs
First 5 sponsors:
Abbott
McMaster University
Halbourne Health
Assistance Publique - Hôpitaux de Paris
Istituto Auxologico Italiano
Command took 1.38 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 60

1 # Read the CSV file and create an RDD
2 import re
3 pharma_RDD = sc.textFile("~/FileStore/tables/" + fileroot1 + "/" + fileroot1 + ".csv")
4 header = pharma_RDD.first()
5 pharma_RDD = pharma_RDD.filter(lambda row: row != header)
6
7 # Split each row based on commas by excluding commas enclosed within double quotes
8 pharma_RDD = pharma_RDD.map(lambda x: re.split(',(?:(?:"[^"]*"")*[^"]*$)', x))
9

► (1) Spark Jobs
Command took 0.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 61

1 #Extract the contents of parent company field and store it in a new variable
2 pharma_Parent_RDD = pharma_RDD.map(lambda x: x[1])
3

Command took 0.09 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 63

1 pharma_Parent_RDD.take(10)

► (1) Spark Jobs
["Abbott Laboratories",
 "AbbVie",
 "AbbVie",
 "Abbott Laboratories",
 "Johnson & Johnson",
 "Abbott Laboratories",
 "Abbott Laboratories",
 "Johnson & Johnson",
 "Johnson & Johnson",
 "Johnson & Johnson",
 "Abbott Laboratories"]
Command took 0.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 64

1 pharma_companies_group = set(pharma_Parent_RDD.distinct().collect())
2

► (1) Spark Jobs
Command took 0.78 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 65

1 #Remove the double quotes
2 parent_companies_group = {company.replace("'", '') for company in pharma_companies_group}
3 print(parent_companies_group)

{'Novo Holdings A/S', 'Perrigo', 'Daichi Sankyo', 'IDEXX Laboratories', 'GileadSmithKline', 'CytRx', 'Aemphastar Pharmaceuticals', 'SciClone Pharmaceuticals', 'Curaxis Pharmaceuticals', 'Bausch Health', 'Bristol Myers Squibb', 'Genentech', 'Hannibalsoft', 'Gilead Sciences', 'GEI Limited', 'Givaudan Orotica', 'Merck & Johnson', 'SELLAC Life Sciences Group', 'Viatris', 'AVCO Pharmaceuticals', 'Eli Lilly Animal Health', 'Pfizer', 'Wockhardt Limited', 'Phibro Animal Health', 'Otsuka Pharmaceutical', 'Jazz Pharmaceuticals', 'Roche', 'Hikma Pharmaceuticals', 'Pacira BioSciences', 'Merck', 'Inidivior Inc.', 'Sun Pharmaceuticals', 'Purdue Pharma', 'Akorn Inc.', 'Biocryst Pharmaceuticals', 'AstraZeneca', 'Amwell Pharmaceuticals', 'United Therapeutics', 'Parwexel International', 'Eisai', 'Lamett Co.', 'Apotex Corp.', 'Incyte Corp.', 'Organon & Co.', 'Takeda Pharmaceutical', 'Agen', 'Anika Therapeutics', 'Astellas Pharma', 'Aceto', 'Endo International', 'Alkami', 'Abbott Laboratories', 'Boehringer Ingelheim', 'Merck KGaA (BD)', 'Cura Inc.', 'Dr. Reddy's Laboratories', 'Lundbeck', 'Nutraceutical International Corp.', 'Camres', 'Biogen Idec', 'Sanofi', 'Imys Therapeutics', 'KV Pharmaceutical', 'Eli Lilly', 'Amyt Pharma', 'Regeneron Pharmaceuticals', 'UCB', 'Taro Pharmaceutical Industries', 'Arbor Pharmaceuticals', 'Teva Pharmaceutical Industries', 'AbbVie'}
Command took 0.18 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Cmd 67

1 # filter out all the sponsors that are not pharmaceutical companies
2 sponsor_non_pharma_RDD = sponsor2023RDD.filter(lambda x: x not in pharma_Parent_RDD)
3
4 # count the number of clinical trials sponsored by each non-pharmaceutical company
5 sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
6
7 # get the top 10 non-pharma sponsors by number of clinical trials sponsored
8 sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])
9
10 # display the corresponding results
11 for sponsor, count in sponsor_non_pharma_top10_RDD:
12     print(sponsor, count)

► (1) Spark Jobs
National Cancer Institute (NCI) 3410
Assut University 3335
Cairo University 3023
Assistance Publique - Hôpitaux de Paris 2951
Mayo Clinic 2766
M.D. Anderson Cancer Center 2702
Novartis Pharmaceuticals 2393
National Institute of Allergy and Infectious Diseases (NIAID) 2340
Massachusetts General Hospital 2263
National Taiwan University Hospital 2181
Command took 17.47 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
```

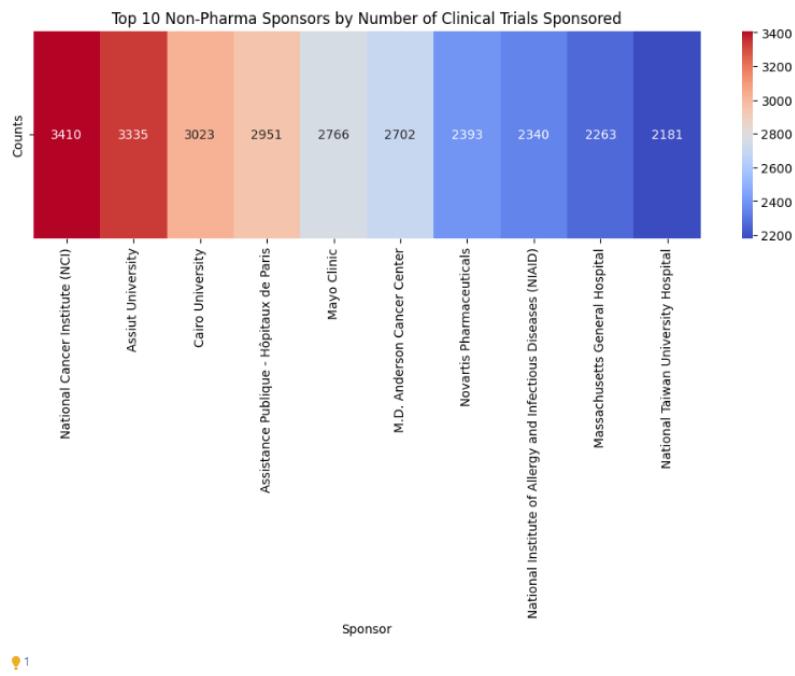
```

Code 40
1 # Filter out all the sponsors that are not pharmaceutical companies
2 sponsor_non_pharma_RDD = sponsorRDDRDD.filter(lambda x: x not in pharma_Parent_RDD)
3
4 # Count the number of clinical trials sponsored by each non-pharmaceutical company
5 sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
6
7 # Get the top 10 non-pharma sponsors by number of clinical trials sponsored
8 sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])
9
10 # Display the corresponding results with header
11 print("Clinical Trials Year")
12 print("-----")
13 print("%(sponsor)s %(count)d" % {"sponsor": sponsor, "count": count})
14 print("-----")
15 for sponsor, count in sponsor_non_pharma_top10_RDD:
16     print("%(sponsor)s %(count)d" % {"sponsor": sponsor, "count": count})
17 print("-----")

* (1) Spark jobs
2022-04-27 12:11:45,488 INFO [main] org.apache.spark.SparkContext: Command took 22.79 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDT_Task1_27_Apr_2

```

Visualizing the answer



Main Idea : The code begins by obtaining a list of all pharmaceutical companies from the `pharmaDF` RDD's `Parent_Company` column, filters the clinical trial dataset (`clinicaltrial_rdd`) to exclude pharmaceutical companies by checking if the `Sponsor` column does not contain any of the extracted pharmaceutical companies, groups the filtered RDD (`non_pharmaceutical_sponsors`) by the `Sponsor` column and counts the number of clinical trials each sponsor has sponsored, displays the result.

Spark SQL Approach

```
Cmd 57
1 -- Step 1: Retrieve a list of pharmaceutical companies from pharma_table
2 WITH pharma_companies AS (
3     SELECT DISTINCT Parent_Company
4     FROM pharma_table
5 )
6
7 -- Step 2: Identify non-pharmaceutical sponsors and count the number of clinical trials they have sponsored
8 SELECT c.Sponsor, COUNT(*) AS num_trials
9 FROM clinicaltrial_table AS c
10 -- Step 3: Left join with pharma_companies to filter out pharmaceutical sponsors
11 LEFT JOIN pharma_companies AS p ON c.Sponsor = p.Parent_Company
12 -- Step 4: Only keep non-pharmaceutical sponsors (i.e., sponsors not in the pharma_companies list)
13 WHERE p.Parent_Company IS NULL
14 -- Step 5: Group by sponsor and count the number of clinical trials they have sponsored
15 GROUP BY c.Sponsor
16 -- Step 6: Order by the count in descending order and limit the results to the top 10
17 ORDER BY num_trials DESC
18 LIMIT 10;
19
20 (5) Spark Jobs
```

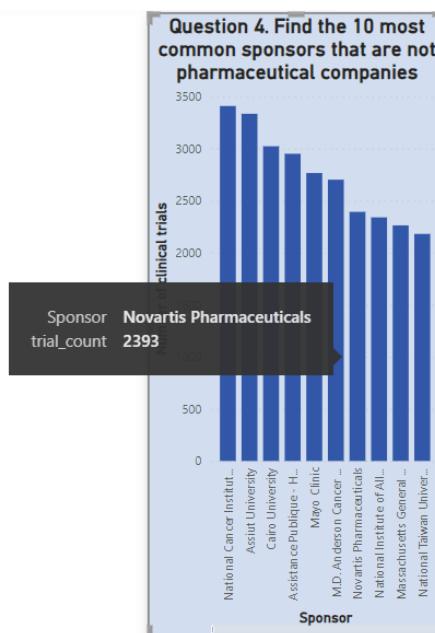
Table ▼ +

	Sponsor	num_trials
1	National Cancer Institute (NCI)	3410
2	Assiut University	3335
3	Cairo University	3023
4	Assistance Publique - Hôpitaux de Paris	2951
5	Mayo Clinic	2766
6	M.D. Anderson Cancer Center	2702
7	Novartis Pharmaceuticals	2393
8	National Institute of Allergy and Infectious Diseases (NIAID)	2340
9	Massachusetts General Hospital	2263
10	National Taiwan University Hospital	2181

↓ 10 rows | 8.24 seconds runtime

Command took 8.24 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTE_TASK1_SQL_1

Visualizing the answer in POWER BI



Main Idea: The code begins by obtaining a list of all pharmaceutical companies from the pharma_table Parent_Company column, filters the clinical trial _table to exclude pharmaceutical companies by checking if the Sponsor column does not contain any of the extracted pharmaceutical companies, groups the filtered result by the Sponsor column and counts the number of clinical trials each sponsor has sponsored, displays the result. The visualization plotted by establishing a relationship between Sponsor column and Parent_company column in the clinicaltrial_table and pharma_table respectively.

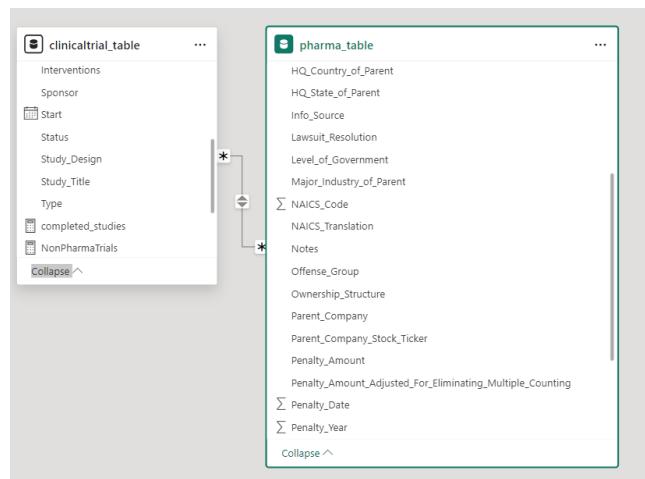


Table 1.4 Discussion of Results

Result	
Sponsor	count
National Cancer Institute (NCI)	3410
Assiut University	3335
Cairo University	3023
Assistance Publique - Hôpitaux de Paris	2951
Mayo Clinic	2766

M.D. Anderson Cancer Center	2702
Novartis Pharmaceuticals	2393
National Institute of Allergy and Infectious Diseases (NIAID)	2340
Massachusetts General Hospital	2263
National Taiwan University Hospital	2181

1.10 Plot number of completed studies for each month in 2023

Dataframe approach

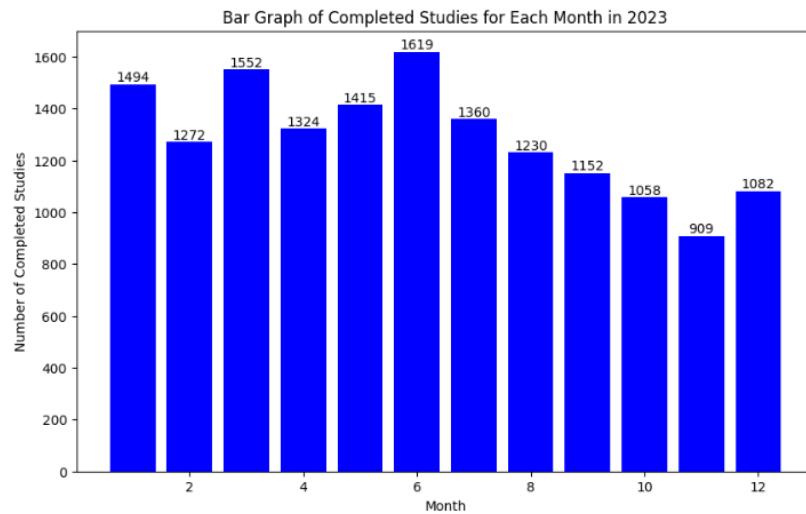
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import pyspark.sql.functions as F
4 from pyspark.sql import DataFrame
5 from pyspark.sql.functions import col
6
7 def process_clinicaltrial(clinicaltrial_df: DataFrame, fileroot: str, trial_year: int):
8
9     # Process clinicaltrial data and generate bar and scatter plots for completed studies each month in the specified trial year.
10
11     Args:
12         clinicaltrial_df (DataFrame): Open DataFrame containing the clinicaltrial data.
13         fileroot (str): The file root string to determine which operations to perform.
14         trial_year (int): The year of interest for the analysis.
15
16     Returns:
17         None
18     """
19
20     if fileroot == "clinicaltrial_NESTED":
21
22         # Extract the completion column to timestamp data type
23         clinicaltrial_df = clinicaltrial_df.withColumn("Start", col("Start").cast("timestamp"))
24         clinicaltrial_df = clinicaltrial_df.withColumn("Completion", col("Completion").cast("timestamp"))
25
26
27         # Function to plot studies each month for the specified completion status and year
28         def plot_studies_each_month(completion_status, study_year):
29             # Convert the completion status parameter to lowercase
30             completion_status = completion_status.lower()
31
32             # Filter the DataFrame for studies with the specified completion status and in the specified study year
33             studies_completed = clinicaltrial_df.filter(
34                 F.lower(clinicaltrial_df["Status"]) == completion_status) &
35                 (F.year(clinicaltrial_df["Completion"]) == study_year)
36
37             # (F.year(clinicaltrial_df["Completion"]) == study_year)
38
39             # Extract the month from the "Completion" column
40             studies_completed = studies_completed.withColumn(
41                 "month", F.month(clinicaltrial_df["Completion"]))
42
43             # Group by the extracted month and count the number of studies
44             monthly_basis_results = studies_completed.groupby("month").count()
45
46             # Display the count of completed studies for each month, ordered by month
47             monthly_basis_results.orderby("month").show()
48
49             # Collect the results to lists for plotting
50             monthly_number = [row.month for row in monthly_basis_results.collect()]
51             no_of_completed_studies = [row["count"] for row in monthly_basis_results.collect()]
52
53             return monthly_number, no_of_completed_studies
54
55
56         # Call the function with the desired completion status and study year
57         month_number, no_of_completed_studies = plot_studies_each_month("Completed", trial_year)
58
59         # Create a bar graph
60         plt.figure(figsize=(10, 6))
61         bars = plt.bar(month_number, no_of_completed_studies, color='blue')
62
63         # Add counts on top of each bar
64         for bar in bars:
65             y_val = bar.get_height() # Get the height of the bar (count)
66             x_pos = bar.get_x() # Get the position of the bar
67             bar.get_x() + bar.get_width() / 2, # Center the text over the bar
68             y_val + bar.get_height() / 2, # Position the text above the bar
```

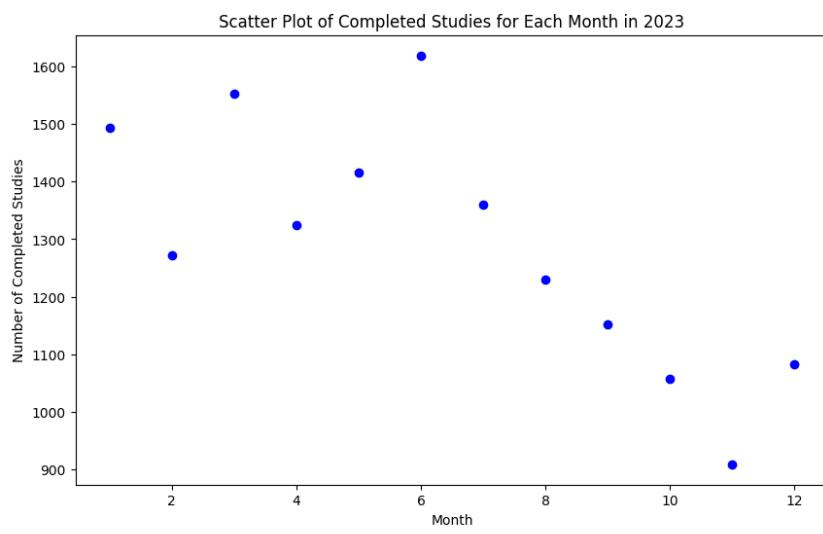
```

48     no_of_completed_studies = [row["count"] for row in monthly_basis_results.collect()]
49
50     return month_number, no_of_completed_studies
51
52 # Call the function with the desired completion status and study year
53 month_number, no_of_completed_studies = plot_studies_each_month("Completed", trial_year)
54
55 # Create a bar graph
56 plt.figure(figsize=(10, 6))
57 bars = plt.bar(month_number, no_of_completed_studies, color='blue')
58
59 # Add counts on top of each bar
60 for bar in bars:
61     y_val = bar.get_height() # Get the height of the bar (count)
62
63     bar.get_x() + bar.get_width() / 2, # Center the text over the bar
64     y_val, # Place the text at the top of the bar
65     str(y_val), # Convert count to string
66     ha='center', # Center-align the text
67     va='bottom' # Place the text at the bottom (above the bar)
68 )
69
70 # Add labels and title
71 plt.xlabel('Month')
72 plt.ylabel('Number of Completed Studies')
73 plt.title('Bar Graph of Completed Studies Each Month In [trial_year]')
74
75 # Display the bar plot
76 plt.show()
77
78 # Create a scatter plot
79 plt.figure(figsize=(10, 6))
80 alt.scatter(month_number, no_of_completed_studies, color='blue', marker='o')
81
82 # Display the scatter plot
83 plt.show()
84
85 elif fileroot == "ClinicalTrial_2021": # Only include completed studies in the specified trial year
86     # Group by month and count the number of completed studies
87     monthly_completed_studies_df = clinicaltrial_df.filter(
88         (F.col("Status") == "Completed") &
89         (F.year(F.to_date(F.col("Completion"), "'WW yyyy")) == trial_year)
90     )
91
92     # Order the Dataframe by 'month_year' in ascending order
93     monthly_completed_studies_df = monthly_completed_studies_df.orderBy('month_year', ascending=True)
94
95     # Show the monthly completed studies in ascending order of months
96     monthly_completed_studies_df.show()
97
98     # Convert the Dataframe to a Pandas Dataframe
99     monthly_completed_studies_pd_df = monthly_completed_studies_df.toPandas()
100
101     # Plotting the data
102     plt.figure(figsize=(10, 6))
103     plt.bar(monthly_completed_studies_pd_df['month_year'], monthly_completed_studies_pd_df['count'], marker='o')
104     plt.xlabel('Number of Completed Studies Each Month In [trial_year]')
105     plt.ylabel('Month')
106     plt.title('Number of Completed Studies')
107     plt.xticks(rotation=90) # Rotate the x-axis labels for better readability
108     plt.grid(True) # Add grid lines
109     plt.show() # Display the plot
110
111
112 process_clinicaltrial(clinicaltrial_df, fileroot, trial_year)
113
114
115 (5) Spark Jobs
116
117 +-----+-----+
118 |month|count|
119 +-----+-----+
120 | 1| 1494|
121 | 2| 1272|
122 | 3| 1552|
123 | 4| 1324|
124 | 5| 1415|
125 | 6| 1619|
126 | 7| 1360|
127 | 8| 1230|
128 | 9| 1152|
129 | 10| 1058|
130 | 11| 909|
131 | 12| 1082|
132 +-----+-----+

```

Visualizing the answer





Main Idea : Initially, the Start and Completion columns of the clinical trial DataFrame (clinicaltrial_df) are converted from string data types to timestamp data. A function is used to filter the clinical trial DataFrame based on the specified completion status and study year. The month is extracted, grouped from Completion column, and count taken. The final results are displayed.

RDD Approach

```

1 split_rdd = clinicaidtrial2023R00.map(lambda line: line.split('`t'))
2 split_rdd.take(2)

# (1) Spark Rdd
#     [
#         'Collection',
#         'Identifier',
#         'Funder',
#         'Funder Type',
#         'Type',
#         'Study Design',
#         'Status',
#         'Completion'],
#     'NCT03930717',
#     'Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India',
#     'PRIDE',
#     'COMPLETED',
#     'Medical Health Issue (E.G., Depression, Psychosis, Personality Disorder, Substance Abuse)',
#     'DESCRIPTION: PRIDE Step 1 problem-solving intervention[BEHAVIORAL: Enhanced usual care]',
#     'Seargent',
#     'Harvard Medical School (HHS and HSDM)|London School of Hygiene and Tropical Medicine',
#     '2018-01-01',
#     'Other',
#     'INTERVENTIONAL',
#     'Allocation: RANDOMIZED|Intervention Model: PARALLEL|Masking: DOUBLE [INVESTIGATOR,- OUTCOMES_ASSESSOR] Primary Purpose: TREATMENT',
#     '2018-01-01',
#     '2019-03-31',
#     'Command took 1.28 seconds -- by r.vellankandaparamarthikayagam@ncl.ac.in at 4/27/2024, 12:11:05 PM on DHT_Task1_27_Apr_2

CMD 72

1 #filtering to include status and completion year
2 split_R00B00 = split_rdd.filter(lambda row:[row[11] and row[3] == "COMPLETED" and row[11].startswith("2023"))]

Command took 0.89 seconds -- by r.vellankandaparamarthikayagam@ncl.ac.in at 4/27/2024, 12:11:05 PM on DHT_Task1_27_Apr_2

CMD 73

```

Remya.Vellakkadaparambu.Karthikeyan.rdd Python v

```

File Edit View Run Help Last edit was 2 minutes ago New cell UI OFF ▾
Content took 14.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Run all BDTT_task1_27_Apr_2 Share Publish ▾
End 73 Synapse ▾
Completed successfully by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk on 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
1 completed_month_status = (
2     '91' : 0,
3     '92' : 0,
4     '93' : 0,
5     '94' : 0,
6     '95' : 0,
7     '96' : 0,
8     '97' : 0,
9     '98' : 0,
10    '99' : 0,
11    '10' : 0,
12    '11' : 0,
13    '12' : 0,
14 )
15 # Updating the count for each month
16 for row in complete_2023R00.collect():
17     completion_date_status = row[13] #Accessing 13th column
18     completion_month_status = completion_date_status.split('-')[1] #Extracting the month
19     study_completed = row[3]
20     completed_month_status[completion_month_status] += 1
21
22 #Results
23 print("-----")
24 print("Month | Number of completed studies of the month |")
25 print("-----")
26 for month, count in completed_month_status.items():
27     print(f"{month}| {count}|")
28 print("-----")
29
30
Command took 14.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

```

► (1) Spark Jobs

Month	Number of completed studies of the month
01	1494
02	1272
03	1552
04	1324
05	1415
06	1619
07	1360
08	1230
09	1152
10	1058
11	909
12	1082

Command took 14.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

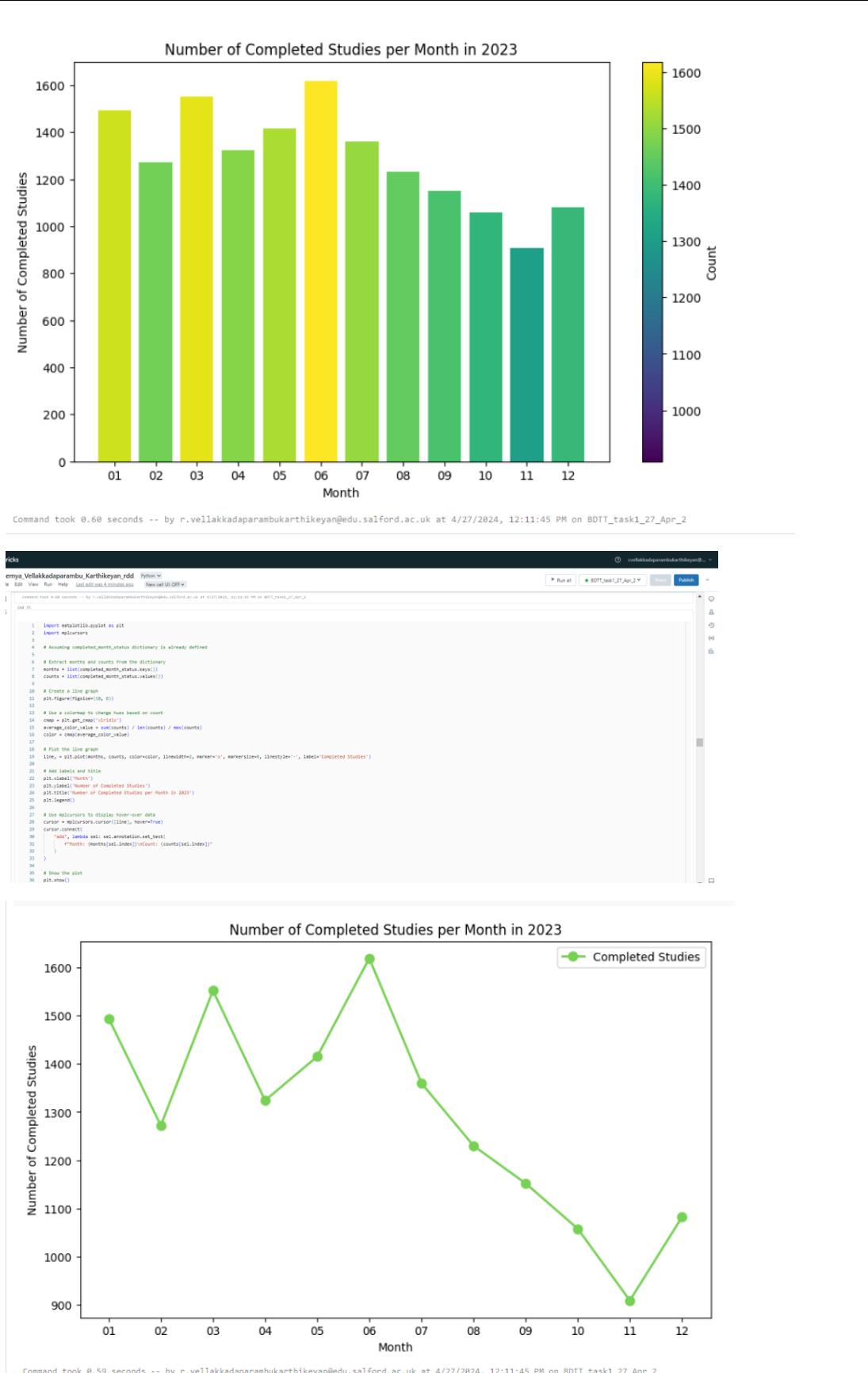
Visualizing the answer

ya.Vellakkadaparambu.Karthikeyan_rdd Python v

```

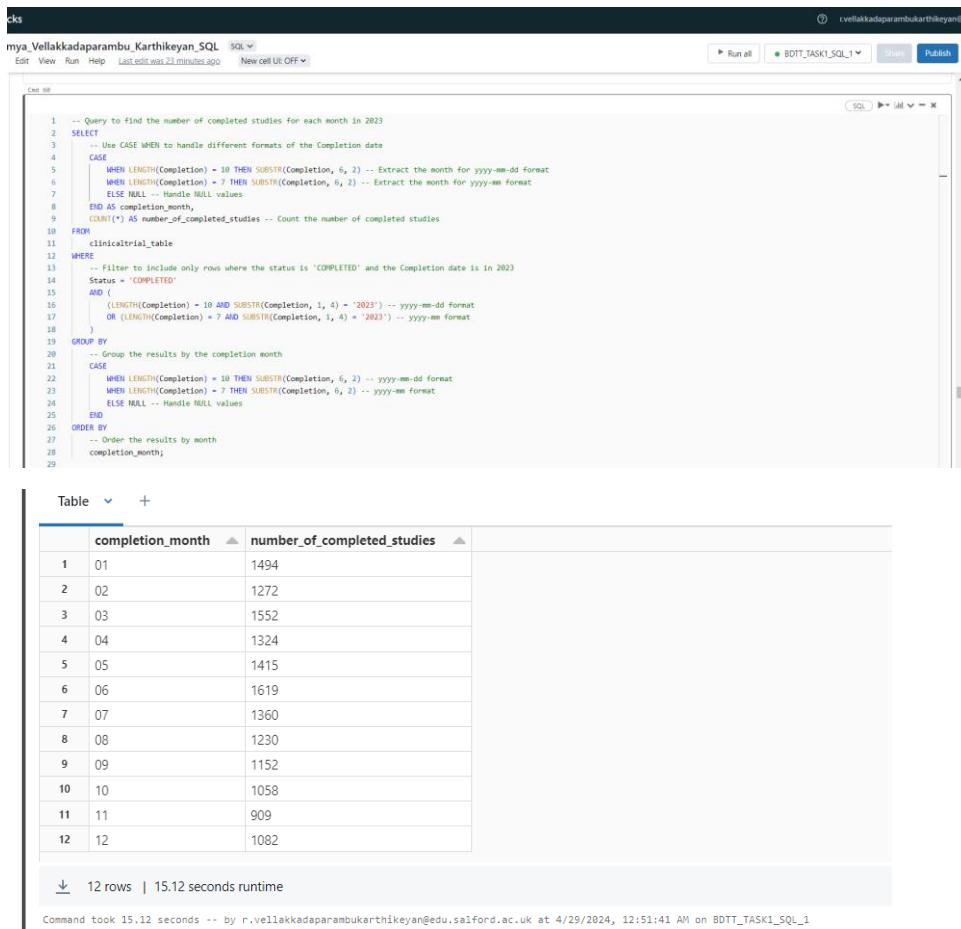
File Edit View Run Help Last edit was 2 minutes ago New cell UI OFF ▾
Content took 14.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
Run all BDTT_task1_27_Apr_2 Share Publish ▾
End 74 Synapse ▾
Completed successfully by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk on 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
1 import matplotlib.pyplot as plt
2 import mplcursors
3
4 # Assuming completed_month_status dictionary is already defined
5
6 # Extract months and counts from the dictionary
7 months = list(completed_month_status.keys())
8 counts = list(completed_month_status.values())
9
10 # Create a bar chart
11 plt.figure(figsize=(10, 6))
12
13 # Use a colormap to change bars based on count
14 cmap = plt.get_cmap('viridis') # Use 'viridis' colormap or any other preferred colormap
15 colors = cmap(count / max(counts) for count in counts)
16
17 bars = plt.bar(months, counts, color=colors)
18
19 # Add labels and title
20 plt.xlabel('Month')
21 plt.ylabel('Number of Completed Studies')
22 plt.title('Number of Completed Studies per Month in 2023')
23
24 # Add a color bar to represent the hue scale
25 sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(min=min(counts), vmax=max(counts)))
26 sm.set_array([])
27 plt.colorbar(sm, label='Count')
28
29 # Use mcursors to display hover-over data
30 cursor = mplcursors.cursor(bars, hover=True)
31 cursor.connect(
32     "add", lambda sel: sel.annotation.set_text(
33         f"Month: {months[sel.index]} \nCount: {counts[sel.index]}"))
34
35
36 # Show the plot
37 plt.show()

```



Main Idea: The clinicaltrial_rdd, is split using the tab delimiter ('\t') to transform each line of data into a list of columns. The split RDD is filtered to only include rows where the study status is "COMPLETED" and the completion date starts with "2023". A dictionary named completed_month_status is initialized with keys representing months (01 to 12) and values set to 0. This dictionary will store the count of completed studies for each month. The code iterates through the complete_2023RDD and extracts the completion date for each study. It then extracts the month from the completion date and increments the corresponding count in the completed_month_status dictionary. The results are displayed and visualized.

Spark SQL Approach



```

1 -- Query to find the number of completed studies for each month in 2023
2
3 SELECT
4   -- Use CASE WHEN to handle different formats of the Completion date
5   CASE
6     WHEN LENGTH(Completion) = 10 THEN SUBSTR(Completion, 6, 2) -- Extract the month for yyyy-mm-dd format
7     WHEN LENGTH(Completion) = 7 THEN SUBSTR(Completion, 6, 2) -- Extract the month for yyyy-mm format
8     ELSE NULL -- Handle NULL values
9   END AS completion_month,
10  COUNT(*) AS number_of_completed_studies -- Count the number of completed studies
11
12 FROM
13   clinicaltrial_table
14 WHERE
15   -- Filter to include only rows where the status is 'COMPLETED' and the Completion date is in 2023
16   Status = 'COMPLETED'
17   AND
18     (LENGTH(Completion) = 10 AND SUBSTR(Completion, 1, 4) = '2023') -- yyyy-mm-dd format
19     OR (LENGTH(Completion) = 7 AND SUBSTR(Completion, 1, 4) = '2023') -- yyyy-mm format
20   )
21 GROUP BY
22   -- Group the results by the completion month
23   CASE
24     WHEN LENGTH(Completion) = 10 THEN SUBSTR(Completion, 6, 2) -- yyyy-mm-dd format
25     WHEN LENGTH(Completion) = 7 THEN SUBSTR(Completion, 6, 2) -- yyyy-mm format
26     ELSE NULL -- Handle NULL values
27   END
28 ORDER BY
29   -- Order the results by month
30   completion_month;
31

```

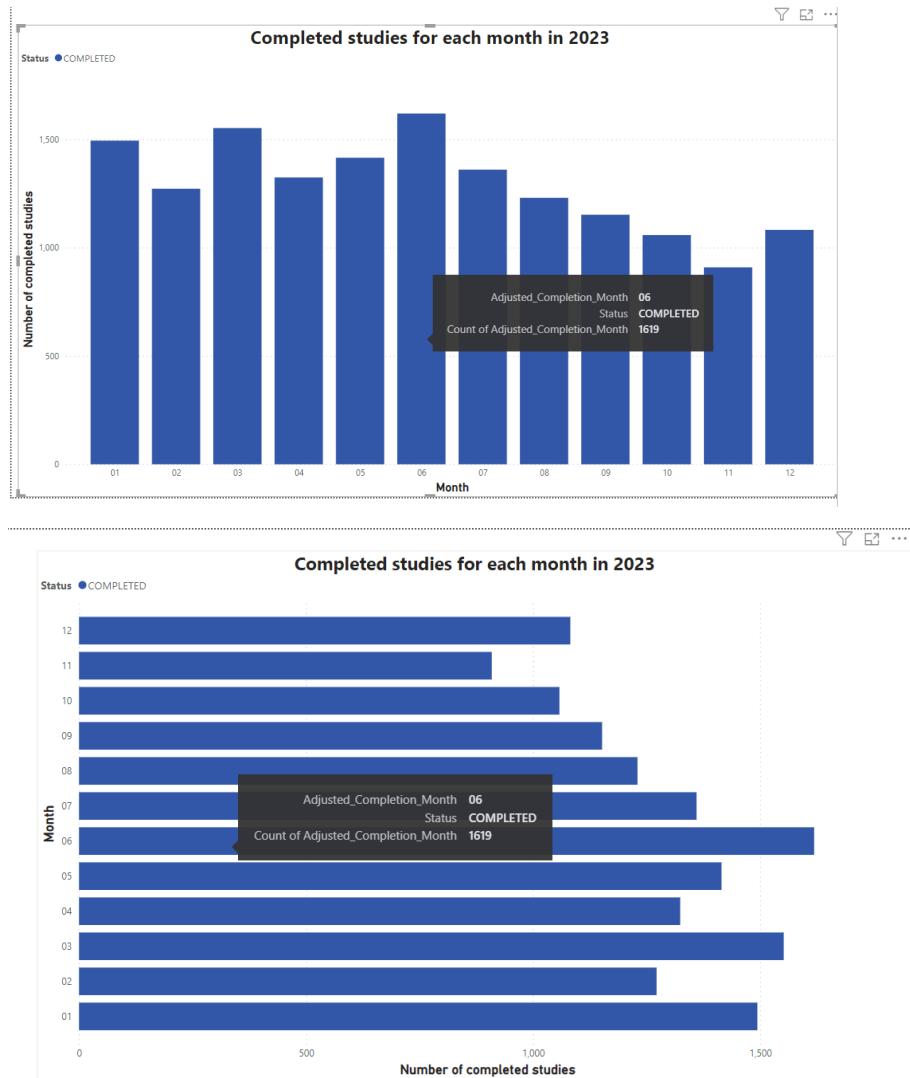
Table +

completion_month	number_of_completed_studies
1 01	1494
2 02	1272
3 03	1552
4 04	1324
5 05	1415
6 06	1619
7 07	1360
8 08	1230
9 09	1152
10 10	1058
11 11	909
12 12	1082

↓ 12 rows | 15.12 seconds runtime

Command took 15.12 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTE_TASK1_SQL_1

Visualizing the answer in POWER BI

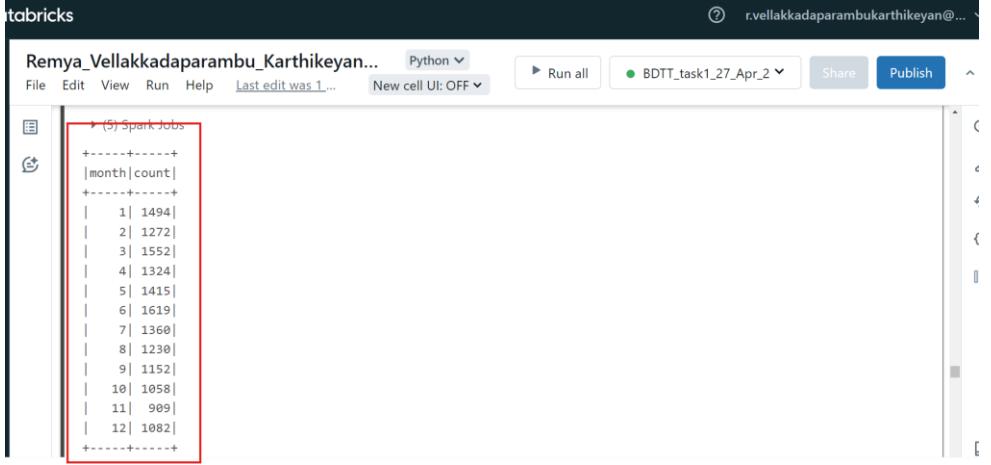


Main Idea: The visualization in Power BI utilized the following calculated columns

```
1 Adjusted_Completion =
2   IF(
3     ISBLANK(clinicaltrial_table[Completion]),
4     "0000-00-00",
5     IF(
6       LEN(clinicaltrial_table[Completion]) = 7,
7       clinicaltrial_table[Completion] & "-01",
8       clinicaltrial_table[Completion]
9     )
10   )
11
```

1 Adjusted_Completion_Month = MID([Adjusted_Completion], 6, 2)

Table 1.5 Discussion of Results

Result
 <pre>Remya_Vellakkadaparambu_Karthikeyan... Python File Edit View Run Help Last edit was 1... Run all Share Publish +---+---+ month count +---+---+ 1 1494 2 1272 3 1552 4 1324 5 1415 6 1619 7 1360 8 1230 9 1152 10 1058 11 909 12 1082 +---+---+</pre> <p>Insights: The monthly distribution of completed studies suggests that clinical trial completions may follow a seasonal pattern. The information can be valuable for healthcare institutions and researchers in planning and managing future clinical trials. Understanding the seasonal variation in completion rates may allow for more efficient resource allocation and better strategic planning for clinical research activities.</p>

1.11 Additional Analyses and Visualization in Dataframe Approach

The additional analyses executed in the dataframe approach and the visualizations performed using python's matplotlib, and plotly are explained below.

