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: 2 nd May 2024

Contents

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

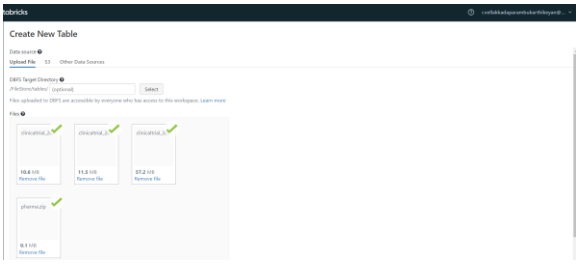
The following points and figures give the setup requirements for the task implementation in Databricks.

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

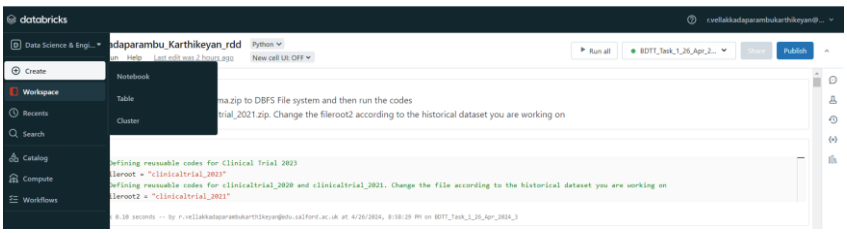
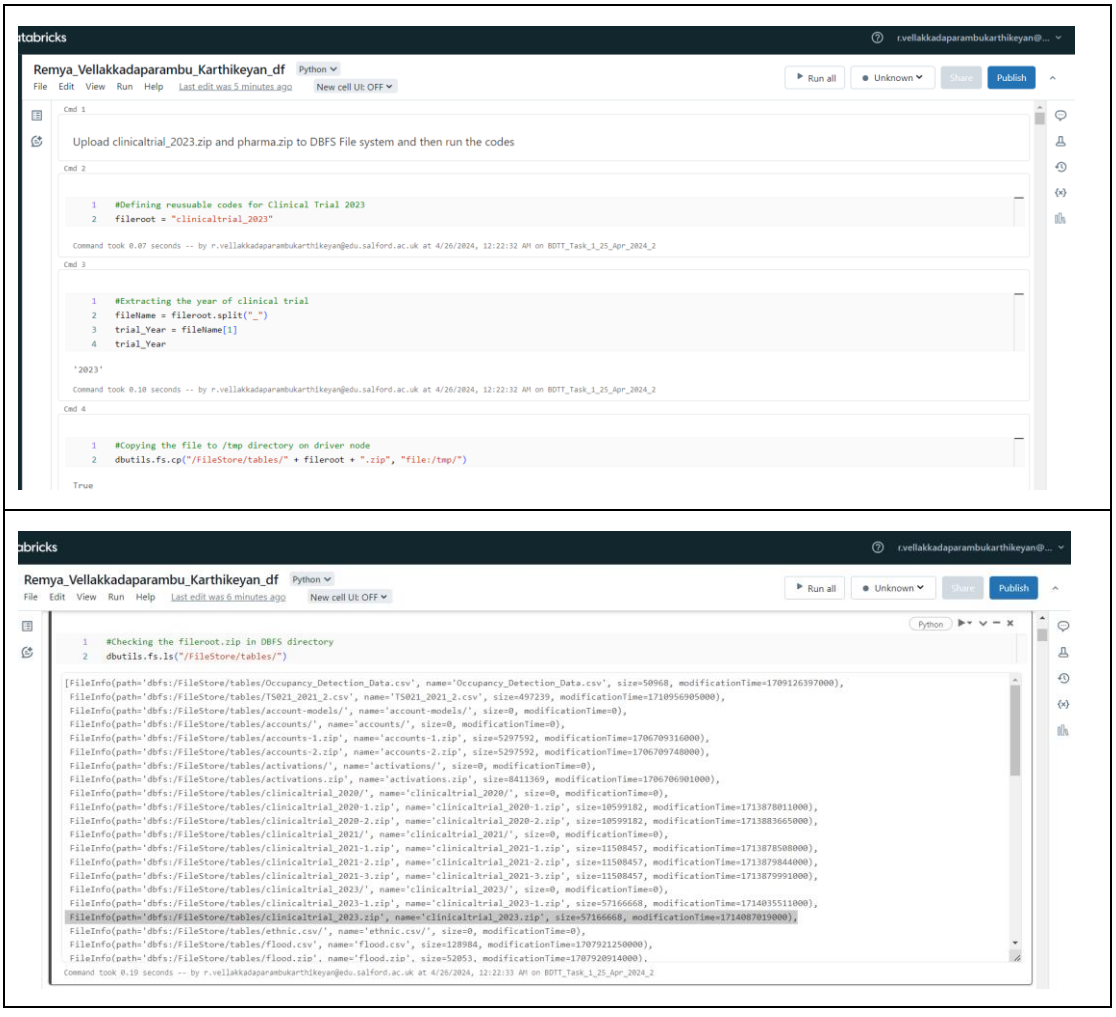


Figure 1.1 Set up requirements

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.



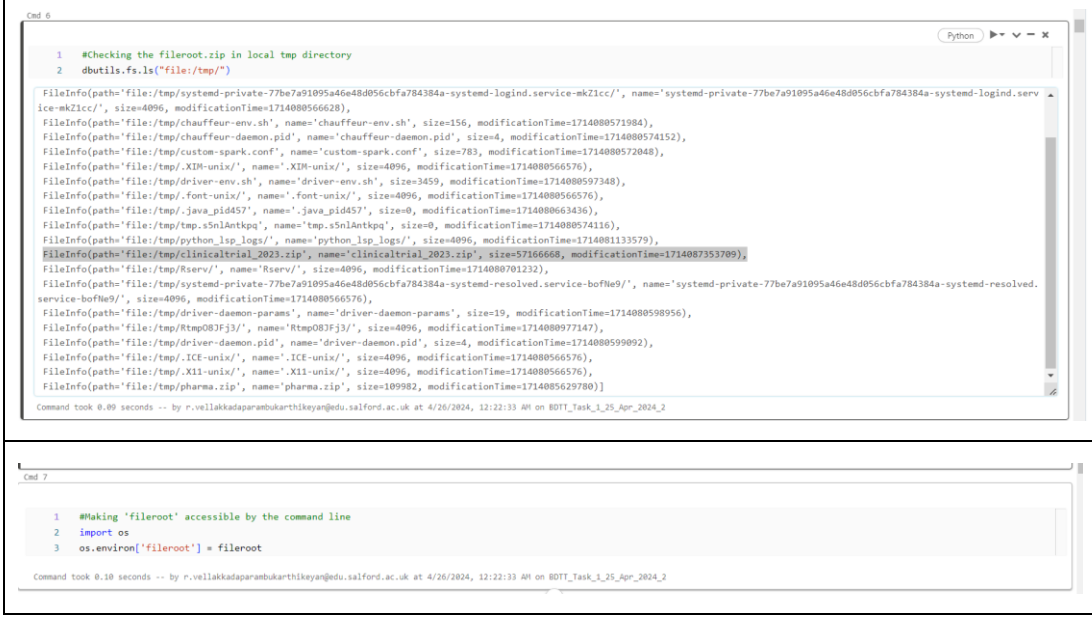


Figure 1.2 Building reusable and rerunnable codes by parameterizing fileroot

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

Moving file from the **/tmp** directory to the DBFS directory

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:

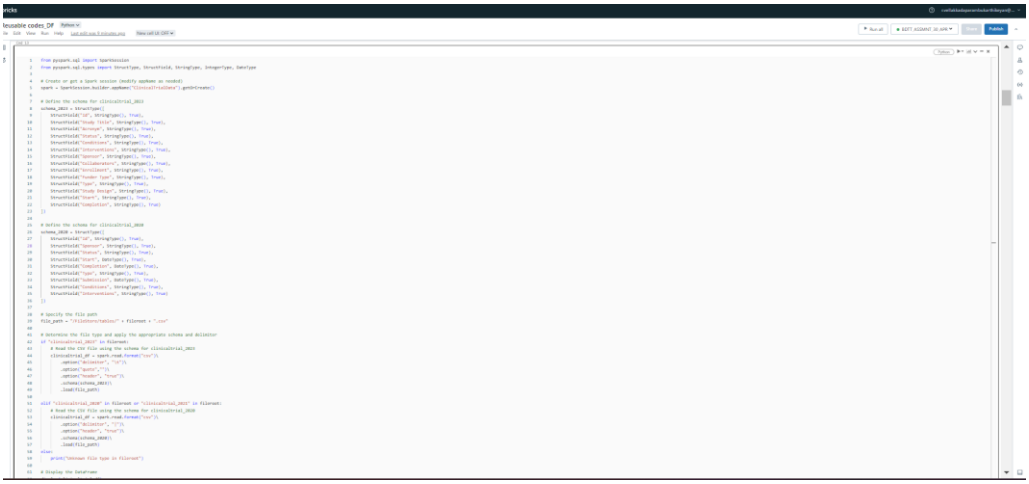
- Delimiter considerations: The delimiter is set to a tab character (**\t**), suggesting the CSV file uses tab-separated values, and the **quote** option is left empty, indicating that no quoting character is used. The **header** option is set to **true** to interpret the first line as column names, 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.