Analysis 1:

Question: Find the frequency distributions of "completed" and "recruiting" statuses in the clinicaltrial dataset

```

1  import pyspark.sql.functions as F
2  import reagent.utils as rt
3
4  def analyse_clinicaltrial_status(clinicaltrial_df, fileroot):
5
6      # Analyze the status distribution in a clinical trial DataFrame and plot the distribution.
7
8      Args:
9          clinicaltrial_df (DataFrame): A Spark DataFrame containing the clinical trial data.
10         fileroot (str): The root identifier of the file, e.g., 'clinicaltrial_2023', 'clinicaltrial_2024', etc.
11
12     Returns:
13         None
14
15     If fileroot == 'clinicaltrial_2023':
16         # Filter the DataFrame to include only records with 'completed' or 'recruiting' status
17         status_filtered_df = clinicaltrial_df.filter(
18             (F.col('clinicaltrial_df.Status') == 'completed') | 
19             (F.col('clinicaltrial_df.Status') == 'recruiting')
20         )
21
22         # Group by the 'Status' column and count the frequency of each status
23         status_distribution = status_filtered_df.groupBy("Status").count()
24
25         # Display the status distribution
26         status_distribution.show()
27
28         # Plot the status distribution
29         status_label = [row.Status for row in status_distribution.collect()]
30         status_counts = [row['count'] for row in status_distribution.collect()]
31
32         plt.figure(figsize=(8, 6))
33         plt.bar(status_label, status_counts, color='blue')
34         plt.xlabel('Status')
35         plt.ylabel('Count')
36         plt.title('Distribution of Study Statuses: Completed and Recruiting')
37         plt.show()
38     else:
39         print("This additional analysis is not applicable for fileroot = (fileroot)")
40
41     analyse_clinicaltrial_status(clinicaltrial_df, fileroot)
42

```

Main Idea: The code filters the clinicaltrial_df DataFrame to include only records where the study status is either "completed" or "recruiting." After filtering the DataFrame, the code groups the data by the Status column using the groupBy method. It then calculates the count of records for each status using the count function. The results are displayed.

Discussion of Result:

```

▶ (2) Spark Jobs
▶   status_filtered_df: pyspark.sql.dataframe.DataFrame = [id: string, Study Title: string ... 12 more fields]
▶   status_distribution: pyspark.sql.dataframe.DataFrame = [Status: string, count: long]

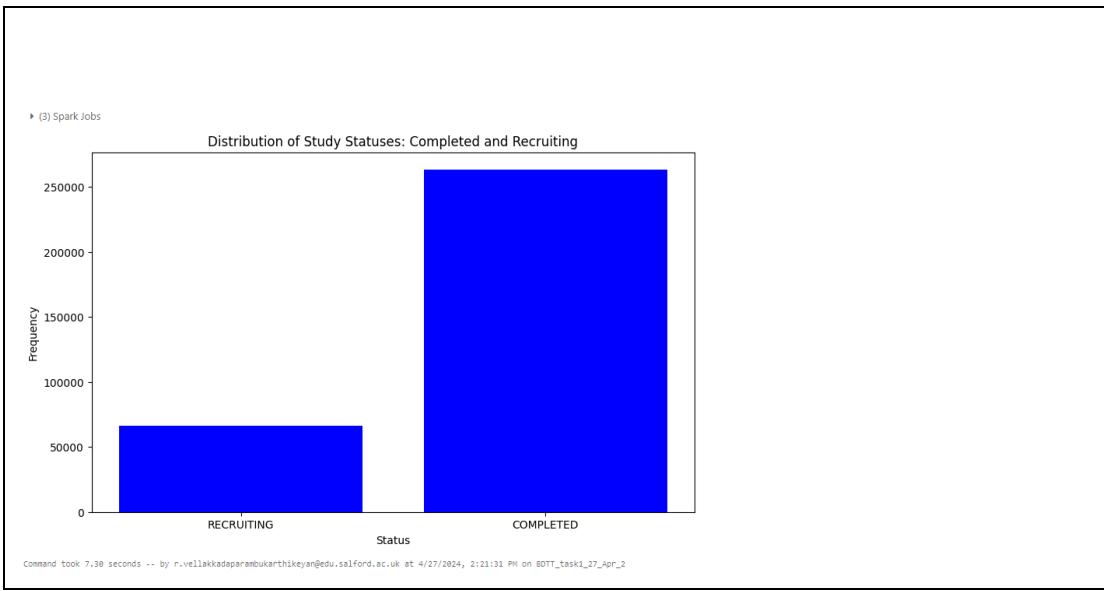
+-----+-----+
| Status| count|
+-----+-----+
|RECRUITING| 66158|
| COMPLETED| 263498|
+-----+-----+

Command took 6.03 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 2:21:31 PM on BDTE_task1_27_Apr_2

```

Insights : This analysis is useful for gaining insights into the current state of clinical trials in the dataset. By understanding the frequency distribution of these statuses, healthcare institutions and researchers can assess the progress of ongoing studies and identify areas where more resources or support may be needed. Additionally, comparing the counts of completed and recruiting studies can help gauge the overall pace and productivity of clinical research in the dataset.

Visualization 1:



Analysis 2:

Question : Find the distribution of clinical trial start dates by year and month (2015-2023)

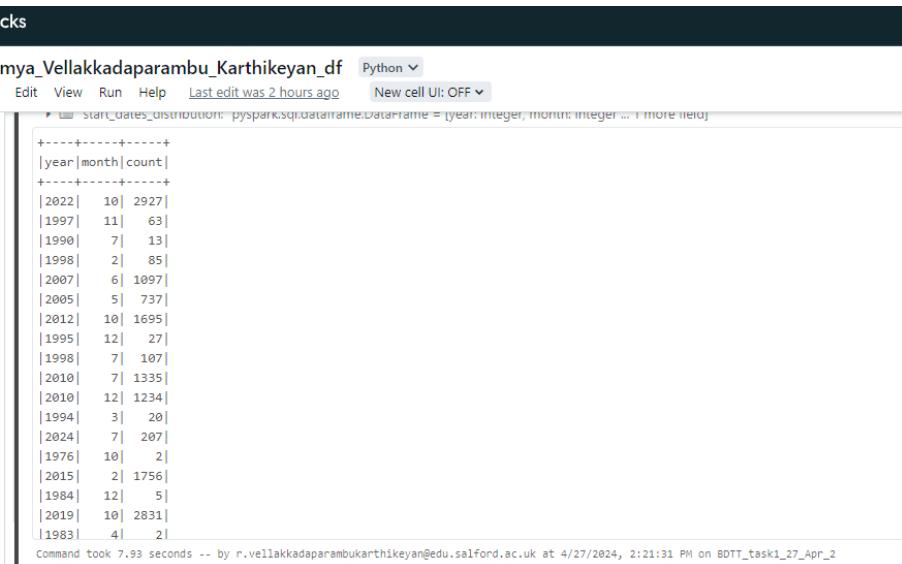
```

1  #!/usr/bin/python
2
3  import pyspark.sql.functions as Fnc
4  import metastore as msit
5  import os
6
7  def analyze_study_start_dates(clinicaltrial_df, fileroot):
8      """
9          Analyze and plot the distribution of study start dates in a clinical trial database.
10
11         Args:
12             clinicaltrial_df (DataFrame): A spark DataFrame containing the clinical trial data.
13             fileroot (str): The root identifier of the file, e.g., "clinicaltrial_N01", "clinicaltrial_N02", etc.
14
15         Returns:
16             None
17
18     """
19     if fileroot == "clinicaltrial_N01":
20         # Create a scatter plot showing the distribution of the "start" column
21         start_date_distribution = clinicaltrial_df.withColumn("year", Fnc.year("start")) \
22             .withColumn("month", Fnc.month("start")) \
23             .groupby("year", "month") \
24             .count()
25
26         # Display the results
27         start_date_distribution.show()
28
29         # Filter the data to include only the years from 2015 to 2023
30         filtered_data = start_date_distribution.filter(
31             (start_date_distribution.year >= 2015) & (start_date_distribution.year <= 2023)
32         )
33
34         # Create a scatter plot
35         plt.figure(figsize=(10, 4))
36
37         # Create a color palette with 12 colors (one for each month)
38         colors = sns.color_palette("husl", 12)
39
40         # Iterate through each month and plot a scatter plot with each month in different colors
41         for month in range(1, 13): # months from 1 to 12
42             # Filter data for the current month
43             monthly_data = filtered_data.filter(filtered_data.month == month)
44
45             # Extract year and count for the current month
46             years = [row['year'] for row in monthly_data.collect()]
47             counts = [row['count'] for row in monthly_data.collect()]
48
49             # Plot data points for the current month in a specific color
50             plt.scatter(years, counts, color=colors[month - 1], label=f'month {month}', alpha=0.7)
51
52             # Add labels and title
53             plt.xlabel('Year')
54             plt.ylabel('Number of Studies Started')
55             plt.title('Distribution of Study Start Dates from 2015 to 2023')
56
57             # Add a legend with a reduced font size
58             plt.legend(loc='upper left', fontsize=8)
59
60             # Show the plot
61             plt.show()
62
63     else:
64         print(f"This additional analysis is not applicable for fileroot = {fileroot}")
65
66
67     analyze_study_start_dates(clinicaltrial_df, fileroot)

```

Main Idea: The code begins by extracting the year and month from the 'Start' column of the clinical trial data. Once the year and month columns have been created, the code groups the data based on these columns. After grouping the data, the code counts the number of clinical trials in each group. The results are displayed.

Discussion of Result:

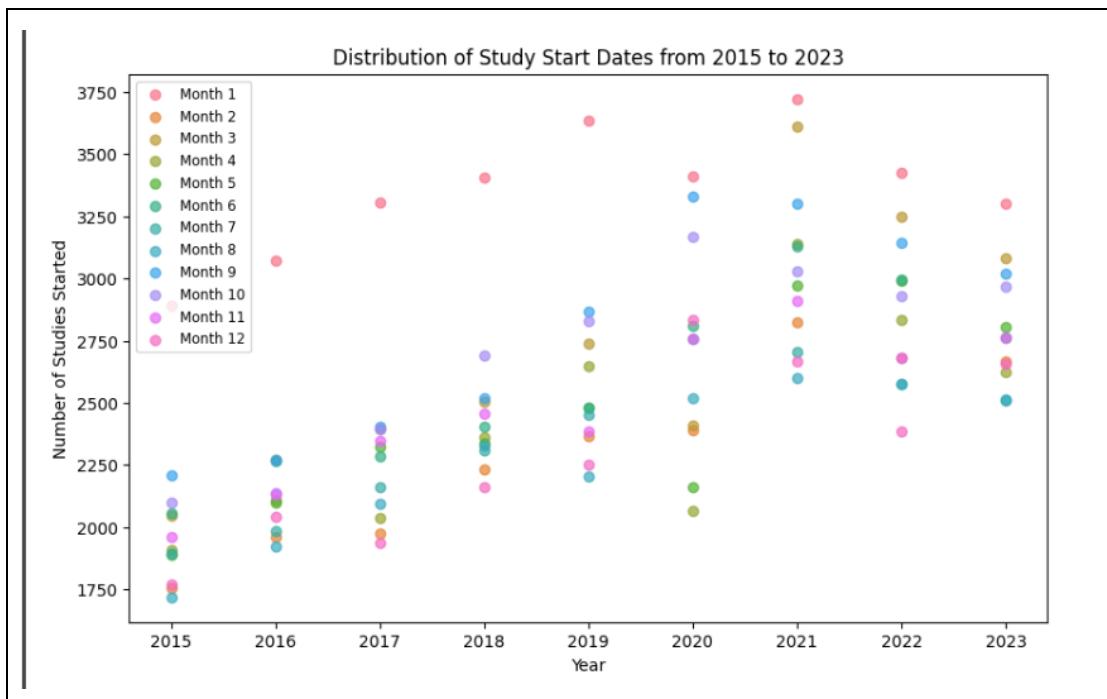


year	month	count
2022	10	2927
1997	11	63
1990	7	13
1998	2	85
2007	6	1097
2005	5	737
2012	10	1695
1995	12	27
1998	7	107
2010	7	1335
2010	12	1234
1994	3	20
2024	7	207
1976	10	2
2015	2	1756
1984	12	5
2019	10	2831
1983	4	21

Command took 7.93 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 2:21:31 PM on BDTE_task1_27_Apr_2

Insights: The analysis provide insights into when clinical trials are most frequently initiated. Understanding the trends in the initiation of new studies can be useful for planning and resource allocation in clinical research. Additionally, observing fluctuations over the years may suggest changes in the pace of research in priorities within the medical research community.

Visualization 2:



Analysis 3:

Question : What are the most common types of interventions in clinical trials?

```
1 import sys
2 import os
3 import pandas as pd
4 import numpy as np
5
6 def analyze_interventions(clinicaltrial_df, fileroot):
7
8     # Analyze the top 5 intervention types in a clinical trial DataFrame.
9
10    args:
11        clinicaltrial_df (DataFrame): a spark DataFrame containing the clinical trial data.
12        fileroot (str): the root identifier of the file, e.g., 'clinicaltrial_0002', 'clinicaltrial_0003', etc.
13        filename (str): the name of the output file.
14
15    Returns:
16        None
17
18    ---
19
20    # If fileroot == "clinicaltrial_0001"
21    # Step 1: Clean the "interventions" column by replacing all double quotes ("") with no space
22    cleaned_interventions_df = clinicaltrial_df.withColumn(
23        "interventions",
24        F.regexp_replace("interventions", '\"', '')
25    )
26
27    # Step 2: Extract the word before the colon (:)
28    cleaned_interventions_df = cleaned_interventions_df.withColumn(
29        "intervention_type",
30        F.split(cleaned_interventions_df["interventions"], ":").get(0)
31    )
32
33    # Step 3: Remove leading and trailing spaces from the intervention type
34    cleaned_interventions_df = cleaned_interventions_df.withColumn(
35        "intervention_type",
36        F.trim(cleaned_interventions_df["intervention_type"])
37    )
38
39    # Step 4: Group by "intervention_type" and count the occurrences, order by count in descending order
40    intervention_counts = cleaned_interventions_df.groupby("intervention_type").count().orderBy(Func.desc("count"))
41
42
43    # Step 5: Remove leading and trailing spaces from the intervention type
44    cleaned_interventions_df = cleaned_interventions_df.withColumn(
45        "intervention_type",
46        F.col("cleaned_interventions_df["intervention_type"]")
47    )
48
49
50    # Step 6: Group by "intervention_type" and count the occurrences, order by count in descending order
51    intervention_counts = cleaned_interventions_df.groupby("intervention_type").count()
52
53    # Step 7: Display only the top 5 counts
54    top_5_intervention_counts = intervention_counts.limit(5).collect()
55
56    # Convert the top 5 intervention counts to a pandas DataFrame
57    df = pd.DataFrame([
58        {"intervention_type": row["intervention_type"], "count": row["count"]} for row in top_5_intervention_counts
59    ])
60
61
62    # Create an intervention bar chart using Plotly
63    fig = px.bar(
64        df,
65        x="intervention_type",
66        y="count",
67        color_discrete_map={
68            "count": "#ff6347"
69        },
70        title="Top 5 Intervention Types and Their Counts"
71    )
72
73
74    # Customize the layout for better visibility
75    fig.update_layout(
76        font=dict(
77            family="Interstate, sans-serif",
78            size=14
79        ),
80        margin_title="Intervention types",
81        yaxis_title="Count"
82    )
83
84
85    # Display the interactive plot
86    fig.show()
87 else:
88     print("This additional analysis is not applicable for fileroot = " + fileroot)
89
90
91 analyze_interventions(clinicaltrial_df, fileroot)
```

Main Idea: After cleaning the 'Interventions' column, the code groups the data by the 'Intervention' column, counts the occurrences of each type, orders the results in descending order by count and then limits the output to the top 5 most common intervention types.

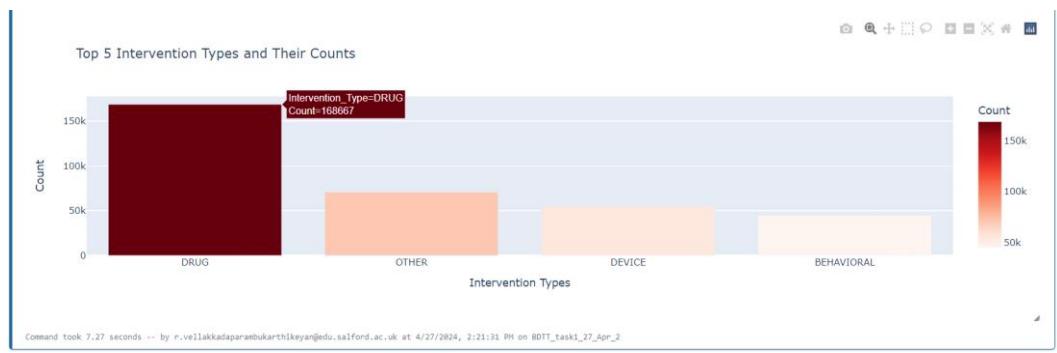
Discussion of Result:

```
70
71   analyze_interventions(clinicaltrial_df, fileroot)
72

▶ (2) Spark Jobs
Row(Intervention_Type='DRUG', count=168667)
Row(Intervention_Type='OTHER', count=70663)
Row(Intervention_Type='DEVICE', count=54989)
Row(Intervention_Type=None, count=47947)
Row(Intervention_Type='BEHAVIORAL', count=44874)
```

Insights : The results highlight the top 5 most common intervention types, which can be valuable for understanding trends in clinical research. Identifying the most frequent intervention types can inform decisions about resource allocation, research focus, and potential areas of innovation within the medical and pharmaceutical industries. It can also provide insights into the current state of clinical research and potential opportunities for further exploration.

Visualization 3:

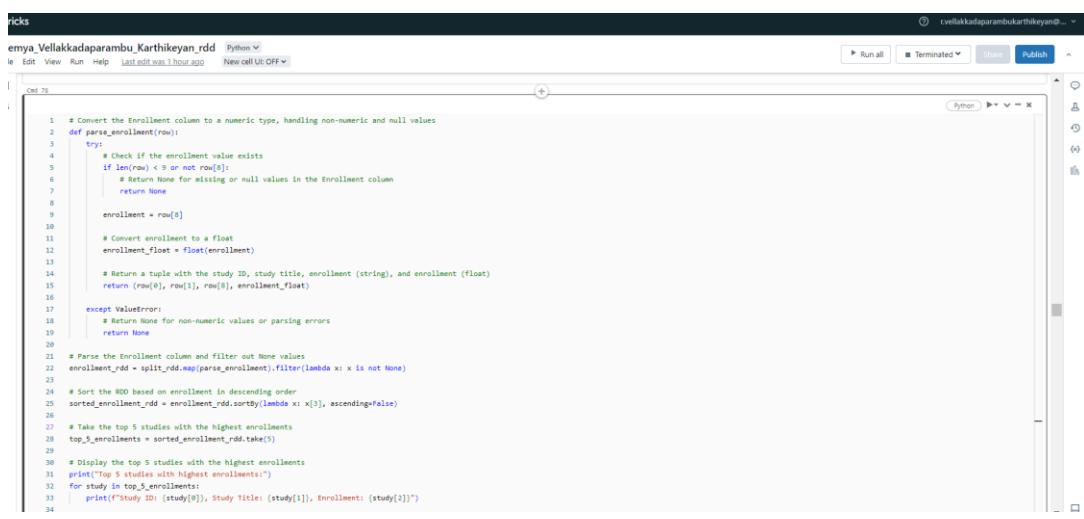


1.12 Additional Analyses and Visualization in RDD Approach

The additional analyses executed in the RDD approach, and the visualizations performed using python's matplotlib, and plotly are explained below.

Analysis 1:

Question : What are the top 5 clinical trials with the highest enrollments?



```
# Convert the Enrollment column to a numeric type, handling non-numeric and null values
def parse_enrollment(row):
    try:
        # Check if the enrollment value exists
        if len(row) < 3 or not row[2]:
            # Return None for missing or null values in the Enrollment column
            return None
        else:
            enrollment = row[2]
            # Convert enrollment to a float
            enrollment_float = float(enrollment)
            # Return a tuple with the study ID, study title, enrollment (string), and enrollment (float)
            return (row[0], row[1], row[2], enrollment_float)
    except ValueError:
        # Return None for non-numeric values or parsing errors
        return None

# Parse the Enrollment column and filter out None values
enrollment_rdd = split_rdd.map(parse_enrollment).filter(lambda x: x is not None)

# Sort the RDD based on enrollment in descending order
sorted_enrollment_rdd = enrollment_rdd.sortBy(lambda x: x[3], ascending=False)

# Take the top 5 studies with the highest enrollments
top_5_enrollments = sorted_enrollment_rdd.take(5)

# Display the top 5 studies with the highest enrollments
print("Top 5 studies with highest enrollments:")
for study in top_5_enrollments:
    print(f"Study ID: {study[0]}, Study Title: {study[1]}, Enrollment: {study[2]}")
```

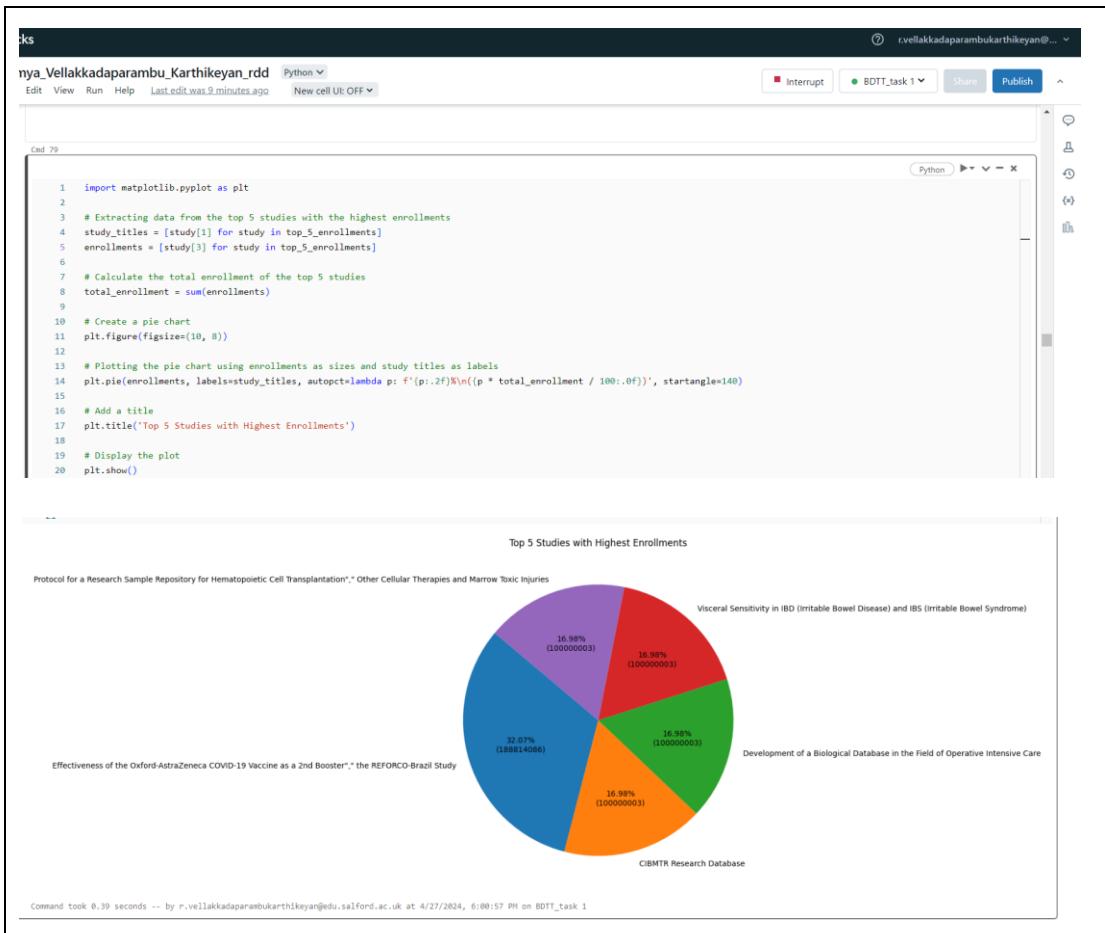
Main Idea: The code converts the 'Enrollment' column from a string type to a numeric type, filters out null values from the RDD, and filtered RDD is sorted in descending order based on the enrollment float value.

Discussion of Result:

```
Top 5 studies with highest enrollments:
Study ID: NCT05697705, Study Title: Effectiveness of the Oxford-AstraZeneca COVID-19 Vaccine as a 2nd Booster", " the REFORCO-Brazil Study, Enrollment: 188814085.0
Study ID: NCT01660809, Study Title: CIBNTR Research Database, Enrollment: 99999999.0
Study ID: NCT03014427, Study Title: Development of a Biological Database in the Field of Operative Intensive Care, Enrollment: 99999999.0
Study ID: NCT02421705, Study Title: Visceral Sensitivity in IBD (Irritable Bowel Disease) and IBS (Irritable Bowel Syndrome), Enrollment: 99999999.0
Study ID: NCT04920474, Study Title: Protocol for a Research Sample Repository for Hematopoietic Cell Transplantation", " Other Cellular Therapies and Marrow Toxic Injuries, Enrollment: 9
999999.0
```

Insights : The results highlight the studies with the highest enrollments, which may be indicative of large-scale clinical trials that can provide robust data for analysis. These findings can be valuable for researchers and decision-makers, as they offer insights into which studies have the potential for the most significant impact due to their large sample sizes.

Visualization 1:



Analysis 2:

Question: How many clinical trials have been sponsored by Mayo Clinic and Massachusetts General Hospital, respectively?



Main Idea: The code filters the RDD to include only studies sponsored by "Mayo Clinic" and "Massachusetts General Hospital" and takes the count.

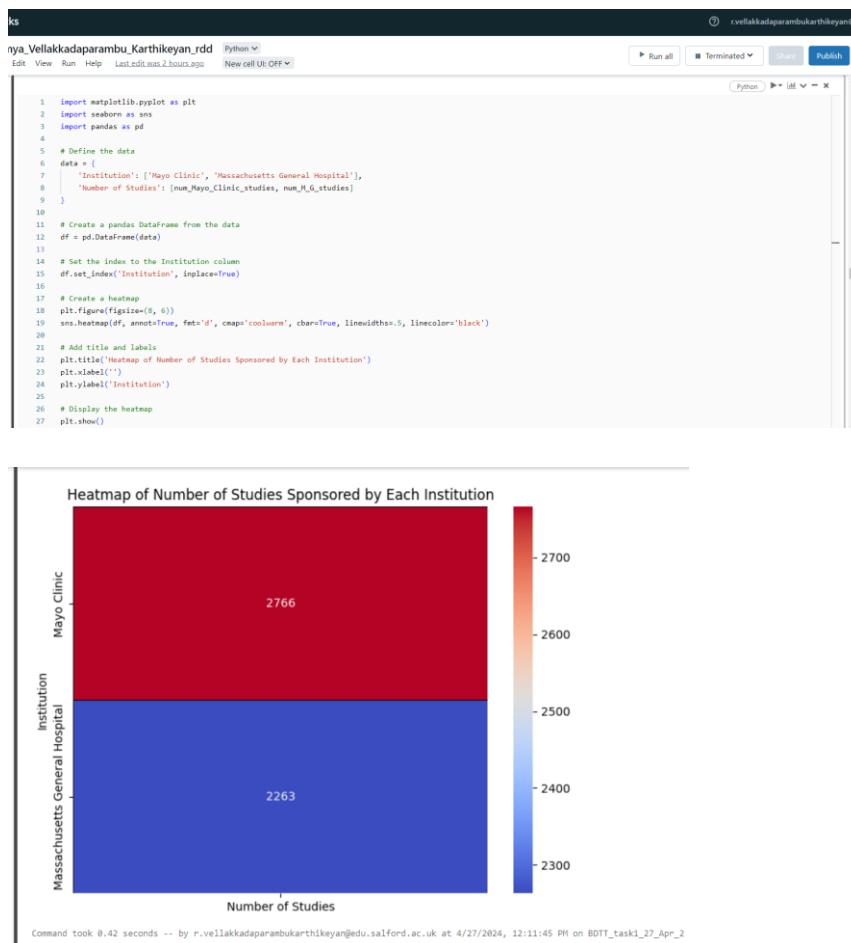
Discussion of Result:

► (2) Spark Jobs

```
Number of studies sponsored by Mayo Clinic: 2766  
Number of studies sponsored by Massachusetts General Hospital: 2263
```

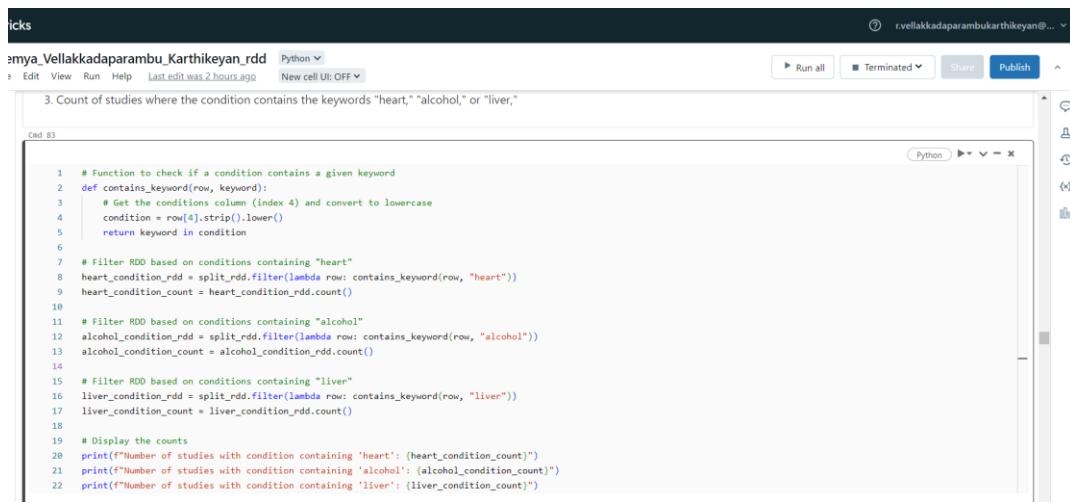
Insights : By comparing the number of studies sponsored by "Mayo Clinic" and "Massachusetts General Hospital" , researchers and decision-makers can gain an understanding of each institution's level of involvement in clinical research. This information can be valuable for assessing research priorities and potential collaborations.

Visualization 2:



Analysis 3:

Question: How many clinical trials involve conditions related to heart, alcohol, and liver, respectively?

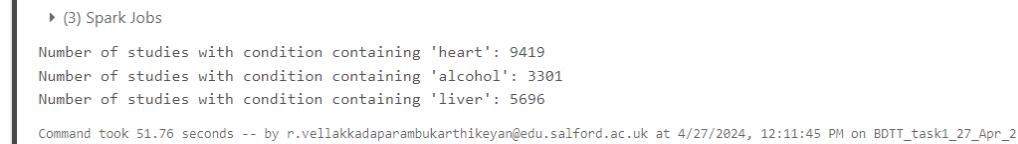


The screenshot shows a Jupyter Notebook interface with a Python kernel. The code cell contains the following Python script:

```
1 # Function to check if a condition contains a given keyword
2 def contains_keyword(row, keyword):
3     # Get the conditions column (index 4) and convert to lowercase
4     condition = row[4].strip().lower()
5     return keyword in condition
6
7 # Filter RDD based on conditions containing "heart"
8 heart_condition_rdd = split_rdd.filter(lambda row: contains_keyword(row, "heart"))
9 heart_condition_count = heart_condition_rdd.count()
10
11 # Filter RDD based on conditions containing "alcohol"
12 alcohol_condition_rdd = split_rdd.filter(lambda row: contains_keyword(row, "alcohol"))
13 alcohol_condition_count = alcohol_condition_rdd.count()
14
15 # Filter RDD based on conditions containing "liver"
16 liver_condition_rdd = split_rdd.filter(lambda row: contains_keyword(row, "liver"))
17 liver_condition_count = liver_condition_rdd.count()
18
19 # Display the counts
20 print("Number of studies with condition containing 'heart': (heart_condition_count)")
21 print("Number of studies with condition containing 'alcohol': (alcohol_condition_count)")
22 print("Number of studies with condition containing 'liver': (liver_condition_count)")
```

Main Idea: The code filters the RDD to include only studies where the conditions contain the keywords "heart", "alcohol", and "liver" and the count of each keyword based conditions are aggregated.

Discussion of Result:



▶ (3) Spark Jobs

```
Number of studies with condition containing 'heart': 9491
Number of studies with condition containing 'alcohol': 3301
Number of studies with condition containing 'liver': 5696

Command took 51.76 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
```

Insights : The results can help identify trends and potential gaps in clinical research related to the "heart", "alcohol", and "liver" conditions. A lower count of trials involving alcohol or liver conditions may suggest less focus in these areas or a potential need for more research.

Visualization 3:



1.13 Additional Analyses and Visualization in Spark SQL Approach

Analysis 1:

Question: Show the distribution of clinical trials across different funder types

```

maya_Vellakkadaparambu_Karthikyan_SQL SQL
Edit View Run Help Last edit was 35 minutes ago New cell UI: OFF
Run all 8DTT_TASK1_SQL_1 Publish
1 -- Query to find the distribution of clinical trials across different funder types
2
3 -- Funder_Type and its corresponding count of trials
4
5 Funder_Type,
6 COUNT(*) AS trial_count
7
8 FROM
9 clinicaltrial_table
10
11 GROUP BY
12 Funder_Type
13
14 ORDER BY
15 trial_count DESC
16
17
18

```

Main Idea: The code retrieves the distribution of clinical trials across different funder types, providing the count of trials for each type of funder. The results are grouped by the Funder_Type column and ordered by the count of trials in descending order, showing which funder types are associated with the most trials.

Discussion of Result:

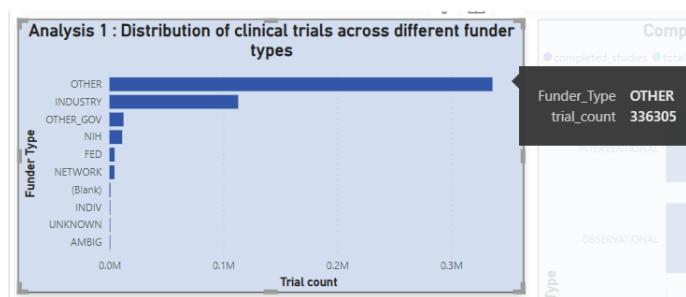
	Funder_Type	trial_count
1	OTHER	336305
2	INDUSTRY	113048
3	OTHER_GOV	12377
4	NIH	11193
5	FED	4526
6	NETWORK	4410
7	null	891
8	INDIV	575
9	UNKNOWN	94
10	AMBIG	3

10 rows | 6.36 seconds runtime

Command took 6.36 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on 8DTT_TASK1_SQL_1

Insights : The result can help to find which funder types are most prevalent in clinical trials, allowing you to identify which types of funders are more actively involved in the research. This information can be useful for understanding trends in funding sources and potentially targeting specific funder types for future research partnerships.

Visualization 1 in POWER BI:



Analysis 2:

Question: What are the most common study types in the clinicaltrial_table dataset, and how many of these studies have been completed?

```
Cmd 66
1 SELECT
2     Type AS study_type,
3     COUNT(*) AS total_studies,
4     SUM(CASE WHEN Status = 'COMPLETED' THEN 1 ELSE 0 END) AS completed_studies
5 FROM
6     clinicaltrial_table
7 GROUP BY
8     Type
9 ORDER BY
10    completed_studies DESC, total_studies DESC;
11
```

Main Idea: The code provides an analysis of the distribution of clinical trials across different study types, focusing on both the total number of studies and the number of completed studies.

Discussion of Result:

▶ (2) Spark Jobs

Table +

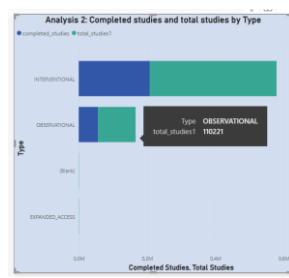
	study_type	total_studies	completed_studies
1	INTERVENTIONAL	371382	207447
2	OBSERVATIONAL	110221	56050
3	null	891	1
4	EXPANDED_ACCESS	928	0

↓ 4 rows | 6.50 seconds runtime

Command took 6.50 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTT_TASK1_SQL_1

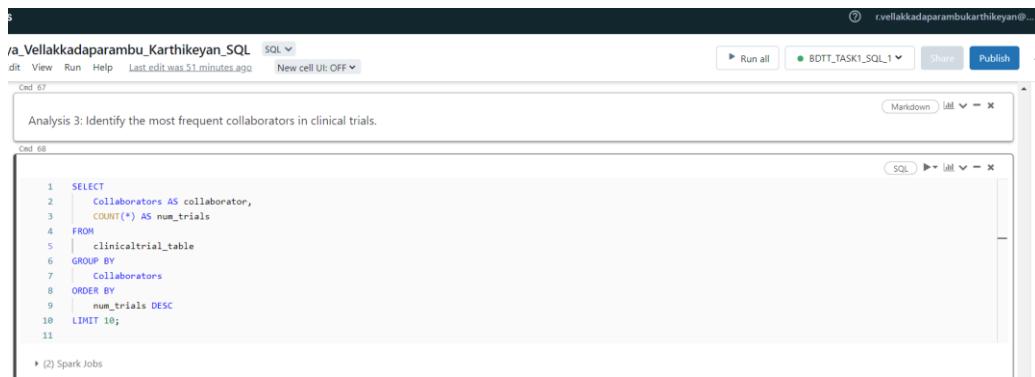
Insights : By ordering the results based on completed studies and total studies, it can identify which study types are more commonly completed, providing valuable insights into trends in research completion across various trial types.

Visualization 2 in POWER BI:



Analysis 3:

Question: Who are the top 10 collaborators involved in the clinical trials, and how many trials have they participated in?



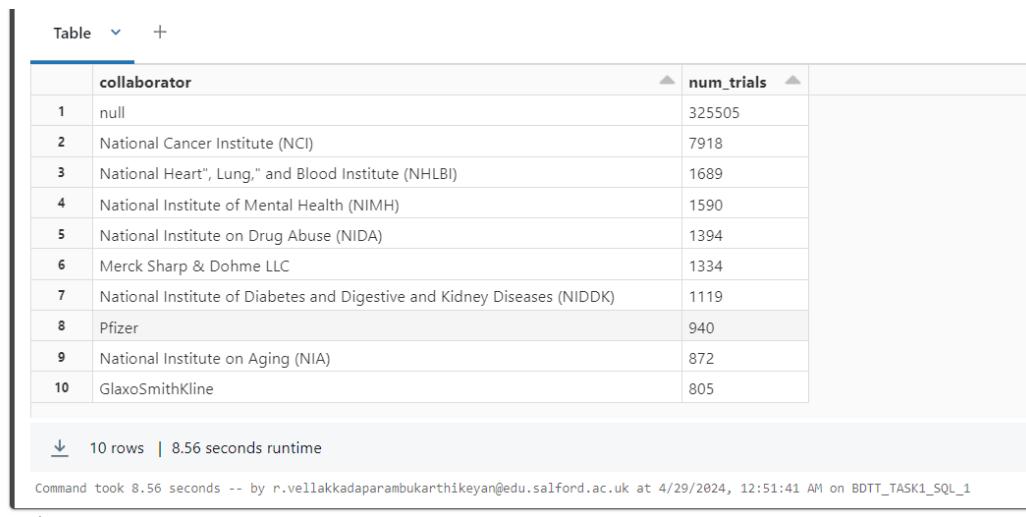
The screenshot shows a Jupyter Notebook cell with the following content:

```
5
/a_Vellakkadaparambu_Karthikeyan_SQL.ipynb SQL
dit View Run Help Last edit was 51 minutes ago New cell UI: OFF
Cell 67
Analysis 3: Identify the most frequent collaborators in clinical trials.
Cell 68
1 SELECT
2   Collaborators AS collaborator,
3   COUNT(*) AS num_trials
4 FROM
5   clinicaltrial_table
6 GROUP BY
7   Collaborators
8 ORDER BY
9   num_trials DESC
10 LIMIT 10;
11
```

Below the cell, it says "(2) Spark Jobs".

Main Idea: The code identifies the most active collaborators in the clinicaltrial_table dataset by counting the number of trials they have participated in and ranking them in descending order.

Discussion of Result:



The screenshot shows the results of the SQL query as a table:

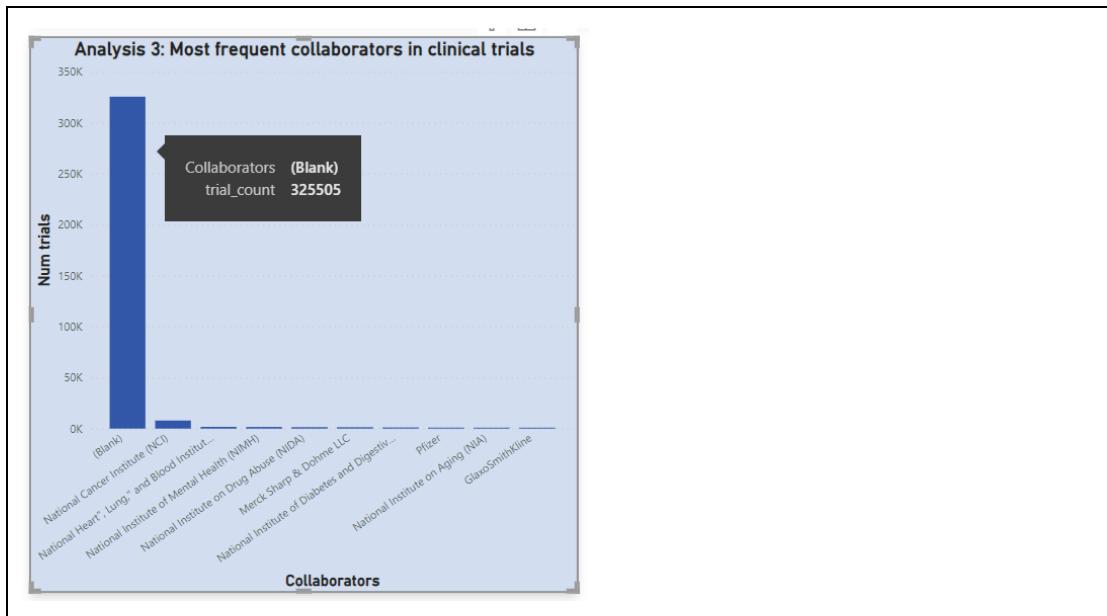
	collaborator	num_trials
1	null	325505
2	National Cancer Institute (NCI)	7918
3	National Heart, Lung, and Blood Institute (NHLBI)	1689
4	National Institute of Mental Health (NIMH)	1590
5	National Institute on Drug Abuse (NIDA)	1394
6	Merck Sharp & Dohme LLC	1334
7	National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK)	1119
8	Pfizer	940
9	National Institute on Aging (NIA)	872
10	GlaxoSmithKline	805

10 rows | 8.56 seconds runtime

Command took 8.56 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 12:51:41 AM on BDTE_TASK1_SQL_1

Insights : The result reveals which collaborators are most frequently involved in clinical trials, providing insights into their level of engagement and potential influence in the clinical trial landscape.