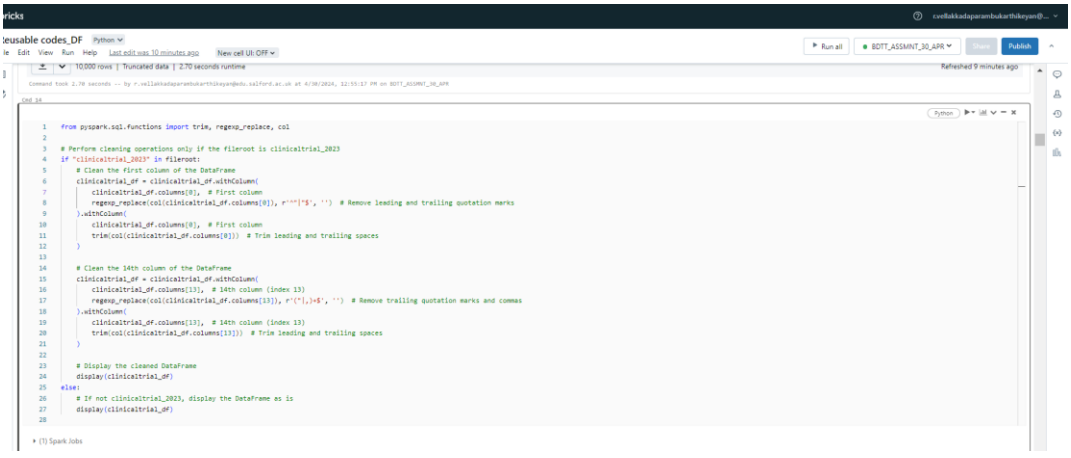
- Trimming Spaces: The leading and trailing spaces are removed from the data in the specified column. This ensures that the data values are consistent and free of extraneous whitespace.

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

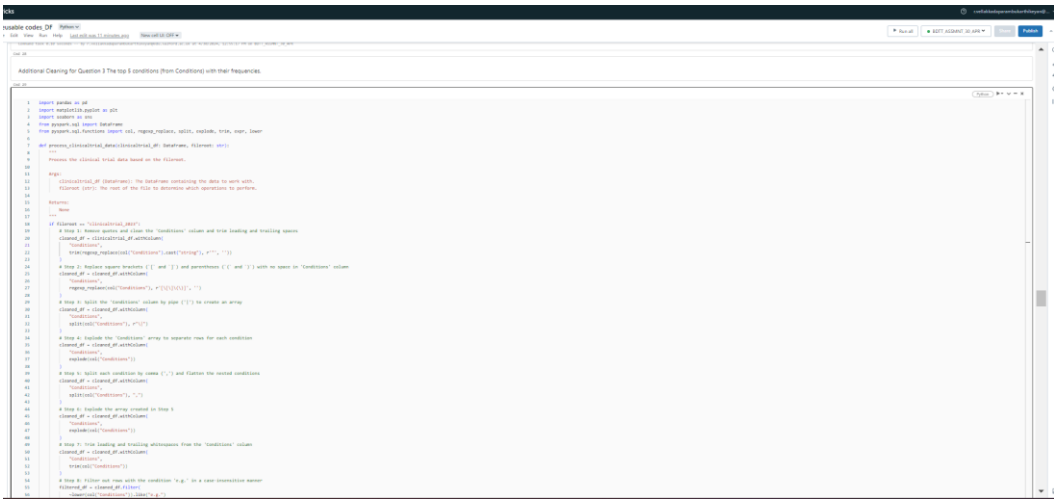


Removing Unwanted Characters



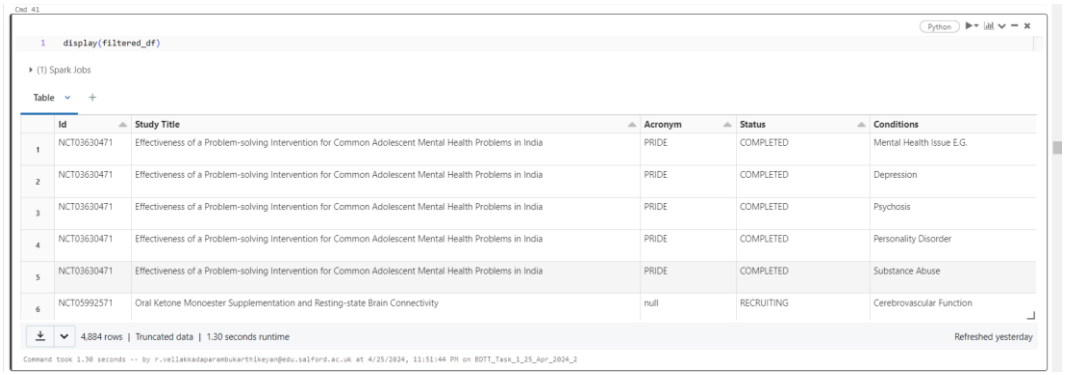
```
from pyspark.sql.functions import trim, regexp_replace, col

# Perform cleaning operations only if the fileroot is clinicaltrial_2023
if "clinicaltrial_2023" in fileroot:
    # Clean the first column of the DataFrame
    clinicaltrial_df = clinicaltrial_df.withColumn(
        clinicaltrial_df.columns[0],  # First column
        regexp_replace(col(clinicaltrial_df.columns[0]), r'^"|"$', '')  # Remove leading and trailing quotation marks
    ).withColumn(
        clinicaltrial_df.columns[0],  # First column
        trim(col(clinicaltrial_df.columns[0]))  # Trim leading and trailing spaces
    )

    # Clean the 14th column of the DataFrame
    clinicaltrial_df = clinicaltrial_df.withColumn(
        clinicaltrial_df.columns[13],  # 14th column (index 13)
        regexp_replace(col(clinicaltrial_df.columns[13]), r'^"|",?$', '')  # Remove trailing quotation marks and commas
    ).withColumn(
        clinicaltrial_df.columns[13],  # 14th column (index 13)
        trim(col(clinicaltrial_df.columns[13]))  # Trim leading and trailing spaces
    )

    # Display the cleaned DataFrame
    display(clinicaltrial_df)
else:
    # If not clinicaltrial_2023, display the DataFrame as is
    display(clinicaltrial_df)
```

Trimming Spaces



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

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql import DataFrame
from pyspark.sql.functions import col, regexp_replace, split, explode, trim, expr, lower

def process_clinicaltrial_data(clinicaltrial_df: DataFrame, filereset: str):
    """
    Process the clinical trial data based on the filereset.

    Args:
        clinicaltrial_df (DataFrame): The DataFrame containing the data to work with.
        filereset (str): The reset of the file to determine which operations to perform.

    Returns:
        None
    """
    if filereset == "clinicaltrial_2024":
        # Step 1: Remove quotes and clean the 'Conditions' column and trim leading and trailing spaces
        cleaned_df = clinicaltrial_df.withColumn(
            "Conditions",
            trim(regexp_replace(col("Conditions").cast("string"), f"^\"|\"$", ''))
        )
        # Step 2: Replace square brackets ('[' and ']') and parentheses ('(' and ')') with no space in 'Conditions' column
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            regexp_replace(col("Conditions"), r"[\[\]\(\)]", '')
        )
        # Step 3: Split the 'Conditions' column by pipe ('|') to create an array
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            split(col("Conditions"), r"\|")
        )
        # Step 4: Explode the 'Conditions' array to separate rows for each condition
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            explode(col("Conditions"))
        )
        # Step 5: Split each condition by comma (',') and flatten the nested conditions
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            split(col("Conditions"), ",")
        )
        # Step 6: Explode the array created in Step 5
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            explode(col("Conditions"))
        )
        # Step 7: Trim leading and trailing whitespaces from the 'Conditions' column
        cleaned_df = cleaned_df.withColumn(
            "Conditions",
            trim(col("Conditions"))
        )
        # Step 8: Filter out rows with the condition 'e.g.' in a case-insensitive manner
        filtered_df = cleaned_df.filter(
            ~lower(col("Conditions")).like("e.g.")
```



Cmd 41

```
display(filtered_df)
```

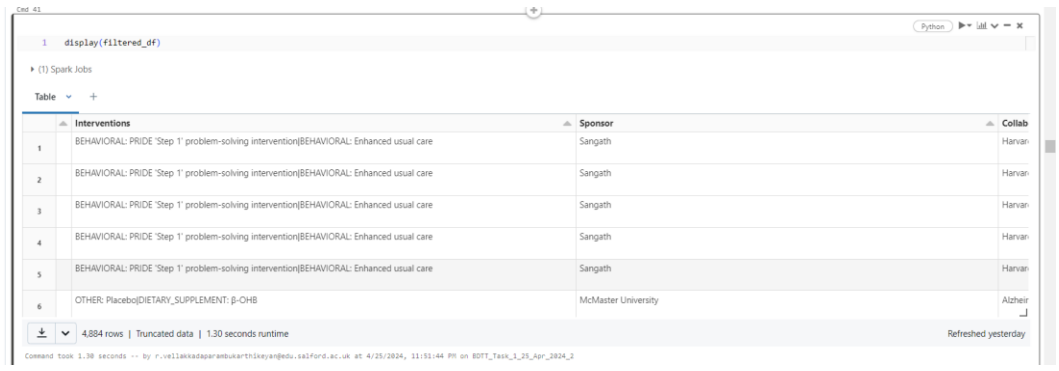
(1) Spark Jobs

Table

	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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/25/2024, 11:51:44 PM on BDTT_Task_1_25_Apr_2024_2



Cmd 41