Visualization 3 in POWER BI:



1.14 Visualization of Clinical Trial 2023 in Power BI

A dashboard is created by writing a csv file from the clinicaltrial_df obtained from the data frame approach. The dashboard and details are given in Figure 1.6 and Table 1.6 respectively.

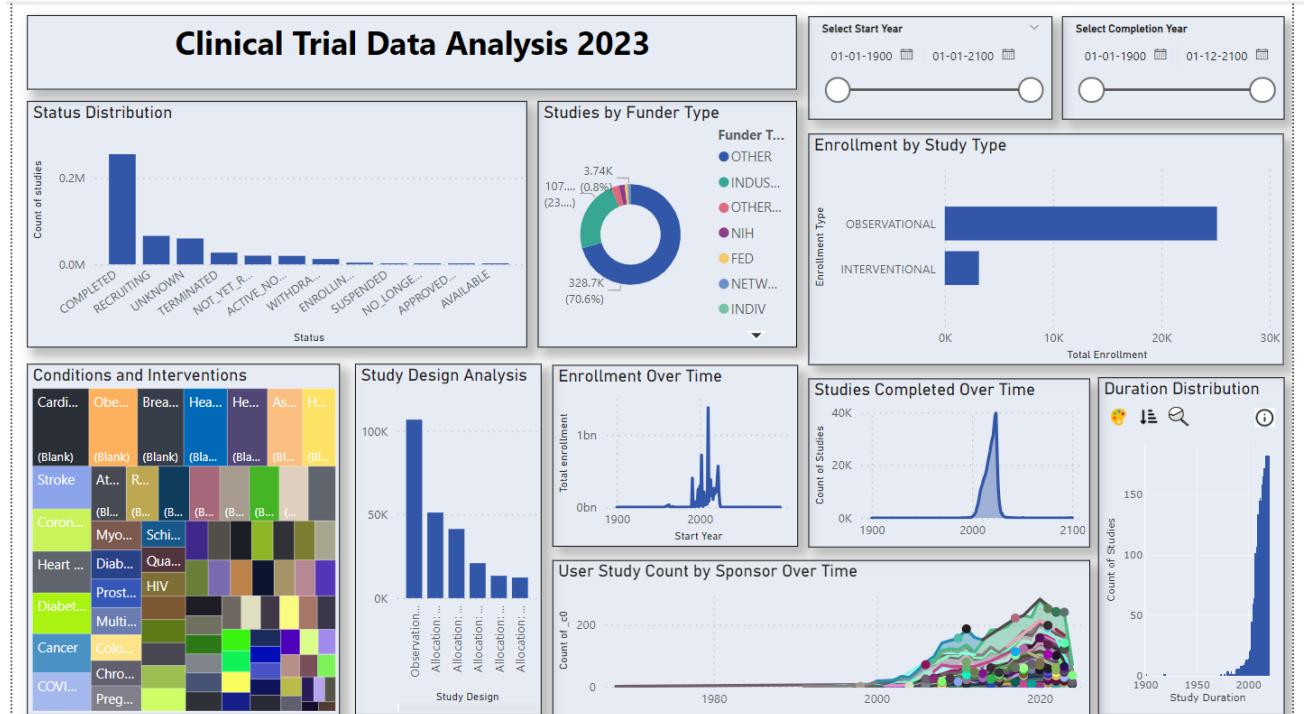
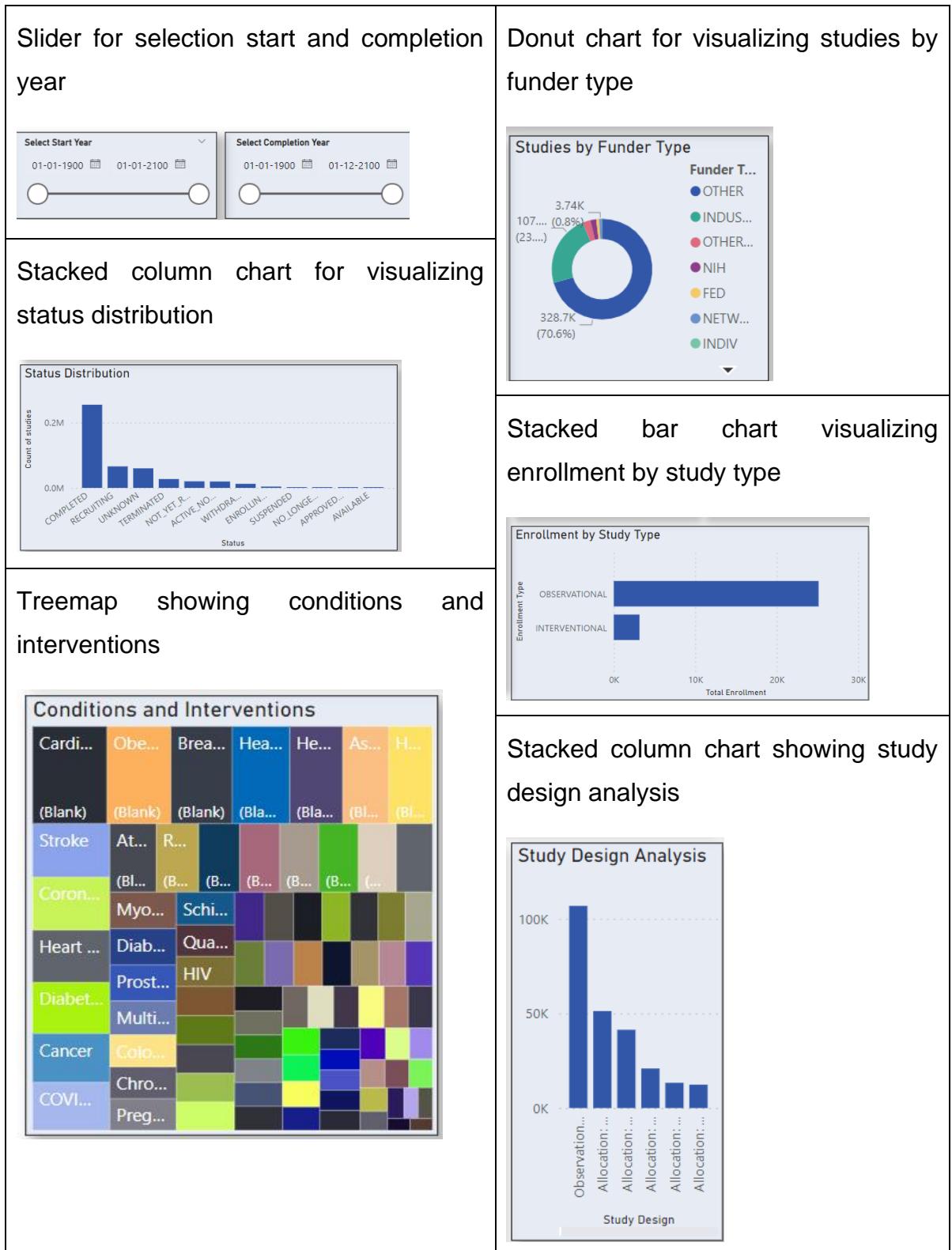
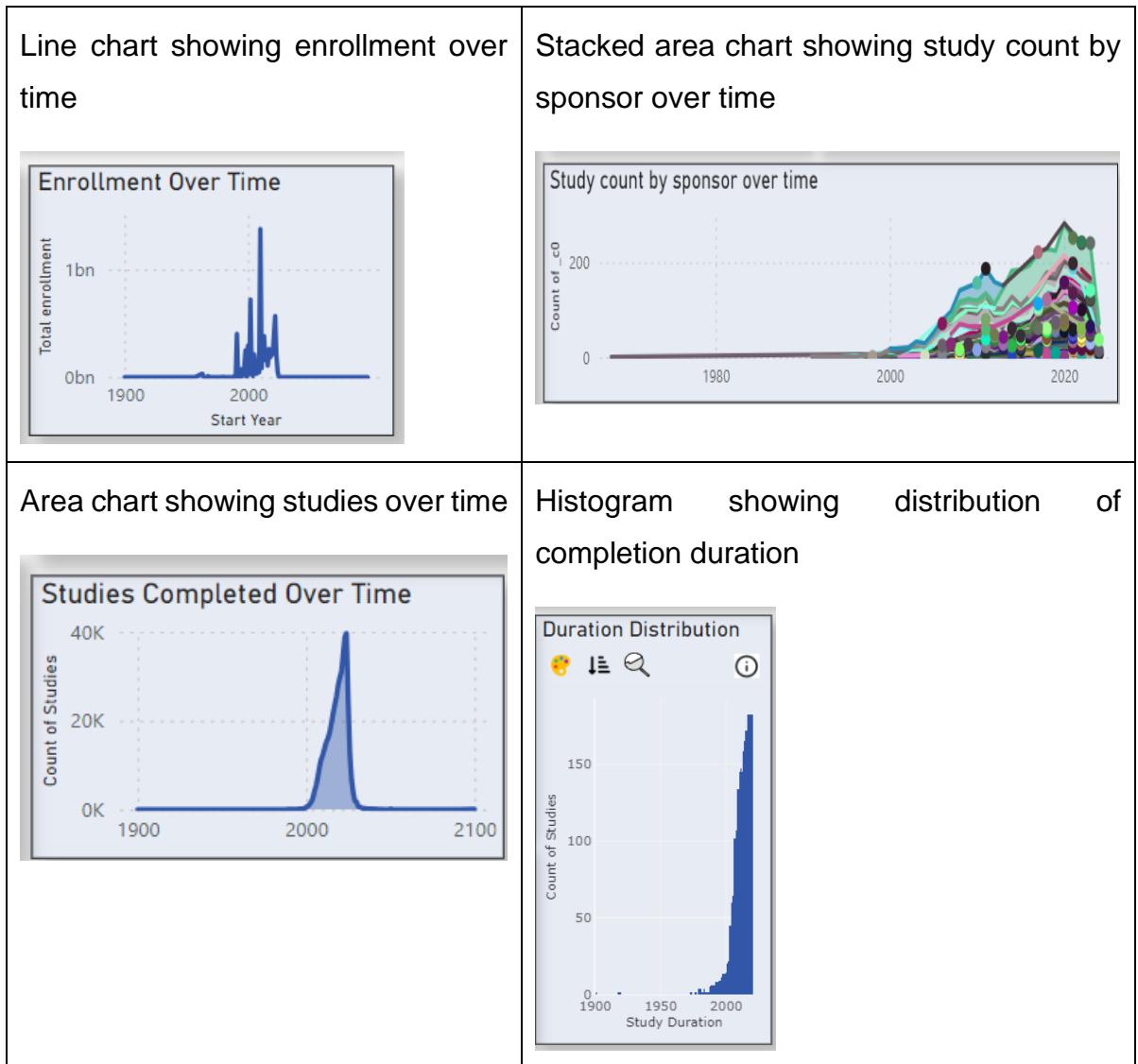


Figure 1.6 Clinical trial 2023 dashboard in Power BI

Table 1.6 Power BI dashboard details





1.15 Analyses of historical dataset Clinicaltrial_2020 and Clinicaltrial_2021

All the notebooks are general and reusable code for different versions of data. Here the results are shown the RDD approach as additional analyses in historical datasets are included in this approach. The same results for question 1- 5 are achieved in Dataframe and Spark approaches as well.

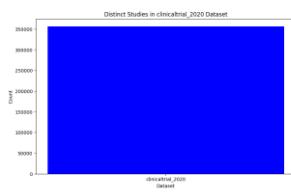
1.15.1 The number of studies in the dataset, checking distinct studies explicitly.

Clinicaltrial_2020

```
Cmd 107
1 # Counting the distinct studies by extracting the first element from each line
2 count = RDD.map(lambda line: line.split("|")[0]).distinct().count()
3
4 # Print the number of distinct studies found in the RDD
5 print(f"The number of distinct studies in {fileroot2} dataset is {count}.")
```

► (1) Spark Jobs
The number of distinct studies in clinicaltrial_2020 dataset is 356466.
Command took 4.06 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1

Visualization

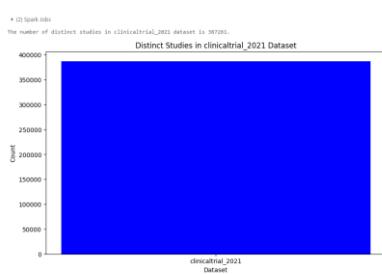


Clinicaltrial_2021

```
Cmd 107
1 # Counting the distinct studies by extracting the first element from each line
2 count = RDD.map(lambda line: line.split("|")[0]).distinct().count()
3
4 # Print the number of distinct studies found in the RDD
5 print(f"The number of distinct studies in {fileroot2} dataset is {count}.")
```

► (1) Spark Jobs
The number of distinct studies in clinicaltrial_2021 dataset is 387261.
Command took 3.83 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/27/2024, 6:40:12 PM on BDTT_task 1

Visualization



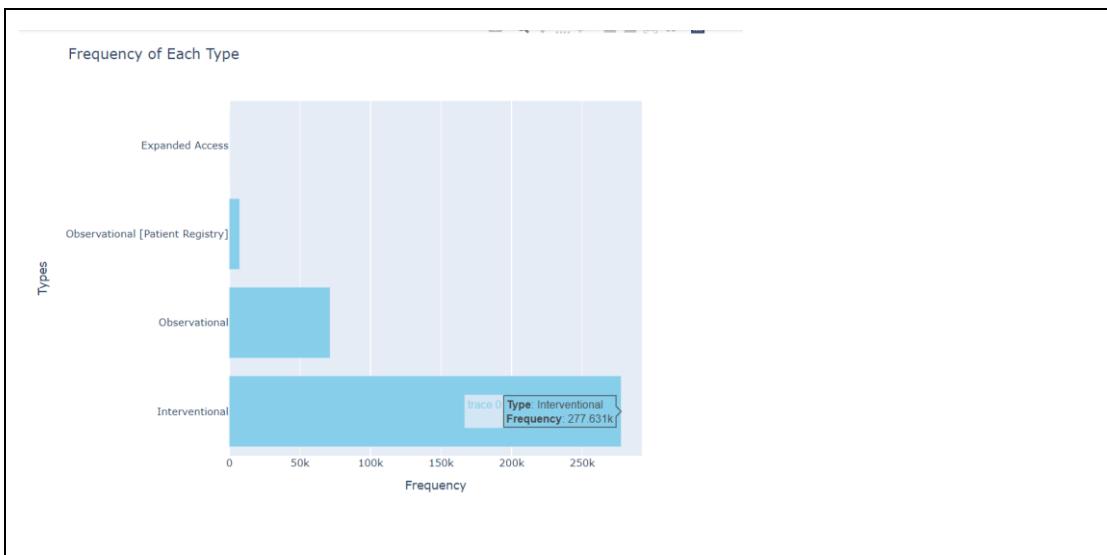
1.15.2 List all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type.

Clinicaltrial_2020

```
Cmd 110
1 # display the obtained result
2 print("All types with their frequencies:")
3 for types, count in all_type:
4     print(types, count)
```

All types with their frequencies:
Interventional 2798
Observational 71458
Observational [Patient Registry] 7332
Expanded Access 69
Command took 0.18 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/27/2024, 9:40:58 PM on BDTT_task 1

Visualization



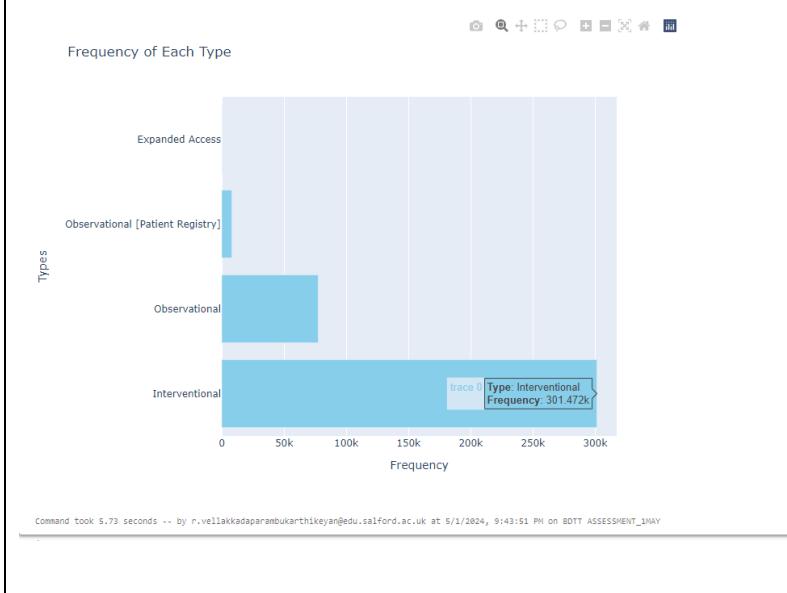
Clinicaltrial_2021

```
Cmd 115
1 # display the obtained result
2 print("All types with their frequencies:")
3 for types, count in all_Type:
4     print(types, count)

All types with their frequencies:
Interventional 301472
Observational 77540
Observational [Patient Registry] 8180
Expanded Access 69

Command took 0.89 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:48:12 PM on BDTT_task_1
```

Visualization



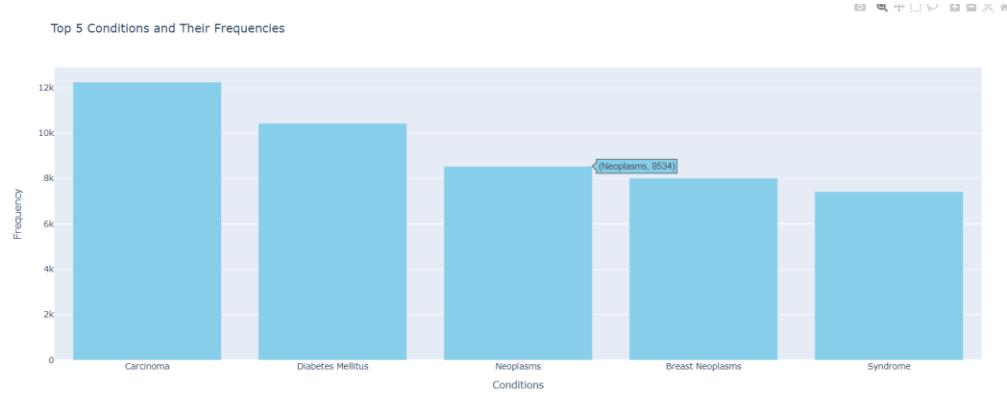
1.15.3 The top 5 conditions (from Conditions) with their frequencies.

Clinicaltrial_2020

```
Cmd 123 Python ►▼▼—x
1 # add up the frequencies for each condition and get the top 5 conditions
2 condition_top5 = condition_frequency.reduceByKey(lambda a, b: a + b).takeOrdered(5, key=lambda x: -x[1])
3
4 # display the corresponding result
5 print("Top 5 conditions and their frequencies:")
6 for condition, count in condition_top5:
7     print(condition, count)

▶ (1) Spark Jobs
Top 5 conditions and their frequencies:
Carcinoma 12245
Diabetes Mellitus 10425
Neoplasms 8534
Breast Neoplasms 8009
Syndrome 7419
Command took 2.01 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1
```

Visualization

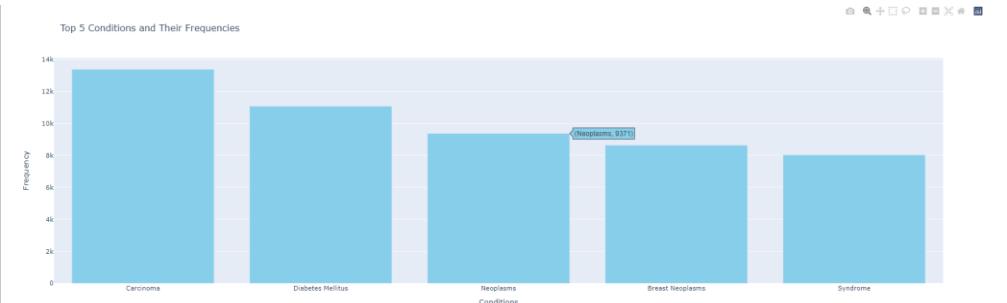


Clinicaltrial_2021

```
Cmd 123 Python ►▼▼—x
1 # add up the frequencies for each condition and get the top 5 conditions
2 condition_top5 = condition_frequency.reduceByKey(lambda a, b: a + b).takeOrdered(5, key=lambda x: -x[1])
3
4 # display the corresponding result
5 print("Top 5 conditions and their frequencies:")
6 for condition, count in condition_top5:
7     print(condition, count)

▶ (1) Spark Jobs
Top 5 conditions and their frequencies:
Carcinoma 13389
Diabetes Mellitus 11080
Neoplasms 9371
Breast Neoplasms 8640
Syndrome 8032
Command took 2.99 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/27/2024, 6:40:12 PM on BDTT_task 1
```

Visualization



1.15.4 Find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

Clinicaltrial_2020

```
Det 100
1 # Filter out all the sponsors that are not pharmaceutical companies
2 sponsor_non_pharma_RDD = sponsor_RDD.filter(lambda x: x not in pharma_Parent_RDD)
3
4 # Count the number of clinical trials sponsored by each non-pharmaceutical company
5 sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
6
7 # Get the top 10 non-pharma sponsors by number of clinical trials sponsored
8 sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])
9
10 # Display the corresponding results with header
11 print("Trial Year")
12 print("-----")
13 print("%s (%d)" % ("Sponsor", "Count"))
14 print("%s (%d)" % ("-----", "-----"))
15 for sponsor, count in sponsor_non_pharma_top10_RDD:
16     print("%s (%d)" % (sponsor, count))
17 print("%s (%d)" % ("-----", "-----"))

* (1) Spark Jobs
2020:
-----
|Sponsor          |Count|
-----+-----+
National Cancer Institute (NCI) | 1180|
M.D. Anderson Cancer Center    | 1120|
Merck Sharp & Dohme Corp.   | 1184|
Mayo Clinic           | 1097|
Assistance Publique - Hôpitaux de Paris | 2643|
Novartis Pharmaceuticals | 1982|
Massachusetts General Hospital | 1823|
Institut Universitaire De Technologie De Paris | 1806|
Roche                | 1781|
National Taiwan University Hospital | 1729|
-----+-----+
```

Command took 1.88 seconds -- by r.vellikkal@paravurathinayagam.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDFT_Task_1

Clinicaltrial_2021

```
Det 100
1 # Filter out all the sponsors that are not pharmaceutical companies
2 sponsor_non_pharma_RDD = sponsor_RDD.filter(lambda x: x not in pharma_Parent_RDD)
3
4 # Count the number of clinical trials sponsored by each non-pharmaceutical company
5 sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)
6
7 # Get the top 10 non-pharma sponsors by number of clinical trials sponsored
8 sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])
9
10 # Display the corresponding results with header
11 print("Trial Year")
12 print("-----")
13 print("%s (%d)" % ("Sponsor", "Count"))
14 print("%s (%d)" % ("-----", "-----"))
15 for sponsor, count in sponsor_non_pharma_top10_RDD:
16     print("%s (%d)" % (sponsor, count))
17 print("%s (%d)" % ("-----", "-----"))

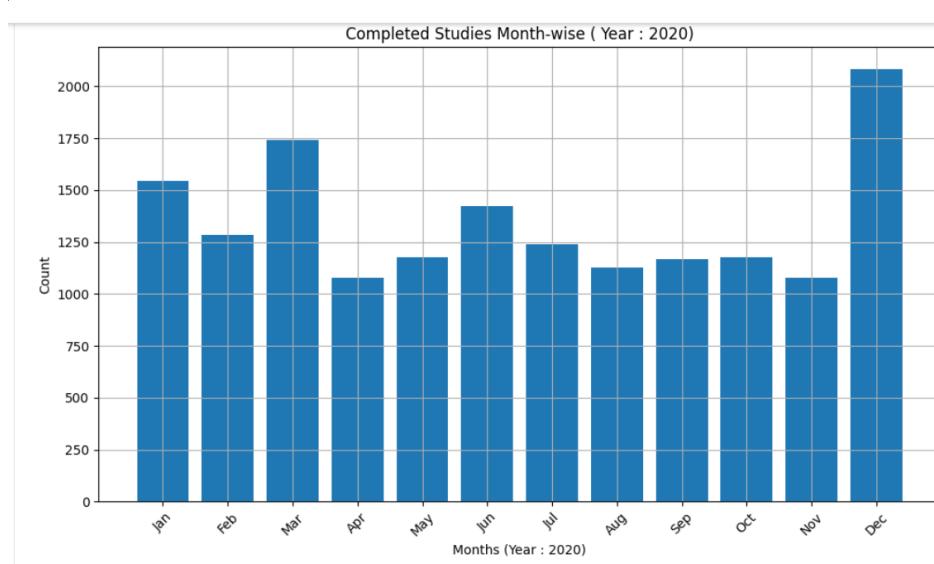
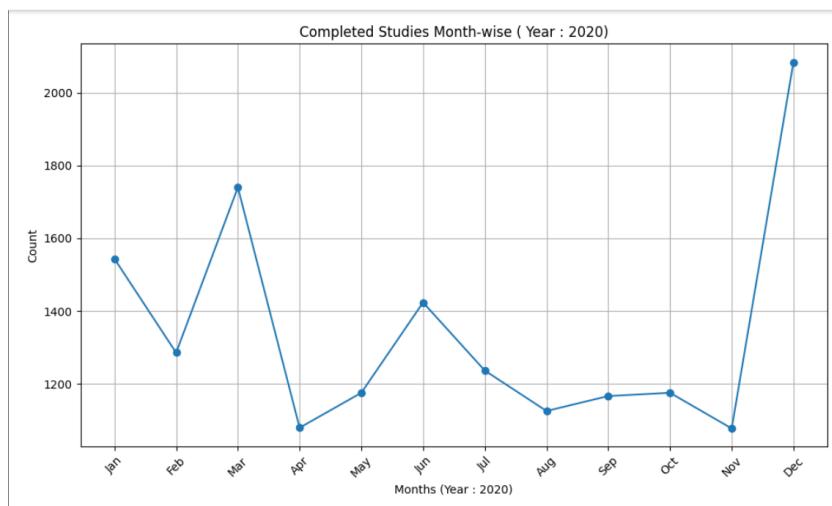
* (1) Spark Jobs
2021:
-----
|Sponsor          |Count|
-----+-----+
National Cancer Institute (NCI) | 3218|
M.D. Anderson Cancer Center    | 2441|
Assistance Publique - Hôpitaux de Paris | 2360|
Mayo Clinic           | 2388|
Merck Sharp & Dohme Corp.   | 2354|
Novartis Pharmaceuticals | 2088|
Massachusetts General Hospital | 1971|
Takeda               | 1950|
JoulePharm-La Roche | 1828|
-----+-----+
```

Command took 2.81 seconds -- by r.vellikkal@paravurathinayagam.salford.ac.uk at 4/27/2024, 6:01:12 PM on BDFT_Task_1

1.15.5 Plot number of completed studies for each month in 2023. Include visualization as well as a table of all the values.

Clinicaltrial_2020

Visualization



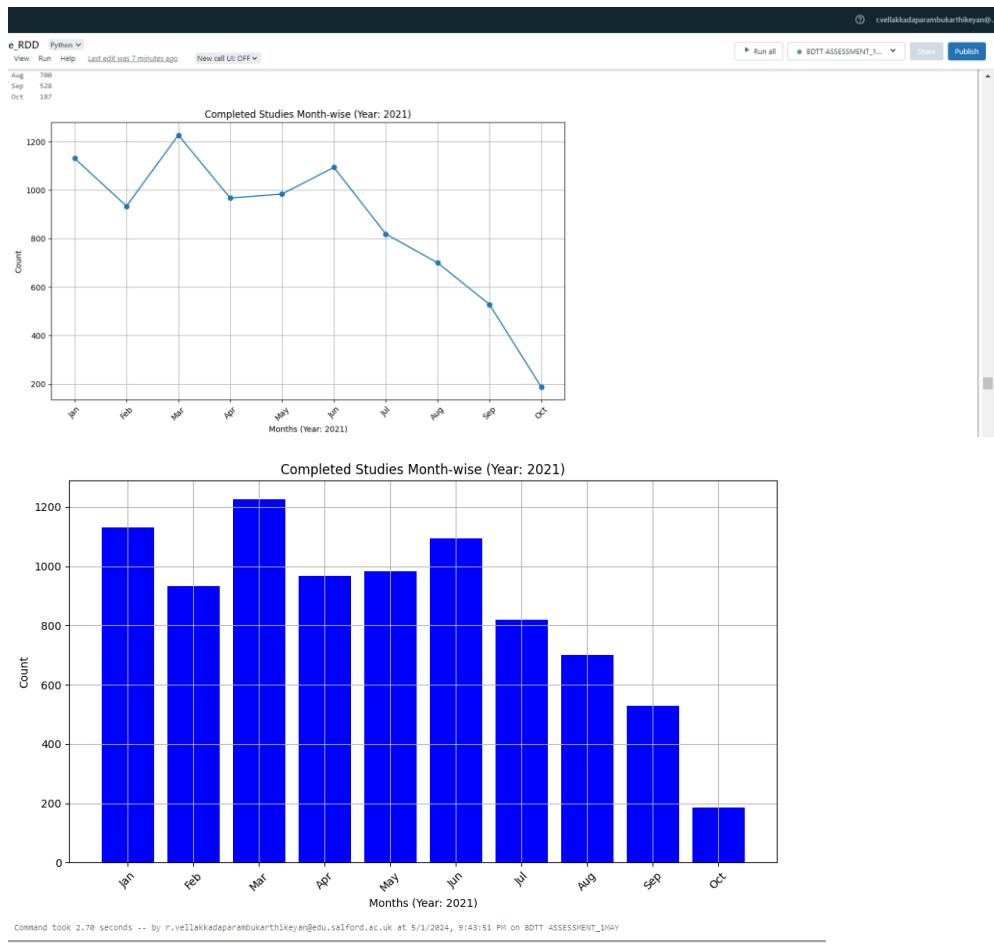
Clinicaltrial 2021

```

5   status_Filter_RDD = RDD.filter(lambda x: Status in x.split("|")[2] and trialYear in x.split("|")[4])
6
7   # Map each row to a tuple containing the month and year of the study completion date
8   # trialYear_RDD = status_Filter_RDD.map(lambda x: (datetime.strptime(x[0][4], "%Y-%m"), datetime.strptime(x[0][4], "%Y-%m").strftime("%Y")))
9
10  # Count the number of studies completed for each month
11  monthwise_Count_RDD = trialYear_RDD.countByValue()
12
13  # Sorting the counts by month
14  monthwise_Count_Sort = sorted(monthwise_Count_RDD.items(), key=lambda x: datetime.strptime(x[0][4], "%Y-%m").month)
15
16  # Extract the counts for each month
17  num_Counts = [(month, year, count) for (month, year), count in monthwise_Count_Sort if year == trialYear]
18
19  # Create a list of months
20  months = [month for (month, year), count in monthwise_Count_Sort if year == trialYear]
21
22  return months, num_Counts
23
24  # Get the counts for completed studies month-wise
25  monthwise_Count = completedStudiesMonthwise("Completed", trial_Year2)
26
27  # Print months and counts in two columns
28  print("Month\tCount")
29  for month, count in zip(monthwise_Count[0], monthwise_Count[1]):
30      print(f'{month}\t{count}')
31
32
33  # (1) Spark Jobs
34
35  Month\tCount
36  Jan\t1311
37  Feb\t954
38  Mar\t1227
39  Apr\t942
40  May\t664
41  Jun\t1094
42  Jul\t749
43  Aug\t700
44  Sep\t528
45  Oct\t187

```

Visualization



1.16 Additional Analysis and Visualization for historical dataset (codes - cmd 49 to 53 in the RDD notebook)

The additional visualization applies for both historical dataset for years 2020 and 2021, as the code is reusable. The visualizations shown here are for trial year 2020.

Listing studies started in a given year (2005), with study Id and sponsor details

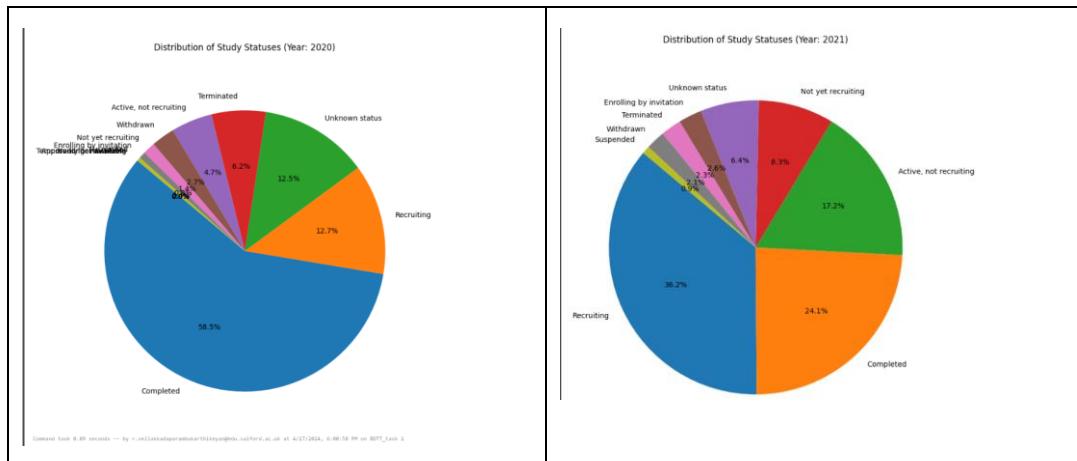
```
Cmd 143
1 # Define the user-defined function to list studies started in a specific year
2 def studies_started_in_year(data_rdd, year):
3     # Filter RDD to select only the rows where the study start year matches the input year
4     studies_year = data_rdd.filter(lambda line: year in line.split("\t")[3])
5
6     # Extract study ID, sponsor, and start date from each row
7     study_info = studies_year.map(lambda line: (line.split("\t")[0], line.split("\t")[1], line.split("\t")[3]))
8
9     # Return the study information
10    return study_info
11
12    # Input the year
13    input_year = "2005" # Example: Input year as a string
14
15    # Call the function with the input year and retrieve the study information
16    studies_year_info = studies_started_in_year(RDD, input_year)
17
18    # Print the study information
19    for study_id, sponsor, start_date in studies_year_info.collect():
20        print(f"Study ID: {study_id}, Sponsor: {sponsor}, Start Date: {start_date}")
21
```

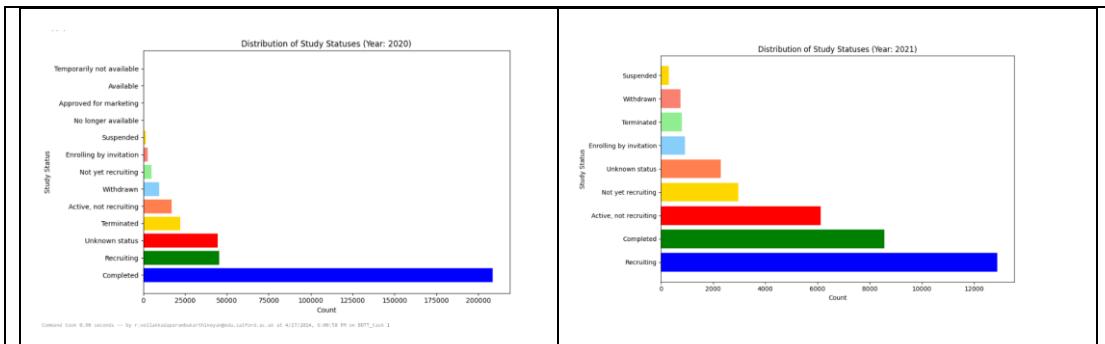
► (1) Spark Jobs

Study ID	Sponsor	Start Date
NCT02750828	The University of Hong Kong	Aug 2005
NCT02754824	Werner Anderl	Sep 2005
NCT02758644	Joaquin de Haro, M.D.	Jan 2005
NCT00370953	Mayo Clinic	Jun 2005
NCT00374647	Celgene Corporation	Sep 2005
NCT00374785	Aspirin	Jan 2005
NCT00375958	Therdyne	Aug 2005
NCT00378375	National Taiwan University Hospital	Mar 2005
NCT00379933	Ann & Robert H Lurie Children's Hospital of Chicago	Sep 2005
NCT00380042	National Taiwan University Hospital	Aug 2005
NCT00386475	University of Medicine and Dentistry of New Jersey	Jan 2005
NCT00387087	University of Pittsburgh	Aug 2005
NCT00387583	The University of Texas Health Science Center, Houston	Sep 2005
NCT00388120	University of Medicine and Dentistry of New Jersey	May 2005
NCT00388470	National Taiwan University Hospital	Aug 2005
NCT00388740	National Taiwan University Hospital	Jan 2005
NCT00389568	National Institute of Allergy and Infectious Diseases (NIAID)	Jun 2005
NCT00392219	National Taiwan University Hospital	Mar 2005
NCT00397712	University of Minnesota	Jun 2005

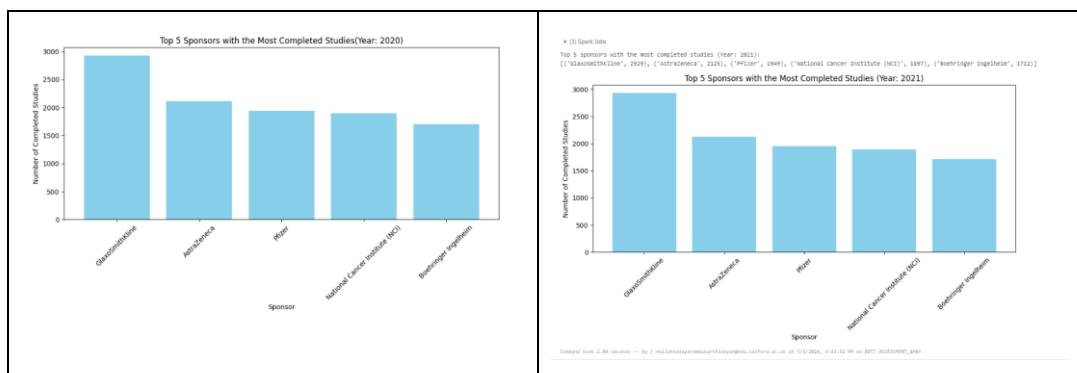
Study ID	Sponsor	Start Date
NCT03895404	McMaster University	Sep 2021
NCT03909225	University of Pisa	Jan 2021
NCT03909795	Pullectra, Inc.	Dec 2021
NCT03909900	Mannheim Medical Clinic	Apr 2021
NCT03926118	Martin Reili	Apr 2021
NCT03925194	Heidelberg University	Oct 2021
NCT03923097	Chinese University of Hong Kong	Oct 2021
NCT03926339	National Heart, Lung, and Blood Institute (NHLBI)	Nov 2021
NCT03927778	W. Lee Dugger Cancer Center Research Institute	Sep 2021
NCT03929005	Willentrop Immunotherapeutics, Inc.	May 2021
NCT03931515	National Institute of Allergy and Infectious Diseases (NIAID)	Nov 2021
NCT03932275	Carmen Giacomantonio	Sep 2021
NCT03927066	Mayo Clinic	May 2021
NCT03925077	University of Alabama at Birmingham	Feb 2021
NCT03925078	University of Michigan Hospital	May 2021
NCT03929329	Hospices Civilis de Lyon	Apr 2021
NCT03929462	Laboratoire TDM-IMAG	Mar 2021
NCT03926520	Brent Forrester	Jan 2021
NCT0392625	Chris Lascola, MD	Dec 2021

Distribution of Study Statuses





Top 5 sponsors with the most completed studies



1.17 Conclusion

Parameters	DF	RDD	SQL
Questions 1 – 5 (Same answer obtained in all three approaches)	Yes	Yes	Yes
Visualization (No built-in visualizations utilized in this report)	Matplotlib, Plotly		Power BI
Unzipping inside Databricks system	Yes	Yes	Yes
Proper cleanup procedures to remove any unnecessary files and folders	Yes	Yes	Yes

3 further analyses of the data (total 9 further analyses)	Yes	Yes	Yes
Dashboard for clinical trial 2023 dataset in Power BI	Yes (POWER BI)		
Reusable codes (file root parameterization used)	Yes	Yes	Yes
Solving the problems like defining and using user defined functions	Yes	Yes	No
Successfully implementing new Spark functions	Yes (E.G : REGEXP_REPLACE, TRIM, LEN(), APPEND(), CAST(), etc		
Visualization for all 9 further analyses	Yes	Yes	Yes

The analysis on clinical trials data is successfully executed in Databricks using data frame, RDD and Spark SQL approach. The analysis provides a comprehensive overview of the clinical trials landscape and offer valuable insights for strategic decision-making in the healthcare and research sectors.

Task 2: Building a Collaborative Filtering Recommender System for Users based on ‘Purchase’ and ‘Play’ Behavior

2.1 Introduction

The online video game distribution services have revolutionized the gaming industry by offering a convenient and accessible way to buy games without the need for physical copies. Users browse a wide selection of games across various genres, and do the same with exclusive deals and discounts. The service usually combines features such as cloud storage for game saves, automatic updates, and social features like opponent or friend matchmakings. The online video game distribution services have become an integral part of modern gaming, offering a streamlined and efficient way to enjoy the latest and greatest titles.

The report presents a comprehensive exploratory analysis and implementation of a recommender system for Steam, an online video game distribution service. The dataset utilized in this project, ‘**steam-200k.csv**’, contains information about different members’ interactions with various games on the Steam platform. Specifically, it includes data about the games - members have purchased and played, as well as the number of hours spent playing each game.

The dataset consists of four columns: a unique identifier for each member, the name of the game they purchased or played, their behavior (either ‘purchase’ or ‘play’), and a value corresponding to either the number of hours played or a purchase indicator.

The **main objective** of the report is :

To leverage both ‘purchase’ and ‘play’ behaviors as implicit user feedback to train a collaborative filtering recommender system using MLlib in Databricks

platform. The recommender system design is based on three approaches and they are:

- 1) Based on only '*purchase*' member behavior
- 2) Based on only '*play*' member behavior
- 3) Based on both '*purchase*' and '*play*' member behavior

A detailed exploration of the data, outlining the structure and characteristics of the dataset is carried out in databricks and Power BI prior to building machine learning based recommender system. The sections below details the steps on loading the dataset into a Spark DataFrame, pre-processing the data, and splitting it into training and test sets. It also discusses the choice of collaborative filtering algorithm based on '*purchase*' and '*play*' behavior and provides an analysis of the model's performance based on various evaluation metrics.

Furthermore, the report explores recommendations generated by the model, providing insights into how users' gaming experiences can be enhanced based on their past behavior and preferences. Through this analysis, the report aims to demonstrate the potential of recommender systems in the context of online gaming platforms and the advantages they offer for personalizing user experiences.

2.2 Set up Requirements

The following points and figures gives the set up requirements for the ML algorithm implementation in Databricks.

After logging in to the Databricks workspace, perform the following :

- Create a new cluster or use an existing cluster with the necessary resources (CPU, memory, etc.) and with appropriate runtime version for the type of work you want to perform (e.g., Apache Spark, ML runtime).
- Upload or import your data to the Databricks environment (e.g., from your local machine, cloud storage, or a database).

- Create a notebook or a job where you will write your ML code.
- In your notebook, import the necessary packages and libraries for the task (e.g., **pyspark**, **mlflow**, etc.).

Figure 2.1 shows the related figures

Creating cluster

Databricks

BDTT_TASK_2_20_APY_2

Configuration Notebooks (0) Libraries Event log Spark UI Driver logs Metrics Apps Spark compute UI - Master More ... Terminate Edit

Databricks Runtime Version
14.3 LTS 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.

Instances Spark JDBC/ODBC

Availability zone us-west-2b

Cluster details

Databricks

Compute

All-purpose compute Job compute

Filter compute you have access to Created by

State	Name	Runtime	Active memory	Active cores	Active DBU / h	Source	Creator	Notebooks	Actions
Running	BDTT_TASK_2_20_APY_2	14.3 ML	15 GB	2 cores	1	UI	r.vellakkadaparambukarthikeyan...	-	⋮

Uploading .csv file

Databricks

Create New Table

Data source Upload File S3 Other Data Sources

DBFS Target Directory /FileStore/tables/ (optional) Select

Files

steam-200k.csv
8.1 MB Remove file

File uploaded to /FileStore/tables/steam_200k-1.csv

Create Table with UI Create Table in Notebook

Creating notebook

The screenshot shows the Databricks Notebook interface. A new cell is being created with the code:

```
1 %%sh
2 pip install matplotlib seaborn pandas
```

The notebook title is "Remya Velakkadaparambu Karthikeyan_ML". The cell status is "New cell UI: OFF".

Installing libraries matplotlib, pandas, seaborn, plotly and ML flow and autolog machine learning runs

The screenshot shows the Databricks Notebook interface with the command output for installing libraries:

```
1 %%sh
2 pip install matplotlib seaborn pandas
```

The output shows the installation process for various packages like numpy, scipy, and tensorflow, along with the creation of MLflow and Autolog configurations.

Figure 2.1 Setting up databricks environment for building recommender system

2.3 Data Management

The data from **steam-200k.csv** is made accessible to the notebook, by following the steps as given in Figure 2.2.

Highlighting steam-200k.csv in DBFS file system

The screenshot shows the Databricks Notebook interface with the command output for listing files in DBFS:

```
1 dbutils.fs.ls("/FileStore/tables")
```

The output lists various CSV files including "steam-200k.csv".

Converting CSV file to Spark dataframe and adding headers

```
Cmd 7
```

```
1 # Load the CSV file into a Spark DataFrame, inferring data types and without a header row
2 steam = spark.read.csv("/FileStore/tables/steam_200k.csv", header=False, inferSchema=True)
3
4 # Rename the columns to the desired headers
5 steam = steam.toDF("ID", "game_Name", "mem_Type", "value")
6

▶ (2) Spark Jobs
▶   steam: pyspark.sql.dataframe.DataFrame
      ID: integer
      game_Name: string
      mem_Type: string
      value: double

Command took 5.59 seconds -- by r.vellekkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 11:00:44 AM on BDTT_TASK_2_20_APP_1
```

Figure 2.2 Data management -convering csv file to steam dataframe

2.3 Data Preprocessing and Preparation Before Training

2.3.1 Data Pre-processing

The data frame structure, schema, total rows, etc are identified as part of pre-processing step in the steam dataframe. Table 2.1 shows preprocessing steps.

Table 2.1 Pre processing on steam dataframe.

Structure of dataframe

```
Cmd 7
```

```
1 # Load the CSV file into a Spark DataFrame, inferring data types and without a header row
2 steam = spark.read.csv("/FileStore/tables/steam_200k.csv", header=False, inferSchema=True)
3
4 # Rename the columns to the desired headers
5 steam = steam.toDF("ID", "game_Name", "mem_Type", "value")
6

▶ (2) Spark Jobs
▶   steam: pyspark.sql.dataframe.DataFrame
      ID: integer
      game_Name: string
      mem_Type: string
      value: double

Command took 5.53 seconds -- by r.vellekkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 11:00:44 AM on BDTT_TASK_2_20_APP_1
```

Removing rows with missing values and display of head 20 rows

```
1 # Remove rows with any missing values (NaN) from the DataFrame and then display the first 20 rows
2 steam = steam.dropna()
3 steam.show(truncate= False)

▶ (1) Spark Jobs
▶ steam: pyspark.sql.dataframe.DataFrame = [ID: integer, game_Name: string ... 2 more fields]

+-----+-----+-----+
|ID    |game_Name|mem_Type|value|
+-----+-----+-----+
|151603712|[The Elder Scrolls V Skyrim|purchase|1.0 |
|151603712|[The Elder Scrolls V Skyrim|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|[HumblePop          |purchase|1.0 |
|151603712|[HumblePop          |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 1.71 seconds -- by r.vellakkadparanmukarthikey@edu.salford.ac.uk at 4/20/2024, 11:01:56 AM on BDTT_TASK_2_20_APP_I
```

2.3.2 Data Preparation before Training

The dataframe `steam` consists of game name (integer), game name (string), member behavior (string), and value corresponding to member behavior (double) in the dataframe. To run Alternating Least Squares (ALS) matrix factorization using MLLib, we need to have integer ID values for both users and items, and this dataset does not contain IDs for the games.

As data preparation, the distinct game names are identified and game Id (integer) is self generated for each game and added this into the existing DataFrame. Table 2.2 shows the data preparation.

Table 2.2 Data Preparation before training

Generating distinct game id to the games

Total number of distinct games

```
1 # Get the count of distinct games in the distinct_games_df Dataframe
2 distinct_game_count = distinct_games_df.count()
3
4 # Display the count of distinct games
5 print(f"Total number of distinct games in distinct_games_df: {distinct_game_count}")
6

► (2) SparkJobs

Total number of distinct games in distinct_games_df: 5155
Command took 3.75 seconds -- by r.vellakkadaparamakarthikeyan@cs.salford.ac.uk at 4/20/2024, 11:06:56 AM on EDT, TASH_2_28_APR_1
```

Last 20 game names

```
1 # Get the last 20 rows of the Dataframe
2 last_20_rows = dataset_games_df.tail(20)
3
4 # Display the last 20 rows
5 for row in last_20_rows:
6     print(row)
7
8 #%% Spark jobs
9
10 game_name='Samperio's Sulfit and the Golden Touch', game_ID=2126
11 game_name='Many Drew Lights, Camera, Curse!', game_ID=21137
12 game_name='The Last of Us Part II: Left Behind - A Shattered', game_ID=193198
13 game_name='T33 - Player Profile (Profile ID:HEV)', game_ID=21219
14 game_name='Habitat', game_ID=2148
15 game_name='Alpha Character Generator', game_ID=21514
16 game_name='Dawn of War III', game_ID=21515
17 game_name='TEK Accessoire Pack', game_ID=2143
18 game_name='T33 - Player Profile (Index : Nu', game_ID=21244
19 game_name='Dawn of War III', game_ID=2145
20 game_name='GUNFAT (Windows 4.17)', game_ID=2146
21 game_name='Slithering 2 Anthology', game_ID=2147
22 game_name='Family Guy The Quest for Stuff Multiverse', game_ID=2148
23 game_name='Dawn of War III', game_ID=2149
24 game_name='Sandstorm', game_ID=2150
25 game_name='Circuit', game_ID=2151
26 game_name='The Last of Us Part II: Thrive', game_ID=2152
27 game_name='Generator Studio Standard', game_ID=2153
28 game_name='Shark Attack Doomsday 2', game_ID=2154
29 game_name='Dagger's Project Free Starter Pack', game_ID=2155

[Output Text] 1.28 seconds
```

Merging distinct games_df and steam df

Deleting redundant game name column

```
1 # Join the steam and distinct_games DataFrame on the game name (game_name)
2 joined_HF = steam[steam['game_name'].str.contains('game_name')] .join(distinct_games_df, steam['game_name'] == distinct_games_df['game_name'], "inner")
3
4 # Drop the game_name column from the joined DataFrame
5 joined_HF = joined_HF.drop(['game_name'])
6
7 # Display the joined DataFrame without the game_name column repetition
8 joined_HF.show(truncate=False)
9
```

▶ (1) Spark 100k

```
10 # Create DataFrame
11 gameRDD = spark.read.csv('gameData.csv', header=True, inferSchema=True)
```

2.4 Exploratory Data Analysis on Dataframes with and without Game IDs

The exploratory data analysis is conducted on both steam dataframe, distinct_games_df and joined_df. Table 2.3 shows the exploratory data analysis.

Table 2.3 Exploratory Data Analysis

Schema of steam DF

```
Cmd 14
1 # View the schema of the DataFrame
2 print("\nSchema of the steam DataFrame:")
3 steam.printSchema()
4
```

```
Schema of the steam DataFrame:
root
 |-- ID: integer (nullable = true)
 |-- game_Name: string (nullable = true)
 |-- mem_Type: string (nullable = true)
 |-- value: double (nullable = true)
```

```
Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:16:35 AM on BOTT_TASK_2_20_APR_1
```

Total rows in steam df

```
Cmd 15
1 # Count the total number of rows in the DataFrame
2 total_rows = steam.count()
3 print(f"\nTotal number of rows in the steam DataFrame: {total_rows}")
4
5
```

```
(2) Spark Jobs
```

```
Total number of rows in the steam DataFrame: 200000
Command took 2.98 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:17:04 AM on BOTT_TASK_2_20_APR_1
```

Unique user behavior types

```
Cmd 16
1 # Get unique values in mem_Type and game_Name
2 print("\nUnique values in the mem_Type column:")
3 steam.select("mem_Type").distinct().show()
4
5
```

```
(2) Spark Jobs
```

```
Unique values in the mem_Type column:
+-----+
|mem_Type|
+-----+
|purchase|
| play|
+-----+
```

```
Command took 2.57 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:17:39 AM on BOTT_TASK_2_20_APR_1
```

Unique values of game name

```
1 print("\nunique values in the game_Name column:")
2 steam.select("game_Name").distinct().show(truncate=False)
3
4
5
```

▶ (2) Spark Jobs

```
unique values in the game_Name column:
+-----+
|game_Name|
+-----+
|Left 4 Dead 2|
|Fallout 4|
|Dragon Age Origins - Ultimate Edition|
|Path of Exile|
|Tomb Raider|
|Left 4 Dead|
|Realm of the Mad God|
|BioShock Infinite|
|The Elder Scrolls V Skyrim|
|SEGA Genesis & Mega Drive Classics|
|HumblePop|
|The Banner Saga|
|Grand Theft Auto IV|
|Dead Island Epidemic|
|Marvel Heroes 2015|
|Elldavid|
+-----+
```

Command took 1.89 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/28/2024, 11:18:49 AM on EOTT_TASK_2_20_APR_3

Count of member behavior - 'purchase' and 'play' in dataset

```
1 # Group by mem_Type and count the number of rows in each group
2 print("\nGroup by mem_Type and count the number of rows in each group:")
3 steam.groupBy("mem_Type").count().show()
```

▶ (2) Spark Jobs

```
Group by mem_Type and count the number of rows in each group:
+-----+
|mem_Type| count|
+-----+
|purchase|129511|
| play| 70489|
+-----+
```

Command took 3.48 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/28/2024, 11:20:00 AM on EOTT_TASK_2_20_APR_3

Users with purchase behavior

```
1 # Filter the DataFrame to include only rows where mem_Type is "purchase"
2 purchase_df = steam.filter(steam.mem_Type == "purchase")
3 purchase_df.show()
4
5
6 (1) Spark Jobs
7
8 purchase_df: pyspark.sql.DataFrame
9   ID: Integer
10   game_Name: string
11   mem_Type: string
12   value: double
13
14 +---+-----+-----+-----+
15 |ID| game_Name|mem_Type|value|
16 +---+-----+-----+-----+
17 |15160731|The Elder Scrolls V: Skyrim|purchase| 1.0|
18 |15160732|Fallout 4|purchase| 1.0|
19 |15160732|Path of Exile|purchase| 1.0|
20 |15160732|BioShock Infinite|purchase| 1.0|
21 |15160732|Left 4 Dead|purchase| 1.0|
22 |15160732|Team Fortress 2|purchase| 1.0|
23 |15160732|Tomb Raider|purchase| 1.0|
24 |15160732|The Banner Saga|purchase| 1.0|
25 |15160732|Dragon Age: Origins|purchase| 1.0|
26 |15160732|Fallout 3 - Game ...|purchase| 1.0|
27 |15160732|SEGA Genesis & Me...|purchase| 1.0|
28 |15160732|Grand Theft Auto IV|purchase| 1.0|
29
30 Command took 1.19 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/28/2024, 11:21:41 AM on EOTT_TASK_2_20_APR_3
```

Game name and their count of occurrences

```

1 # Group the filtered DataFrame by game_Name and count the occurrences
2 purchase_count_df = purchase_df.groupby("game_Name").count()
3
4 purchase_count_df.show(truncate = False)

> (2) Spark Jobs
> [1] purchase_count_df: pyspark.sql.DataFrame[game_Name:string, count:long]

+-----+-----+
|game_Name|count|
+-----+-----+
|Dota 2|4841 |
|METAL GEAR SOLID V THE PHANTOM PAIN|168 |
|LEGO Batman The Videogame|111 |
|RDR|185 |
|Doomsday|111 |
|Legend of Grimrock|158 |
|Diablo III: Original Sin|171 |
|Halo: Combat Evolved Anniversary|4 |
|Sanctuary RPG Black Edition|5 |
|Snuggie Truck|9 |
|Unreal Flight|108 |
|Dungeons 2|6 |
|Dumb's Revenge|14 |
|Hassleheart|1 |
|DHF Hellball Challenge 12 & Artbook|1 |
|MONSTROUS Soundtrack & Artbook|2 |
|Dota 2: Dire vs. Radiant|76 |
|Call of Duty: Modern Warfare 2 - Multiplayer|134 |
Command took 1.99 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:13:38 AM on BDTT_TASK_2_28_APR_1

```

Games and count in descending order

```

Cmd 22
1 # Order the results in descending order based on the count
2 ordered_purchase_count_df = purchase_count_df.orderBy("count", ascending=False)
3
4
> (2) Spark Jobs
> [1] ordered_purchase_count_df: pyspark.sql.DataFrame[game_Name:string, count:long]

+-----+-----+
|game_Name|count|
+-----+-----+
|Dota 2|4841 |
|Team Fortress 2|2323 |
|Unreal Flight|185 |
|Counter-Strike Global Offensive|1412 |
|Half-Life 2: Lost Coast|981 |
|Counter-Strike Source|979 |
|Left 4 Dead|955 |
|Counter-Strike|858 |
|Halo: Reach|853 |
|Garry's Mod|733 |
|The Elder Scrolls V: Skyrim|727 |
|Robocraft|686 |
|Counter-Strike Condition Zero: Deleted Scenes|579 |
|Counter-Strike Condition Zero|579 |
|Heroes & Generals|558 |
|Halo: Combat Evolved|551 |
|ISL War's Civilization V|596 |
Command took 3.21 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:20:00 AM on BDTT_TASK_2_28_APR_1

```

Game with highest number of purchase occurrences

```

Cmd 23
1 # Get the game_Name with the highest number of "purchase" occurrences
2 top_game = ordered_purchase_count_df.first()
3
4
5 # Display the game_Name with the highest number of "purchase" occurrences and its count
6 print(f"The game with the highest number of 'purchase' occurrences is: {top_game['game_Name']}, with {top_game['count']} occurrences.")

> (2) Spark Jobs
The game with the highest number of 'purchase' occurrences is: Dota 2, with 4841 occurrences.
Command took 2.64 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:29:11 AM on BDTT_TASK_2_28_APR_1

```

Games with 'Play' behavior

```

1 from pyspark.sql.functions import sum
2
3 # Filter the DataFrame to include only rows where mem_type is "play"
4 play_df = steam.filter(steam.mem_type == "play")
5 play_df.show(truncate = False)
6
7
8 (1) Spark Jobs
9 play_df: pyspark.sql.dataframe.DataFrame = [ID: integer, game_Name: string ... 2 more fields]
+-----+-----+
|ID |game_Name |mem_type|value|
+-----+-----+
|151603712|The Elder Scrolls V Skyrim |play |273.0 |
|151603712|Fallout 4 |play |87.0 |
|151603712|Sports |play |14.9 |
|151603712|Fallout: New Vegas |play |12.1 |
|151603712|Left 4 Dead 2 |play |8.9 |
|151603712|HumblePop |play |8.5 |
|151603712|Path of Exile |play |8.1 |
|151603712|Poly Bridge |play |7.5 |
|151603712|Left 4 Dead |play |3.3 |
|151603712|Team Fortress 2 |play |2.8 |
|151603712|Tom Raider |play |2.5 |
|151603712|The Banner Saga |play |2.0 |
|151603712|Grand Theft Auto V: Episode 1 - Pacific |play |1.4 |
|151603712|BioShock Infinite |play |1.3 |
|151603712|Dragon Age: Origins - Ultimate Edition|play |1.3 |
|151603712|Fallout 3 - Game of the Year Edition|play |0.8 |
|151603712|SEGA Genesis & Mega Drive Classics |play |0.8 |
|151603712|Grand Theft Auto IV |play |0.6 |
+-----+
Command took 0.95 seconds ... by r.vellankadeparebukarthekeyan@edufxford.ac.uk at 4/28/2024, 11:38:29 AM on EOTT_TASK_2_28_APP_1

```

User and cumulative play time

```

1 from pyspark.sql.functions import sum, round
2
3 # Group the filtered DataFrame by ID and calculate the cumulative play time (sum of 'value' column)
4 cumulative_play_time_df = play_df.groupby("ID").agg(sum("value").alias("cumulative_play_time"))
5
6 # Round the cumulative_play_time column to two decimal places
7 cumulative_play_time_df = cumulative_play_time_df.withColumn("cumulative_play_time", round(cumulative_play_time_df["cumulative_play_time"], 2))
8
9 # Display the DataFrame with cumulative play time rounded to two decimal places
10 cumulative_play_time_df.show(truncate=False)
11
12
13 (1) Spark Jobs
14 cumulative_play_time_df: pyspark.sql.dataframe.DataFrame = [ID: integer, cumulative_play_time: double]
+-----+
|ID |cumulative_play_time|
+-----+
|16187221|1617.0 |
|16470920|697.0 |
|24878374|6.0 |
|99992274|6.5 |
|17441510|25.6 |
|151603712|2.5 |
|17860173|671.8 |
|17913285|32.0 |
|12941210|1.2 |
|74557142|0.1 |
|19272990|21.0 |
|16470920|1.2 |
|24880130|25.8 |
|16502366|5.0 |
|13845569|112.0 |
|35264447|125.4 |
|18787855|31.2 |
|16854630|116.0 |
+-----+
Command took 0.23 seconds ... by r.vellankadeparebukarthekeyan@edufxford.ac.uk at 4/28/2024, 11:31:38 AM on EOTT_TASK_2_28_APP_1

```

Users with highest play time

```

1 (1) Spark Jobs
2 cumulative_play_time_df: pyspark.sql.dataframe.DataFrame = [ID: integer, cumulative_play_time: double]
+-----+
|ID |cumulative_play_time|
+-----+
|71863790|11754.0 |
|18998980|11651.7 |
|18864947|11523.1 |
|12832650|10479.1 |
|351383649|9640.0 |
|416046432|9546.3 |
|40798067|9427.2 |
|12573819|9174.8 |
|14235519|8172.9 |
|14544507|8137.1 |
|52547959|7836.7 |
|13863519|7805.1 |
|15932163|7218.5 |
|161615403|7190.4 |
|47663596|7161.0 |
|67649495|7152.1 |
|17875130|7052.5 |
|18818751|6820.9 |
+-----+
Command took 2.18 seconds ... by r.vellankadeparebukarthekeyan@edufxford.ac.uk at 4/28/2024, 11:32:48 AM on EOTT_TASK_2_28_APP_1

```

Top 5 user based on playtime

```
Cmd 27
```

```
1 # Take the top 5 IDs from the ordered DataFrame
2 top_5_ids_df = cumulative_play_time_df.limit(5)
3
4 # Display the top 5 IDs with their corresponding cumulative play time
5 top_5_ids_df.show(truncate=False)
6
7
```

```
* (2) Spark Jobs
* [1] top_5_ids_df: pyspark.sql.DataFrame = [ID: integer, cumulative_play_time: double]
```

ID	cumulative_play_time
73817395	11754.0
1059962	11651.7
108630947	10853.2
26762388	10470.1
153382649	9640.0

```
Command took 2.35 seconds -- by r.vellakkadaparambukarthikayan@edu.salford.ac.uk at 4/20/2024, 11:36:53 AM on EOTT_TASK_2_29_APR_1
```

Frequency of occurrence of game IDs

```
Cmd 33
```

```
1 # Distribution of games
2 game_distribution = joined_df.groupBy("game_ID").count().orderBy(desc("count")).toPandas()
3 print("\nTop 10 most frequent games:")
4 print(game_distribution.head(10))
```

```
* (6) Spark Jobs
```

```
Top 10 most frequent games:
   game_ID  count
0      1    9682
1     3894    4646
2     2354    2789
3     3439    2632
4      70    1752
5      45    1693
6     3698    1424
7     2678    1397
8     2618    1394
9     2228    1271
```

```
Command took 6.09 seconds -- by r.vellakkadaparambukarthikayan@edu.salford.ac.uk at 4/20/2024, 11:47:57 AM on EOTT_TASK_2_20_APR_1
```

Top 10 frequent games

```
Cmd 34
```

```
1 # Distribution of games
2 game_distribution = joined_df.groupBy("game_Name").count().orderBy(desc("count")).toPandas()
3 print("\nTop 10 most frequent games:")
4 print(game_distribution.head(10))
```

```
* (6) Spark Jobs
```

```
Top 10 most frequent games:
   game_Name  count
0          Dota 2    9682
1  Team Fortress 2    4646
2  Counter-Strike Global Offensive    2789
3        Unturned    2632
4       Left 4 Dead 2    1752
5  Counter-Strike Source    1693
6        Counter-Strike    1424
7       Garry's Mod    1397
8  The Elder Scrolls V Skyrim    1394
9        Warframe    1271
```

```
Command took 6.72 seconds -- by r.vellakkadaparambukarthikayan@edu.salford.ac.uk at 4/20/2024, 11:49:12 AM on EOTT_TASK_2_20_APR_1
```

Top 10 games based on play time

```

1 # Top games in terms of play time
2 top_games_play_time = joined_df.filter((joined_df["Game_Type"] == "play")\
3 .groupby("game_Name")\
4 .agg(sum("value").alias("total_play_time"))\
5 .orderBy("total_play_time")\
6 .toPandas()
7
8 print("\nTop 10 games by play time:")
9 print(top_games_play_time.head(10))

*(0) SparkJobs

Top 10 games by play time:
   game_Name total_play_time
0      Dota 2    981684.6
1 Counter-Strike Global Offensive 322771.6
2      Game Forte 2    127392.5
3      Counter-Strike    134261.1
4      Sid Meier's Civilization V    99821.3
5      Counter-Strike Source    96075.5
6      The Elder Scrolls V Skyrim 78889.3
7      Garry's Mod    49725.3
8 Call of Duty Modern Warfare 2 - Multiplayer 428099.9
9      Left 4 Dead 2    33596.7

Command took 5.81 seconds -- by r.vellakkadaprambuka@hikayedu.salford.ac.uk at 4/28/2024, 11:58:48 AM on RDTT_TASK_2_28_Apr_1

```



```

1 from pyspark.sql import SparkSession
2
3 # Initialize a spark session
4 spark = SparkSession.builder.getOrCreate()
5
6 # Register the DataFrame as a temporary table
7 spark.createDataFrame(joined_df).createOrReplaceTempView("joined_df")
8
9 # calculate the average play time per game and round the time to 2 decimal places
10 avg_play_time_per_game = spark.sql("""
11     SELECT game_ID, round(sum(value), 2) AS avg_play_time
12     FROM joined_df
13     GROUP BY game_ID
14 """)
15
16 # Display the results
17 avg_play_time_per_game.show()
18

*(0) SparkJobs
+---+-----+
| game_ID | avg_play_time |
+---+-----+
| 21221 | 16.99 |
| 30031 | 36.81 |
| 3751 | 5.53 |
| 3749 | 5.96 |
| 1443 | 40.41 |
| 433 | 4.59 |
| 4081 | 3.28 |
| 1543 | 20.13 |
| 1589 | 0.21 |
| 2166 | 9.41 |
| 1445 | 3.82 |
| 3997 | 4.21 |
| 1029 | 2.11 |
| 4841 | 1.04 |
| 348 | 5.46 |
| 1298 | 4.85 |
| 1327 | 5.61 |
| 3897 | 315.36 |
+---+-----+
Command took 3.81 seconds -- by r.vellakkadaprambuka@hikayedu.salford.ac.uk at 4/28/2024, 11:58:42 PM on RDTT_TASK_2_28_Apr_1

```

2.5 Visualization on Dataframes using Python Matplotlib, and Plotly

The data visualization is performed using python matplotlib, plotly libraries and a dashboard is constructed using Power BI. Table 2.4 shows the visualization using matplotlib and plotly.

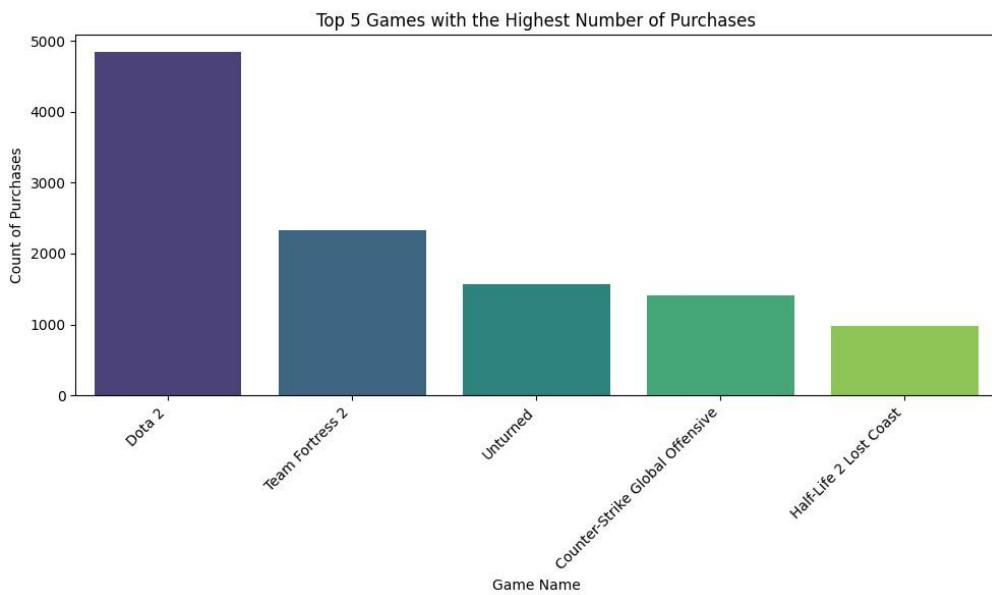
Table 2.4 Visualization using Python Matplotlib and Plotly

Top 5 games with highest number of purchases
--

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # Convert the purchase_count_df DataFrame to a Pandas DataFrame
6 purchase_count_pandas_df = purchase_count_df.toPandas()
7
8 # Sort the DataFrame in descending order based on the count
9 purchase_count_pandas_df = purchase_count_pandas_df.sort_values(by='count', ascending=False)
10
11 # Select the top 5 rows (top 5 games with the highest number of purchases)
12 top_5_games = purchase_count_pandas_df.head(5)
13
14 # Plot a vertical bar chart
15 plt.figure(figsize=(10, 6))
16 sns.barplot(x='game_name', y='count', data=top_5_games, palette='viridis')
17
18 # Set plot labels and title
19 plt.xlabel('Game Name')
20 plt.ylabel('Count of Purchases')
21 plt.title('Top 5 Games with the Highest Number of Purchases')
22
23 # Rotate x-axis labels for better readability
24 plt.xticks(rotation=45, ha='right')
25
26 # Show the plot
27 plt.tight_layout()
28 plt.show()
29

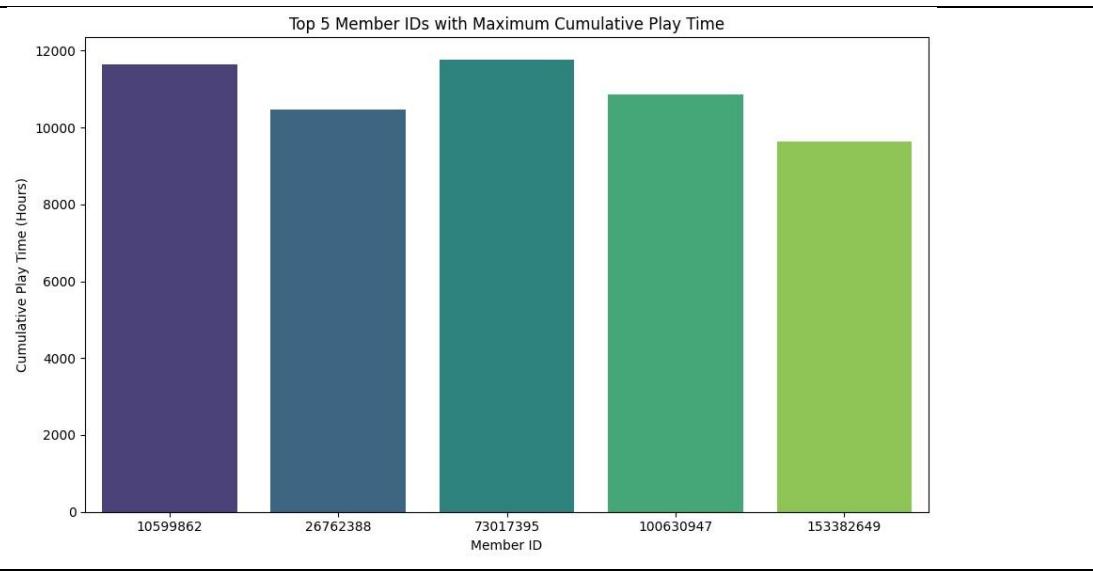
```



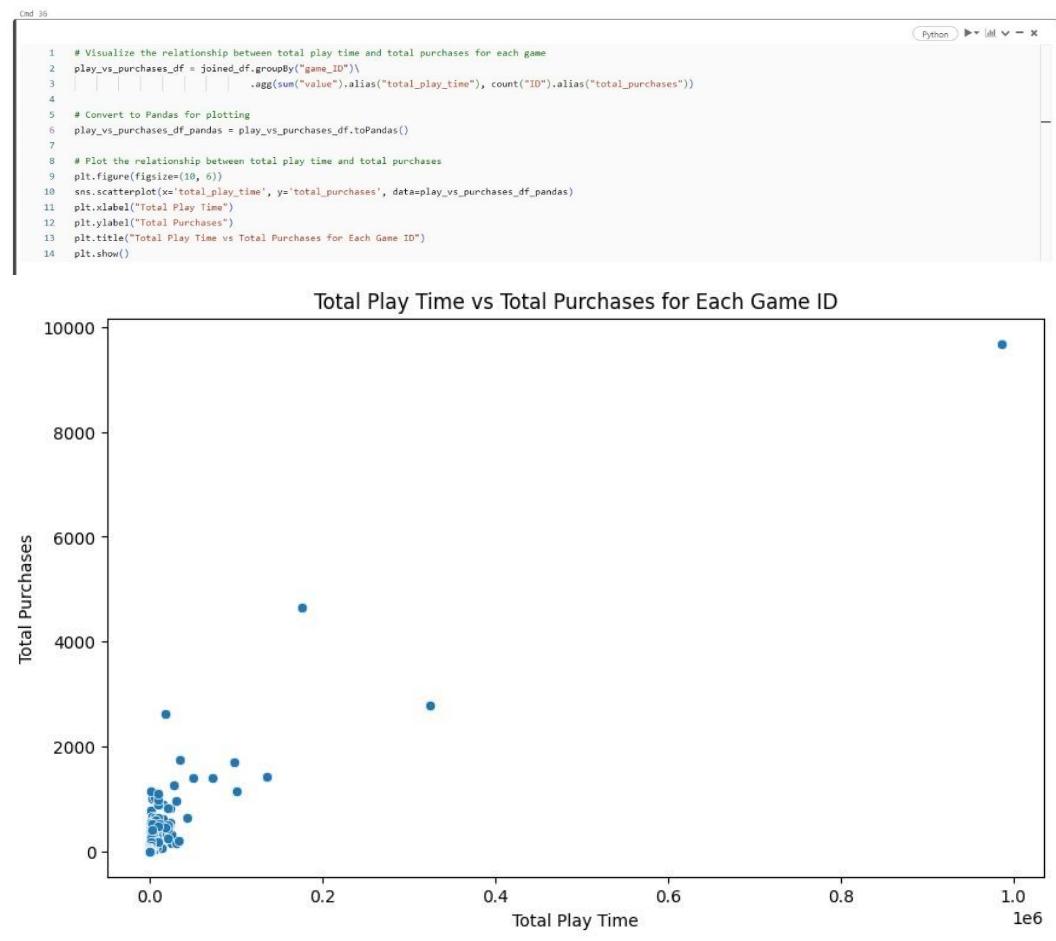
Top 5 users with maximum cumulative play time

```
Cmd 28

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 # Convert the top 5 IDs Dataframe to a Pandas DataFrame
6 top_5_ids_pandas_df = top_5_ids_df.toPandas()
7
8 # Plot a horizontal bar chart
9 plt.figure(figsize=(10, 6))
10 sns.barplot(y='cumulative_play_time', x='ID', data=top_5_ids_pandas_df, palette='viridis')
11
12 # Set plot labels and title
13 plt.xlabel('Member ID')
14 plt.ylabel('Cumulative Play Time (Hours)')
15 plt.title('Top 5 Member IDs with Maximum Cumulative Play Time')
16
17 # Show the plot
18 plt.tight_layout()
19 plt.show()
```



Play Time versus Total Purchases

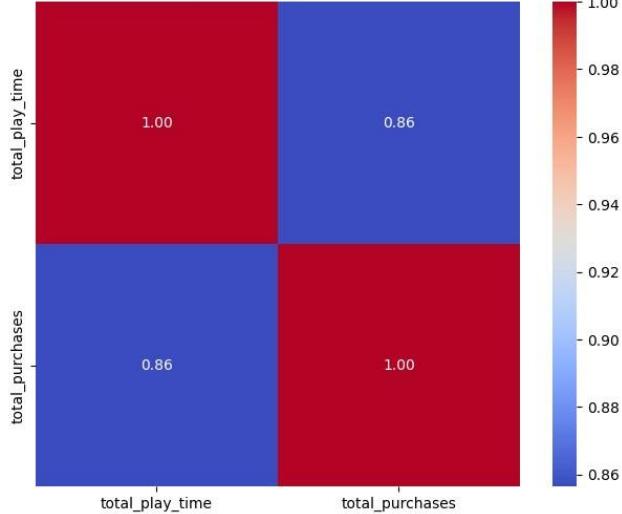


Correlation Heat map: Total Play time and Total Purchases

Cmd 37

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3 import pandas as pd
4 from pyspark.sql.functions import sum, count
5
6 # Aggregate the data by game name and calculate total play time and total purchases
7 game_aggregated_df = joined_df.groupby("game_Name")\
8     .agg(sum("value").alias("total_play_time"), count("ID").alias("total_purchases"))
9
10 # Convert the aggregated DataFrame to a Pandas DataFrame
11 game_aggregated_df_pandas = game_aggregated_df.toPandas()
12
13 # Calculate the correlation matrix between total play time and total purchases
14 correlation_matrix = game_aggregated_df_pandas[['total_play_time', 'total_purchases']].corr()
15
16 # Plot the correlation heat map
17 plt.figure(figsize=(8, 6))
18 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True, fmt='.2f')
19 plt.title("Correlation Heatmap: Total Play Time and Total Purchases")
20 plt.show()
21
```

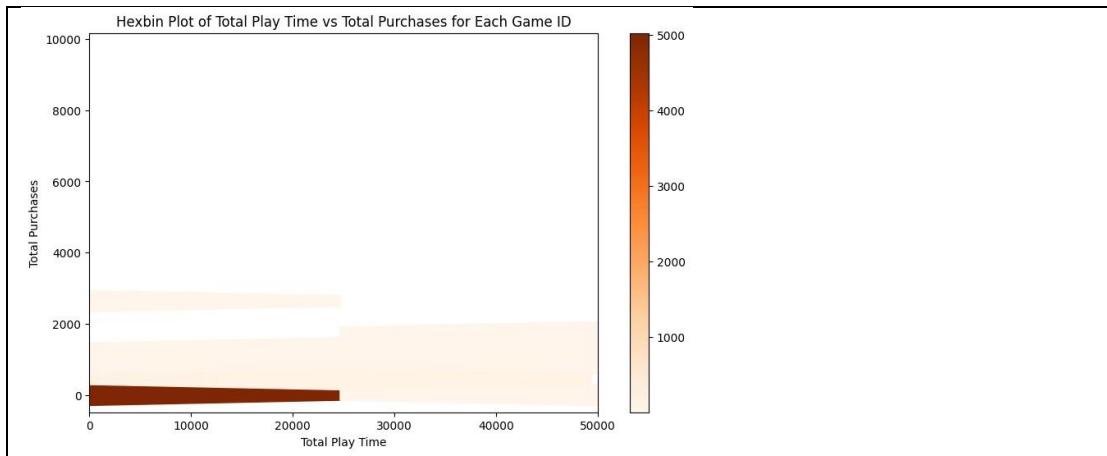
Correlation Heatmap: Total Play Time and Total Purchases