```
display(filtered_df)
```

(1) Spark Jobs

Table

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

4,884 rows | Truncated data | 1.30 seconds runtime — Refreshed yesterday

Command took 1.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/25/2024, 11:51:44 PM on BDTT_Task_1_25_Apr_2024_2

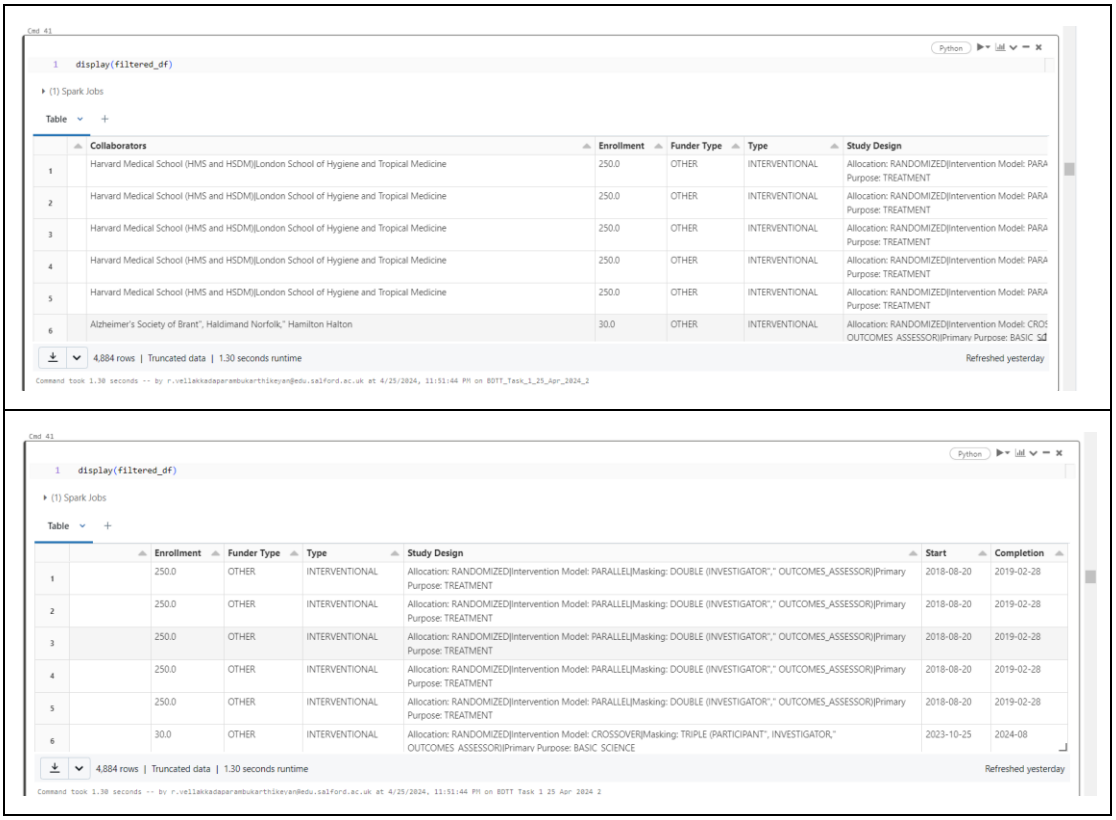


Figure 1.4 Data cleaning and preparation in PySpark DataFrame approach

Delimiter considerations

```python
from pyspark import SparkContext
from pyspark.sql.functions import trim, regexp_replace
import re

# Assuming sc is your SparkContext and the RDD is already created
# sc = SparkContext.getOrCreate()

# Function to clean the first field in a line
def clean_first_field(line):
    # Split the line into fields using tab as the delimiter
    fields = line.split('\t')

    # Clean the first field (first column) by removing leading and trailing quotation marks and trimming spaces
    first_field = fields[0]
    first_field = re.sub(r'^"|"$', '', first_field)  # Remove leading and trailing quotation marks
    first_field = first_field.strip()  # Trim leading and trailing spaces

    # Replace the cleaned first field in the list of fields
    fields[0] = first_field

    # Reconstruct the line by joining the fields with the tab delimiter
    cleaned_line = '\t'.join(fields)

    return cleaned_line

# Function to clean the 14th field in a line
def clean_14th_field(line):
    # Split the line into fields using tab as the delimiter
    fields = line.split('\t')

    # Check if the line has at least 14 fields
    if len(fields) < 14:
        # Return the line unchanged if there aren't enough fields
        return line
```

Removing unwanted characters and Trimming spaces

```
    first_field = first_field.strip()  # Trim leading and trailing spaces

    # Replace the cleaned first field in the list of fields
    fields[0] = first_field

    # Reconstruct the line by joining the fields with the tab delimiter
    cleaned_line = '\t'.join(fields)

    return cleaned_line

# Function to clean the 14th field in a line
def clean_14th_field(line):
    # Split the line into fields using tab as the delimiter
    fields = line.split('\t')

    # Check if the line has at least 14 fields
    if len(fields) < 14:
        # Return the line unchanged if there aren't enough fields
        return line

    # Clean the 14th field (14th column) by removing trailing commas and quotation marks
    # `fields[13]` represents the 14th column since indexing starts at 0
    fields[13] = re.sub(r'("),+$', '', fields[13])  # Remove trailing quotation marks and commas
    fields[13] = fields[13].strip()  # Trim leading and trailing spaces

    # Reconstruct the line by joining the fields with the tab delimiter
    cleaned_line = '\t'.join(fields)

    return cleaned_line
```

```
# 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

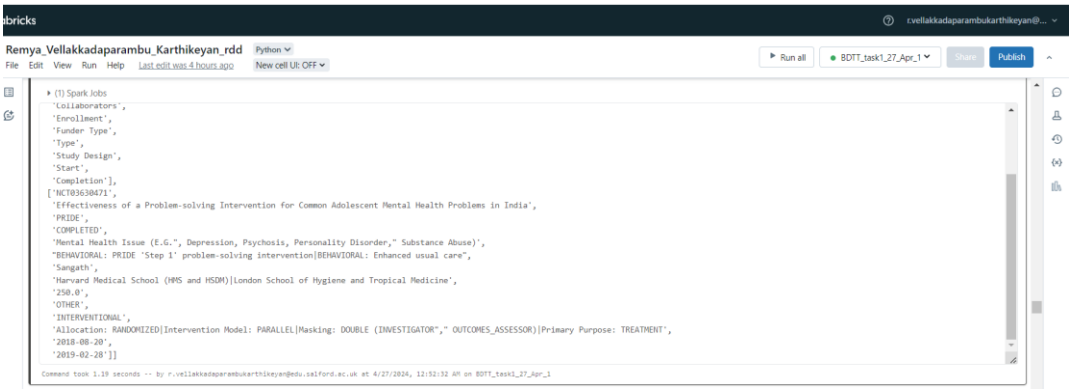
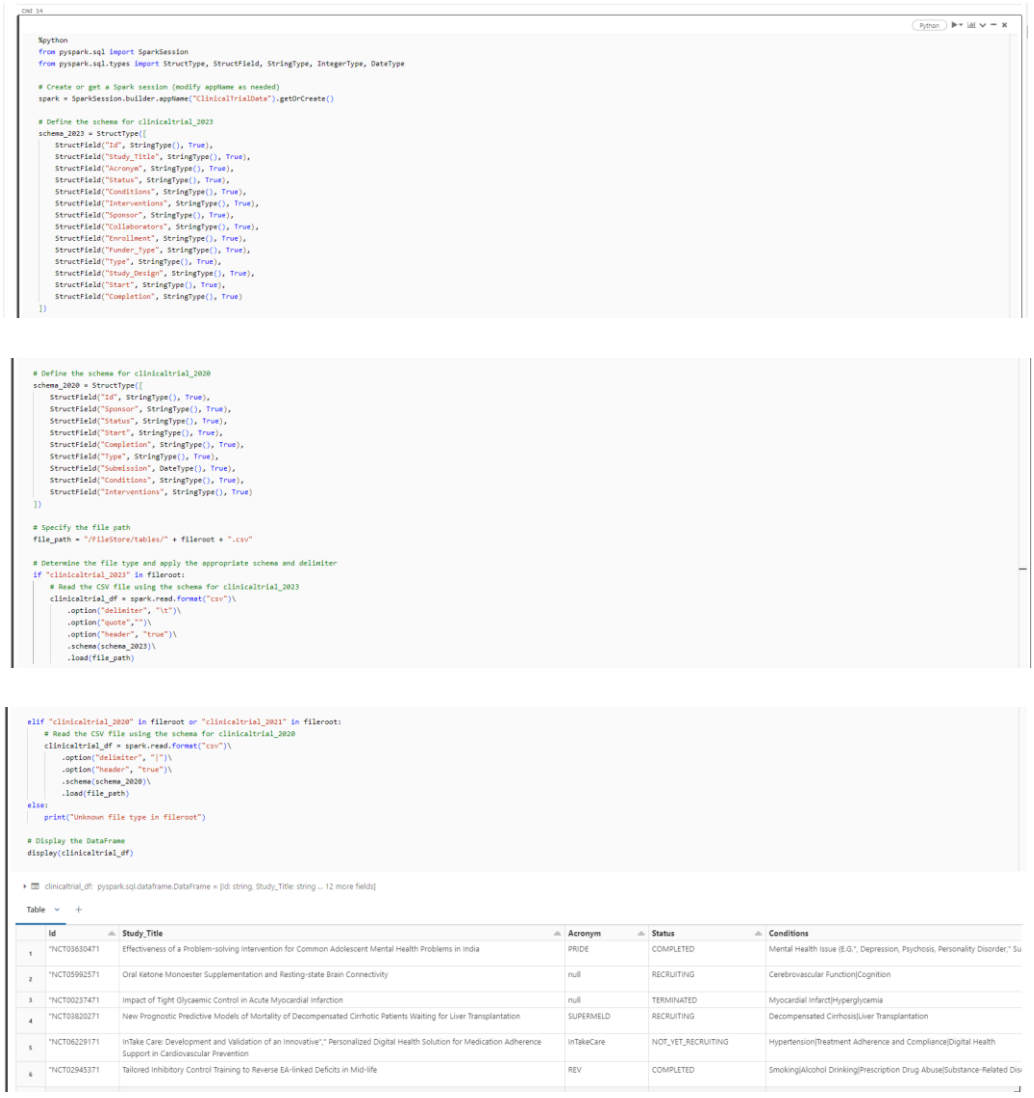