Total Play time versus Total Purchases for Each Game ID

Cmd 38

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 # Convert to Pandas for plotting
5 play_vs_purchases_df_pandas = play_vs_purchases_df.toPandas()
6
7 # Create a hexbin plot to visualize the relationship between total play time and total purchases
8 plt.figure(figsize=(10, 6))
9 hexbin_plot = plt.hexbin(play_vs_purchases_df_pandas['total_play_time'], play_vs_purchases_df_pandas['total_purchases'], gridsize=20, cmap='Oranges', mincnt=1)
10
11 # Add a color bar to the plot for reference
12 plt.colorbar(hexbin_plot)
13
14 # Add labels and title
15 plt.xlabel("Total Play Time")
16 plt.ylabel("Total Purchases")
17 plt.title("Hexbin Plot of Total Play Time vs Total Purchases for Each Game ID")
18
19 # Set the x-axis range to display only up to 50,000
20 plt.xlim(0, 50000)
21
22 # Show the plot
23 plt.show()
24
```



2.6 Visualization on Dataframes using Power BI

The data from the joined_df containing the original data with integer based game ID is loaded into the Power BI by following the steps given in Figure 2.

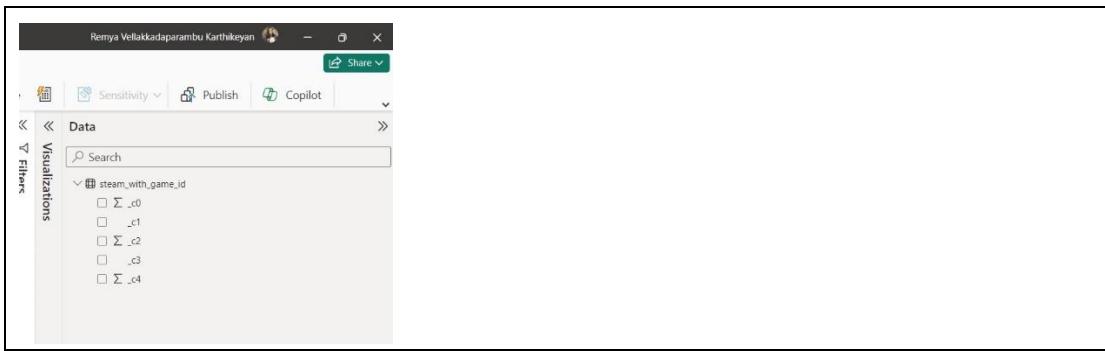
Saving the joined_df as csv file and then to spark SQL Delta table

Table is saved in default database

Getting the server hostname and http path from ML cluster

Copying the authentication values to the databricks interface in Power BI

The table loaded to the Power BI



Viewing sample data loaded to Power BI

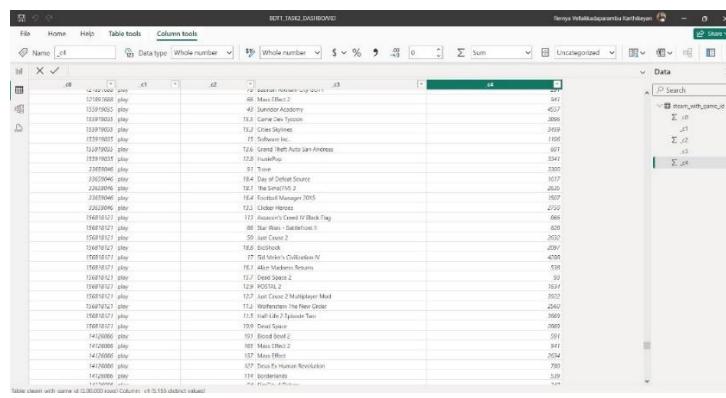


Figure 2.2 Loading data from databricks to Power BI

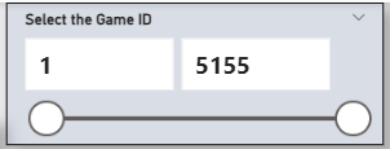
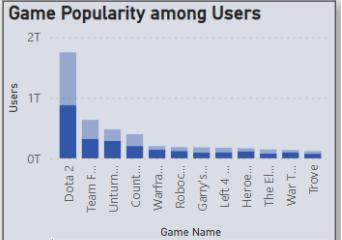
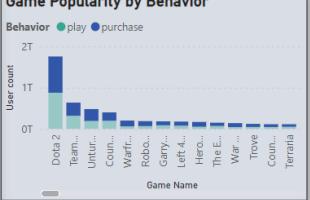
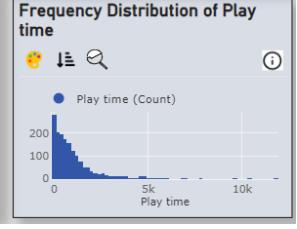
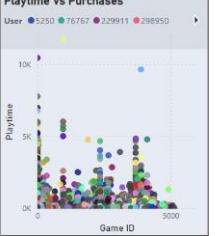
Figure 2.3 shows the dashboard created using Power BI on the data uploaded from Databricks

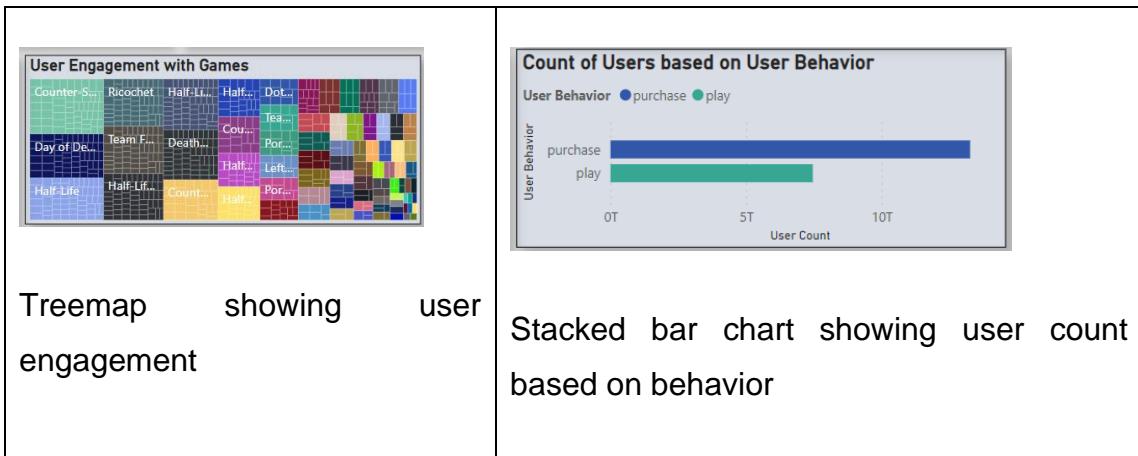


Figure 2.3 dashboard build on data loaded from databricks to Power BI

Analyzing each visualization closer in Table 2.4.

Table 2.4 Close up view of visualization using PowerBI

 <p>Slider to select game ID</p>	 <p>Game Popularity among Users Users Game Name</p>	
 <p>User Distribution on Behavior Users and Purchase / Play Behavior purchase play</p>	<p>Popular games among users based on behavior (clustered column chart)</p>	
<p>Ribbon chart showing user distribution based on behavior</p>	<p>5155 Distinct Game Count</p>	
<p>Total games count (card)</p>	<p>12.39K Distinct User Count</p>	
 <p>Purchase Vs Play Behavior purchase (36%) play (64%)</p>	 <p>Game Popularity by Behavior Behavior play purchase User count Game Name</p>	
<p>Pie chart showing purchase and play proportion</p>	<p>Stacked column chart showing popular games based on behavior</p>	
 <p>Frequency Distribution of Play time Play time (Count) Play time</p>	 <p>Playtime Vs Purchases User Playtime Game ID</p>	 <p>Average Play Time 48.88 0.00 97.76</p>
<p>Histogram showing frequency distribution of playtime</p>	<p>Scatter plot showing playtime vs purchases of the games</p>	<p>Gauge showing average playtime</p>



2.7 Collaborative Filtering Recommender System based on User Behavior ‘Purchase’

The steps involved in the machine learning algorithm can be briefed as follows.

Step 1: Filter the Data for 'purchase' type

```

In [42]:
1 from pyspark.ml.recommendation import ALS
2 from pyspark.ml.evaluation import RegressionEvaluator
3 from pyspark.sql.functions import col
4
5 # Step 1: Filter the Data for 'purchase' type
6 purchase_joined_df = joined_df.filter(col("item_type") == "purchase")
7 purchase_joined_df.show(truncate = False)

```

(3) Spark Jobs

151603711 purchase 1.0	injustice	1168
151603712 purchase 1.0	Fallout New Vegas	1921
151603712 purchase 1.0	Left 4 Dead 2	79
151603712 purchase 1.0	Huntshop	1341
151603712 purchase 1.0	Path of Exile	1563
151603712 purchase 1.0	Poly Bridge	4317
151603712 purchase 1.0	Republique	1593
151603712 purchase 1.0	Team Fortress 2	3694
151603712 purchase 1.0	Tomb Raider	1755
151603712 purchase 1.0	The Banner Saga	3342
151603712 purchase 1.0	Dead Island Epidemic	3541
151603712 purchase 1.0	Eldritch Infinite	2393
151603712 purchase 1.0	Fable Anniversary - Ultimate Edition	24
151603712 purchase 1.0	Fallout 3 - Game of the Year Edition	4624
151603712 purchase 1.0	SEGA Genesis & Mega Drive Classics	3274
151603712 purchase 1.0	Grand Theft Auto IV	3518
151603712 purchase 1.0	Realm of the Mad God	2120
151603711 purchase 1.0	Marvel Heroes 2015	3542

only showing top 20 rows

Step 2: Split the data into training (80%) and test (20%) sets

```

In [43]:
1 (training,test) = purchase_joined_df.randomSplit([0.8,0.2],seed = 100)
2
3 # training: pyspark.sql.dataframe.DataFrame = [D: integer, itemType:string ... 3 more fields]
4 # test: pyspark.sql.dataframe.DataFrame = [D: integer, itemType:string ... 3 more fields]

```

Step 3: Train the ALS model

```

Cmd 44

1 from pyspark.ml.recommendation import ALS
2 als = ALS(maxIter = 5, regParam = 0.01, userCol = "ID", itemCol = "game_ID", ratingCol = "value", seed = 100)
3 model = als.fit(training)

▶ [9] Spark Jobs
▶ [1] Mlflow run
Logged 1 run to an experiment in MLflow. Learn more

2024/04/20 11:16:43 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '8cd49fb727014587a21fdb187fd36894', which will track hyperparameters, performance metric
s, model artifacts, and lineage information for the current pyspark.ml workflow
2024/04/20 11:16:44 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/databricks/python/lib/python3.10/site-packages/mlflow/data/spark_dataset.py:159:
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, i
t 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."
2024/04/20 11:17:10 WARNING mlflow.pyspark.ml: Model ALS_890ded271951 will not be autologged because it is not allowlisted or or because one or more of its nested models are not allowli
sted. 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.pyspar
k.ml.autolog docs for more details).

Command took 29.36 seconds -- by r.vellakkadaparambukarthikeyan@eduford.ac.uk at 4/20/2024, 12:16:41 PM on BDT_TASK_2_20_APR_1

```

MaxIter = 5, regParam = 0.01

Step 4: Evaluate the model's performance using the test data

```

Cmd 45

1 predictions = model.transform(test).dropna()
2 predictions.show()

▶ [5] Spark Jobs
▶ [2] predictions: pyspark.sql.DataFrame = [ID:integer, game_ID:prediction ... 4 more fields]
+----+-----+-----+
| ID|game_ID|prediction|
+----+-----+-----+
| 5298|purchase| 1.0|Counter-Strike: Global Offensive ...| 4510.0|0.9753596 |
| 5299|purchase| 1.0|Half-Life 2 Deathmatch ...| 4411.0|0.9784899 |
| 5300|purchase| 1.0|Call of Duty: Warzone ...| 4482.0|0.9803991 |
| 5301|purchase| 1.0|Call of Duty: Warzone ...| 3747.0|0.9805621 |
| 7667|purchase| 1.0|Team Fortress 2 ...| 2562.0|0.9100848 |
| 7668|purchase| 1.0|Call of Duty: Warzone ...| 3851.0|0.9810368 |
| 7669|purchase| 1.0|Call of Duty: Warzone ...| 3601.0|0.9810315 |
| 7670|purchase| 1.0|Call of Duty: Warzone ...| 3541.0|0.9810365 |
| 7671|purchase| 1.0|Call of Duty: Warzone ...| 3997.0|0.9180671 |
| 7672|purchase| 1.0|Deathmatch Classics ...| 1972.0|0.9155388 |
| 7673|purchase| 1.0|Half-Life: Blue Shift ...| 4461.0|0.9454339 |
| 7674|purchase| 1.0|Call of Duty: Warzone ...| 3976.0|0.9810351 |
| 7675|purchase| 1.0|Call of Duty: Warzone ...| 12521.0|0.9101928 |
| 103398|purchase| 1.0|Counter-Strike: Global Offensive ...| 30891.0|0.9580938 |
| 103399|purchase| 1.0|Counter-Strike: Global Offensive ...| 20441.0|0.9571225 |
| 103400|purchase| 1.0|Call of Duty: Warzone ...| 6291.0|0.9652074 |
| 103401|purchase| 1.0|Call of Duty: Warzone ...| 3651.0|0.9570571 |
| 103412|purchase| 1.0|Team Fortress 2 ...| 3747.0|1.0027679 |
Command took 0.38 seconds -- by r.vellakkadaparambukarthikeyan@eduford.ac.uk at 4/20/2024, 13:24:35 PM on BDT_TASK_3_20_APR_1

```

Step 5: Explore some of the resulting recommendations

Displaying sorted predictions

```

Cmd 46

1 from pyspark.sql.functions import desc
2
3 # Order the predictions DataFrame in descending order of ID
4 predictions_sorted = predictions.orderBy(desc("ID"))
5
6 # Display the sorted DataFrame
7 predictions_sorted.show()

▶ [5] Spark Jobs
▶ [2] predictions_sorted: pyspark.sql.DataFrame = [ID:integer, game_ID:prediction ... 4 more fields]
+----+-----+-----+
| ID|game_ID|prediction|
+----+-----+-----+
| 30940240|purchase| 1.0|Unturned ...| 34391.0|0.9521162 |
| 30925240|purchase| 1.0|A.V.A - Alliance ...| 18791.0|0.9600036 |
| 30925240|purchase| 1.0|Team Fortress 2 ...| 389410.0|0.9719154 |
| 30925240|purchase| 1.0|Call of Duty: Warzone ...| 34801.0|0.9810329 |
| 30982291|purchase| 1.0|Grand Theft Auto V ...| 2591.0|0.8741915 |
| 30905291|purchase| 1.0|Happy Wars! ...| 25111.0|0.9361343 |
| 30895132|purchase| 1.0|Counter-Strike: Global Offensive ...| 18680.0|0.8822688 |
| 30895132|purchase| 1.0|Call of Duty: Warzone ...| 42521.0|0.9012882 |
| 30895132|purchase| 1.0|Call of Duty: Warzone ...| 24801.0|0.7741528 |
| 30833596|purchase| 1.0|The Land of the Lost ...| 8311.0|0.9544407 |
| 30783842|purchase| 1.0|Metal Gear Online ...| 24801.0|0.7590557 |
| 30735101|purchase| 1.0|Marvel Heroes 2015 ...| 35421.0|0.8495149 |
| 30699168|purchase| 1.0|Heroes & Generals ...| 19421.0|0.9201645 |
| 30665058|purchase| 1.0|Call of Duty: Warzone ...| 15921.0|0.9810341 |
| 30665058|purchase| 1.0|The Elder Scrolls ...| 41311.0|0.9141792 |
| 30651701|purchase| 1.0|Bomberman ...| 34791.0|0.9151715 |
| 30654752|purchase| 1.0|Team Fortress 2 ...| 38941.0|0.9880236 |
| 30654752|purchase| 1.0|Warframe ...| 22201.0|0.9914115 |
Command took 9.48 seconds -- by r.vellakkadaparambukarthikeyan@eduford.ac.uk at 4/20/2024, 12:25:54 PM on BDT_TASK_2_20_APR_1

```

Predictions listed by ordering game Id in ascending order

```
Code 47
1 # Order the predictions DataFrame in ascending order of game_ID
2 ordered_predictions = predictions.orderBy("game_ID", ascending=True)
3
4 # Display the ordered DataFrame
5 ordered_predictions.show(truncate=False)
6
```

(5) Spark jobs

ID	mem_Type	value	game_Name	game_ID	prediction
41160759	purchase	1.0	[Data 2]	[1]	[0.8325339]
41360645	purchase	1.0	[Data 2]	[1]	[0.83884656]
75689918	purchase	1.0	[Data 2]	[1]	[0.91355434]
76787637	purchase	1.0	[Data 2]	[1]	[0.78468263]
994406	purchase	1.0	[Data 2]	[1]	[0.88912126]
43684632	purchase	1.0	[Data 2]	[1]	[0.82608081]
44321815	purchase	1.0	[Data 2]	[1]	[0.8233659]
37490443	purchase	1.0	[Data 2]	[1]	[0.8579175]
12141711	purchase	1.0	[Data 2]	[1]	[0.82463825]
44865715	purchase	1.0	[Data 2]	[1]	[0.91552737]
47384303	purchase	1.0	[Data 2]	[1]	[0.78100336]
29810000	purchase	1.0	[Data 2]	[1]	[0.8777396]
11750777	purchase	1.0	[Data 2]	[1]	[0.814169]
45974869	purchase	1.0	[Data 2]	[1]	[0.8265135]
47457723	purchase	1.0	[Data 2]	[1]	[0.7585415]
4834220	purchase	1.0	[Data 2]	[1]	[0.87821746]
23579751	purchase	1.0	[Data 2]	[1]	[0.8998766]
46534663	purchase	1.0	[Data 2]	[1]	[0.9181451]

Command took 8.81 seconds -- by r.vellankkaperumal@keyen.edu.sa from 10.20.10.1 at 4/26/2024, 12:27:17 PM on EDT_TASK_2_20_APR_1

Ordering the predictions based on user ID (descending) and game ID (ascending)

```
1 from pyspark.sql.functions import desc, asc
2
3 # Order the predictions DataFrame based on ID in descending order and game_ID in ascending order
4 ordered_predictions = predictions.orderBy(desc("ID"), asc("game_ID"))
5
6 # Show the ordered predictions
7 ordered_predictions.show()
8
```

(5) Spark jobs

ID	mem_Type	value	game_Name	game_ID	prediction
389404240	purchase	1.0	Unturned	3439	0.9521162
389262440	purchase	1.0	A.V.A - Alliance ...	1879	0.9600036
38918895	purchase	1.0	Team Fortress 2	3804	0.87179154
389052991	purchase	1.0	Unturned	3439	0.9862124
389052991	purchase	1.0	Brawlhalla	299	1.0741915
389052991	purchase	1.0	Happy Wars!	2511	1.0346134
388695132	purchase	1.0	Counter-Strike Ne...	1866	0.8632668
388653033	purchase	1.0	thelhunter	4352	1.0030802
38848736	purchase	1.0	Metal War Online ...	2400	0.7774258
388385380	purchase	1.0	The Lord of the R...	835	0.99574447
387688442	purchase	1.0	Metal War Online ...	2400	0.7390567
387535018	purchase	1.0	Marvel Heroes 2015	3542	0.8495149
386993682	purchase	1.0	Warface	1505	0.9383248
386993682	purchase	1.0	Heroes & Generals	1940	0.9383248
38661701	purchase	1.0	The Elder Scrolls: ...	451	0.9041794
386567532	purchase	1.0	Unturned	3439	0.81121715
386567532	purchase	1.0	Warframe	2220	0.8974415
386547532	purchase	1.0	Team Fortress 2	3804	0.88980236

Command took 0.11 seconds -- by r.vellankkaperumal@keyen.edu.sa from 10.20.10.1 at 4/26/2024, 11:37:25 PM on EDT_TASK_3_20_APR_1

Evaluating RMSE and MAE

```

Cmd 49
1 from pyspark.ml.evaluation import RegressionEvaluator
2 evaluator = RegressionEvaluator(metricName = "rmse",labelCol = "value",predictionCol = "prediction")
3 rmse = evaluator.evaluate(predictions)
4 print(" Root mean square error is : %g" %rmse)
5 # Create a RegressionEvaluator for MAE
6 evaluator_mae = RegressionEvaluator(metricName="mae", labelCol="value", predictionCol="prediction")
7 mae = evaluator_mae.evaluate(predictions)
8 print("Mean absolute error (MAE) is: %f" % (mae))

* (10) Spark Jobs
Root mean square error is = 0.0756261
Mean absolute error (MAE) is: 0.03364780978809928
Command took 16.58 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:40:23 PM on BDIT_TASK_2_20_APR_1

```

Predictions ordered in descending order

```

1 # Order the predictions DataFrame based on prediction in descending order
2 ordered_predictions = predictions.orderBy("prediction", ascending=False)
3
4 # Show the ordered predictions
5 ordered_predictions.show(truncate=False)
6

* (5) Spark Jobs
* ordered_predictions: pyspark.sql.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

+---+-----+-----+-----+
|ID |mem_Type|value|game_Name| game_ID|prediction|
+---+-----+-----+-----+
|19005236 |purchase|1.0 |Star Trek D-A-C |1842 |1.6195754 |
|14544587 |purchase|1.0 |Star Trek D-A-C |1842 |1.5327085 |
|167049628 |purchase|1.0 |Night & Magic Heroes Online |775 |1.3611796 |
|103650936 |purchase|1.0 |Night & Magic Heroes Online |775 |1.252822 |
|18604016 |purchase|1.0 |Football Manager 2012 |3773 |1.2432587 |
|102894728 |purchase|1.0 |Destination Sol |4130 |1.2335975 |
|40008833 |purchase|1.0 |RollerCoaster Tycoon Deluxe |4123 |1.2223594 |
|10598808 |purchase|1.0 |Pro Evolution Soccer 2015 |1224 |1.2104812 |
|305130721 |purchase|1.0 |Night & Magic Heroes Online |775 |1.1860325 |
|22849279 |purchase|1.0 |Football Manager 2011 |233 |1.1849642 |
|116032656 |purchase|1.0 |A Virus Named TOH |516 |1.1748672 |
|175450855 |purchase|1.0 |Praetorians |1080 |1.1743008 |
|243664576 |purchase|1.0 |Sheltered |171 |1.1585559 |
|747644096 |purchase|1.0 |Assetto Corsa |1932 |1.1546407 |
|4466715 |purchase|1.0 |Transformers Fall of Cybertron |3093 |1.1545993 |
|179377677 |purchase|1.0 |Enclave |4353 |1.1401118 |
|113423671 |purchase|1.0 |Metro 2033 |3434 |1.1396382 |
|130758906 |purchase|1.0 |Super Killer Hornet Resurrection|1260 |1.1395532 |

```

Predictions ordered in ascending order

```

Cmd 50
1 # Drop rows with missing values
2 predictions = predictions.dropna()
3
4 # Order the predictions DataFrame based on prediction in descending order
5 ordered_predictions = predictions.orderBy("prediction", ascending=True)
6
7 # Show the ordered predictions
8 ordered_predictions.show(truncate=False)
9

* (5) Spark Jobs
* predictions: pyspark.sql.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
* ordered_predictions: pyspark.sql.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

+---+-----+-----+-----+
|ID |mem_Type|value|game_Name| game_ID|prediction|
+---+-----+-----+-----+
|19005236 |purchase|1.0 |Star Trek D-A-C |1842 |1.6195754 |
|14544587 |purchase|1.0 |Star Trek D-A-C |1842 |1.5327085 |
|167049628 |purchase|1.0 |Night & Magic Heroes Online |775 |1.3611796 |
|103650936 |purchase|1.0 |Night & Magic Heroes Online |775 |1.252822 |
|18604016 |purchase|1.0 |Football Manager 2012 |3773 |1.2432587 |
|102894728 |purchase|1.0 |Destination Sol |4130 |1.2335975 |
|40008833 |purchase|1.0 |RollerCoaster Tycoon Deluxe |4123 |1.2223594 |
|10598808 |purchase|1.0 |Pro Evolution Soccer 2015 |1224 |1.2104812 |
|305130721 |purchase|1.0 |Night & Magic Heroes Online |775 |1.1860325 |
|22849279 |purchase|1.0 |Football Manager 2011 |233 |1.1849642 |
|116032656 |purchase|1.0 |A Virus Named TOH |516 |1.1748672 |
|175450855 |purchase|1.0 |Praetorians |1080 |1.1743008 |
|243664576 |purchase|1.0 |Sheltered |171 |1.1585559 |
|747644096 |purchase|1.0 |Assetto Corsa |1932 |1.1546407 |
|4466715 |purchase|1.0 |Transformers Fall of Cybertron |3093 |1.1545993 |
|179377677 |purchase|1.0 |Enclave |4353 |1.1401118 |
|113423671 |purchase|1.0 |Metro 2033 |3434 |1.1396382 |
|130758906 |purchase|1.0 |Super Killer Hornet Resurrection|1260 |1.1395532 |

```

SQL queries on predictions based on temporary view

```
Cmd 53
1 %sql
2
3 select ID,game_ID, game_Name, prediction from myPredictions order by prediction desc limit 10
▶ (3) Spark Jobs
▶ └─ _sqldf: pyspark.sql.DataFrame = [ID: integer, game_ID: integer ... 2 more fields]
Table ┴ +
```

ID	game_ID	game_Name	prediction
1	89005236	Star Trek D-A-C	1.6195754
2	1454587	Star Trek D-A-C	1.5327085
3	197949629	Might & Magic Heroes Online	1.3611796
4	203650036	Might & Magic Heroes Online	1.252822
5	18604016	Football Manager 2012	1.2432387
6	202894728	Desiration Sol	1.2335975
7	40080833	RollerCoaster Tycoon Deluxe	1.2223594
8	1059862	Pro Evolution Soccer 2015	1.2104812
9	305130721	Might & Magic Heroes Online	1.1860325
10	42849279	Football Manager 2011	1.1849642

↓ 10 rows | 8.65 seconds runtime Refreshed now

① This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[53]`. Learn more

Command took 8.65 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 12:44:18 PM on BDIT_TASK_2_20_APP_3

SQL queries on prediction of game names with word ‘wars’ (same queries are also run on python in the codes)

```
Cmd 54
1 %sql
2 select ID,game_ID, game_Name, prediction from myPredictions where game_Name like '%wars%' order by prediction desc limit 10
▶ (5) Spark Jobs
▶ └─ _sqldf: pyspark.sql.DataFrame = [ID: integer, game_ID: integer ... 2 more fields]
Table ┴ +
```

ID	game_ID	game_Name	prediction
1	130021142	Star Wars Empire at War Gold	1.0901763
2	160546873	Happy Wars	1.0863229
3	247871700	Magicka Wizard Wars	1.085828
4	235264279	Lambda Wars Beta	1.0503274
5	243077634	Magicka Wizard Wars	1.0471183
6	44866715	Star Wars Dark Forces	1.0420889
7	117488479	Star Wars - Jedi Knight II Jedi Outcast	1.0390089
8	309052991	Happy Wars	1.0346134
9	9925590	Worms Clan Wars	1.0327645
10	117488479	Star Wars Jedi Knight Dark Forces II	1.0336353

↓ 10 rows | 11.55 seconds runtime Refreshed now

① This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[54]`. Learn more

Command took 11.55 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 12:45:08 PM on BDIT_TASK_2_20_APP_3

SQL queries on prediction for game IDs (18, 24)

Cmd 55

```

1 %sql
2 select ID,game_ID, game_Name, prediction from myPredictions where game_ID In ( 18, 24) order by prediction desc limit 10

```

▶ (5) Spark Jobs

▀ _sqldf: pyspark.sql.dataframe.DataFrame = [ID: integer, game_ID: integer ... 2 more fields]

Table

ID	game_ID	game_Name	prediction
1	153956921	RoboTex	1.0216267
2	58977564	Call of Duty Modern Warfare 2 - Multiplayer	1.0179827
3	63263519	Call of Duty Modern Warfare 2 - Multiplayer	1.0179724
4	232207754	RoboTex	1.0093927
5	86912006	RoboTex	1.0091228
6	102825821	RoboTex	1.0047473
7	57798235	Call of Duty Modern Warfare 2 - Multiplayer	1.0031015
8	54999247	Call of Duty Modern Warfare 2 - Multiplayer	1.0024006
9	21861124	Call of Duty Modern Warfare 2 - Multiplayer	1.0017794
10	71704418	Call of Duty Modern Warfare 2 - Multiplayer	1.0005744

▀ 10 rows | 6.04 seconds runtime

Refreshed 16 hours ago

▀ This result is stored as PySpark data frame _sqldf and in the IPython output cache as Out[325]. Learn more

Command took 6.04 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/19/2024, 8:38:38 PM on BDIT_TASK_19_APR_2024

Function giving recommendation filtered by game name

```

1 #Function give recommendations filtered by game name
2 def recommendationsByGameName(name):
3     filter = '%' + name + '%'
4     recommendationsList = ordered_predictions.orderBy("prediction",ascending=False).filter(ordered_predictions.game_Name.like(filter)).limit(10).show(truncate=False)
5     return recommendationsList

```

Command took 0.06 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:51:52 PM on BDIT_TASK_2_20_APR_1

Cmd 59

```

1 recommendationsByGameName('Duty')

```

▶ (5) Spark Jobs

ID	[mem_type]value game_Name	[game_ID prediction]
1302916505	[purchase]1.0 Call of Duty World at War	[2379 1.0478029
12643609	[purchase]1.0 Commandos Beyond the Call of Duty	[583 1.0333916
192103372	[purchase]1.0 Call of Duty Modern Warfare 3 - Multiplayer	[2029 1.0239547
18836199	[purchase]1.0 Call of Duty 4 Modern Warfare	[2950 1.0220948
58977564	[purchase]1.0 Call of Duty Modern Warfare 2	[3851 1.0180221
58977564	[purchase]1.0 Call of Duty Modern Warfare 2 - Multiplayer	[18 1.0179827
63263519	[purchase]1.0 Call of Duty Modern Warfare 2 - Multiplayer	[18 1.0179724
134274586	[purchase]1.0 Call of Duty Ghosts	[2849 1.014595
67349307	[purchase]1.0 Commandos Beyond the Call of Duty	[583 1.0132535
189458261	[purchase]1.0 Call of Duty World at War	[2379 1.0126683

Command took 7.31 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:49:19 PM on BDIT_TASK_2_20_APR_1

Function giving recommendation filtered by game ID

Cmd 60

```

1 #Function give recommendations filtered by game id
2 def recommendationsByGameID(game_IDs):
3     # Filter ordered predictions based on provided game_IDs
4     filtered_predictions = ordered_predictions.filter(ordered_predictions.game_ID.isin(game_IDs))
5
6     # Order by prediction in descending order and limit to top 10 results
7     top_recommendations = filtered_predictions.orderBy("prediction", ascending=False).limit(10)
8
9     # Show the top recommendations without truncation
10    top_recommendations.show(truncate=False)
11
12    # Return the DataFrame containing the top recommendations
13    return top_recommendations

```

Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:51:44 PM on BDIT_TASK_2_20_APR_1

```

Cmd 61:
1   recommendationsByGameID(1)

* (5) Spark Jobs
+-----+-----+
|ID |mem_Type|value|game_Name|game_ID|prediction|
+-----+-----+
|2320262151|purchase|1.0 |Dota 2 |1 |0.8967617 |
|2022372661|purchase|1.0 |Dota 2 |1 |0.8967617 |
|2352905431|purchase|1.0 |Dota 2 |1 |0.8967617 |
|1977328821|purchase|1.0 |Dota 2 |1 |0.9734962 |
|1883667961|purchase|1.0 |Dota 2 |1 |0.9734962 |
|2375812291|purchase|1.0 |Dota 2 |1 |0.9734962 |
|2708593361|purchase|1.0 |Dota 2 |1 |0.96982646 |
|2053393101|purchase|1.0 |Dota 2 |1 |0.96187985 |
|1121348321|purchase|1.0 |Dota 2 |1 |0.96187985 |
|2158323581|purchase|1.0 |Dota 2 |1 |0.9604784 |
+-----+-----+

DataFrame[ID: int, mem_Type: string, value: double, game_Name: string, game_ID: int, prediction: float]
Command took 0.15 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 12:51:58 PM on BDIT_TASK_2_20_APR_1

```

Function giving recommendation filtered by user ID

```

Cmd 62:
1   #Function give recommendations filtered by user id
2   def recommendationsByUserID(ID):
3       # Filter ordered predictions based on provided user ID
4       user_recommendations = ordered_predictions.filter(ordered_predictions.ID == ID)
5
6       # Order the filtered results by prediction in descending order
7       ordered_user_recommendations = user_recommendations.orderBy("prediction", ascending=False)
8
9       # Show all the recommendations for the user ID without truncation
10      ordered_user_recommendations.show(truncate=False)
11
12      # Return the DataFrame containing the recommendations for the user ID
13      return ordered_user_recommendations
14

Command took 0.06 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 12:53:42 PM on BDIT_TASK_2_20_APR_1

```

```

Cmd 63:
1   recommendationsByUserID(59945701)

* (5) Spark Jobs
+-----+-----+
|ID |mem_Type|value|game_Name |game_ID|prediction|
+-----+-----+
|59945701|purchase|1.0 |Psychonauts |1967 |1.0001882 |
|59945701|purchase|1.0 |Cities in Motion 2 |2731 |1.000336 |
|59945701|purchase|1.0 |Orcs Must Die! |3213 |1.0004743 |
|59945701|purchase|1.0 |Company of Heroes Opposing Fronts|4170 |1.0012639 |
|59945701|purchase|1.0 |FINAL FANTASY XIII |3895 |1.0000488 |
|59945701|purchase|1.0 |Magicka |2242 |0.9993508 |
|59945701|purchase|1.0 |Fallout New Vegas Dead Money |3497 |0.9978061 |
|59945701|purchase|1.0 |Company of Heroes 2 |561 |0.99668574 |
|59945701|purchase|1.0 |The Elder Scrolls V: Skyrim |2610 |0.98496807 |
+-----+-----+

DataFrame[ID: int, mem_Type: string, value: double, game_Name: string, game_ID: int, prediction: float]
Command took 7.70 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 12:55:18 PM on BDIT_TASK_2_20_APR_1

```

Step 6: Generate top 10 recommendations for each user. Display the first 5 recommendations for each user

Top 10 recommendations for users

```

Cmd 65
1 userRecs.show(truncate=False)
2
3 # (4) Spark Jobs
4
5 +-----+
6 |ID|recommendations
7 +-----+
8 |176767|[({4905, 1.5527669}, {1842, 3.5277522}, {1829, 1.2875135}, {2523, 1.1803238}, {4228, 1.1769731}, {775, 1.1496066}, {518, 1.1320826}, {4094, 1.1023601}, {3609, 1.0937335}, {4661, 1.0923998}]|
9 |144736|[({4905, 1.7227709}, {1842, 1.7155082}, {1829, 1.3510804}, {518, 1.2394433}, {2523, 1.2377787}, {4228, 1.2054207}, {775, 1.1891726}, {951, 1.1681924}, {2957, 1.1608532}, {3352, 1.1634471}]|
10 |22991|[({4905, 1.6115963}, {1842, 1.6015139}, {1829, 1.089021}, {4228, 1.2287848}, {2523, 1.210366}, {775, 1.19924}, {518, 1.180209}, {2957, 1.1410441}, {3110, 1.1392164}, {3609, 1.1381712}]|
11 |83501|[({1842, 1.7261893}, {4905, 1.7173065}, {1829, 1.3506272}, {2523, 1.234631}, {518, 1.234618}, {4228, 1.1937861}, {775, 1.1763262}, {2957, 1.1623528}, {3392, 1.1609262}, {951, 1.1587856}]|
12 |94836|[({4905, 1.6897531}, {1842, 1.5981476}, {1829, 1.2944885}, {4228, 1.2025548}, {518, 1.1995176}, {775, 1.1813927}, {2523, 1.1740778}, {4094, 1.1588467}, {951, 1.1468440}, {3609, 1.1458846}]|
13 |975449|[({1842, 1.7232509}, {4905, 1.5228887}, {1829, 1.3044542}, {2523, 1.1808693}, {518, 1.1195247}, {3192, 1.0912535}, {2957, 1.0755268}, {4094, 1.0619423}, {3110, 1.0530982}, {3437, 1.0524952}]|
14 |1260792|[({4905, 1.720592}, {1842, 1.7194513}, {1829, 1.3480662}, {518, 1.2338754}, {2523, 1.2318735}, {4228, 1.1968924}, {775, 1.1799517}, {951, 1.1609999}, {2957, 1.1603899}, {3392, 1.1591319}]|
15 |2531540|[({4905, 1.6623276}, {1842, 1.6524051}, {1829, 1.255583}, {775, 1.2145339}, {518, 1.203613}, {4228, 1.1838813}, {2523, 1.1808993}, {3609, 1.1664426}, {3110, 1.1594594}, {2957, 1.1588467}]

Command took 31.45 seconds -- by r.vellakkadapanburkarthikeyan@edu.salford.ac.uk at 4/28/2024, 12:57:22 PM on BDT_TASK_2_29_APR_2024

```

Recommendations for specific user

```

Cmd 67
1 from pyspark.sql.functions import explode, col
2
3 # Function to get game recommendations for a specific user ID and join with game details
4 def gameRecommendationsForUser(user_ID):
5     # Filter the recommendations for the specific user ID from userRecs
6     user_recommendations = userRecs.filter(col("ID") == user_ID)
7
8     # Explode the recommendations column to work with each recommendation separately
9     user_recommendations_exploded = user_recommendations.select("ID", explode("recommendations").alias("recommendation"))
10
11     # Join the exploded recommendations with the distinct games DataFrame
12     # Use the game_ID from the recommendations and the distinct_games_df DataFrame to join
13     recommended_games = user_recommendations_exploded.join(distinct_games_df,
14         user_recommendations_exploded["recommendation.game_ID"] == distinct_games_df["game_ID"],
15         how="inner")
16     .select("game_ID", "game_Name", "recommendation.rating")
17     .orderBy(col("recommendation.rating").desc())
18
19     # Display the recommended games with their ratings (predictions)
20     recommended_games.show(truncate=False)
21
22     return recommended_games
23
24 # Call the function for a specific user ID (e.g., user ID = 1)
25 gameRecommendationsForUser(76767)
26

```

```

Cmd 67
1 gameRecommendationsForUser(109323647)
2
3 # (4) Spark Jobs
4
5 +-----+
6 |game_ID|game_Name|rating|
7 +-----+
8 |1842|Star Trek D-A-C|1.8329778|
9 |4905|Duke Nukem Forever|1.6623802|
10 |1829|Construction Simulator 2015|1.4040481|
11 |2523|Construction Simulator 2015 Vertical Skyline|1.2796733|
12 |518|Cities in Motion|1.2415711|
13 |4228|The LEGO Movie - Videogame|1.2242678|
14 |3392|SONIC THE HEDGEHOG 4 Episode II|1.2024618|
15 |3110|Railroad Tycoon 2 Platinum|1.1917119|
16 |4096|Freedom Fall|1.1873692|
17 |2957|Tank Universal|1.187072|
18
19
20 DataFrame[game_ID: int, game_Name: string, rating: float]
21
22 Command took 25.75 seconds -- by r.vellakkadapanburkarthikeyan@edu.salford.ac.uk at 4/19/2024, 8:30:30 PM on BDT_TASK_2_19_APR_2024

```

All the predictions given in the section 2.7, 2.8, 2.9 are for maxIter 5 and regParam 0.01. The section 2.10 explains about hyperparameter tuning applied to the recommender system and the best model results.

2.8 Collaborative Filtering Recommender System based on 'Play' User Behavior

The steps involved in the machine learning algorithm can be briefed as follows.

Step 1: Filter the Data for 'play' type



```

1 from pyspark.sql.functions import col
2
3 # Step 1: Filter the data for 'play' type
4 play_joined_df = joined_df.filter(col("mem_Type") == "play")
5
6 # Display the play_joined_df DataFrame
7 play_joined_df.show(truncate=False)
8

```

(3) Spark Jobs

play_joined_dt: pyspark.sql.DataFrame[Dataframe = [ID: integer, mem_Type: string ... 3 more fields]]

ID	mem_Value	game_Name	game_ID
151680712	play	[275,0]The Elder Scrolls V Skyrim	[2810]
151680712	play	[87,4]Fallout 4	[411]
151680712	play	[14,9]Fallout 3	[3889]
151680712	play	[12,1]Fallout: New Vegas	[5621]
151680712	play	[8,8]Left 4 Dead 2	[70]
151680712	play	[8,5]HomePop	[3741]
151680712	play	[8,2]Path of exile	[1265]
151680712	play	[7,5]Grand Theft Auto V	[4117]
151680712	play	[7,1]Left 4 Dead	[10929]
151680712	play	[2,8]Team Fortress 2	[5894]
151680712	play	[2,5]Tom Raider	[1755]
151680712	play	[2,0]The Banner Saga	[3342]
151680712	play	[1,4]Dead Island: Riptide	[1561]
151680712	play	[1,1]Grand Theft Auto Infinite	[2730]
151680712	play	[1,1]Dragon Age: Origins - Ultimate Edition	[654]
151680712	play	[0,8]Fallout 3 - Game of the Year Edition	[4024]
151680712	play	[0,8]Sega Genesis & Mega Drive Classics	[374]
151680712	play	[0,6]Grand Theft Auto IV	[3518]

Command took 1.70 seconds -- by r.vellakalappanbukurthikeyandu.ssf@ordi.ac.uk at 4/20/2024, 2:29:29 PM on BDT_TASK_2_2B_APP_1

Step 2: Split the data into training (80%) and test (20%) sets



```

1 # Split the data into training (80%) and test (20%) sets
2 (train_play, test_play) = play_joined_df.randomSplit([0.8, 0.2], seed=100)
3

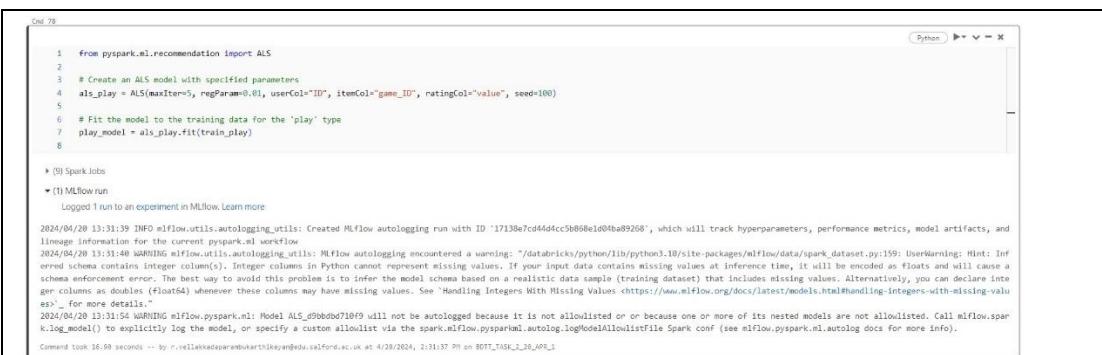
```

train_play: pyspark.sql.DataFrame[DataFrame = [ID: integer, mem_Type: string ... 3 more fields]]

test_play: pyspark.sql.DataFrame[DataFrame = [ID: integer, mem_Type: string ... 3 more fields]]

Command took 0.86 seconds -- by r.vellakalappanbukurthikeyandu.ssf@ordi.ac.uk at 4/20/2024, 2:30:21 PM on BDT_TASK_2_2B_APP_1

Step 3: Train the ALS model



```

1 from pyspark.ml.recommendation import ALS
2
3 # Create an ALS model with specified parameters
4 als_play = ALS(maxIter=5, regParam=0.01, userCol="ID", itemCol="game_ID", ratingCol="value", seed=100)
5
6 # Fit the model to the training data for the 'play' type
7 play_model = als_play.fit(train_play)
8

```

(9) Spark Jobs

(1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

2024/04/20 13:31:38 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '17130e7cd44d4cc5b868e1d04ba89268', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow.

2024/04/20 13:31:40 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/data/rvella/python/lib/python3.10/site-packages/mlflow/dataset.py":159: UserWarning: Hint: Inferring 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 (FloatType) 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>)."

2024/04/20 13:31:54 WARNING mlflow.pyspark.ml: Model ALS_5b0bdb70f9 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.K.logModel() 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 36.59 seconds -- by r.vellakalappanbukurthikeyandu.ssf@ordi.ac.uk at 4/20/2024, 2:31:37 PM on BDT_TASK_2_2B_APP_1

MaxIter = 5, regParam = 0.01

Step 4: Evaluate the model's performance using the test data

```

1 # Use the trained model (play_model) to generate predictions on the test data (test_play)
2 play_predictions = play_model.transform(test_play).dropna()
3
4 # Display the predictions without truncation
5 play_predictions.show()

+---+
| ID|mem_type|value| game_Name|game_ID| prediction|
+---+
| 5259| play| 13.0| Team Fortress 2| 1018| 30.695396|
| 7507| play| -4.4| Mass Effect: Deadly Nexus| 3446| 0.495291|
| 7676| play| 12.5| Call of Duty: Black Ops 2| 24088| 66.532946|
| 7677| play| 25.0| Counter-Strike: Source| 45| -347.70978|
| 7677| play| 365.0| Counter-Strike| 3690| -483.53235|
| 298590| play| 0.1| Deus Ex Game of the Year| 3077| 7.0383816|
| 298590| play| 0.2| Dragon Age: Origin...| 654| 361.30627|
| 298590| play| 0.2| Euro Truck Simulator| 3079| -53.969936|
| 298590| play| 0.2| Octodad Dadliest...| 1019| -0.83708967|
| 298590| play| 0.2| PROTOTYPE 2| 3870| 43.14355|
| 298590| play| 0.2| Scribblenauts Unlim...| 2564| 27.161074|
| 298590| play| 0.2| Shadow Warrior| 1334| 13.16288|
| 298590| play| 0.4| Portal| 2464| -0.0000004|
| 298590| play| 0.4| Portal 2| 1208| 4.8061293|
| 298590| play| 0.5| Counter-Strike: Global Offens...| 3690| 466.48820|
| 298590| play| 0.6| Age of Empires II...| 246| -28.974216|
| 298590| play| 0.7| Alpha Protocol| 48| 5.8887914|
| 298590| play| 0.8| Surgeon Simulator| 4354| 4.9576592|

```

Command took 6.84 seconds -- by r.vellikkadaparamakarthikeyan@eduroadford.ac.uk at 4/20/2024, 2:32:30 PM on 8077_TASK_2_28_APP_1

Step 5: Explore some of the resulting recommendations

Displaying sorted predictions

```

+---+
| ID|mem_type|value| game_Name|game_ID| prediction|
+---+
| 1009764273| play| 0.6| Team Fortress 2| 3894| 5.09093|
| 10095132| play| 1.5| Brahmastra| 2991| -0.020365105|
| 100866738| play| 0.7| Magic: The Gathering| 2282| -0.01734924|
| 100933901| play| 10.0| Counter-Strike Global Offens...| 2300| 0.7995980|
| 100933901| play| 0.1| Portal| 2464| -0.0000004|
| 100971738| play| 0.3| Arcepion 3 2 Gates...| 3735| 8.0538874|
| 100971738| play| 0.5| Limbo| 173| 2.0410746|
| 100971738| play| 0.7| Team Fortress 2| 3894| 2.3996201|
| 100971738| play| 1.0| Portal| 2464| -0.0000004|
| 100971738| play| 2.1| Red Crucible| 1039| 0.0000000000000001|
| 100971738| play| 0.2| Dungeon Defenders| 1531| 3.1790954|
| 100971738| play| 7.0| Team Fortress 2| 3894| 2.1598748|
| 100424080| play| 0.2| Unturned| 3439| 0.417923|
| 100424080| play| 0.4| Counter-Strike: Global Offens...| 2300| 0.7995989|
| 100708054| play| 0.5| Assetto Corsa| 352| 0.5553084|
| 100407398| play| 2.5| Assetto Corsa| 352| 0.5553089|
| 100126013| play| 0.2| Warframe| 1368| 0.15216118|
| 100129381| play| 0.9| Age of Mythology| 2773| 0.94345534|
| 100380171| play| 0.7| Half-Life 2| 361| -0.01820663|
+---+
only showing top 20 rows


```

Command took 0.03 seconds -- by r.vellikkadaparamakarthikeyan@eduroadford.ac.uk at 4/20/2024, 2:33:10 PM on 8077_TASK_2_28_APP_2

Predictions listed by ordering game Id in ascending order

```

+---+
| ID|mem_type|value| game_Name|game_ID| prediction|
+---+
| 58899462| play| 0.2| Dots 2| [1| 14.5671107| |
| 30780104| play| 11.0| Dots 2| [1| -3.1818922| |
| 86287245| play| 2.6| Dots 2| [1| -4.1594954| |
| 45264645| play| 123.0| Dots 2| [1| 167.51844| |
| 19971908| play| 100.0| Dots 2| [1| 97.27800| |
| 22467994| play| 111.8| Dots 2| [1| 104.03020| |
| 36557643| play| 150.0| Dots 2| [1| -17.724874| |
| 4824107| play| 191.8| Dots 2| [1| -1289.0221| |
| 59825286| play| 7.0| Dots 2| [1| 144.35585| |
| 8585433| play| 33.0| Dots 2| [1| 378.38028| |
| 51272054| play| 129.0| Dots 2| [1| -12.120076| |
| 30405090| play| 129.0| Dots 2| [1| -1.4248010| |
| 61772065| play| 122.0| Dots 2| [1| 120.91584| |
| 8865447| play| 11.3| Dots 2| [1| 0.4827721| |
| 10962398| play| 1319.0| Dots 2| [1| -29.3795803| |
| 12526579| play| 0.5| Dots 2| [1| 559.71967| |
| 61380811| play| 11.0| Dots 2| [1| 156.96729| |
| 35701646| play| 1764.0| Dots 2| [1| 199.95795| |


```

Command took 4.57 seconds -- by r.vellikkadaparamakarthikeyan@eduroadford.ac.uk at 4/20/2024, 2:34:04 PM on 8077_TASK_2_28_APP_3

Ordering the predictions based on user ID (descending) and game ID (ascending)

```

1 from pyspark.sql.functions import desc, asc
2
3 # Order the play_predictions DataFrame based on ID in descending order and game_ID in ascending order
4 ordered_play_predictions = play_predictions.orderBy(desc("ID"), asc("game_ID"))
5
6 # Show the ordered play prediction
7 ordered_play_predictions.show()

```

(5) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
309404240	play	13.0	Unturned	3439	2.6611562
309062959	play	0.7	Heroes & Generals	1942	-0.0263592
309062975	play	0.7	Team Fortress 2	3073	2.3500001
3088695132	play	0.5	Brawlhalla	2991	0.02655105
308468736	play	0.7	Magic: The Gathering	3282	-0.01751492
307915393	play	10.1	Counter-Strike Global Offensive	2354	-0.7895598
307674835	play	0.4	Codename: CURE	631	0.024094163
306971738	play	0.5	Lineage II	173	2.041676-6
306971738	play	0.3	Awaypoint: 3 Gates...	3751	8.953887e-4
306971738	play	0.3	Team Fortress 2	3073	2.3500001
306862038	play	0.1	Nidhogg	1941	0.02777231
304971849	play	0.2	Dungeon Defenders II	1533	3.1792054
304555788	play	7.0	Team Fortress 2	3894	2.8585746
304224083	play	0.2	Unturned	3439	0.4173023
3040637281	play	0.2	Carpe Diem	3254	0.13705693
303789664	play	0.5	Undertale	3939	-8.555284
303467988	play	2.5	Assetto Corsa	1932	-0.5633089
303526013	play	0.2	WARFACE	1360	0.15210181

Command took 0.21 seconds -- by r.veliakkadaparamakumar@keyone.edu.sa on 4/26/2024, 2:34:13 PM on EOTT_TASK_2_10_APR_1

Evaluating RMSE and MAE

```

1 from pyspark.ml.evaluation import RegressionEvaluator
2
3 # Create a RegressionEvaluator for RMSE
4 play_evaluator = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")
5
6 # Evaluate the RMSE of play_predictions
7 play_rmse = play_evaluator.evaluate(play_predictions)
8 print(f"Root Mean Square Error (RMSE) for 'play' predictions is: {play_rmse:.4f}")
9
10 # Create a RegressionEvaluator for MAE
11 evaluator_play_mae = RegressionEvaluator(metricName="mae", labelCol="value", predictionCol="prediction")
12 play_mae = evaluator_play_mae.evaluate(play_predictions)
13 print(f"Mean absolute error (MAE) is: {play_mae}")
14

```

(10) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
307620122	play	111.6	APK Reloaded	1897	137074.508
30207081	play	133.0	Area 3	1530	15152.443
31795818	play	0.3	Counter-Strike	3690	13423.424
111162598	play	13.9	Men of War: Assault Squad	2653	8594.098
23492094	play	16.6	Sid Meier's Civilization V	963	8349.804
67720709	play	39.0	Counter-Strike	3690	7740.3496
52056164	play	127.0	Train Simulator	856	7487.0523
52056164	play	128.0	Counter-Strike Global Offensive	1235	7487.0523
23492094	play	0.3	Warframe	2679	1688.3272
30790853	play	10.2	Area 3	1530	16698.2183
11112884	play	0.4	LEGO MARVEL Super Heroes	225	6318.757
61653738	play	11.6	PlanetSide 2	3442	8175.6459
105163018	play	5.9	Europa Universalis IV	2804	5873.382
62263745	play	120.0	Counter-Strike	3690	5492.3223
190305835	play	4.7	Mount & Blade Warband	226	5374.555
80384871	play	18.2	Sid Meier's Civilization V	963	5336.6436
2099617175	play	130.0	APK Reloaded	1897	4868.489
151608301	play	11.2	Data 2	1	4776.953

Command took 11.75 seconds -- by r.veliakkadaparamakumar@keyone.edu.sa on 4/26/2024, 2:46:13 PM on EOTT_TASK_2_10_APR_1

Predictions ordered in descending order

```

1 # Order the predictions DataFrame based on prediction in descending order
2 ordered_play_predictions = play_predictions.orderBy("prediction", ascending=False)
3
4 # Show the ordered predictions
5 ordered_play_predictions.show(truncate=False)
6

```

(5) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
176520122	play	111.6	APK Reloaded	1897	137074.508
30207081	play	133.0	Area 3	1530	15152.443
31795818	play	0.3	Counter-Strike	3690	13423.424
111162598	play	13.9	Men of War: Assault Squad	2653	8594.098
23492094	play	16.6	Sid Meier's Civilization V	963	8349.804
67720709	play	39.0	Counter-Strike	3690	7740.3496
52056164	play	127.0	Train Simulator	856	7487.0523
52056164	play	128.0	Counter-Strike Global Offensive	1235	7487.0523
23492094	play	0.3	Warframe	2679	1688.3272
30790853	play	10.2	Area 3	1530	16698.2183
11112884	play	0.4	LEGO MARVEL Super Heroes	225	6318.757
61653738	play	11.6	PlanetSide 2	3442	8175.6459
105163018	play	5.9	Europa Universalis IV	2804	5873.382
62263745	play	120.0	Counter-Strike	3690	5492.3223
190305835	play	4.7	Mount & Blade Warband	226	5374.555
80384871	play	18.2	Sid Meier's Civilization V	963	5336.6436
2099617175	play	130.0	APK Reloaded	1897	4868.489
151608301	play	11.2	Data 2	1	4776.953

Command took 0.35 seconds -- by r.veliakkadaparamakumar@keyone.edu.sa on 4/26/2024, 2:43:10 PM on EOTT_TASK_2_10_APR_1

SQL queries on predictions based on temporary view

```
Cod_87
1 %sql
2 select ID,game_ID, game_Name, prediction from my_play_Predictions order by prediction desc limit 10

```

(5) Spark jobs

`_sqldf: pyspark.sql.DataFrame` = [ID: integer, game_ID: integer ... 2 more fields]

ID	game_ID	game_Name	prediction
1	176929122	1897	APB Reloaded
2	20207081	1530	Arma 3
3	57905818	2690	Counter-Strike
4	111362598	2053	Men of War Assault Squad
5	23492094	963	Sid Meier's Civilization V
6	87740709	3690	Counter-Strike
7	82866146	856	Train Simulator
8	50811344	2354	Counter-Strike Global Offensive
9	23492094	2670	Garry's Mod
10	87040683	1530	Arma 3

↓ 10 rows | 6.76 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[104]`. Learn more

Command took 6.76 seconds -- by r.vellakkadparambukarthikeyan@edu.salford.ac.uk at 4/29/2024, 2:46:11 PM on BDTT_TASK_2_20_APR_1

SQL queries on prediction of game names with word ‘wars’ (same queries are also run on python in the codes)

```
Cod_98
1 %sql
2 select ID,game_ID, game_Name, prediction from my_play_Predictions where game_Name like '%Wars%' order by prediction desc limit 10

```

(5) Spark Jobs

`_sqldf: pyspark.sql.DataFrame` = [ID: integer, game_ID: integer ... 2 more fields]

ID	game_ID	game_Name	prediction
1	69009454	3255	Guild Wars
2	102111851	2488	Infinity Wars - Animated Trading Card Game
3	42061089	3255	Guild Wars
4	82385352	247	Star Wars Jedi Knight Jedi Academy
5	103260848	628	Star Wars - Battlefront II
6	11748479	247	Star Wars Jedi Knight Jedi Academy
7	102263723	2488	Infinity Wars - Animated Trading Card Game
8	11373749	1594	Star Wars Knights of the Old Republic
9	44321815	247	Star Wars Jedi Knight Jedi Academy
10	43955374	2488	Infinity Wars - Animated Trading Card Game

↓ 10 rows | 6.97 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[105]`. Learn more

Command took 6.97 seconds -- by r.vellakkadparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 2:48:14 PM on BDTT_TASK_2_20_APR_1

SQL queries on prediction for game IDs (18, 24)

Cmd 89

```

1 %sql
2 | select ID,game_ID, game_Name, prediction from my_play_Predictions where game_ID In ( 18, 24) order by prediction desc limit 10

```

(5) Spark Jobs

```

▶ _sqldf: pyspark.sql.dataframe.DataFrame = [ID: integer, game_ID: integer ... 2 more fields]

```

Table +

ID	game_ID	game_Name	prediction
1	37422528	Call of Duty Modern Warfare 2 - Multiplayer	3615.7046
2	43908860	Call of Duty Modern Warfare 2 - Multiplayer	1941.437
3	61189155	Call of Duty Modern Warfare 2 - Multiplayer	1614.936
4	125017535	Call of Duty Modern Warfare 2 - Multiplayer	1547.8528
5	74347105	Call of Duty Modern Warfare 2 - Multiplayer	1097.4738
6	78309377	Call of Duty Modern Warfare 2 - Multiplayer	597.69934
7	55334302	Call of Duty Modern Warfare 2 - Multiplayer	515.8308
8	56436989	Call of Duty Modern Warfare 2 - Multiplayer	483.5675
9	82352462	Call of Duty Modern Warfare 2 - Multiplayer	418.8179
10	55751308	Call of Duty Modern Warfare 2 - Multiplayer	299.90842

↓ 10 rows | 6.50 seconds runtime

Refreshed now

① This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[106]`. Learn more

Command took 6.50 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:49:19 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by game name

Cmd 92

```

1 #Function to give recommendation filtered by name
2 def recommendationsByGamePlayName(name):
3     filter = '%' + name + '%'
4     recommendationsList = ordered_play_predictions.orderBy("prediction", ascending=False).filter(ordered_play_predictions.game_Name.like(filter)).limit(10).show
5     return recommendationsList

```

Command took 0.87 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:53:33 PM on BDTT_TASK_2_20_APR_1

Cmd 93

Cmd 93

```

1 recommendationsByGamePlayName('Duty')

```

(5) Spark Jobs

ID	meas_Type	value	game_Name	game_ID	prediction
37422528	play	79.0	[Call of Duty Modern Warfare 2 - Multiplayer]18	18	3615.7046
93189271	play	102.0	[Call of Duty Modern Warfare 3 - Multiplayer]2029	2029	2198.523
43908860	play	155.0	[Call of Duty Modern Warfare 2 - Multiplayer]18	18	1941.437
29753085	play	0.1	[Call of Duty Modern Warfare 3 - Multiplayer]2029	2029	1799.5476
8776918	play	46.0	[Call of Duty Modern Warfare 3 - Multiplayer]2029	2029	1663.0079
61189155	play	227.0	[Call of Duty Modern Warfare 2 - Multiplayer]18	18	1614.936
75134679	play	82.0	[Call of Duty Modern Warfare 3 - Multiplayer]2029	2029	1565.4069
125017535	play	6.4	[Call of Duty Modern Warfare 2 - Multiplayer]18	18	1547.8528
107377573	play	287.0	[Call of Duty Black Ops II - Multiplayer]	1682	1104.032
74347105	play	345.0	[Call of Duty Modern Warfare 2 - Multiplayer]18	18	1097.4738

Command took 0.88 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:54:06 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by game ID

```

Cmd 94
1 #Function to give recommendation filtered by game id
2 def recommendationsByGamePlayID(game_IDs):
3     # Filter ordered predictions based on provided game_IDs
4     filtered_play_predictions = ordered_play_predictions.filter(ordered_play_predictions.game_ID.isin(game_IDs))
5
6     # Order by prediction in descending order and limit to top 10 results
7     top_recommendations_play = filtered_play_predictions.orderBy("prediction", ascending=False).limit(10)
8
9     # Show the top recommendations without truncation
10    top_recommendations_play.show(truncate=False)
11
12    # Return the DataFrame containing the top recommendations
13    return top_recommendations_play
14

Command took 0.04 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:54:58 PM on BDTT_TASK_2_20_APP_1

```

```

Cmd 95
1 recommendationsByGamePlayID(1)

* (5) Spark Jobs
+-----+-----+
|ID |mem_Type|value |game_Name|game_ID|prediction|
+-----+-----+
|151600301|play |1.2 |Data 2 |1 |4776.953 |
|79319875|play |29.0 |Data 2 |1 |1824.8347 |
|59096740|play |1794.0|Data 2 |1 |914.5463 |
|21535912|play |8.0 |Data 2 |1 |878.619 |
|78053539|play |8.2 |Data 2 |1 |874.91876 |
|83975567|play |318.0|Data 2 |1 |868.5405 |
|104721942|play |3.3 |Data 2 |1 |889.60004 |
|134223421|play |0.3 |Data 2 |1 |729.785 |
|43360645|play |2390.0|Data 2 |1 |675.4644 |
|44321815|play |1054.0|Data 2 |1 |665.3171 |
+-----+-----+

DataFrame[ID: int, mem_Type: string, value: double, game_Name: string, game_ID: int, prediction: float]
Command took 7.81 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:02:01 PM on BDTT_TASK_2_20_APP_1

```

Function giving recommendation filtered by user ID

```

Cmd 96
1 #Function to give recommendation filtered by user id
2 def recommendationsByUserPlayID(ID):
3     # Filter ordered predictions based on provided user ID
4     user_recommendations_play = ordered_play_predictions.filter(ordered_play_predictions.ID == ID)
5
6     # Order the filtered results by prediction in descending order
7     ordered_user_recommendations_play = user_recommendations_play.orderBy("prediction", ascending=False)
8
9     # Show all the recommendations for the user ID without truncation
10    ordered_user_recommendations_play.show(truncate=False)
11
12    # Return the DataFrame containing the recommendations for the user ID
13    return ordered_user_recommendations_play
14

Command took 0.04 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:02:53 PM on BDTT_TASK_2_20_APP_1

```

```

Cmd 97
1 recommendationsByUserPlayID(43908860)

* (5) Spark Jobs
+-----+-----+
|ID |mem_Type|value |game_Name|game_ID|prediction|
+-----+-----+
|43908860|play |155.0|Call of Duty Modern Warfare 2 - Multiplayer|18 |1941.437 |
|43908860|play |24.0 |Far Cry 3 |4119 |81.10817 |
|43908860|play |4.7 |Half-Life 2 |346 |36.620537 |
|43908860|play |17.4 |Call of Duty Black Ops |1971 |29.139709 |
+-----+-----+

DataFrame[ID: int, mem_Type: string, value: double, game_Name: string, game_ID: int, prediction: float]
Command took 6.04 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:03:20 PM on BDTT_TASK_2_20_APP_1

```

Step 6: Generate top 10 recommendations for each user. Display the first 5 recommendations for each user

Top 10 recommendations for users

```
Cmd 99
1 userRecs_play.show(truncate=False)
2
3 # (2) Spark Jobs
4
5 +-----+
6 |ID      |recommendations
7 |
8 +-----+
9 |76767  |[(2203, 4199.766), (3255, 1953.0667), (3698, 1605.7485), (356, 1575.394), (1451, 1459.061), (2085, 1327.3253), (2588, 1316.5726), (4355, 1239.2446), (3203, 1154.2031), (23
10 3, 1193.4972)]
11 |1229911 |[(3698, 6215.9077), (1586, 4334.606), (2542, 4317.0996), (2106, 3783.7915), (4922, 3399.3816), (1901, 3299.0073), (1098, 3040.7278), (467, 2633.402), (3501, 2625.207), (13
12 98, 2582.9548)]
13 |1948368 |[(2804, 263.8825), (467, 186.04253), (1897, 174.1855), (3203, 143.24677), (1554, 117.32468), (2830, 108.03887), (2295, 86.523636), (45, 82.99983), (2540, 80.149956), (124
14 2, 69.65848)]
15 |1975484 |[(3698, 241.05455), (2295, 210.02835), (317, 192.05493), (3203, 171.73044), (1586, 140.03279), (1398, 104.18551), (2085, 103.45003), (2106, 93.82693), (2830, 89.28183),
16 (312, 85.370804)]
17 |1268792 |[(310, 0.5887631), (2342, 0.56960773), (3690, 0.45999994), (2804, 0.4796917), (317, 0.46284655), (290, 0.31627083), (427, 0.30924162), (2295, 0.3053782), (1241, 0.292914
18 48), (3255, 0.26125038)]
19 |12531540 |[(3203, 1571.48489), (2690, 912.8006), (1507, 901.0282), (2106, 791.1619), (226, 708.7439), (3255, 707.1464), (2540, 702.1155), (1451, 701.0715), (1897, 697.9845), (3177, 6
20 87.53107)]
21 |1275325 |[(3177, 857.6364), (2295, 706.35846), (3255, 640.4785), (1241, 615.049), (1451, 556.9387), (2085, 530.3844), (4355, 406.3319), (1897, 379.24884), (1586, 370.82578), (2540,
22 324.94928)]
23 |13450426 |[(2342, 310.23495), (4922, 286.8648), (4125, 226.24982), (1118, 157.65955), (2295, 137.453), (3177, 130.57567), (312, 129.53894), (1451, 122.648735), (290, 101.58583), (10
24 Command took 22.18 seconds -- by r.vellakkadaparumbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:04:22 PM on BDIT_TASK_2_28_APR_1
```

Recommendations for specific user

```
Cmd 100
1 from pyspark.sql.functions import explode, col
2
3 # Function to get game recommendations for a specific user ID and join with game details
4 def gameRecommendationsForUserPlay(userID):
5     # Filter the recommendations for the specific user ID from userRecs
6     user_recommendationsPlay = userRecs_play.filter(col("ID") == userID)
7
8     # Explode the recommendations column to work with each recommendation separately
9     user_recommendationsPlay_exploded = user_recommendationsPlay.select("ID", explode("recommendations").alias("recommendation"))
10
11     # Join the exploded recommendations with the distinct game information
12     # use the game_ID from the recommendations and the distinct_games_DF DataFrame to join
13     recommended_gamePlay = user_recommendationsPlay_exploded.join(distinct_games_DF,
14         user_recommendationsPlay_exploded["recommendation.game_ID"] == distinct_games_DF["game_ID"],
15         "left")
16
17     .select("game_ID", "game_name", "recommendation.rating")
18     .orderBy(col("recommendation.rating").desc())
19
20     # Display the recommended games with their ratings (predictions)
21     recommended_gamePlay.show(truncate=False)
22
23     return recommended_gamePlay
24
25 # Call the function for a specific user ID (e.g., user ID = 1)
26 gameRecommendationsForUserPlay(1)
27
28 # (1) Spark Jobs
29
30 +-----+
31 |game_ID|game_Name          |rating   |
32 +-----+
33 |1229911|Football Manager 2010 |1459.766|
34 |1229911|Football Manager 2012 |1398.2817|
35 |1229911|Final Fantasy XIV: A Realm Returns |1368.7486|
36 |1394 |Genshin Impact |1357.394|
37 |1394 |Grand Theft Auto V |1352.3231|
38 |1242 |Grand Theft Auto VI |1327.2328|
39 |1312 |Leisure's Army: Praying Grounds |1331.5726|
40 |13255|Lionheart: The Longest Alliance |1324.2814|
41 |1118 |Sight of Flame |1319.2811|
42 |1229911|Football Manager 2011 |1181.4972|
43
44 +-----+
45
46 DataFrame[game_ID: int, game_Name: string, rating: float]
47
48 Command took 20.34 seconds -- by r.vellakkadaparumbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:04:49 PM on BDIT_TASK_2_28_APR_1
```

```
Cmd 101
1 gameRecommendationsForUserPlay(109323647)
2
3 # (4) Spark Jobs
4
5 +-----+
6 |game_ID|game_Name          |rating   |
7 +-----+
8 |2295 |X-Plane 10 Global - 64 Bit |4658.6436|
9 |4125 |Football Manager 2014 |3426.2288|
10 |2342 |Baldur's Gate Enhanced Edition |3129.1882|
11 |1507 |Football Manager 2015 |1866.6298|
12 |3177 |Marvel Puzzle Quest |1725.3738|
13 |312 |OMSI 2 |1520.0271|
14 |1530 |Arma 3 |1469.7559|
15 |4678 |NOBUNAGA'S AMBITION Kakushin with Power Up Kit |1329.8258|
16 |1118 |Nobunaga's Ambition Souzou with Power Up Kit |1268.7926|
17 |3255 |Guild Wars |1179.1804|
18
19 +-----+
20
21 DataFrame[game_ID: int, game_Name: string, rating: float]
22
23 Command took 26.21 seconds -- by r.vellakkadaparumbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:14:31 PM on BDIT_TASK_2_28_APR_1
```

2.9 Collaborative Filtering Recommender System based on User Behavior ‘Purchase’ and ‘Play’

The steps involved in the machine learning algorithm can be briefed as follows.

Step 1: Filter the Data for ‘purchase’ and ‘play’ type



```

1 from pyspark.sql.functions import col
2
3 # Step 1: Filter the data for 'play' type
4 purchase_play_df = joined_df.filter(col("item_type").isin(["purchase", "play"]))
5
6 # Display the (key, joined_df) DataFrame
7 purchase_play_df.show(truncate=False)
  
```

Spark Jobs

ID	item_type	value	game_name	game_ID
151508712	purchase	1.0	[The Elder Scrolls V Skyrim]2010	
151508712	play	173.0	[The Elder Scrolls V Skyrim]2010	
151508712	purchase	1.0	Fallout 4	411
151508712	play	14.0	Fallout 4	411
151508712	purchase	1.0	[Spore]	3869
151508712	play	14.9	[Spore]	3869
151508712	purchase	1.0	Fallout New Vegas	19221
151508712	play	17.1	Fallout New Vegas	19221
151508712	purchase	1.0	(Left 4 Dead 2)	70
151508712	play	16.8	(Left 4 Dead 2)	70
151508712	purchase	1.0	(Left 4 Dead)	13481
151508712	play	8.5	(Left 4 Dead)	13481
151508712	purchase	1.0	[Path of Exile]	1585
151508712	play	16.1	[Path of Exile]	1585
151508712	purchase	1.0	(Left 4 Dead)	1419
151508712	play	7.5	(Left 4 Dead)	1417
151508712	purchase	1.0	(Left 4 Dead)	1993
151508712	play	12.3	(Left 4 Dead)	1992

Current time 2:34 seconds -- by r.vellakkadaparambukarthikayangenu.salford.ac.uk at 4/20/2024, 4:07:54 PM on EDT_TASK_2_20_APACHE

Step 2: Split the data into training (80%) and test (20%) sets



```

Cmd 189
  
```

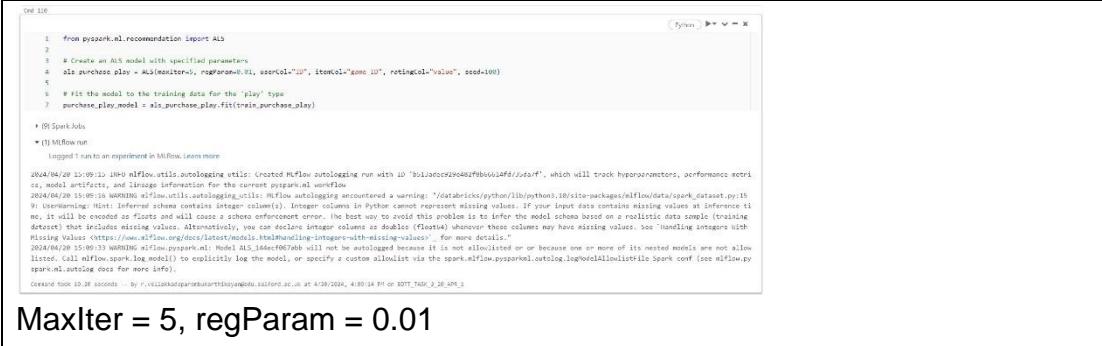
```

1 # Split the data into training (80%) and test (20%) sets
2 (train_purchase_play, test_purchase_play) = purchase_play_df.randomSplit([0.8, 0.2], seed=100)
  
```

train_purchase_play: pyspark.sql.DataFrame = [ID:integer, item_type:string ... 3 more fields]
test_purchase_play: pyspark.sql.DataFrame = [ID:integer, item_type:string ... 3 more fields]

Command took 0.88 seconds -- by r.vellakkadaparambukarthikayangenu.salford.ac.uk at 4/20/2024, 4:09:44 PM on EDT_TASK_2_20_APACHE

Step 3: Train the ALS model



```

Cmd 190
  
```

```

1 from pyspark.ml.recommendation import ALS
2
3 # Create an ALS model with specified parameters
4 als_purchase_play = ALS(maxIter=5, regParam=0.01, userCol="ID", itemCol="game_ID", ratingCol="value", seed=100)
5
6 # Fit the model to the training data for the 'play' type
7 purchase_play_model = als_purchase_play.fit(train_purchase_play)
  
```

(8) Spark Jobs

(1) Miflow run

Logged 1 runs to an experiment in Miflow. Learn more

2024/04/20 15:09:20 INFO Miflow: miflow autologging with file: /mnt/miflow/miflow-autologging/run with ID 'b013a0c924e404fb0b04fd2/cuda/f'. which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow.

2024/04/20 15:09:20 INFO Miflow: WARMING! Miflow autologging encountered a warning: "/databricks/python/lib/python3.10/site-packages/miflow/data/spark_dataset.py:15: Warning: Infered schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference time, it must be encoded as float and will cause a schema error. The best way to solve this problem is to infer the model schema based on a realistic data sample (training data), then encode missing values as float. (Floats allow the columns may have missing values. See 'Handling integers with Missing Values' (<https://www.miflow.org/docs/latest/models.html#handling-integers-with-missing-values>). For more details.)"

2024/04/20 15:09:20 INFO Miflow: Model ALS_14decf047eb will not be autologged because it is not allowedlisted or because one or more of its nested models are not allowedlisted. Call miflow.spark.logModel() to explicitly log model, or specify a custom allowedlist via the spark.miflow.pysparkML.outgoingLogModelAllowedList file spark.miflow.logModelSeeAllowedListForMoreInfo.

Current time 3:30 seconds -- by r.vellakkadaparambukarthikayangenu.salford.ac.uk at 4/20/2024, 4:09:14 PM on EDT_TASK_2_20_APACHE

MaxIter = 5, regParam = 0.01

Step 4: Evaluate the model's performance using the test data

```

Cmd 111
1 # Use the trained model (play_model) to generate predictions on the test data (test_play)
2 purchase_play_predictions = purchase_play_model.transform(test_purchase_play).dropna()
3
4 # Display the predictions without truncation
5 purchase_play_predictions.show()

> (5) Spark jobs
> purchase_play_predictions: pyspark.sql.DataFrameDataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+-----+-----+-----+
| ID|mem_Type|value| game_Name|game_ID| prediction|
+-----+-----+-----+
| 52261| play| 13.0| Portal 2| 10181| -0.34277421|
| 52260| purchase| 1.0| Day of Defeat| 1597| -7.9243903|
| 52260| purchase| 1.0| Half-Life| 2374| 1.1765294|
| 52589| purchase| 1.0| Half-Life Blue Shift| 446| -0.61897085|
| 52589| purchase| 1.0| Team Fortress 2| 3894| 0.7384777|
| 76757| play| 1.2| Half-Life| 2374| 9.411664|
| 76757| play| 9.7| call of Duty Mode...| 2029| -839.78143|
| 76757| play| 12.5| call of Duty Blac...| 2308| -0.020597812|
| 76757| play| 15.0| Portal 2| 10181| -0.8205275|
| 76757| play| 15.9| call of Duty Mode...| 1636| 9.657413|
| 76757| play| 24.0| Banished| 3918| 172.95218|
| 76757| play| 25.0| Counter-Strike So...| 451| 1.0000000|
| 76757| purchase| 1.0| Banished| 3918| 172.95218|
| 76757| purchase| 1.0| call of Duty Mode...| 2029| -839.78143|
| 76757| purchase| 1.0| call of Duty W...| 2379| 359.8921|
| 76757| purchase| 1.0| Half-Life| 2374| 9.411664|
| 76757| purchase| 1.0| The Stanley Parable| 2333| -3.2984027|
| 76757| purchase| 1.0| Thief 2| 1532| 8.939316|
Command took 4.72 seconds -- by r.vellakkadaparamoukarankeyn@eduro.salford.ac.uk at 4/20/2024, 4:11:02 PM on DOTT_TASK_2_29_APR_1

```

Step 5: Explore some of the resulting recommendations

Displaying sorted predictions

```

Cmd 112
1 from pyspark.sql.functions import desc
2
3 # Order the play_predictions DataFrame in descending order of ID
4 purchase_play_predictions_sorted = purchase_play_predictions.orderBy(desc("ID"))
5
6 # Display the sorted DataFrame without truncation
7 purchase_play_predictions_sorted.show()

> (5) Spark jobs
> purchase_play_predictions_sorted: pyspark.sql.DataFrameDataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+-----+-----+-----+
| ID|mem_Type|value| game_Name|game_ID| prediction|
+-----+-----+-----+
| 369464248| play| 2.7| Mitos Is The Game| 3159| 0.0028512|
| 369253771| play| 0.6| MicroWorld Super| 549| 0.0007685|
| 369253771| purchase| 2.0| Braudhalla| 299| 0.9657595|
| 369238508| play| 0.3| Data 2| 1| 0.00000714|
| 369188095| purchase| 1.0| Counter-Strike Ne...| 1860| 6.087011|
| 369188095| purchase| 1.0| Braudhalla| 299| 3.1880617|
| 369101805| purchase| 1.0| Braudhalla| 299| 6.087011|
| 3691017542| purchase| 1.0| Data 2| 1| 6.099981|
| 369095872| play| 5.2| Data 2| 1| 0.00000714|
| 369095872| play| 0.2| Heroes & Generals| 1942| 0.9949225|
| 369038665| purchase| 1.0| Data 2| 1| 2.2999935|
| 368925132| purchase| 1.0| Data 2| 1| 3.2999892|
| 368880998| purchase| 1.0| Team Fortress 2| 3894| 0.5099923|
| 368760273| play| 0.6| Team Fortress 2| 3894| 0.50997247|
| 368760273| purchase| 1.0| Unturned| 3439| 0.50861636|
| 368695132| play| 0.6| Counter-Strike Ne...| 1860| 0.00000711|
| 368653033| purchase| 1.0| theHunter| 4352| 0.01086299|
| 368693685| play| 12.3| Counter-Strike Gl...| 2354| 0.999998|
| 368648738| play| 0.6| War Thunder| 4030| 1.0000000|
Command took 4.88 seconds -- by r.vellakkadaparamoukarankeyn@eduro.salford.ac.uk at 4/20/2024, 4:12:05 PM on DOTT_TASK_2_29_APR_1

```

Predictions listed by ordering game Id in ascending order

```

Cmd 113
1 # Order the play_predictions DataFrame in ascending order of game_ID
2 ordered_purchase_play_predictions = purchase_play_predictions.orderBy("game_ID", ascending=True)
3
4 # Display the ordered DataFrame without truncation
5 ordered_purchase_play_predictions.show(truncate=False)

> (5) Spark jobs
> ordered_purchase_play_predictions: pyspark.sql.DataFrameDataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+-----+-----+-----+
| ID|mem_Type|value| game_Name|game_ID| prediction|
+-----+-----+-----+
| 14533726| purchase| 1.0| Data 2| 1| 1.2984814|
| 11377349| play| 2.2| Data 2| 1| 55.000546|
| 31665921| purchase| 1.0| Data 2| 1| 47.107937|
| 9128165 | purchase| 1.0| Data 2| 1| -5.616095 |
| 21369581| play| 28.0| Data 2| 1| 1.6765032|
| 19910536| purchase| 1.0| Data 2| 1| 14.000000 |
| 25638909| purchase| 1.0| Data 2| 1| 10.000000 |
| 2883971| play| 3.1| Data 2| 1| -1.4884745|
| 17539772| play| 1.1| Data 2| 1| 56.000000 |
| 15325705| purchase| 1.0| Data 2| 1| 14.000000 |
| 10218551| play| 19.3| Data 2| 1| 56.000000 |
| 14544557| purchase| 1.0| Data 2| 1| 94.97991 |
| 32647734| purchase| 1.0| Data 2| 1| 18.363960 |
| 48296163| purchase| 1.0| Data 2| 1| 81.000000 |
| 25051518| play| 15.9| Data 2| 1| 149.000000 |
| 17538772| purchase| 1.0| Data 2| 1| 56.000000 |
| 11605721| play| 24.5| Data 2| 1| -51.377018|
Command took 7.58 seconds -- by r.vellakkadaparamoukarankeyn@eduro.salford.ac.uk at 4/20/2024, 4:12:05 PM on DOTT_TASK_2_29_APR_1

```

Ordering the predictions based on user ID (descending) and game ID (ascending)

```
Cmd 114
```

```

1   from pyspark.sql.functions import desc, asc
2
3   # Order the play_predictions DataFrame based on ID in descending order and game_ID in ascending order
4   ordered_purchase_play_predictions = purchase_play_predictions.orderBy(desc("ID"), asc("game_ID"))
5
6   # Show the ordered play predictions
7   ordered_purchase_play_predictions.show()

```

(5) Spark Jobs

```

ordered_purchase_play_predictions: pyspark.sql.DataFrame[{"ID": integer, "mem_Type": string ... 4 more fields}]

| ID|mem_Type|value|game_Name|game_ID|prediction|
+---+-----+-----+-----+-----+-----+
| 3090404248| play | 2.0 | Mitos Is The Game | 1135 | 0.9828512 |
| 3092653771| play | 2.0 | Brauhalla | 299 | 0.9965795 |
| 3092653771| play | 0.8 | MicroWolts Surge | 949 | 0.9986785 |
| 3091880398| play | 1.0 | Counter-Strike: Global Offensive | 1 | 0.9999791 |
| 3091880398| purchase | 1.0 | Counter-Strike: Global Offensive | 1880 | 0.9999791 |
| 3091880398| purchase | 1.0 | Brauhalla | 299 | 3.1880617 |
| 309197542| purchase | 1.0 | Dota 2 | 2 | 6.699981 |
| 309058572| play | 5.2 | Dota 2 | 1 | 0.99999714 |
| 309052991| play | 0.2 | Heroes & Generals | 1942 | 0.9999234 |
| 3089388661| purchase | 1.0 | Dota 2 | 1 | 2.2999355 |
| 308925132| purchase | 1.0 | Dota 2 | 1 | 3.7999892 |
| 308880098| purchase | 1.0 | Team Fortress 2 | 3894 | 0.9999923 |
| 308760273| purchase | 1.0 | Unturned | 3439 | 0.9806136 |
| 308760273| play | 0.6 | Team Fortress 2 | 3894 | 0.9997247 |
| 308695132| play | 0.6 | Counter-Strike: Global Offensive | 1 | 0.9999211 |
| 308653033| purchase | 1.0 | Unturned | 4352 | 0.011086299 |
| 308503685| play | 12.3 | Counter-Strike Global Offensive | 2354 | 0.9999861 |
| 308468736| play | 0.7 | Magic Duels | 3282 | 1.0007724 |

```

Command took 1.04 seconds -- by r.vellakkadaparanbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:13:34 PM on BD7_TASK_2_20_APR_1

Evaluating RMSE and MAE

```
Cmd 115
```

```

1   from pyspark.ml.evaluation import RegressionEvaluator
2
3   # Create a RegressionEvaluator for RMSE
4   purchase_play_evaluator = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")
5
6   # Evaluate the RMSE of play_predictions
7   purchase_play_rmse = purchase_play_evaluator.evaluate(purchase_play_predictions)
8   print(f"Root Mean Square Error (RMSE) for 'purchase' and 'play' predictions is: {purchase_play_rmse:.4f}")
9
10  purchase_play_evaluator.setMetricName("mae")
11
12  # Change the metricName of the evaluator to 'mae' to evaluate Mean Absolute Error (MAE)
13  evaluator.setMetricName("mae")
14
15  # Evaluate the model on the full training data using MAE
16  purchase_play_mae = purchase_play_evaluator.evaluate(purchase_play_predictions)
17  print(f"Root Mean Square Error (RMSE) for 'purchase' and 'play' predictions is: {purchase_play_mae:.4f}")

(10) Spark Jobs
Root Mean Square Error (RMSE) for 'purchase' and 'play' predictions is: 207.90978
Root Mean Square Error (RMSE) for 'purchase' and 'play' predictions is: 39.5405

Command took 12.46 seconds -- by r.vellakkadaparanbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:16:48 PM on BD7_TASK_2_20_APR_1

```

Predictions ordered in descending order

```
Cmd 116
```

```

1   # Order the predictions DataFrame based on prediction in descending order
2   ordered_purchase_play_predictions = purchase_play_predictions.orderBy("prediction", ascending=False)
3
4   # Show the ordered predictions
5   ordered_purchase_play_predictions.show(truncate=False)

```

(5) Spark Jobs