Figure 1.5 Data cleaning and preparation in PySpark RDD approach

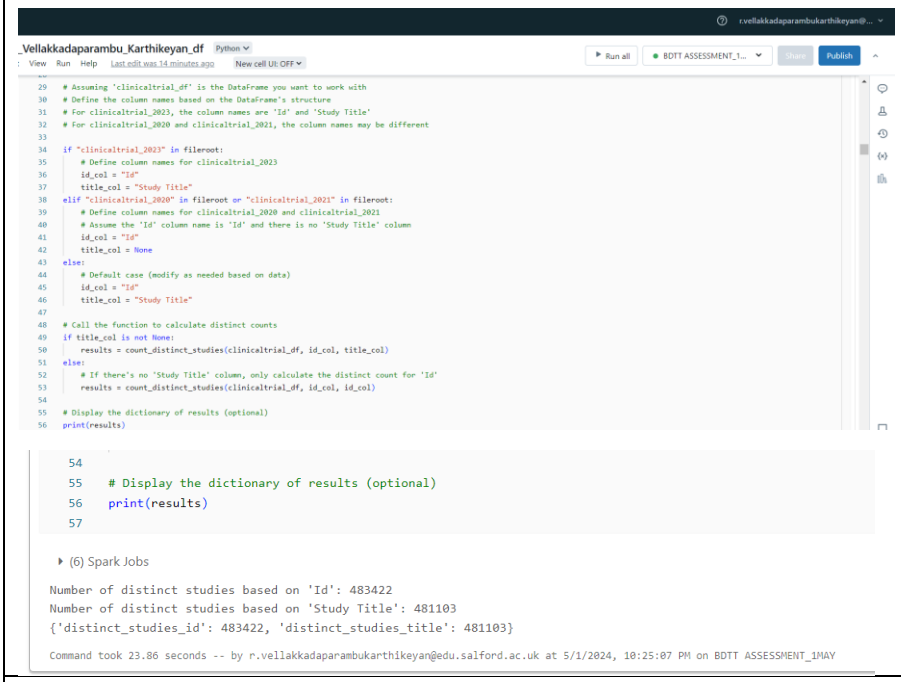


The data cleaning and preparation in Spark SQL approach foolows the same codes as in DataFrame approach as the clinicaltrial table is created from dataframe.

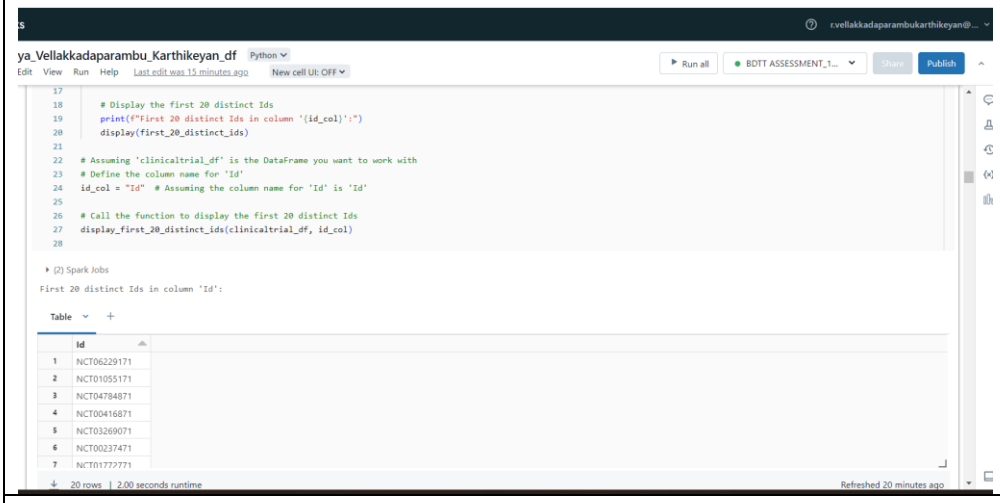
Figure 1.6 Data cleaning and preparation in Spark SQL approach

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

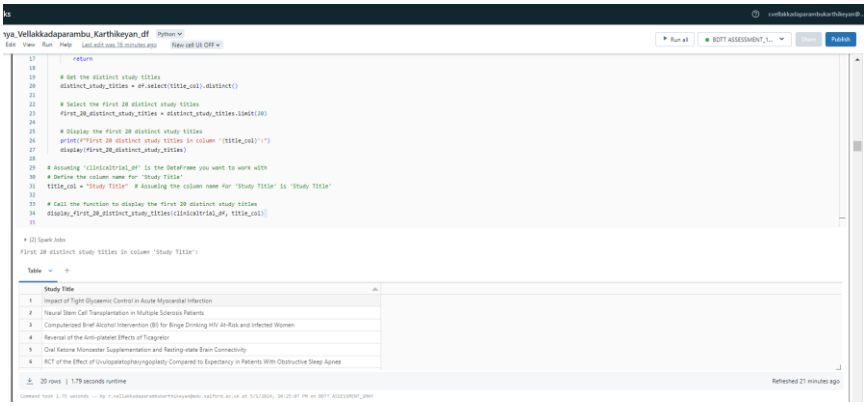
Dataframe approach



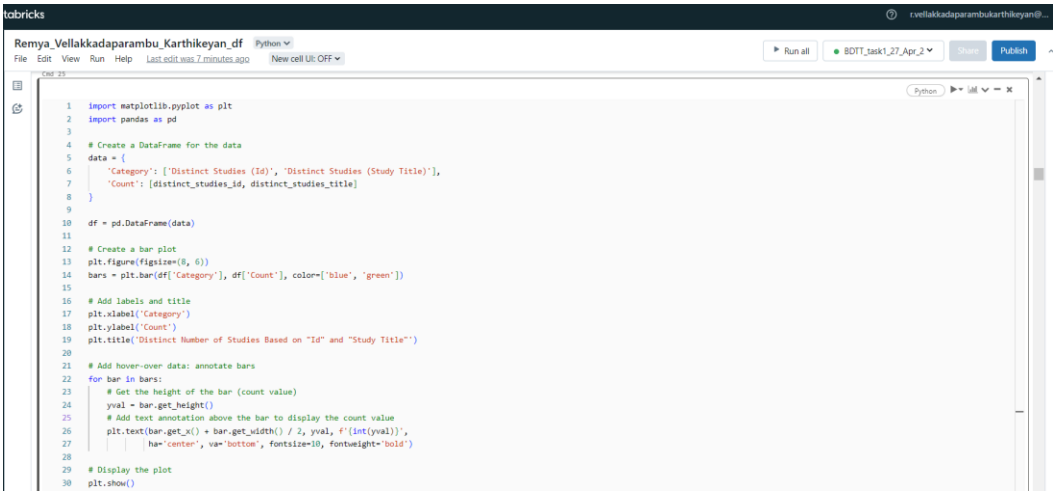
Distinct 20 studies based on ‘Id’

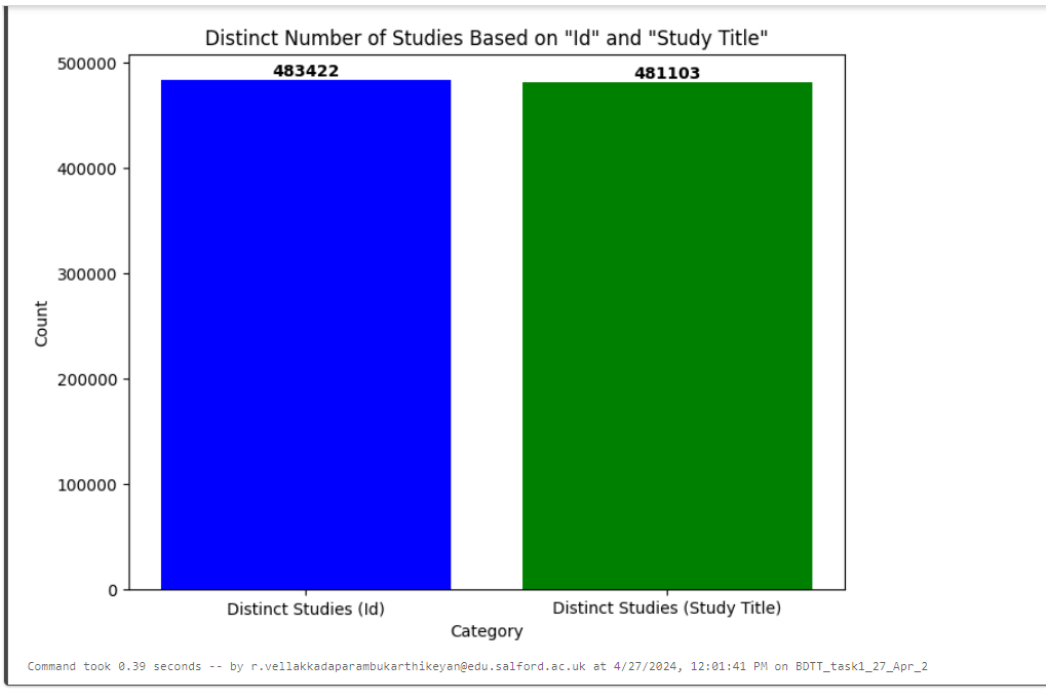


Distinct 20 studies based on ‘Study Title’