```

ordered_purchase_play_predictions: pyspark.sql.DataFrame[{"ID": integer, "mem_Type": string ... 4 more fields}]

| ID|mem_Type|value|game_Name|game_ID|prediction|
+---+-----+-----+-----+-----+-----+
| 15382349| purchase | 1.0 | Team Fortress 2 | 3894 | 9639.875 |
| 15382349| purchase | 1.0 | Counter-Strike: Global Offensive | 1 | 9639.875 |
| 198754517| purchase | 1.0 | GameMaker Studio | 158 | 6475.678 |
| 81054122| play | 1137.0 | Football Manager 2014 | 4125 | 6101.131 |
| 81054122| purchase | 1.0 | Football Manager 2014 | 4125 | 6101.131 |
| 708911479| purchase | 1.0 | Sid Meier's Civilization VI | 963 | 6012.927 |
| 708911479| purchase | 1.0 | Sid Meier's Civilization VI | 963 | 5998.934 |
| 5959791| purchase | 1.0 | Sid Meier's Civilization VI | 963 | 5719.993 |
| 98388239| purchase | 1.0 | Dota 2 | 1 | 5141.942 |
| 120725272| purchase | 1.0 | Dota 2 | 1 | 4493.739 |
| 579056517| purchase | 1.0 | Team Fortress Source | 45 | 4494.4238 |
| 120725272| purchase | 1.0 | Team Fortress Source | 45 | 4494.4238 |
| 141798461| purchase | 1.0 | Dota 2 | 1 | 4000.983 |
| 12088241| play | 3426.0 | Counter-Strike | 3698 | 4010.8364 |
| 12088241| purchase | 1.0 | Counter-Strike | 3698 | 4010.8364 |
| 111709355| purchase | 1.0 | Garry's Mod | 2670 | 3995.7262 |
| 187490795| purchase | 1.0 | Dota 2 | 1 | 3511.488 |
| 199977295| purchase | 1.0 | Dota 2 | 1 | 3229.658 |

```

Command took 5.77 seconds -- by r.vellakkadaparanbukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:17:49 PM on BD7_TASK_2_20_APR_1

SQL queries on predictions based on temporary view

```
Cmd 119
1 %sql
2 select ID,game_ID, game_Name, prediction from my_purchase_play_Predictions order by prediction desc limit 10
3
4 (5) Spark Jobs
5 +_sqldf: pyspark.sql.DataFrame = [ID:integer, game_ID:integer ... 2 more fields]
6
7 Table v +
8
9 +-----+-----+-----+-----+
10 | ID | game_ID | game_Name | prediction |
11 +-----+-----+-----+-----+
12 | 1 | 153382649 | Team Fortress 2 | 9639.675 |
13 | 2 | 180753437 | GameMaker Studio | 6475.676 |
14 | 3 | 180753437 | GameMaker Studio | 6475.676 |
15 | 4 | 81054122 | Football Manager 2014 | 6101.131 |
16 | 5 | 81054122 | Football Manager 2014 | 6101.131 |
17 | 6 | 70487610 | Sid Meier's Civilization V | 6012.9277 |
18 | 7 | 101414179 | Data 2 | 5981.9834 |
19
20 +-----+-----+-----+-----+
21 | 10 rows | 7.53 seconds runtime |
22
23 (1) This result is stored as PySpark data frame _sqldf and in the IPython output cache as Out[142]. Learn more
24 Command took 7.53 seconds -- by r.velilakkadaparambukarthikkeyan@edu.salford.ac.uk at 4/20/2024, 4:29:58 PM on BOTT_TASK_2_20_APR_1
```

SQL queries on prediction of game names with word ‘wars’ (same queries are also run on python in the codes)

```
Cmd 128
1 %sql
2 select ID,game_ID, game_Name, prediction from my_purchase_play_Predictions where game_Name like '%Wars%' order by prediction desc limit 10
3
4 (5) Spark Jobs
5 +_sqldf: pyspark.sql.DataFrame = [ID:integer, game_ID:integer ... 2 more fields]
6
7 Table v +
8
9 +-----+-----+-----+-----+
10 | ID | game_ID | game_Name | prediction |
11 +-----+-----+-----+-----+
12 | 1 | 51822361 | Guild Wars | 165.87782 |
13 | 2 | 176223416 | Star Wars Knights of the Old Republic | 162.67662 |
14 | 3 | 26762388 | Star Wars Knights of the Old Republic | 55.265408 |
15 | 4 | 101656933 | Star Wars Knights of the Old Republic | 47.519855 |
16 | 5 | 101656933 | Star Wars Knights of the Old Republic | 47.519855 |
17 | 6 | 34177747 | Star Wars Knights of the Old Republic | 38.568573 |
18 | 7 | 86168617 | Star Wars Knights of the Old Republic | 35.940487 |
19
20 +-----+-----+-----+-----+
21 | 10 rows | 6.30 seconds runtime |
22
23 (1) This result is stored as PySpark data frame _sqldf and in the IPython output cache as Out[143]. Learn more
24 Command took 6.30 seconds -- by r.velilakkadaparambukarthikkeyan@edu.salford.ac.uk at 4/20/2024, 4:21:44 PM on BOTT_TASK_2_20_APR_1
```

SQL queries on prediction for game IDs (18, 24)

Cmd 121

```

1   %%sql
2   select ID, game_ID, game_Name, prediction from my_purchase_play_Predictions where game_ID In ( 18, 24) order by prediction desc limit 10

```

(5) Spark Jobs

`_sqldf: pyspark.sql.dataframe.DataFrame = [ID:integer, game_ID:integer ... 2 more fields]`

Table +

ID	game_ID	game_Name	prediction
1	71704418	Call of Duty Modern Warfare 2 - Multiplayer	1193.99
2	55763300	Call of Duty Modern Warfare 2 - Multiplayer	1137.974
3	21861124	Call of Duty Modern Warfare 2 - Multiplayer	967.4204
4	53621752	Call of Duty Modern Warfare 2 - Multiplayer	940.9795
5	58724834	Call of Duty Modern Warfare 2 - Multiplayer	824.99365
6	24204083	Call of Duty Modern Warfare 2 - Multiplayer	739.83997
7	112849197	Call of Duty Modern Warfare 2 - Multiplayer	732.67883

↓ 10 rows | 5.95 seconds runtime

Refreshed now

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[144]`. Learn more

Command took 5.95 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 4:22:35 PM on BDIT_TASK_2_28_APR_1

Function giving recommendation filtered by game name

Cmd 125

```

1   #Function to get recommendation filtered by game name
2   def recommendationsByGamePurchasePlayName(name):
3       filter = '%' + name + '%'
4       recommendationsList = ordered_purchase_play_predictions.orderBy("prediction", ascending=False).filter(ordered_purchase_play_predictions.game_Name.like(filter)).limit(10).
show(truncate=False)
5       return recommendationsList

```

Command took 0.00 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 4:25:23 PM on BDIT_TASK_2_28_APR_1

Cmd 125

```

1   recommendationsByGamePurchasePlayName('Duty')

```

(5) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
[71704418]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[1193.99]
[55763300]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[1137.974]
[104332808]	[purchase]	1.0	[Call of Duty Black Ops - Multiplayer]	[2308]	[1018.9942]
[21861124]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[967.4204]
[53621752]	[purchase]	1.0	[Call of Duty Modern Warfare 3 - Multiplayer]	[2029]	[952.3881]
[58621752]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[940.9795]
[58724834]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[824.99365]
[24204083]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[739.83997]
[112849197]	[play]	[295.0]	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[732.67883]
[112849197]	[purchase]	1.0	[Call of Duty Modern Warfare 2 - Multiplayer]	[18]	[732.67883]

Command took 6.73 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 4:25:31 PM on BDIT_TASK_2_28_APR_1

Function giving recommendation filtered by game ID

Cmd 126

```

1   #Function to get recommendation filtered by game ID
2   def recommendationsByGamePurchasePlayID(game_IDs):
3       # Filter ordered predictions based on provided game_IDs
4       filtered_purchase_play_predictions = ordered_purchase_play_predictions.filter(ordered_purchase_play_predictions.game_ID.isin(game_IDs))
5
6       # Order by prediction in descending order and limit to top 10 results
7       top_recommendations_purchase_play = filtered_purchase_play_predictions.orderBy("prediction", ascending=False).limit(10)
8
9       # Show the top recommendations without truncation
10      top_recommendations_purchase_play.show(truncate=False)
11
12      # Return the DataFrame containing the top recommendations
13      return top_recommendations_purchase_play

```

Command took 0.00 seconds -- by r.vellakkadaparambukarthikeyan@salford.ac.uk at 4/20/2024, 4:26:54 PM on BDIT_TASK_2_28_APR_1

Function giving recommendation filtered by user ID

```
Cmd 128
```

```
1 #Function to get recommendation filtered by user ID
2 def recommendationsByUserPurchasePlay(ID):
3     # Filter ordered predictions based on provided user ID
4     user_recommendations_purchase_play = ordered_purchase_play_predictions.filter(ordered_purchase_play_predictions.ID == ID)
5
6     # Order the filtered results by prediction in descending order
7     ordered_user_recommendations_purchase_play = user_recommendations_purchase_play.orderBy("prediction", ascending=False)
8
9     # Show all the recommendations for the user ID without truncation
10    ordered_user_recommendations_purchase_play.show(truncate=False)
11
12    # Return the DataFrame containing the recommendations for the user ID
13    return ordered_user_recommendations_purchase_play

Command took 0.11 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:28:20 PM on BDT_TASK_2_20_APR_2024
```

```
Cmd 129
```

```
1 recommendationsByUserPurchasePlayID(43908860)

▶ (3) Spark Jobs
```

ID	mem_Type	value	game_Name	game_ID	prediction
43908860	purchase	1.0	[Portal 2	1918	[9.263415
43908860	purchase	1.0	[Half-life 2 Episode Two	3669	[3.6330867
43908860	play	6.0	[Adventure Capitalist	4001	[1.1660283
43908860	play	155.0	[Call of Duty Modern Warfare 2 - Multiplayer	18	[0.997113
43908860	play	22.0	[Call of Duty Modern Warfare 3	1636	[-3.471014

```
DataFrame[|ID: int, mem_Type: string, value: double, game_Name: string, game_ID: int, prediction: float]

Command took 7.13 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:28:00 PM on BDT_TASK_2_20_APR_2024
```

Step 6: Generate top 10 recommendations for each user. Display the first 5 recommendations for each user

Top 10 recommendations for users

```

Code 131
1 userRecs_purchase_play.show(truncate=False)
2
3 (2) Spark Jobs
4
5 +-----+
6 |ID      |recommendations
7 +-----+
8
9 [76767, [[{1586, 6225.5273}, {52, 5013.1094}, {226, 4566.9277}, {2203, 3625.5723}, {1390, 2522.3167}, {312, 2481.7205}, {356, 2222.3284}, {3842, 2191.7627}, {1224, 2144.9688}, {2750, 2134.2842}], {144736, [[{4125, 844.43835}, {352, 696.23944}, {2203, 464.66013}, {947, 408.0185}, {3773, 402.23822}, {1586, 338.92426}, {233, 283.1874}, {1118, 210.53702}, {4996, 202.04451}, {1530, 201.69913}], {229911, [[{947, 2936.00873}, {1113, 2716.5212}, {3302, 2280.1013}, {185, 1699.7037}, {1118, 1488.9478}, {3437, 1454.61}, {2085, 1293.1644}, {4792, 1238.7642}, {3177, 1224.5564}, {4096, 1133.20811}], {835015, [[{3773, 703.8117}, {352, 688.76685}, {4125, 671.0984}, {2203, 637.8292}, {947, 461.16022}, {233, 414.97903}, {1586, 330.3361}, {1118, 295.18806}, {1530, 259.33594}, {3437, 255.42953}], {948168, [[{4125, 510.78476}, {352, 497.71725}, {947, 495.06613}, {856, 315.11853}, {467, 222.78027}, {1118, 215.57735}, {1586, 214.53067}, {1530, 206.06044}, {3773, 172.83717}, {3177, 142.66028}], {975449, [[{4125, 158.78618}, {2203, 109.891365}, {1224, 79.81523}, {1586, 77.58397}, {1241, 73.47321}, {2804, 57.50179}, {1530, 57.28557}, {352, 54.95613}, {3103, 51.579777}, {233, 47.39927}], {126879, [[{352, 759.405}, {4125, 541.977}, {2203, 486.5319}, {1586, 462.2525}, {947, 335.5011}, {233, 325.59433}, {3773, 310.3086}, {1118, 246.85162}, {2830, 168.94197}, {1291, 157.60779}], {2531540, [[{1224, 1933.53}, {2203, 1674.6844}, {1586, 959.87681}, {1780, 667.76891}, {3602, 621.76497}, {2830, 618.2033}, {1241, 610.1472}, {1897, 556.0581}, {2540, 529.6347}, {233, 501.60793}]]]
1 Command took 27.61 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:29:58 PM on BDTT_TASK_2_20_APR_1

```

Recommendations for specific user

```

Code 132
1 from pyspark.sql.functions import explode, col
2
3 # Function to get game recommendations for a specific user ID and 50+ with game details
4 def gameRecommendationsForUserPurchasePlay(userID):
5     # Filter the recommendations for the specific user ID from userRecs
6     user_recrecommendationsPurchasePlay = userRecs.purchase_play.filter(col("ID") == userID)
7
8     # Exclude the recommendations column to work with each recommendation separately
9     user_recrecommendationsPurchasePlay_exploded = user_recrecommendationsPurchasePlay.select("ID", explode("recommendations").alias("recommendation"))
10
11     # Join the exploded recommendations with the distinct game database
12     # Use the game_ID from the recommendations and the distinct_game_ID database to join
13     recommended_gamesPurchasePlay = user_recrecommendationsPurchasePlay_exploded.join(distinct_games_df,
14         user_recrecommendationsPurchasePlay_exploded["recommendation.game_id"] == distinct_games_df["game_ID"])
15
16     .select("game_ID", "game_name", "recommendation.rating")
17     .orderBy("recommendation.rating").desc()
18
19     # Display the recommended games with their ratings (predictions)
20     recommended_gamesPurchasePlay.show(truncate=False)
21
22
23     return recommended_gamesPurchasePlay
24
25     # Call the function for a specific user ID (e.g., user ID = 1)
26     gameRecommendationsForUserPurchasePlay(109347)
27
28 (4) Spark Jobs
29
30 +-----+
31 |game_ID|game_Name          |rating   |
32 +-----+
33 |1586   |Out of the Park Baseball 16|4225.5273|
34 |1586   |[ArmA 3]                |1013.1094  |
35 |2203   |Football Manager 2013  |1013.1094  |
36 |2203   |Football Manager 2012  |1013.1094  |
37 |2203   |Football Manager 2011  |1013.1094  |
38 |2203   |Football Manager 2010  |1013.1094  |
39 |2203   |Football Manager 2009  |1013.1094  |
40 |2203   |Football Manager 2008  |1013.1094  |
41 |2203   |Football Manager 2007  |1013.1094  |
42 |2203   |Football Manager 2006  |1013.1094  |
43 |2203   |Football Manager 2005  |1013.1094  |
44 |2203   |Football Manager 2004  |1013.1094  |
45 |2203   |Football Manager 2003  |1013.1094  |
46 |2203   |Football Manager 2002  |1013.1094  |
47 |2203   |Football Manager 2001  |1013.1094  |
48 |2203   |Football Manager 2000  |1013.1094  |
49 |2203   |Football Manager 1999  |1013.1094  |
50 |2203   |Football Manager 1998  |1013.1094  |
51 |2203   |Football Manager 1997  |1013.1094  |
52 |2203   |Football Manager 1996  |1013.1094  |
53 |2203   |Football Manager 1995  |1013.1094  |
54 |2203   |Football Manager 1994  |1013.1094  |
55 |2203   |Football Manager 1993  |1013.1094  |
56 |2203   |Football Manager 1992  |1013.1094  |
57 |2203   |Football Manager 1991  |1013.1094  |
58 |2203   |Football Manager 1990  |1013.1094  |
59 |2203   |Football Manager 1989  |1013.1094  |
60 |2203   |Football Manager 1988  |1013.1094  |
61 |2203   |Football Manager 1987  |1013.1094  |
62 |2203   |Football Manager 1986  |1013.1094  |
63 |2203   |Football Manager 1985  |1013.1094  |
64 |2203   |Football Manager 1984  |1013.1094  |
65 |2203   |Football Manager 1983  |1013.1094  |
66 |2203   |Football Manager 1982  |1013.1094  |
67 |2203   |Football Manager 1981  |1013.1094  |
68 |2203   |Football Manager 1980  |1013.1094  |
69 |2203   |Football Manager 1979  |1013.1094  |
70 |2203   |Football Manager 1978  |1013.1094  |
71 |2203   |Football Manager 1977  |1013.1094  |
72 |2203   |Football Manager 1976  |1013.1094  |
73 |2203   |Football Manager 1975  |1013.1094  |
74 |2203   |Football Manager 1974  |1013.1094  |
75 |2203   |Football Manager 1973  |1013.1094  |
76 |2203   |Football Manager 1972  |1013.1094  |
77 |2203   |Football Manager 1971  |1013.1094  |
78 |2203   |Football Manager 1970  |1013.1094  |
79 |2203   |Football Manager 1969  |1013.1094  |
80 |2203   |Football Manager 1968  |1013.1094  |
81 |2203   |Football Manager 1967  |1013.1094  |
82 |2203   |Football Manager 1966  |1013.1094  |
83 |2203   |Football Manager 1965  |1013.1094  |
84 |2203   |Football Manager 1964  |1013.1094  |
85 |2203   |Football Manager 1963  |1013.1094  |
86 |2203   |Football Manager 1962  |1013.1094  |
87 |2203   |Football Manager 1961  |1013.1094  |
88 |2203   |Football Manager 1960  |1013.1094  |
89 |2203   |Football Manager 1959  |1013.1094  |
90 |2203   |Football Manager 1958  |1013.1094  |
91 |2203   |Football Manager 1957  |1013.1094  |
92 |2203   |Football Manager 1956  |1013.1094  |
93 |2203   |Football Manager 1955  |1013.1094  |
94 |2203   |Football Manager 1954  |1013.1094  |
95 |2203   |Football Manager 1953  |1013.1094  |
96 |2203   |Football Manager 1952  |1013.1094  |
97 |2203   |Football Manager 1951  |1013.1094  |
98 |2203   |Football Manager 1950  |1013.1094  |
99 |2203   |Football Manager 1949  |1013.1094  |
100 |2203   |Football Manager 1948  |1013.1094  |
101 |2203   |Football Manager 1947  |1013.1094  |
102 |2203   |Football Manager 1946  |1013.1094  |
103 |2203   |Football Manager 1945  |1013.1094  |
104 |2203   |Football Manager 1944  |1013.1094  |
105 |2203   |Football Manager 1943  |1013.1094  |
106 |2203   |Football Manager 1942  |1013.1094  |
107 |2203   |Football Manager 1941  |1013.1094  |
108 |2203   |Football Manager 1940  |1013.1094  |
109 |2203   |Football Manager 1939  |1013.1094  |
110 |2203   |Football Manager 1938  |1013.1094  |
111 |2203   |Football Manager 1937  |1013.1094  |
112 |2203   |Football Manager 1936  |1013.1094  |
113 |2203   |Football Manager 1935  |1013.1094  |
114 |2203   |Football Manager 1934  |1013.1094  |
115 |2203   |Football Manager 1933  |1013.1094  |
116 |2203   |Football Manager 1932  |1013.1094  |
117 |2203   |Football Manager 1931  |1013.1094  |
118 |2203   |Football Manager 1930  |1013.1094  |
119 |2203   |Football Manager 1929  |1013.1094  |
120 |2203   |Football Manager 1928  |1013.1094  |
121 |2203   |Football Manager 1927  |1013.1094  |
122 |2203   |Football Manager 1926  |1013.1094  |
123 |2203   |Football Manager 1925  |1013.1094  |
124 |2203   |Football Manager 1924  |1013.1094  |
125 |2203   |Football Manager 1923  |1013.1094  |
126 |2203   |Football Manager 1922  |1013.1094  |
127 |2203   |Football Manager 1921  |1013.1094  |
128 |2203   |Football Manager 1920  |1013.1094  |
129 |2203   |Football Manager 1919  |1013.1094  |
130 |2203   |Football Manager 1918  |1013.1094  |
131 |2203   |Football Manager 1917  |1013.1094  |
132 |2203   |Football Manager 1916  |1013.1094  |
133 |2203   |Football Manager 1915  |1013.1094  |
134 |2203   |Football Manager 1914  |1013.1094  |
135 |2203   |Football Manager 1913  |1013.1094  |
136 |2203   |Football Manager 1912  |1013.1094  |
137 |2203   |Football Manager 1911  |1013.1094  |
138 |2203   |Football Manager 1910  |1013.1094  |
139 |2203   |Football Manager 1909  |1013.1094  |
140 |2203   |Football Manager 1908  |1013.1094  |
141 |2203   |Football Manager 1907  |1013.1094  |
142 |2203   |Football Manager 1906  |1013.1094  |
143 |2203   |Football Manager 1905  |1013.1094  |
144 |2203   |Football Manager 1904  |1013.1094  |
145 |2203   |Football Manager 1903  |1013.1094  |
146 |2203   |Football Manager 1902  |1013.1094  |
147 |2203   |Football Manager 1901  |1013.1094  |
148 |2203   |Football Manager 1900  |1013.1094  |
149 |2203   |Football Manager 1999  |1013.1094  |
150 |2203   |Football Manager 1998  |1013.1094  |
151 |2203   |Football Manager 1997  |1013.1094  |
152 |2203   |Football Manager 1996  |1013.1094  |
153 |2203   |Football Manager 1995  |1013.1094  |
154 |2203   |Football Manager 1994  |1013.1094  |
155 |2203   |Football Manager 1993  |1013.1094  |
156 |2203   |Football Manager 1992  |1013.1094  |
157 |2203   |Football Manager 1991  |1013.1094  |
158 |2203   |Football Manager 1990  |1013.1094  |
159 |2203   |Football Manager 1989  |1013.1094  |
160 |2203   |Football Manager 1988  |1013.1094  |
161 |2203   |Football Manager 1987  |1013.1094  |
162 |2203   |Football Manager 1986  |1013.1094  |
163 |2203   |Football Manager 1985  |1013.1094  |
164 |2203   |Football Manager 1984  |1013.1094  |
165 |2203   |Football Manager 1983  |1013.1094  |
166 |2203   |Football Manager 1982  |1013.1094  |
167 |2203   |Football Manager 1981  |1013.1094  |
168 |2203   |Football Manager 1980  |1013.1094  |
169 |2203   |Football Manager 1979  |1013.1094  |
170 |2203   |Football Manager 1978  |1013.1094  |
171 |2203   |Football Manager 1977  |1013.1094  |
172 |2203   |Football Manager 1976  |1013.1094  |
173 |2203   |Football Manager 1975  |1013.1094  |
174 |2203   |Football Manager 1974  |1013.1094  |
175 |2203   |Football Manager 1973  |1013.1094  |
176 |2203   |Football Manager 1972  |1013.1094  |
177 |2203   |Football Manager 1971  |1013.1094  |
178 |2203   |Football Manager 1970  |1013.1094  |
179 |2203   |Football Manager 1969  |1013.1094  |
180 |2203   |Football Manager 1968  |1013.1094  |
181 |2203   |Football Manager 1967  |1013.1094  |
182 |2203   |Football Manager 1966  |1013.1094  |
183 |2203   |Football Manager 1965  |1013.1094  |
184 |2203   |Football Manager 1964  |1013.1094  |
185 |2203   |Football Manager 1963  |1013.1094  |
186 |2203   |Football Manager 1962  |1013.1094  |
187 |2203   |Football Manager 1961  |1013.1094  |
188 |2203   |Football Manager 1960  |1013.1094  |
189 |2203   |Football Manager 1959  |1013.1094  |
190 |2203   |Football Manager 1958  |1013.1094  |
191 |2203   |Football Manager 1957  |1013.1094  |
192 |2203   |Football Manager 1956  |1013.1094  |
193 |2203   |Football Manager 1955  |1013.1094  |
194 |2203   |Football Manager 1954  |1013.1094  |
195 |2203   |Football Manager 1953  |1013.1094  |
196 |2203   |Football Manager 1952  |1013.1094  |
197 |2203   |Football Manager 1951  |1013.1094  |
198 |2203   |Football Manager 1950  |1013.1094  |
199 |2203   |Football Manager 1949  |1013.1094  |
200 |2203   |Football Manager 1948  |1013.1094  |
201 |2203   |Football Manager 1947  |1013.1094  |
202 |2203   |Football Manager 1946  |1013.1094  |
203 |2203   |Football Manager 1945  |1013.1094  |
204 |2203   |Football Manager 1944  |1013.1094  |
205 |2203   |Football Manager 1943  |1013.1094  |
206 |2203   |Football Manager 1942  |1013.1094  |
207 |2203   |Football Manager 1941  |1013.1094  |
208 |2203   |Football Manager 1940  |1013.1094  |
209 |2203   |Football Manager 1939  |1013.1094  |
210 |2203   |Football Manager 1938  |1013.1094  |
211 |2203   |Football Manager 1937  |1013.1094  |
212 |2203   |Football Manager 1936  |1013.1094  |
213 |2203   |Football Manager 1935  |1013.1094  |
214 |2203   |Football Manager 1934  |1013.1094  |
215 |2203   |Football Manager 1933  |1013.1094  |
216 |2203   |Football Manager 1932  |1013.1094  |
217 |2203   |Football Manager 1931  |1013.1094  |
218 |2203   |Football Manager 1930  |1013.1094  |
219 |2203   |Football Manager 1929  |1013.1094  |
220 |2203   |Football Manager 1928  |1013.1094  |
221 |2203   |Football Manager 1927  |1013.1094  |
222 |2203   |Football Manager 1926  |1013.1094  |
223 |2203   |Football Manager 1925  |1013.1094  |
224 |2203   |Football Manager 1924  |1013.1094  |
225 |2203   |Football Manager 1923  |1013.1094  |
226 |2203   |Football Manager 1922  |1013.1094  |
227 |2203   |Football Manager 1921  |1013.1094  |
228 |2203   |Football Manager 1920  |1013.1094  |
229 |2203   |Football Manager 1919  |1013.1094  |
230 |2203   |Football Manager 1918  |1013.1094  |
231 |2203   |Football Manager 1917  |1013.1094  |
232 |2203   |Football Manager 1916  |1013.1094  |
233 |2203   |Football Manager 1915  |1013.1094  |
234 |2203   |Football Manager 1914  |1013.1094  |
235 |2203   |Football Manager 1913  |1013.1094  |
236 |2203   |Football Manager 1912  |1013.1094  |
237 |2203   |Football Manager 1911  |1013.1094  |
238 |2203   |Football Manager 1910  |1013.1094  |
239 |2203   |Football Manager 1909  |1013.1094  |
240 |2203   |Football Manager 1908  |1013.1094  |
241 |2203   |Football Manager 1907  |1013.1094  |
242 |2203   |Football Manager 1906  |1013.1094  |
243 |2203   |Football Manager 1905  |1013.1094  |
244 |2203   |Football Manager 1904  |1013.1094  |
245 |2203   |Football Manager 1903  |1013.1094  |
246 |2203   |Football Manager 1902  |1013.1094  |
247 |2203   |Football Manager 1901  |1013.1094  |
248 |2203   |Football Manager 1900  |1013.1094  |
249 |2203   |Football Manager 1999  |1013.1094  |
250 |2203   |Football Manager 1998  |1013.1094  |
251 |2203   |Football Manager 1997  |1013.1094  |
252 |2203   |Football Manager 1996  |1013.1094  |
253 |2203   |Football Manager 1995  |1013.1094  |
254 |2203   |Football Manager 1994  |1013.1094  |
255 |2203   |Football Manager 1993  |1013.1094  |
256 |2203   |Football Manager 1992  |1013.1094  |
257 |2203   |Football Manager 1991  |1013.1094  |
258 |2203   |Football Manager 1990  |1013.1094  |
259 |2203   |Football Manager 1989  |1013.1094  |
260 |2203   |Football Manager 1988  |1013.1094  |
261 |2203   |Football Manager 1987  |1013.1094  |
262 |2203   |Football Manager 1986  |1013.1094  |
263 |2203   |Football Manager 1985  |1013.1094  |
264 |2203   |Football Manager 1984  |1013.1094  |
265 |2203   |Football Manager 1983  |1013.1094  |
266 |2203   |Football Manager 1982  |1013.1094  |
267 |2203   |Football Manager 1981  |1013.1094  |
268 |2203   |Football Manager 1980  |1013.1094  |
269 |2203   |Football Manager 1979  |1013.1094  |
270 |2203   |Football Manager 1978  |1013.1094  |
271 |2203   |Football Manager 1977  |1013.1094  |
272 |2203   |Football Manager 1976  |1013.1094  |
273 |2203   |Football Manager 1975  |1013.1094  |
274 |2203   |Football Manager 1974  |1013.1094  |
275 |2203   |Football Manager 1973  |1013.1094  |
276 |2203   |Football Manager 1972  |1013.1094  |
277 |2203   |Football Manager 1971  |1013.1094  |
278 |2203   |Football Manager 1970  |1013.1094  |
279 |2203   |Football Manager 1969  |1013.1094  |
280 |2203   |Football Manager 1968  |1013.1094  |
281 |2203   |Football Manager 1967  |1013.1094  |
282 |2203   |Football Manager 1966  |1013.1094  |
283 |2203   |Football Manager 1965  |1013.1094  |
284 |2203   |Football Manager 1964  |1013.1094  |
285 |2203   |Football Manager 1963  |1013.1094  |
286 |2203   |Football Manager 1962  |1013.1094  |
287 |2203   |Football Manager 1961  |1013.1094  |
288 |2203   |Football Manager 1960  |1013.1094  |
289 |2203   |Football Manager 1959  |1013.1094  |
290 |2203   |Football Manager 1958  |1013.1094  |
291 |2203   |Football Manager 1957  |1013.1094  |
292 |2203   |Football Manager 1956  |1013.1094  |
293 |2203   |Football Manager 1955  |1013.1094  |
294 |2203   |Football Manager 1954  |1013.1094  |
295 |2203   |Football Manager 1953  |1013.1094  |
296 |2203   |Football Manager 1952  |1013.1094  |
297 |2203   |Football Manager 1951  |1013.1094  |
298 |2203   |Football Manager 1950  |1013.1094  |
299 |2203   |Football Manager 1949  |1013.1094  |
300 |2203   |Football Manager 1948  |1013.1094  |
301 |2203   |Football Manager 1947  |1013.1094  |
302 |2203   |Football Manager 1946  |1013.1094  |
303 |2203   |Football Manager 1945  |1013.1094  |
304 |2203   |Football Manager 1944  |1013.1094  |
305 |2203   |Football Manager 1943  |1013.1094  |
306 |2203   |Football Manager 1942  |1013.1094  |
307 |2203   |Football Manager 1941  |1013.1094  |
308 |2203   |Football Manager 1940  |1013.1094  |
309 |2203   |Football Manager 1939  |1013.1094  |
310 |2203   |Football Manager 1938  |1013.1094  |
311 |2203   |Football Manager 1937  |1013.1094  |
312 |2203   |Football Manager 1936  |1013.1094  |
313 |2203   |Football Manager 1935  |1013.1094  |
314 |2203   |Football Manager 1934  |1013.1094  |
315 |2203   |Football Manager 1933  |1013.1094  |
316 |2203   |Football Manager 1932  |1013.1094  |
317 |2203   |Football Manager 1931  |1013.1094  |
318 |2203   |Football Manager 1930  |1013.1094  |
319 |2203   |Football Manager 1929  |1013.1094  |
320 |2203   |Football Manager 1928  |1013.1094  |
321 |2203   |Football Manager 1927  |1013.1094  |
322 |2203   |Football Manager 1926  |1013.1094  |
323 |2203   |Football Manager 1925  |1013.1094  |
324 |2203   |Football Manager 1924  |1013.1094  |
325 |2203   |Football Manager 1923  |1013.1094  |
326 |2203   |Football Manager 1922  |1013.1094  |
327 |2203   |Football Manager 1921  |1013.1094  |
328 |2203   |Football Manager 1920  |1013.1094  |
329 |2203   |Football Manager 1919  |1013.1094  |
330 |2203   |Football Manager 1918  |1013.1094  |
331 |2203   |Football Manager 1917  |1013.1094  |
332 |2203   |Football Manager 1916  |1013.1094  |
333 |2203   |Football Manager 1915  |1013.1094  |
334 |2203   |Football Manager 1914  |1013.1094  |
335 |2203   |Football Manager 1913  |1013.1094  |
336 |2203   |Football Manager 1912  |1013.1094  |
337 |2203   |Football Manager 1911  |1013.1094  |
338 |2203   |Football Manager 1910  |1013.1094  |
339 |2203   |Football Manager 1909  |1013.1094  |
340 |2203   |Football Manager 1908  |1013.1094  |
341 |2203   |Football Manager 1907  |1013
```

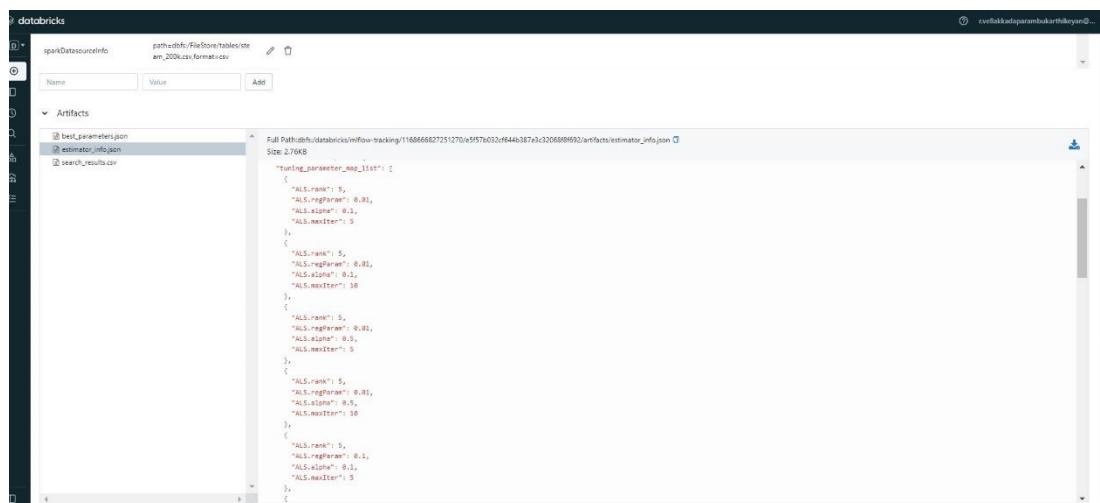
2.10 Selection of Hyperparameters in Recommender System based on User Behavior

The hyper parameter combination employed in the tuning process of all the user behavior based recommender systems are:

Table 2.5 Hyperparameters consideration in the recommender system

Parameter	Values	
Rank	5	10
RegParam	0.01	0.1
Alpha	0.1	0.5
MaxIter	5	10

Figure 2.4 shows the 'estimator_info.json' from the ML flow experiment UI.



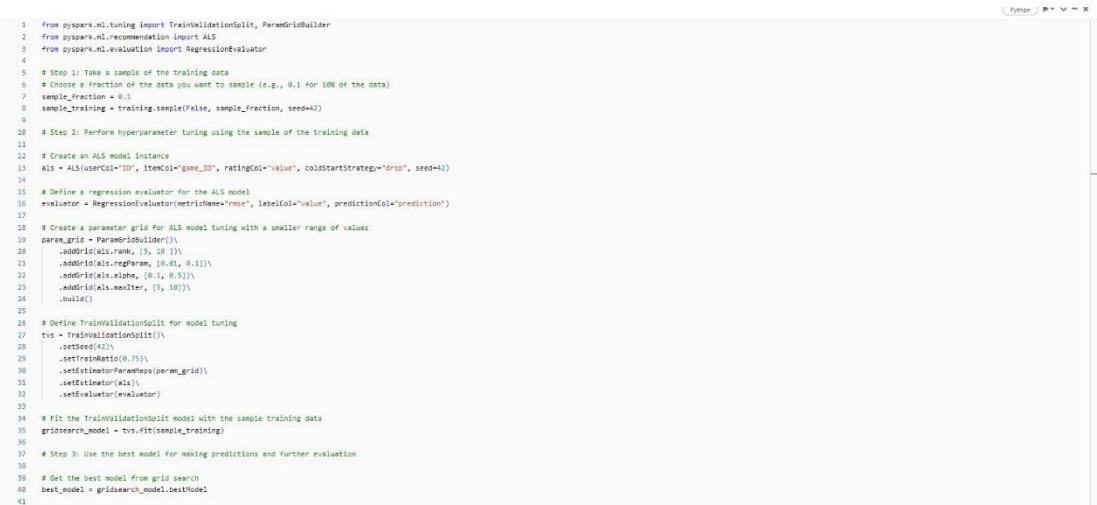
The screenshot shows the Databricks ML Flow experiment interface. On the left, there's a sidebar with 'Artifacts' expanded, showing three files: 'best_parameters.json', 'estimator_info.json', and 'search_results.csv'. The 'estimator_info.json' file is selected and its content is displayed in the main pane. The JSON file contains a single key 'tuning_parameter_map_list' which holds five entries, each representing a different parameter set for an ALS model. Each entry includes 'ALS.rank', 'ALS.regsParam', 'ALS.alpha', and 'ALS.maxIter' parameters.