Visualizing the answer





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

RDD Approach

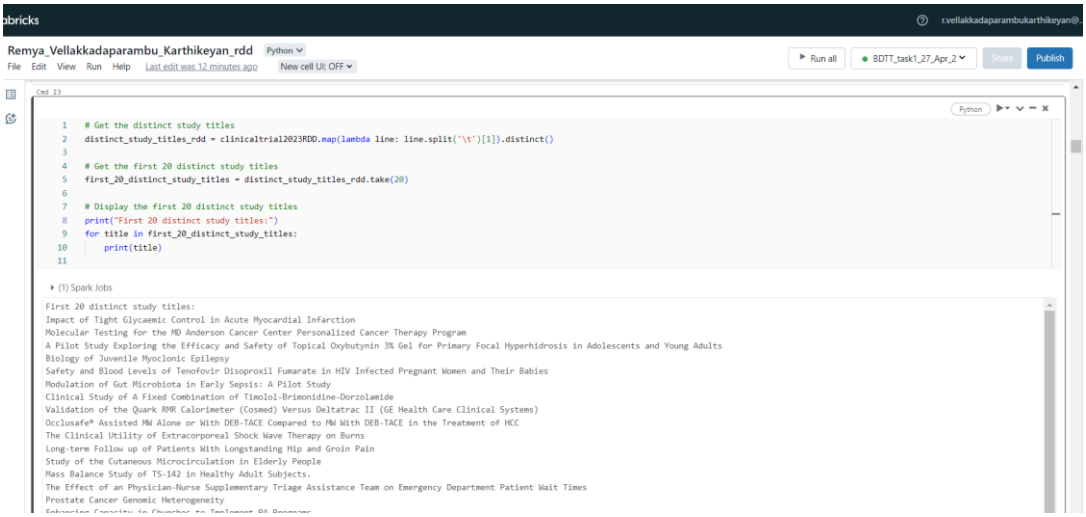
```
# Discard the header from the RDD
clinicaltrial2023RDD_no_header = clinicaltrial2023RDD.zipWithIndex().filter(lambda line_index: line_index[1] != 0).map(lambda line_index: line_index[0])

# Count the distinct number of studies based on the 'Id' column
distinct_studies_id = clinicaltrial2023RDD_no_header.map(lambda line: line.split('\t')[0]).distinct().count()

# Count the distinct number of studies based on the 'Study Title' column
distinct_studies_title = clinicaltrial2023RDD_no_header.map(lambda line: line.split('\t')[1]).distinct().count()

# Display the results
print(f"Number of distinct studies based on 'Id': {distinct_studies_id}")
print(f"Number of distinct studies based on 'Study Title': {distinct_studies_title}")
```

(3) Spark Jobs

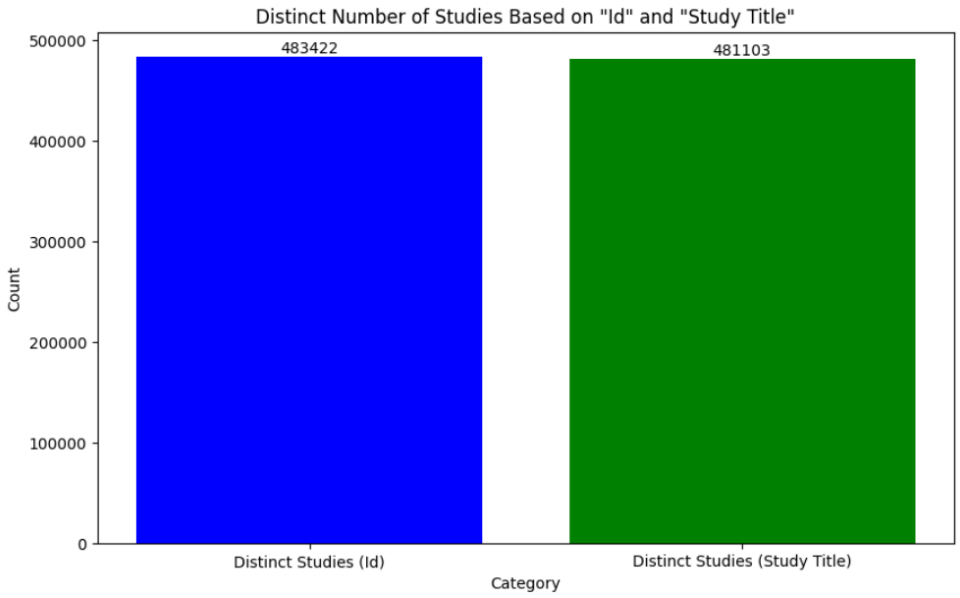
Number of distinct studies based on 'Id': 483422
Number of distinct studies based on 'Study Title': 481103

Distinct 20 studies based on ‘Id’

Distinct 20 studies based on ‘Study Title’

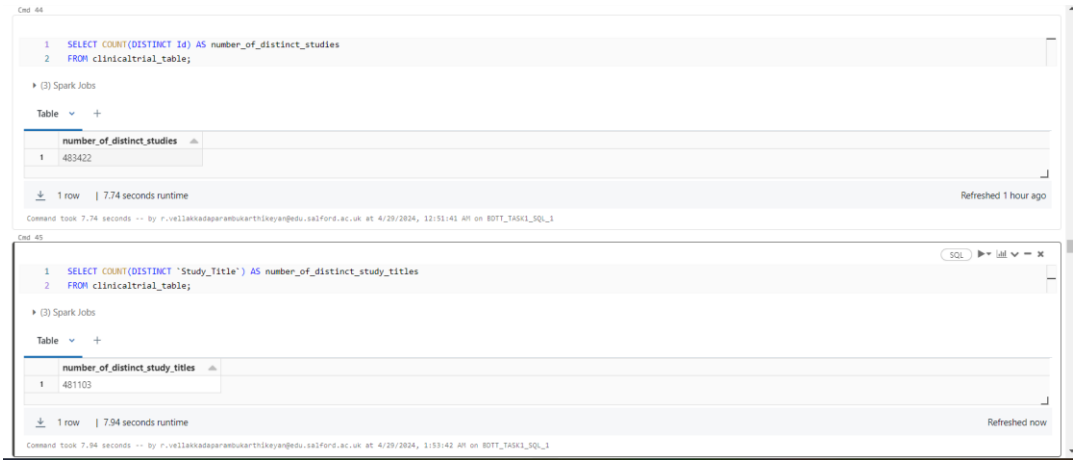


Visualizing the answer



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

Spark SQL Approach



Distinct 20 studies based on ‘Id’

```
SELECT DISTINCT Id
FROM clinicaltrial_table
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

Distinct 20 studies based on ‘Study Title’

```
SELECT DISTINCT Study_Title
FROM clinicaltrial_table
LIMIT 20;
```

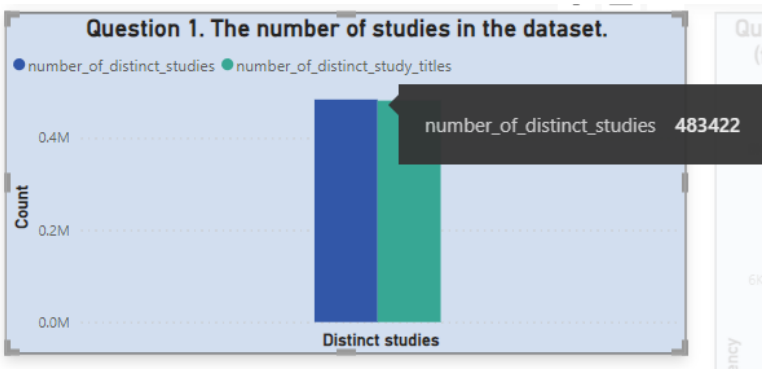
(2) Spark Jobs

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 (BI) for Binge Drinking HIV At-Risk and Infected Women
4	Reversal of the Anti-platelet Effects of Ticagrelor
5	Oral Ketone Monoester 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

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.

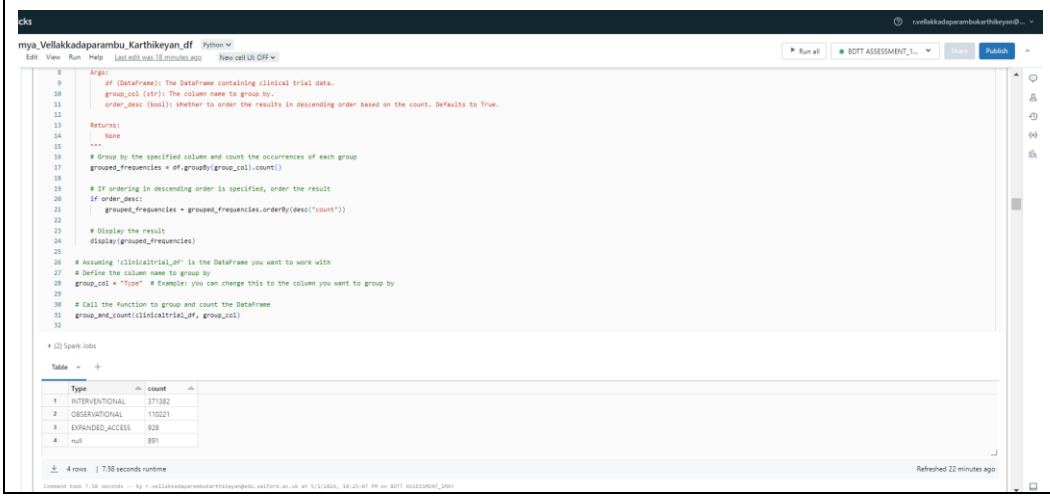
Table 1.1 Discussion of Results of Question 1

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

```
from pyspark.sql.functions import desc
from pyspark.sql import DataFrame

def group_and_count(df: DataFrame, group_col: str, order_desc: bool = True):
    """
    Group the DataFrame by a specified column and count the occurrences.

    Args:
        df (DataFrame): The DataFrame containing clinical trial data.
        group_col (str): The column name to group by.
        order_desc (bool): Whether to order the results in descending order based on the count. Defaults to True.