```
Full Path: dbfs:/databricks/mlflow-tracking/1168666827251270/e557b032/f644b3e7a3c3206898992/artifacts/estimator_info.json
Size: 2.7KB
{
  "tuning_parameter_map_list": [
    {
      "ALS.rank": 5,
      "ALS.regsParam": 0.01,
      "ALS.alpha": 0.1,
      "ALS.maxIter": 5
    },
    {
      "ALS.rank": 5,
      "ALS.regsParam": 0.01,
      "ALS.alpha": 0.1,
      "ALS.maxIter": 10
    },
    {
      "ALS.rank": 5,
      "ALS.regsParam": 0.01,
      "ALS.alpha": 0.5,
      "ALS.maxIter": 5
    },
    {
      "ALS.rank": 5,
      "ALS.regsParam": 0.01,
      "ALS.alpha": 0.5,
      "ALS.maxIter": 10
    },
    {
      "ALS.rank": 5,
      "ALS.regsParam": 0.1,
      "ALS.alpha": 0.1,
      "ALS.maxIter": 5
    }
  ]
}
```

Figure 2.4 Hyperparameter tuning considerations in the recommender system design

2.10.1 Selection of Hyperparameters in Recommender System based on ‘Purchase’ User Behavior

Figure 2.5 shows the hyperparameter tuning with parameters described in Table 2.5. The similar codes for ‘Play’ and ‘Purchase’ and ‘Play’ based recommender systems are given in Figure 2.6 and 2.7 respectively. **Grid search** hyperparameter tuning is employed in this algorithm. To implement grid search in the recommender system using PySpark libraries, the **ALS** model from **pyspark.ml.recommendation** and the **ParamGridBuilder** from **pyspark.ml.tuning** are utilized for collaborative filtering and parameter grid creation respectively. The ranges of values for hyperparameters such as rank, regParam, alpha and maxIter are defined based on Table 2.5. The ALS model is combined with parameter grid in a **TrainValidationSplit** object, setting the evaluation metric. The model is fit to the training data by tuning object. The best hyperparameter configuration is found during the grid search, and is used for predictions and further evaluation. As the run ML flow runs where taking long, the training data is sampled for fast completion of runs.



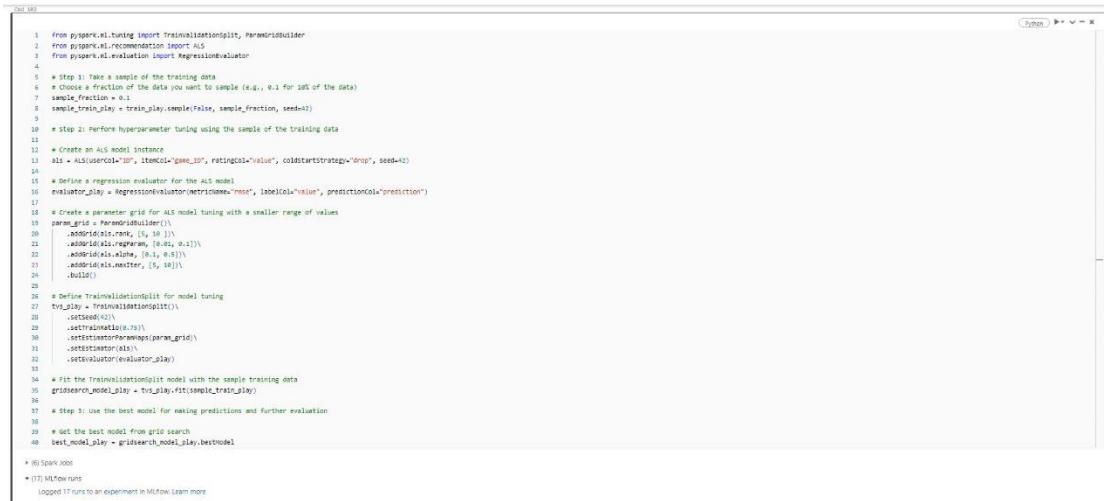
```

1  from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
2  from pyspark.ml.recommendation import ALS
3  from pyspark.ml.evaluation import RegressionEvaluator
4
5  # Step 1: Take a sample of the training data
6  # I choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
7  sample_fraction = 0.1
8  sample_training = training.sample(False, sample_fraction, seed=42)
9
10 # Step 2: Perform hyperparameter tuning using the sample of the training data
11
12 # Create an ALS model instance
13 als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)
14
15 # Define a regression evaluator for the ALS model
16 evaluator = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")
17
18 # Create a parameter grid for ALS model tuning with a smaller range of values
19 param_grid = ParamGridBuilder()
20 .addGrid(als.rank, [3, 10])
21 .addGrid(als.regParam, [0.01, 0.1])
22 .addGrid(als.alpha, [0.5, 0.9])
23 .addGrid(als.maxIter, [5, 10])
24 .build()
25
26 # Define TrainValidationSplit for model tuning
27 tvs = TrainValidationSplit()
28 .setSelected(42)
29 .setTestRatio(0.75)
30 .setEstimatorParamMaps(param_grid)
31 .setEstimator(als)
32 .setEvaluator(evaluator)
33
34 # Fit the TrainValidationSplit model with the sample training data
35 gridsearch_model = tvs.fit(sample_training)
36
37 # Step 3: Use the best model for making predictions and further evaluation
38
39 # Get the best model from grid search
40 best_model = gridsearch_model.bestModel
41
42

```

Figure 2.5 Hyperparameter tuning in recommender system based on ‘Purchase’ user behaviour.

2.10.2 Selection of Hyperparameters in Recommender System based on ‘Play’ User Behavior



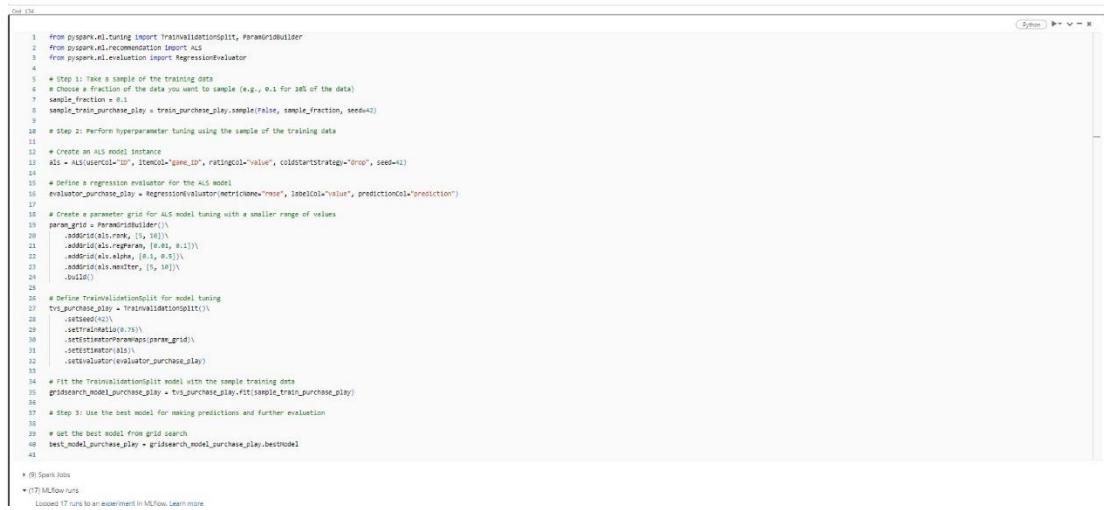
```

1 from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
2 from pyspark.ml.recommendation import ALS
3 from pyspark.ml.evaluation import RegressionEvaluator
4
5 # step 1: Take a sample of the training data
6 # Choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
7 sample_fraction = 0.1
8 sample_train_play = train_play.sample(False, sample_fraction, seed=42)
9
10 # step 2: Perform hyperparameter tuning using the sample of the training data
11
12 # Create an ALS model instance
13 als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)
14
15 # Define a regression evaluator for the ALS model
16 evaluator_play = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")
17
18 # Create a parameter grid for ALS model tuning with a smaller range of values
19 param_grid = ParamGridBuilder()
20 .addGrid(als.rank, [1, 10])
21 .addGrid(als.alpha, [0.01, 0.1])
22 .addGrid(als.n非iter, [5, 10])
23 .build()
24
25 # Define TrainValidationSplit for model tuning
26 tv_play = TrainValidationSplit()
27 .setEstimator(als)
28 .setStratified(True)
29 .setTrainRatio(0.75)
30 .setTuningParams(param_grid)
31 .setEvaluator(evaluator_play)
32
33 # Fit the TrainValidationSplit model with the sample training data
34 gridsearch_model_play = tv_play.fit(sample_train_play)
35
36 # Step 3: Use the best model for making predictions and further evaluation
37
38 # Get the best model from grid search
39 best_model_play = gridsearch_model_play.bestModel
40
41
42 # (0) Spark logs
43 - (17) MLflow runs
44 Logged 17 runs to an experiment in MLflow. Learn more

```

Figure 2.6 Hyperparameter tuning in recommender system based on ‘Play’ user behaviour.

2.10.3 Selection of Hyperparameters in Recommender System based on ‘Purchase’ and ‘Play’ User Behavior



```

1 from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
2 from pyspark.ml.recommendation import ALS
3 from pyspark.ml.evaluation import RegressionEvaluator
4
5 # step 1: Take a sample of the training data
6 # Choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
7 sample_fraction = 0.1
8 sample_train_purchase_play = train_purchase_play.sample(False, sample_fraction, seed=42)
9
10 # step 2: Perform hyperparameter tuning using the sample of the training data
11
12 # Create an ALS model instance
13 als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)
14
15 # Define a regression evaluator for the ALS model
16 evaluator_purchase_play = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")
17
18 # Create a parameter grid for ALS model tuning with a smaller range of values
19 param_grid = ParamGridBuilder()
20 .addGrid(als.rank, [1, 10])
21 .addGrid(als.alpha, [0.01, 0.1])
22 .addGrid(als.nNoniter, [5, 10])
23 .build()
24
25 # Define TrainValidationSplit for model tuning
26 tv_purchase_play = TrainValidationSplit()
27 .setEstimator(als)
28 .setStratified(True)
29 .setTrainRatio(0.75)
30 .setTuningParams(param_grid)
31 .setEvaluator(evaluator_purchase_play)
32
33 # Fit the TrainValidationSplit model with the sample training data
34 gridsearch_model_purchase_play = tv_purchase_play.fit(sample_train_purchase_play)
35
36 # Step 3: Use the best model for making predictions and further evaluation
37
38 # Get the best model from grid search
39 best_model_purchase_play = gridsearch_model_purchase_play.bestModel
40
41
42 # (0) Spark logs
43 - (17) MLflow runs
44 Logged 17 runs to an experiment in MLflow. Learn more

```

Figure 2.7 Hyperparameter tuning in recommender system based on ‘Purchase’ and ‘Play’ user behaviour

2.11 Model Training and Evaluation

Model training in the filtering recommender system involves using the training set to fit a model, such as the ALS model, to learn the patterns in user-item (game) interactions. Once the model is trained, it is evaluated on the testing set to assess its performance.

The two metrics used for evaluating recommender systems are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). RMSE measures the square root of the average of the squared differences between actual and predicted values, providing insight into the magnitude of prediction errors.

MAE calculates the average absolute difference between actual and predicted values, giving an indication of prediction accuracy. By evaluating the model's RMSE and MAE on the testing set, you can determine how well the model generalizes to new, unseen data and how accurately it predicts user preferences and behavior.

2.11.1 Model training and Evaluation of Recommender System based on '*Purchase*' Behavior

In the model training and evaluation phase of a recommender system implementation, the best model obtained from hyperparameter tuning is used to make predictions on both the training and testing data.

To assess the model's performance, predictions are generated using the best model on the full training data. The RMSE and MAE evaluation metrics is applied to the testing data to gauge the model's performance on test data.

Furthermore, the best model is used to generate sorted predictions, which can be valuable for producing top recommendations. Using the top recommendations, personalized recommendations most relevant to the users can be given.

Figure 2.8 shows the model training and evaluation metrics of ‘*purchase*’ based recommender system. Figures 2.9 and 2.10 shows the same for ‘*play*’ and ‘*purchase and play*’ based recommender system.

```
1 # Use the best model directly for making predictions on the full training data
2 # You do not need to refit the model on the full training data as the model is already trained
3
4 # Make predictions on the full training data
5 full_training_predictions = best_model.transform(training)
6
7 # Evaluate the model on the full training data
8 full_training_rmse = evaluator.evaluate(full_training_predictions)
9 print("Root Mean Squared Error (RMSE) on full training data: " + str(full_training_rmse))
10
11 # Change the metricName of the evaluator to 'mse' to evaluate Mean Absolute Error (MAE)
12 evaluator.setMetricName("mse")
13
14 # Evaluate the model on the full training data using MAE
15 full_training_mae = evaluator.evaluate(full_training_predictions)
16 print("Mean Absolute Error (MAE) on full training data: " + str(full_training_mae))
17
18 # If you want to use the model on other data for predictions, you can use 'best_model' directly
19

# (6) Spark Jobs
# full_training_predictions: pyspark.sql.DataFrame = [ID: integer, item:Type.string ... 4 more fields]
Root Mean Squared Error (RMSE) on full training data: 0.16303040217988086
Mean Absolute Error (MAE) on full training data: 0.11972841082340983
Command took 17.95 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 2:07:37 PM on DOTT_TASK_2_28_APR_1

Cmd 78
1 # Select best model and identify the parameters
2 bestModel = gridsearch_model.bestModel
3
4 print("Parameters for the best model:")
5 print("Rank Parameter: %d" % bestModel.rank)
6 print("RegParam Parameter: %g" % bestModel._java_obj.parent().getRegParam())
7 print("MaxIter Parameter: %d" % bestModel._java_obj.parent().getMaxIter())
8 print("ImplicitPrefs Parameter: %s" % bestModel._java_obj.parent().getImplicitPrefs())
9

Parameters for the best model:
Rank Parameter: 10
RegParam Parameter: 0.1
MaxIter Parameter: 10
ImplicitPrefs Parameter: False
Command took 0.84 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 2:14:47 PM on DOTT_TASK_2_28_APR_1

Cmd 74
1 # Make predictions on the test data
2 full_test_predictions = best_model.transform(test)
3
4 # Evaluate the model on the full test data
5 full_test_rmse = evaluator.evaluate(full_test_predictions)
6 print("Root Mean Squared Error (RMSE) on Full test data: " + str(full_test_rmse))
7
8 # Change the metricName of the evaluator to 'mse' to evaluate Mean Absolute Error (MAE)
9 evaluator.setMetricName("mse")
10
11 # Evaluate the model on the full test data using MAE
12 full_test_mae = evaluator.evaluate(full_test_predictions)
13 print("Mean Absolute Error (MAE) on Full test data: " + str(full_test_mae))
14
15 # If you want to use the model on other data for predictions, you can use 'best_model' directly

# (6) Spark Jobs
# full_test_predictions: pyspark.sql.DataFrame = [ID: integer, item:Type.string ... 4 more fields]
Root Mean Squared Error (RMSE) on full test data: 0.17675984442224833
Mean Absolute Error (MAE) on full test data: 0.12262598742586626
Command took 14.11 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 1:51:42 PM on DOTT_TASK_2_28_APR_1

Cmd 70
1 # Transform data with the best model
2 bestGeneratedPredictions = bestModel.transform(test)
3
4 # Drop rows with null value
5 bestGeneratedPredictions = bestGeneratedPredictions.dropna()
6
7 # Order predictions in descending order based on the "prediction" column
8 sorted_predictions = bestGeneratedPredictions.orderBy("prediction", ascending=False)
9
10 # Display the top 10 sorted predictions
11 sorted_predictions.show(10, truncate=False)
12
13

# (6) Spark Jobs
# bestGeneratedPredictions: pyspark.sql.DataFrame = [ID: integer, item:Type.string ... 4 more fields]
# sorted_predictions: pyspark.sql.DataFrame = [ID: integer, item:Type.string ... 4 more fields]
+---+|ID |item_type|value|gate_name |game_ID|prediction|+---+
|15084493 |purchase|1.0 |[GtaOnline] |1620 |0.8094594 ||
|65290208 |purchase|1.0 |[Grand Theft Auto V] |2140 |0.80359 ||
|65274785 |purchase|1.0 |[Grand Theft Auto V] |2047 |0.8035944 ||
|12321493 |purchase|1.0 |[Grand Theft Auto V] |2047 |0.8035944 ||
|45854483 |purchase|1.0 |[Grand Theft Auto Vice City] |3416 |0.8037551 ||
|39727485 |purchase|1.0 |[Portal] |4304 |0.80371665 ||
|20000000 |purchase|1.0 |[Grand Theft Auto V] |2047 |0.80359327 ||
|44884718 |purchase|1.0 |[Madden NFL 18] |3407 |0.80359467 ||
|10306302 |purchase|1.0 |[Day of Defeat] |3397 |0.80359282 ||
|39274480 |purchase|1.0 |[No More Room In Hell] |2165 |0.80359315 ||
+---+
only showing top 10 rows
Command took 18.12 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/28/2024, 2:08:00 PM on DOTT_TASK_2_28_APR_1
```

Figure 2.8 Model training and evaluation metrics of ‘purchase’ based recommender system

2.11.2 Model training and Evaluation of Recommender System based on ‘Play’ Behavior

```
Cmd 183
1 # Use the best model directly for making predictions on the full training data
2 # You do not need to refit the model on the full training data as the model is already trained
3
4 # Make predictions on the full training data
5 full_training_predictions_play = best_model_play.transform(train_play)
6
7 # Evaluate the model on the full training data
8 full_training_rmse_play = evaluator_play.evaluate(full_training_predictions_play)
9 print("Root Mean Squared Error (RMSE) on full training data: (full_training_rmse_play)")
10
11 # Change the metricName of the evaluator to 'mae' to evaluate Mean Absolute Error (MAE)
12 evaluator_play.setMetricName("mae")
13
14 # Evaluate the model on the full training data using MAE
15 full_training_mae_play = evaluator_play.evaluate(full_training_predictions_play)
16 print("Mean Absolute Error (MAE) on full training data: (full_training_mae_play)")
17
18 # If you want to use the model on other data for predictions, you can use 'best_model' directly
19

*(11) Spark Jobs
* full_training_predictions_play: pyspark.sql.DataFrame:Dataframe = [ID: integer, mem_type: string ... 4 more fields]
Root Mean Squared Error (RMSE) on full training data: 185.15931204856726
Mean Absolute Error (MAE) on full training data: 45.70201187313386
Command took 15.85 seconds -- by r.velakkadaparambukarthkeyan@edu.salford.ac.uk at 4/28/2024, 3:43:56 PM on BDTT_TASK_2_28_APR_1

Cmd 184
1 # Select best model and identify the parameters
2
3 print("Parameters for the best model:")
4 print("Rank Parameter: %s" % best_model_play.rank)
5 print("RegParam Parameter: %s" % best_model_play._java_obj.parent().getRegParam())
6 print("MaxIter Parameter: %s" % best_model_play._java_obj.parent().getMaxIter())
7 print("ImplicitPrefs Parameter: %s" % best_model_play._java_obj.parent().getImplicitPrefs())
Parameters for the best model:
Rank Parameter: 10
RegParam Parameter: 0.1
MaxIter Parameter: 10
ImplicitPrefs Parameter: False
Command took 0.12 seconds -- by r.velakkadaparambukarthkeyan@edu.salford.ac.uk at 4/28/2024, 3:47:11 PM on BDTT_TASK_2_28_APR_1

Cmd 185
1 # Transform the test data using the best model
2 transformed_test_play = best_model_play.transform(test_play)
3
4 # Ensure that the "prediction" column is of type "double"
5 transformed_test_play = transformed_test_play.withColumn(
6     "prediction",
7     transformed_test_play["prediction"].cast("double")
8 )
9
10 # Evaluate the model on the full test data
11 evaluator_play.setMetricName("rmse")
12 full_test_rmse_play = evaluator_play.evaluate(transformed_test_play)
13 print("Root Mean Squared Error (RMSE) on full test data: (full_test_rmse_play)")
14
15 # Change the metricName of the evaluator to 'mae' to evaluate Mean Absolute Error (MAE)
16 evaluator_play.setMetricName("mae")
17
18 # Evaluate the model on the full test data using MAE
19 full_test_mae_play = evaluator_play.evaluate(transformed_test_play)
20 print("Mean Absolute Error (MAE) on full test data: (full_test_mae_play)")

*(6) Spark Jobs
* transformed_test_play: pyspark.sql.DataFrame:Dataframe = [ID: integer, mem_type: string ... 4 more fields]
Root Mean Squared Error (RMSE) on test data: 224.7555653942484
Mean Absolute Error (MAE) on full test data: 54.8912263260849
Command took 15.18 seconds -- by r.velakkadaparambukarthkeyan@edu.salford.ac.uk at 4/28/2024, 3:57:00 PM on BDTT_TASK_2_28_APR_1

Cmd 186
1 # Transform data with the best model
2 bestGeneratedPlayPredictions = best_model_play.transform(test_play)
3
4 # Drop rows with null values
5 bestGeneratedPlayPredictions = bestGeneratedPlayPredictions.dropna()
6
7 # Order predictions in descending order based on the 'prediction' column
8 sorted_predictions_play = bestGeneratedPlayPredictions.orderBy("prediction", ascending=False)
9
10 # Display the top 10 sorted predictions
11 sorted_predictions_play.show(10, truncate=False)

*(5) Spark Jobs
* bestGeneratedPlayPredictions: pyspark.sql.DataFrame:Dataframe = [ID: integer, mem_type: string ... 4 more fields]
* sorted_predictions_play: pyspark.sql.DataFrame:Dataframe = [ID: integer, mem_type: string ... 4 more fields]
+---+|mem_type|value|game_name|game_ID|prediction|+---+
|14645159|play|10.6|Counter Strike|1890|1496.7659| |
|88189887|play|1470.0|Mount & Blade Warband|226|948.0586| |
|32126281|play|537.0|Mount & Blade Warband|226|915.0609| |
|49863389|play|1930.0|Counter Strike|3690|871.0834| |
|42657390|play|1953.0|Call of Duty: Warzone|223|855.2032| |
|11937119|play|11.0|Call of Duty Black Ops - Multiplayer|2280|855.2034| |
|61189151|play|13.4|Zombie Panic Source|154|749.6911| |
|99977985|play|122.0|[MyZ]|345|737.81915| |
|24712132|play|177.0|Ultra Street Fighter IV|3858|729.3984| |
|55426012|play|11.0|[Data Z]|1|712.63965| |
+---+
only showing top 10 rows

Command took 0.57 seconds -- by r.velakkadaparambukarthkeyan@edu.salford.ac.uk at 4/28/2024, 3:59:16 PM on BDTT_TASK_2_28_APR_1
```

Figure 2.9 Model training and evaluation metrics of ‘play’ based recommender system

2.11.3 Model training and Evaluation of Recommender System based on ‘Purchase’ and ‘Play’ Behavior

The figure consists of three vertically stacked screenshots of a Jupyter Notebook interface, each showing a code cell and its output.

Screenshot 1:

```

Cmd 135
1 # Use the best model directly for making predictions on the full training data
2 # Transform the training data using the best model
3 full_training_predictions_purchase_play = best_model_purchase_play.transform(train_purchase_play)
4
5 # Convert the prediction column to double type, if necessary
6 full_training_predictions_purchase_play = full_training_predictions_purchase_play.withColumn(
7     "prediction",
8     full_training_predictions_purchase_play["prediction"].cast("double")
9 )
10 # Evaluate the model on the full training data using RMSE
11 evaluator_purchase_play.setMetricName("rmse")
12 full_training_rmse_purchase_play = evaluator_purchase_play.evaluate(full_training_predictions_purchase_play)
13 print(f"Root Mean Squared Error (RMSE) on full training data: {full_training_rmse_purchase_play}")
14
15 # Change the metricName of the evaluator to "mae" to evaluate Mean Absolute Error (MAE)
16 evaluator_purchase_play.setMetricName("mae")
17
18 # Evaluate the model on the full training data using MAE
19 full_training_mae_purchase_play = evaluator_purchase_play.evaluate(full_training_predictions_purchase_play)
20 print(f"Mean Absolute Error (MAE) on full training data: {full_training_mae_purchase_play}")

```

(7) Spark Jobs

full_training_predictions.purchase_play: pyspark.sql.dataframe.DataFrame = [ID:integer, mem_type:string ... 4 more fields]

Root Mean Squared Error (RMSE) on full training data: 143.6439220157775
Mean Absolute Error (MAE) on full training data: 23.767940645524057

Command took 17.57 seconds -- by r.vellakkadaparambukarthikkeyan@edu.salford.ac.uk at 4/20/2024, 4:51:41 PM on DDT_TASK_2_20_APR_1

Screenshot 2:

```

Cmd 136
1 # Select best model and identify the parameters
2
3 print("Parameters for the best model:")
4 print("Rank Parameter: %s" % best_model_purchase_play.rank)
5 print("RegParam Parameter: %s" % best_model_purchase_play._java_obj.parent().getRegParam())
6 print("MaxIter Parameter: %s" % best_model_purchase_play._java_obj.parent().getMaxIter())
7 print("ImplicitPrefs Parameter: %s" % best_model_purchase_play._java_obj.parent().getImplicitPrefs())

```

Parameters for the best model:
Rank Parameter: 10
RegParam Parameter: 0.1
MaxIter Parameter: 10
ImplicitPrefs Parameter: False

Command took 0.07 seconds -- by r.vellakkadaparambukarthikkeyan@edu.salford.ac.uk at 4/20/2024, 4:55:04 PM on DDT_TASK_2_20_APR_1

Screenshot 3:

```

Cmd 137
1 # Perform predictions on the test data using the best model
2 test_predictions_purchase_play = best_model_purchase_play.transform(test_purchase_play)
3
4 # Convert the prediction column to double type, if necessary
5 test_predictions_purchase_play = test_predictions_purchase_play.withColumn(
6     "prediction",
7     test_predictions_purchase_play["prediction"].cast("double")
8 )
9
10 # Evaluate the model on the test data using RMSE
11 evaluator_purchase_play.setMetricName("rmse")
12 test_rmse_pp = evaluator_purchase_play.evaluate(test_predictions_purchase_play)
13 print(f"Root Mean Squared Error (RMSE) on test data: {test_rmse_pp}")
14
15 # Evaluate the model on the test data using MAE
16 evaluator_purchase_play.setMetricName("mae")
17 test_mae_pp = evaluator_purchase_play.evaluate(test_predictions_purchase_play)
18 print(f"Mean Absolute Error (MAE) on test data: {test_mae_pp}")
19

```

(10) Spark Jobs

test_predictions.purchase_play: pyspark.sql.dataframe.DataFrame = [ID:integer, mem_type:string ... 4 more fields]

Root Mean Squared Error (RMSE) on test data: 132.32095410496453
Mean Absolute Error (MAE) on test data: 25.7821727633066

Command took 17.00 seconds -- by r.vellakkadaparambukarthikkeyan@edu.salford.ac.uk at 4/20/2024, 5:00:32 PM on DDT_TASK_2_20_APR_1

```

1 # Transform data with the best model
2 bestGeneratedPurchasePlayPredictions = best_model_purchase_play.transform(test_purchase_play)
3
4 # Drop rows with null values
5 bestGeneratedPurchasePlayPredictions = bestGeneratedPurchasePlayPredictions.dropna()
6
7 # Order predictions in descending order based on the 'prediction' column
8 sorted_predictions_purchase_play = bestGeneratedPurchasePlayPredictions.orderBy("prediction", ascending=False)
9
10 # Display the top 10 sorted predictions
11 sorted_predictions_purchase_play.show(10, truncate=False)

(Spark Jobs)
+---+---+---+
|ID |item_Type|value|game_name |game_ID|prediction|
+---+---+---+
|11111304|purchase|1.0 |Dota 2 |1 |1966.9524 |
|50011344|purchase|1.0 |Total War ATTILA |14678 |2939.3514 |
|50011344|play |10.0 |Darks Souls Prepare to Die Edition|1320 |2862.1328 |
|18812088|purchase|1.0 |Dota 2 |1 |12755.2073 |
|17899545|purchase|1.0 |Counter-Strike |3698 |12525.9932 |
|16471130|purchase|1.0 |Dota 2 |1 |12755.2073 |
|500118751|purchase|1.0 |Team Fortress 2 |3094 |12652.2055 |
|19318921|purchase|1.0 |DayZ |345 |1154.9697 |
|500118751|play |158.0 |The Binding of Isaac: Rebirth |199 |1146.5511 |
|139037163|play |111.2 |The Elder Scrolls V: Skyrim |2610 |1115.3812 |
+---+---+---+
only showing top 10 rows

```

Command took: 8.28 seconds --> tp://vellakkadaparambukarthikeyan@keyen@192.168.1.10:8080 at 4/20/2014, 5:02:16 PM via BOTT_755x_2_20_MH_

Figure 2.10 Model training and evaluation metrics of ‘purchase’ and ‘play’ based recommender system

2.12 MLflow Experiment Tracking

Figures 2.11 to 2.13 shows the MLflow experiment tracking for the recommender system.

Run Name	Created	Dataset	Duration	Source	Models	Metrics	Parameters
luminous-rat-503	1 day ago	dataset (7bb3f270) Train	6.5min	□ BOTT_Ta...	-	-	value rmse prediction
likeable-sheep-342	1 day ago	dataset (3bae533b) Train	18.2s	□ BOTT_Ta...	-	-	-
loud-toon-939	1 day ago	dataset (5c1bd003) Train	6.2min	□ BOTT_Ta...	-	-	value rmse prediction
exultant-carp-283	1 day ago	dataset (5c1bd003) Train	13.8s	□ BOTT_Ta...	-	-	-
melodic-fox-308	1 day ago	dataset (5c1bd003) Train	14.1s	□ BOTT_Ta...	-	-	-
flawless-dolphin-875	1 day ago	dataset (5c1bd003) Train	14.1s	□ BOTT_Ta...	-	-	-
bold-cow-892	1 day ago	-	0.8s	□ BOTT_Ta...	-	-	-
treaky-thief-795	1 day ago	dataset (5c1bd003) Train	14.2s	□ BOTT_Ta...	-	-	-
clumsy-jay-741	1 day ago	dataset (5c1bd003) Train	18.2s	□ BOTT_Ta...	-	-	-
popular-donkey-684	1 day ago	dataset (9323a841) Train	16.0s	□ BOTT_Ta...	-	-	-
BOTT_Task_ML_P...	1 day ago	dataset (90489e4) Train	6.4min	□ BOTT_Ta...	-	-	value rmse prediction
rambunctious-rat-345	1 day ago	dataset (90489e4) Train	6.6min	□ BOTT_Ta...	-	-	value rmse prediction

101 matching runs

Figure 2.11 Experiment Overview (last run name enclosed in red rectangle box)

The screenshot shows the Databricks Experiment UI for a run titled "luminous-rat-SOS - Purchase_Mov_Combined". The top navigation bar includes "Experiments", "Runs", "Jobs", "Clusters", "Metrics", "Logs", and "Artifacts". The main area displays experiment parameters, artifacts, and metrics.

Parameters (14)

Name	Value
RegressorsLabeledTableCol	value
RegressorsLabeledTableName	move
RegressorsLabeledPredictorCol	predictor
RegressorsLabeledThroughOrigin	Value
item_ALS_alpha	0.1
item_ALS_maxIter	10
item_ALS_rank	10
item_ALS_stepSize	0.1
intercept	False
learner	ALS
evaluator	RegressionEvaluator
positive	1
seed	42
nonnegative	0.05

Artifacts

- best_parameters.json
- estimator_info.json
- search_results.csv

Tags (3)

Search Results

Full Path: dbfs:/databricks/milflow-tracking/116866682751270/e5f57b032cf644b387e3c32068f8f692/artifacts/search_results.csv

Size: 1.9KB

params	rmse	param_ALS.rank	param_ALS.regParam	param_ALS.alpha	param_ALS.maxIter
{"ALS.rank": 5, "ALS.regParam": ...}	122.9501820425502	5	0.1	0.5	5
{"ALS.rank": 5, "ALS.regParam": ...}	124.07424296110812	5	0.1	0.5	10
{"ALS.rank": 10, "ALS.regParam": ...}	128.85966205409054	10	0.01	0.1	5
{"ALS.rank": 10, "ALS.regParam": ...}	105.25306757833636	10	0.01	0.1	10
{"ALS.rank": 10, "ALS.regParam": ...}	128.85966295469054	10	0.01	0.5	5
{"ALS.rank": 10, "ALS.regParam": ...}	105.23306757833636	10	0.01	0.5	10
{"ALS.rank": 10, "ALS.regParam": ...}	89.354501064460946	10	0.1	0.1	5
{"ALS.rank": 10, "ALS.regParam": ...}	87.4739142740338	10	0.1	0.1	10
{"ALS.rank": 10, "ALS.regParam": ...}	88.35430366448941	10	0.1	0.5	5
{"ALS.rank": 10, "ALS.regParam": ...}	87.4750142740338	10	0.1	0.5	10

Figure 2.12 Experiment run details (hyperparameter and evaluation metrics)

The screenshot shows the Databricks UI for a run titled "luminous-rat-SOS - Purchase_Mov_Combined". The main area displays experiment parameters, artifacts, and metrics.

Artifacts

- best_parameters.json
- estimator_info.json
- search_results.csv

best_parameters.json

```
{
  "ALS.rank": 10,
  "ALS.regParam": 0.1,
  "ALS.alpha": 0.1,
  "ALS.maxIter": 10
}
```

Figure 2.13 Best hyperparameters

2.13 Recommender System based on ‘Purchase’ Behavior - Prediction Visualization

Figure 2.14 shows the predictions of the recommender system based on ‘Purchase’ behavior

1 display(sorted_predictions)

2 # (1) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
1	132014951	purchase	1 BLOCKADE 3D	1620	0.9094304
2	65398050	purchase	1 Gauntlet	2545	0.905059
3	3527485	purchase	1 Half-Life 2 Deathmatch	443	0.90458244
4	3527485	purchase	1 Ricochet	629	0.9040007
5	43684632	purchase	1 Grand Theft Auto Vice City	3416	0.9037553
6	3527485	purchase	1 Portal	4304	0.90373045
7	101030777	purchase	1 HAWKEN	4073	0.90366197

10,000 rows | Truncated data | 8.35 seconds runtime

Command took 8.35 seconds -- by r.vellakkalaparumalr@vayagedu.salford.ac.uk at 4/21/2024, 9:02:59 PM on 8077_1604_2_20_APR_2

Cell 74

```
1 sorted_predictions_pandas = sorted_predictions.toPandas()
2 print(sorted_predictions_pandas.head(10))
```

(1) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
0	132014951	purchase	1.0 BLOCKADE 3D	1620	0.909430
1	65398050	purchase	1.0 Gauntlet	2545	0.905050
2	3527485	purchase	1.0 Half-Life 2 Deathmatch	443	0.904582
3	3527485	purchase	1.0 Ricochet	629	0.904001
4	43684632	purchase	1.0 Grand Theft Auto Vice City	3416	0.903755
5	3527485	purchase	1.0 Portal	4304	0.903730
6	43684632	purchase	1.0 HAWKEN	4073	0.903662
7	44866735	purchase	1.0 HAWKEN	4073	0.903662
8	43684632	purchase	1.0 Day of Defeat	1997	0.903409
9	3527485	purchase	1.0 No More Room in Hell	2819	0.903381

Command took 8.44 seconds -- by r.vellakkalaparumalr@vayagedu.salford.ac.uk at 4/21/2024, 9:03:04 PM on 8077_1604_2_20_APR_2

Top 10 recommendations

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 # Convert to Pandas Dataframe for plotting
5 sorted_predictions_pandas = sorted_predictions.toPandas()
6
7 # Plotting top 10 predictions as a bar chart
8 sns.set(style="white")
9 sns.barplot(x="sorted_predictions_pandas['game_ID'].head(10)", y="sorted_predictions_pandas['prediction']")
10 plt.xlabel('Game ID')
11 plt.ylabel('Prediction')
12 plt.title('Top 10 Predictions')
13 plt.show()
```

Top 10 Predictions

ID	mem_Type	value	game_Name	game_ID	prediction
0	132014951	purchase	1 BLOCKADE 3D	1620	0.909430
1	65398050	purchase	1 Gauntlet	2545	0.905050
2	3527485	purchase	1 Half-Life 2 Deathmatch	443	0.904582
3	3527485	purchase	1 Ricochet	629	0.904001
4	43684632	purchase	1 Grand Theft Auto Vice City	3416	0.903755
5	3527485	purchase	1 Portal	4304	0.903730
6	101030777	purchase	1 HAWKEN	4073	0.903662
7	43684632	purchase	1 No More Room in Hell	2819	0.903662
8	43684632	purchase	1 Day of Defeat	1997	0.903409
9	3527485	purchase	1 No More Room in Hell	2819	0.903381
10	188051332	purchase	1 Counter-Strike: Global Offensive	1160	0.902971
11	188051332	purchase	1 Team Fortress 2	1161	0.902953
12	22207548	purchase	1 Fallout: New Vegas: Courier's Stash	2307	0.902877
13	2753625	purchase	1 Alien Swarm	3370	0.902866
14	188051332	purchase	1 Team Fortress	2220	0.902820
15	188051332	purchase	1 Red Alert	2611	0.902711
16	99374297	purchase	1 Warface	1805	0.902602
17	43684632	purchase	1 Grand Theft Auto IV	2752	0.902678
18	188051332	purchase	1 Team Fortress 2	2309	0.902673
19	122778021	purchase	1 Team Fortress 2	3894	0.902585
20	22207548	purchase	1 Arma 2: Operation Arrowhead	1199	0.902561
21	22207548	purchase	1 Sleeping Dogs	4236	0.902550
22	188051332	purchase	1 Left 4 Dead	3320	0.902550
23	91807668	purchase	1 Left 4 Dead 2	703	0.902543
24	7559825	purchase	1 Left 4 Dead 2	704	0.902489
25	246936243	purchase	1 Far Cry 2	1013	0.902488
26	246936243	purchase	1 Cricket Heroes	2350	0.902485
27	16645459	purchase	1 Resident Evil Revelations 2 / Biohazard Revelations 4	103	0.902449
				2227	0.902399

Figure 2.14 Predictions of the recommender system based on ‘Purchase’ behavior

2.14 Recommender System based on ‘Play’ Behavior - Prediction Visualization

Figure 2.15 shows the predictions of the recommender system based on 'Play' behavior

Top 10 recommendations



Color coded result visualization (removed from notebook)

remya Vellakkadapambu Karthikeyan, ML Python 5 ★

File Edit View Run Help Last edit: 3 minutes ago New cell U/OFF ▾

Run all BDIT_TASK_2_20 APR_2 Share Publish

```
1 # Convert sorted_predictions to Pandas DataFrame
2 sorted_predictions_of = sorted_predictions_play.toandas()
3
4 # Apply color coding to the table using style.background_gradient
5 styled_table = sorted_predictions_of.style.background_gradient(cmap="coolwarm")
6
7 # Displays the styled table
8 styled_table
9
```

Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction	
0	14465559	play	0.600000	Counter-Strike	5990	1496.703657
1	8018684	play	4.000000	Mount & Blade Warband	276	949.058638
2	1233631	play	253.000000	Mount & Blade Warband	276	949.058638
3	1062309	play	0.30.000000	Counter-Strike	260	871.093374
4	42657009	play	953.000000	Football Manager 2011	233	835.200000
5	11794760	play	8.900000	Call of Duty Black Ops - Multiplayer	2308	788.261841
6	81189155	play	1.400000	Zombie Panic Source	54	749.601074
7	96077905	play	27.000000	DayZ	315	737.000000
8	24721234	play	77.000000	Left 4 Dead	156	729.000000
9	5	play	1.000000	Left 4 Street Fighter IV	1	712.639648
10	26729643	play	4.800000	Zombie Panic Source	54	712.008545
11	106834087	play	1.800000	Counter-Strike Source	45	686.534607
12	87201181	play	118.000000	Call of Duty Black Ops - Multiplayer	2308	618.603098
13	35701646	play	37.000000	The Binding of Isaac	847	598.646411
14	80611940	play	0.000000	Skullgirls	2429	577.554510
15	1062309	play	57.000000	Call of Duty Black Ops - Multiplayer	2308	553.984283
16	14465559	play	71.000000	Arms 2 Operation Arrowhead	1199	520.675858
17	103824897	play	36.000000	Total War ROME II - Emperor Edition	3002	520.225708
18	79017735	play	34.000000	FINAL FANTASY XIV A Realm Reborn	3698	471.965424
19	175001491	play	176.000000	Garry's Mod	2670	468.770599
20	12707785	play	2.200000	Mount & Blade Warband	276	466.842499
21	54003031	play	0.200000	Counter-Strike	3698	450.607910

Figure 2.15 Predictions of the recommender system based on ‘*Play*’ behavior

2.15 Recommender System based on ‘Purchase’ and ‘Play’ Behavior -Prediction Visualization

Figure 2.16 shows the predictions of the recommender system based on ‘Purchase’ and ‘Play’ behavior

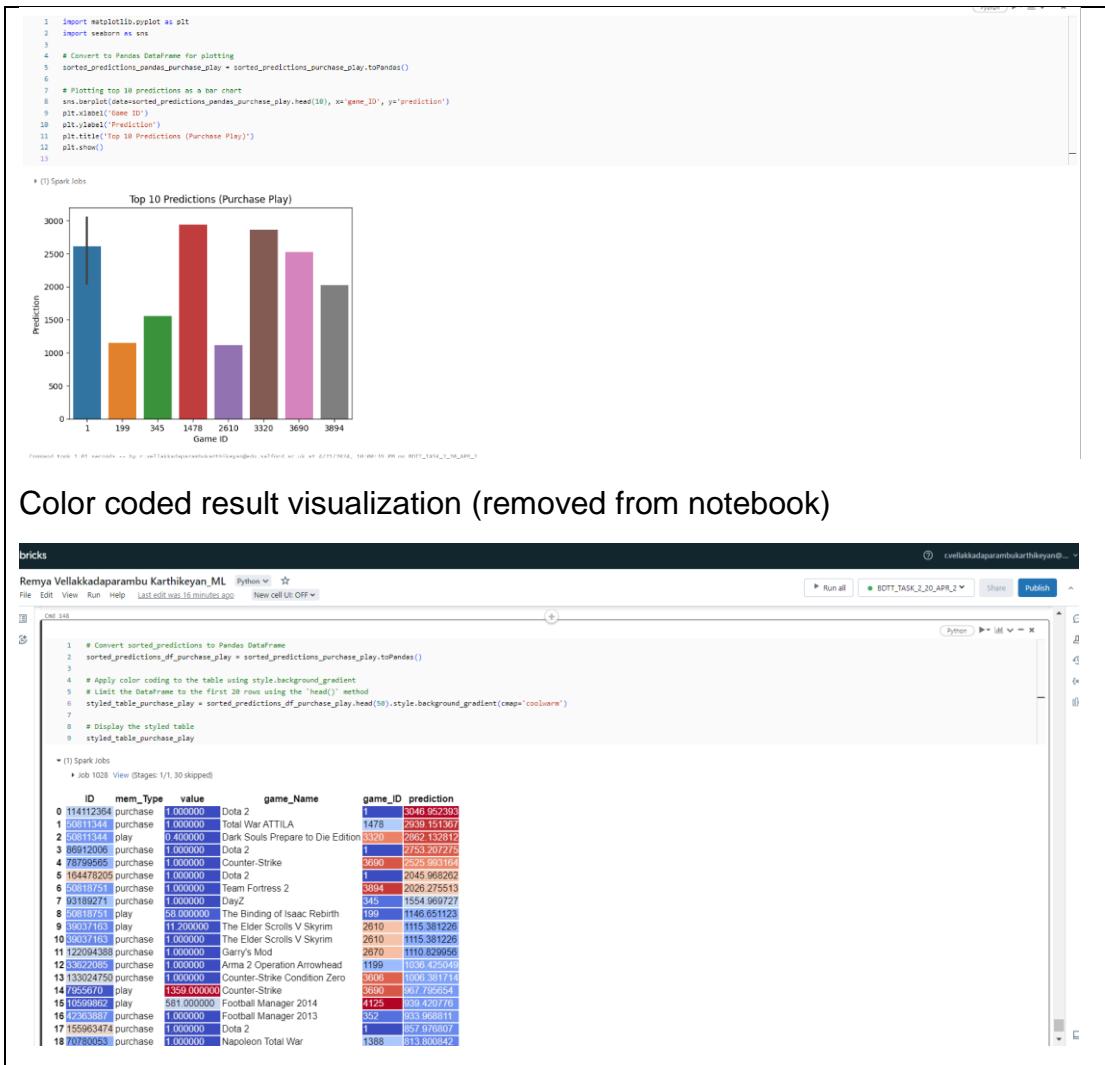


Figure 2.16 Predictions of the recommender system based on ‘Purchase’ and ‘Play’ behavior

2.16 Results and Discussion

The three recommender systems implemented are giving predictions and recommendations based on the rating column (value in the data set which depends on the user or member behavior –‘purchase’ or ‘play’). The detailed results are discussed in previous sections. The hyperparamater tuning based on grid search is performed and the best model is obtained for maxIter = 10, rank =10, alpha = 0.1, regParam = 0.1, ImplicitPrefs = False. The evaluation metrics (rounded to 3 decimal places) on full training and test data under the best model is given in Table 2.6.

Table 2.6 Comparison of evaluation metrics

Recommender system based on behavior	RMSE		MAE	
	Full training data	Full Test data	Full training data	Full Test data
Purchase	0.168	0.177	0.120	0.123
Play	185.139	224.755	45.71	54.79
Purchase and Play	143.644	132.321	23.768	25.761

2.17 Conclusions

Low values of RMSE and MAE on training data indicates the model fits well whereas low values on the test data indicate good generalization to new, unseen data. High difference between the training and test data evaluation metrics indicates the model is fitting noise in the training data.

Table 2.7 Comparison of recommender system models

Recommender system based on behavior	Model fits well	Good generalization to unseen data	Model is fitting noise
Purchase	Yes	Fairly good	No
Play	Yes	No	Yes
Purchase and Play	No	No	Yes

Based on the Table 2.7, the low RMSE and MAE values suggests that recommender system based on '*Purchase*' user behavior is most effective at making accurate recommendations. The collaborative filtering recommender system based on '*Purchase*' user behavior is the best performing system.