    Returns:
        None
    """
    # Group by the specified column and count the occurrences of each group
    grouped_frequencies = df.groupBy(group_col).count()

    # If ordering in descending order is specified, order the result
    if order_desc:
        grouped_frequencies = grouped_frequencies.orderBy(desc("count"))

    # Display the result
    display(grouped_frequencies)

# Assuming 'clinicaltrial_df' is the DataFrame you want to work with
# Define the column name to group by
```



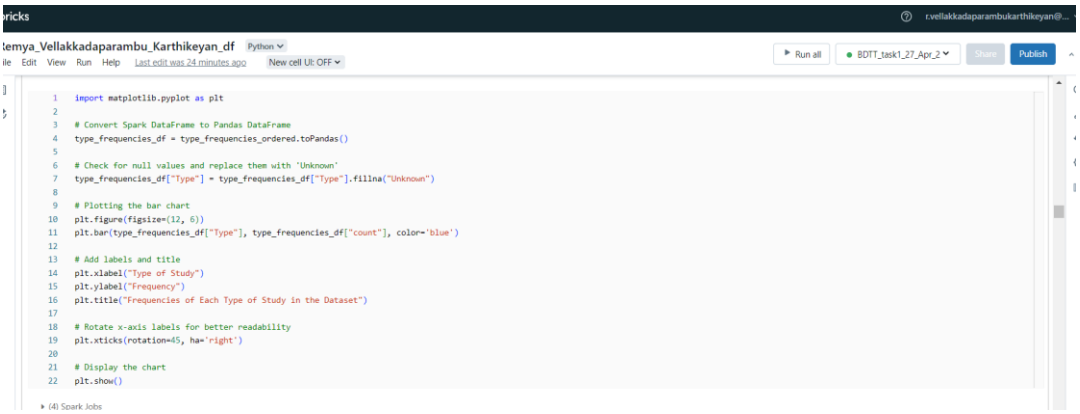
```
mya_Vellakkadaparambu_Karthikeyan_df
Edit View Run Help Last edit was 18 minutes ago New cell UI: OFF
Run all BOTT ASSESSMENT_1... Share Publish

# Assuming 'clinicaltrial_df' is the DataFrame you want to work with
# Define the column name to group by
group_col = "Type"  # Example: you can change this to the column you want to group by

# Call the function to group and count the DataFrame
group_and_count(clinicaltrial_df, group_col)

(2) Spark Jobs

Table
```

	Type	count
1	INTERVENTIONAL	371382
2	OBSERVATIONAL	110221
3	EXPANDED_ACCESS	928
4	null	891

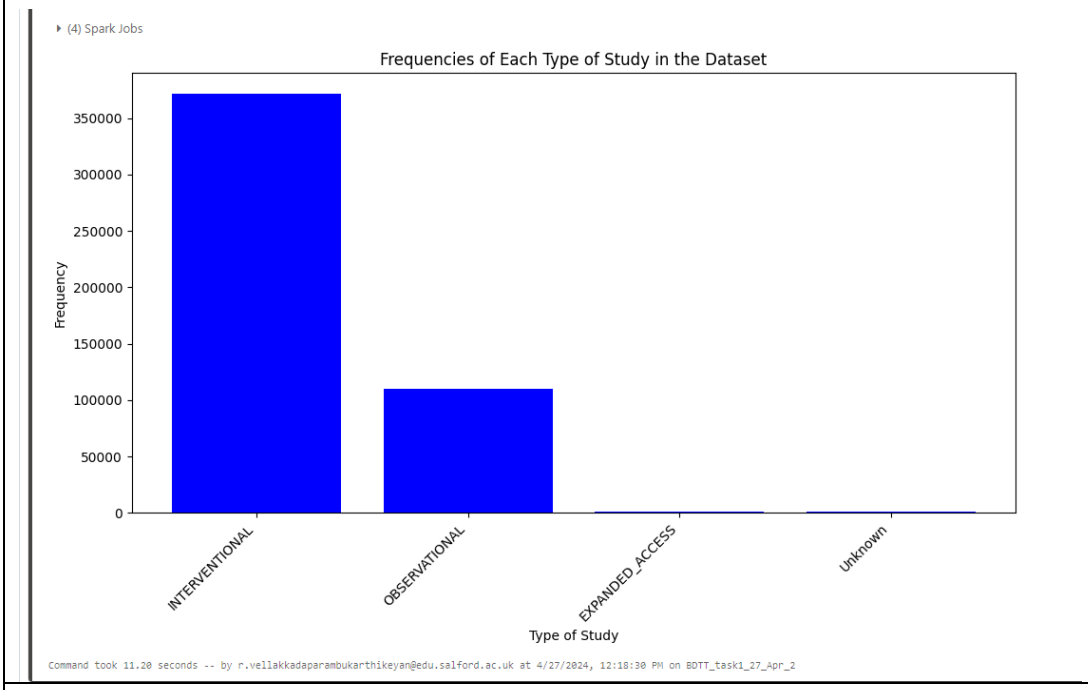
4 rows | 7.58 seconds runtime

Refreshed 22 minutes ago

Visualizing the answer



```
import matplotlib.pyplot as plt

# Convert Spark DataFrame to Pandas DataFrame
type_frequencies_df = type_frequencies_ordered.toPandas()

# Check for null values and replace them with 'Unknown'
type_frequencies_df["Type"] = type_frequencies_df["Type"].fillna("Unknown")

# Plotting the bar chart
plt.figure(figsize=(12, 6))
plt.bar(type_frequencies_df["Type"], type_frequencies_df["count"], color='blue')

# Add labels and title
plt.xlabel("Type of Study")
plt.ylabel("Frequency")
plt.title("Frequencies of Each Type of Study in the Dataset")

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Display the chart
plt.show()
```



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

RDD Approach

```
# Filter out the header line from the RDD using zipWithIndex
clinicaltrial2023RDD_no_header = clinicaltrial2023RDD.zipWithIndex().filter(lambda line_index: line_index[1] > 0).map(lambda line_index: line_index[0])

# Define a function to extract the "Type" column
def extract_type(line):
    fields = line.split('\t')
    if len(fields) > 10:  # Ensure there are at least 11 fields
        return fields[10]  # Return the "Type" column (index 10)
    else:
        return None  # Return None if the line does not have enough fields

# Get the "Type" column while filtering out invalid lines
type_rdd = clinicaltrial2023RDD_no_header.map(extract_type).filter(lambda x: x is not None)

# Perform the counting and sorting
type_counts_rdd = type_rdd.map(lambda type_: (type_, 1)).reduceByKey(lambda a, b: a + b)
type_counts_ordered_rdd = type_counts_rdd.sortBy(lambda x: -x[1])
results = type_counts_ordered_rdd.collect()

# Display the results
print("Type frequencies ordered from most frequent to least frequent:")
for type_, count in results:
    print(f"Type: {type_}, Count: {count}")
```

▸ (4) Spark Jobs

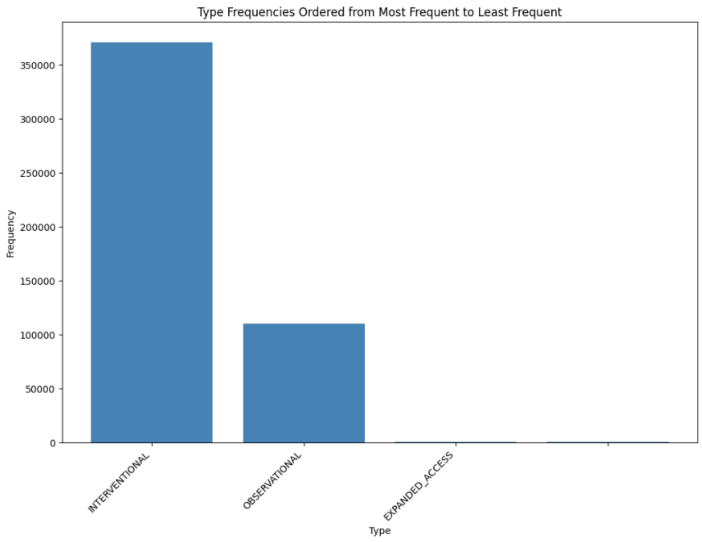
```
Type frequencies ordered from most frequent to least frequent:
Type: INTERVENTIONAL, Count: 371382
Type: OBSERVATIONAL, Count: 110221
Type: EXPANDED_ACCESS, Count: 928
Type: , Count: 889
```

Command took 28.71 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:44 PM on BDTT_task1_27_Apr_2

Visualizing the answer


```
import matplotlib.pyplot as plt

# Get the type frequencies ordered from most frequent to least frequent
type_counts_rdd = type_rdd.map(lambda type_: (type_, 1)).reduceByKey(lambda a, b: a + b)
type_counts_ordered_rdd = type_counts_rdd.sortBy(lambda x: -x[1])
results = type_counts_ordered_rdd.collect()

# Prepare data for plotting
type_names = []
type_counts = []
for type_, count in results:
    type_names.append(type_)
    type_counts.append(count)

# Plotting the type frequencies
plt.figure(figsize=(12, 8))
plt.bar(type_names, type_counts, color='steelblue')
plt.xlabel('Type')
plt.ylabel('Frequency')
plt.title('Type Frequencies Ordered from Most Frequent to Least Frequent')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.show()
```

▸ (3) Spark Jobs

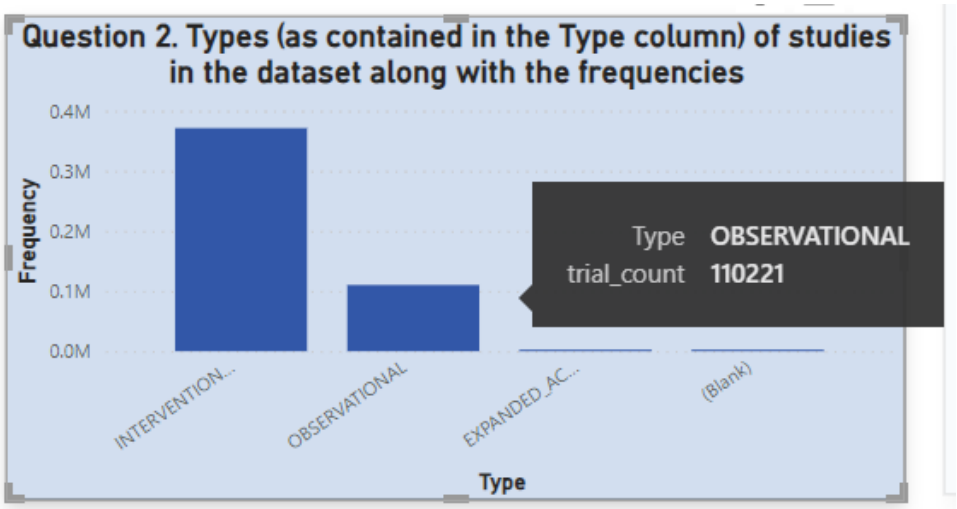


Command took 17.33 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:44 PM on BDTT_task1_27_Apr_2

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

Spark SQL Approach

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

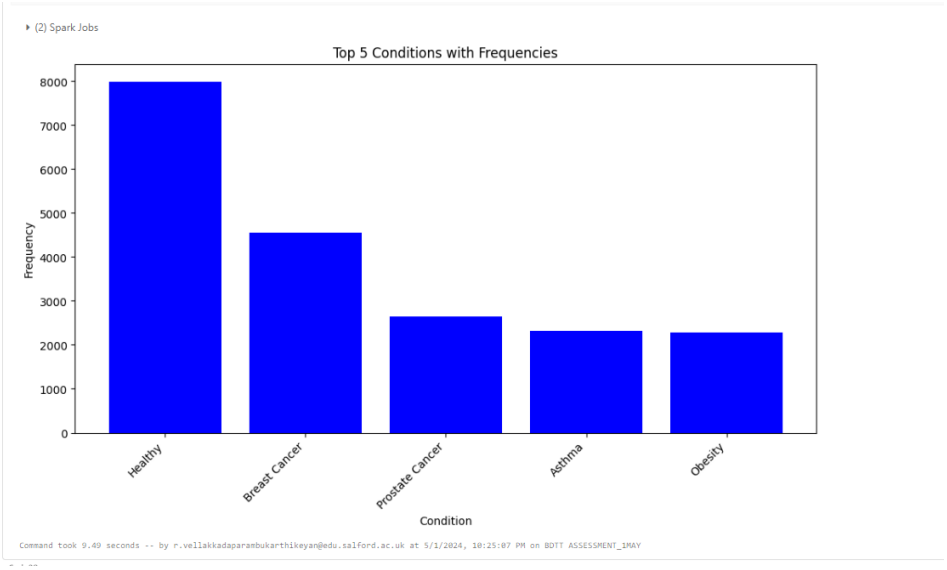
Dataframe approach

Directly finding the top 5 conditions from the dataset

```
from pyspark.sql import DataFrame
from pyspark.sql.functions import desc, col, split, explode, trim, isnull

def display_top_5_conditions(clinicaltrial_df: DataFrame, fileroot: str):
    """
    Display the top 5 conditions and their frequencies based on the given fileroot.

    Args:
        clinicaltrial_df (DataFrame): The DataFrame containing the data to work with.
        fileroot (str): The root of the file to determine which operations to perform.

    Returns:
        None
    """
    if fileroot == "clinicaltrial_2023":
        # For clinicaltrial_2023, group by "Conditions" column and count occurrences
        condition_frequencies = clinicaltrial_df.groupBy("Conditions").count()
        # Order the result from most frequent to least frequent and limit to the top 5
        top_5_conditions = condition_frequencies.orderBy(desc("count")).limit(5)
        # Display the top 5 conditions with their frequencies
        top_5_conditions.show()

    elif fileroot in ("clinicaltrial_2020", "clinicaltrial_2021"):
        # For clinicaltrial_2020 and clinicaltrial_2021, split the 'conditions' column by commas
        conditions_df = clinicaltrial_df.select(split('conditions', ',').alias('condition_list'))
        # Use explode to create a new DataFrame with one row per condition
        exploded_conditions_df = conditions_df.select(explode('condition_list').alias('condition'))
        # Filter out empty strings and whitespace-only conditions
        exploded_conditions_df = exploded_conditions_df.filter(
            ~isnull(exploded_conditions_df['condition']) & (trim(exploded_conditions_df['condition']) != "")
        )
        # Calculate the frequency of each condition using groupBy and count
        condition_frequency_df = exploded_conditions_df.groupBy('condition').count()
        # Order the conditions by frequency in descending order and limit to the top 5
        top_5_conditions_df = condition_frequency_df.orderBy(desc('count')).limit(5)
        # Display the top 5 conditions and their frequencies
        print("Top 5 conditions and their frequencies:")
        top_5_conditions_df.show()
```

```
a_Vellakkadaparambu_Karthikeyan_df
Run all
BDTT ASSESSMENT_1...
Share
Publish

# Use explode to create a new DataFrame with one row per condition
exploded_conditions_df = conditions_df.select(explode('condition_list').alias('condition'))
# Filter out empty strings and whitespace-only conditions
exploded_conditions_df = exploded_conditions_df.filter(
    ~isnull(exploded_conditions_df['condition']) & (trim(exploded_conditions_df['condition']) != "")
)
# Calculate the frequency of each condition using groupBy and count
condition_frequency_df = exploded_conditions_df.groupBy('condition').count()
# Order the conditions by frequency in descending order and limit to the top 5
top_5_conditions_df = condition_frequency_df.orderBy(desc('count')).limit(5)
# Display the top 5 conditions and their frequencies
print("Top 5 conditions and their frequencies:")
top_5_conditions_df.show()

display_top_5_conditions(clinicaltrial_df, fileroot)

(2) Spark Jobs

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

Command took 8.39 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 5/1/2024, 10:25:07 PM on BDTT ASSESSMENT_1DAY
```

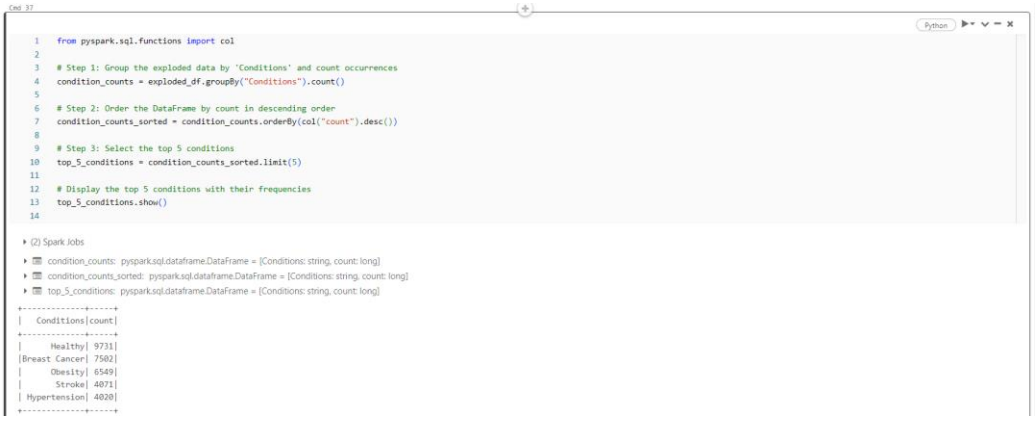
Visualization



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

```
from pyspark.sql.functions import col, regexp_replace, split, explode, trim, expr, lower

# Step 1: Split the 'Conditions' column by pipe ('|') to create an array
cleaned_df = clinicaltrial2023DF.withColumn(
    "Conditions",
    split(col("Conditions"), r"\|")
)

# Step 2: Explode the 'Conditions' array to separate rows for each condition
exploded_df = cleaned_df.withColumn(
    "Conditions",
    explode(col("Conditions"))
)
```

cleaned_df: pyspark.sql.dataframe.DataFrame = [Id: string, Study Title: string ... 12 more fields]
exploded_df: pyspark.sql.dataframe.DataFrame = [Id: string, Study Title: string ... 12 more fields]
Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:16:30 PM on BDTT_tasks1_27_Apr_2

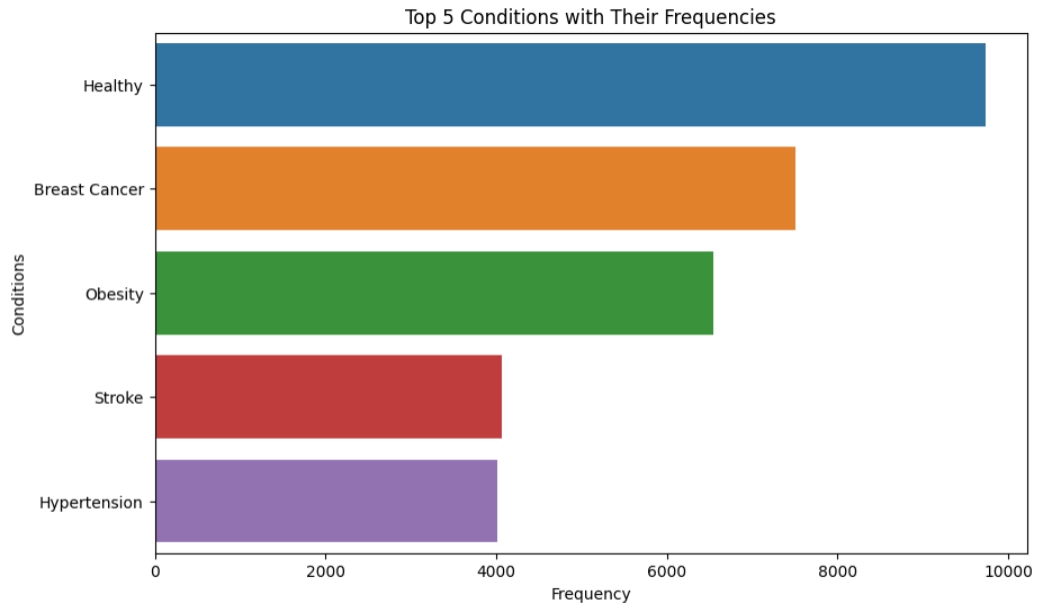


```
from pyspark.sql.functions import col

# Step 1: Group the exploded data by 'Conditions' and count occurrences
condition_counts = exploded_df.groupBy("Conditions").count()

# Step 2: Order the DataFrame by count in descending order
condition_counts_sorted = condition_counts.orderBy(col("count").desc())

# Step 3: Select the top 5 conditions
top_5_conditions = condition_counts_sorted.limit(5)

# Display the top 5 conditions with their frequencies
top_5_conditions.show()
```

(2) Spark Jobs
condition_counts: pyspark.sql.dataframe.DataFrame = [Conditions: string, count: long]
condition_counts_sorted: pyspark.sql.dataframe.DataFrame = [Conditions: string, count: long]
top_5_conditions: pyspark.sql.dataframe.DataFrame = [Conditions: string, count: long]

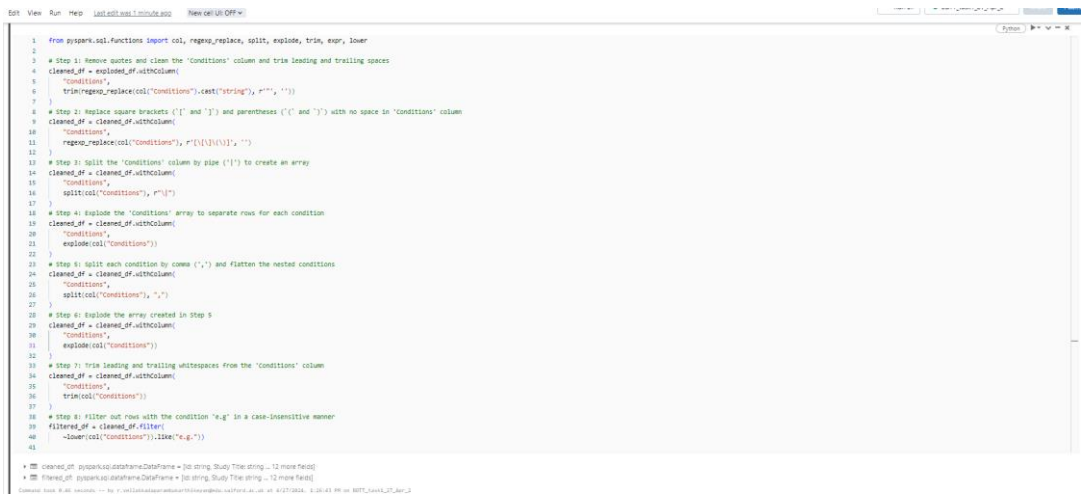
Conditions	count
Healthy	9731
Breast Cancer	7502
Obesity	6549
Stroke	4071
Hypertension	4020

Visualization



Command took 9.79 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 1:23:52 PM on BDTT_task1_27_Apr_2

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



```
from pyspark.sql.functions import col, regexp_replace, split, explode, trim, expr, lower

# Step 1: Remove quotes and clean the 'Conditions' column and trim leading and trailing spaces
cleaned_df = exploded_df.withColumn(
    "Conditions",
    trim(regexp_replace(col("Conditions").cast("string"), r'"', ''))
)
# Step 2: Replace square brackets ('[' and ']') and parentheses ('(' and ')') with no space in 'Conditions' column
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    regexp_replace(col("Conditions"), r'[\[\]\(\)]', '')
)
# Step 3: Split the 'Conditions' column by pipe ('|') to create an array
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    split(col("Conditions"), r"\|")
)
# Step 4: Explode the 'Conditions' array to separate rows for each condition
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    explode(col("Conditions"))
)
# Step 5: Split each condition by comma (',') and flatten the nested conditions
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    split(col("Conditions"), ",")
)
# Step 6: Explode the array created in Step 5
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    explode(col("Conditions"))
)
# Step 7: Trim leading and trailing whitespaces from the 'Conditions' column
cleaned_df = cleaned_df.withColumn(
    "Conditions",
    trim(col("Conditions"))
)
# Step 8: Filter out rows with the condition 'e.g' in a case-insensitive manner
filtered_df = cleaned_df.filter(
    ~lower(col("Conditions")).like("e.g.")
)
```

```
from pyspark.sql.functions import col

# Step 10: Calculate the top 5 conditions with their frequencies
top_5_conditions = filtered_df.groupBy("Conditions").count()  # Group by 'Conditions' and count occurrences
top_5_conditions = top_5_conditions.orderBy(col("count").desc())  # Order by count in descending order
top_5_conditions = top_5_conditions.limit(5)  # Select the top 5 conditions

# Display the top 5 conditions with their frequencies
top_5_conditions.show()
```

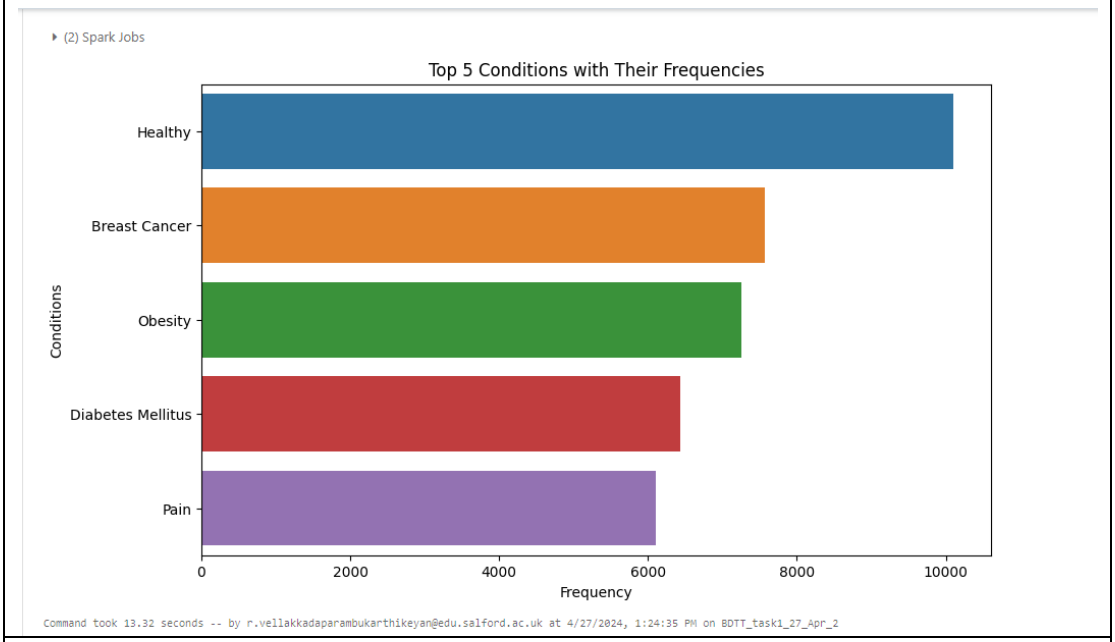
(2) Spark Jobs

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

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

Command took 12.23 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 1:24:18 PM on BDTT_task1_27_Apr_2

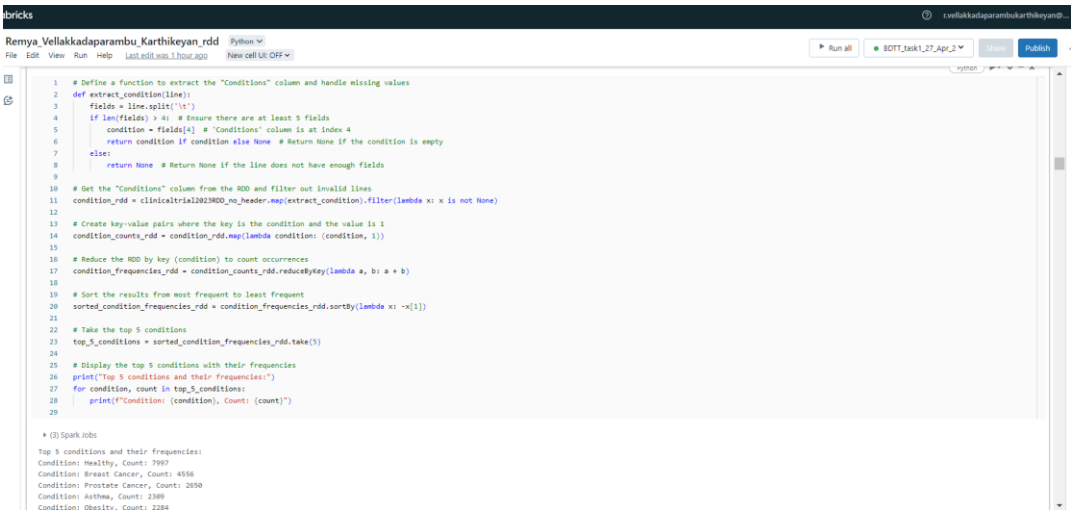
Visualization



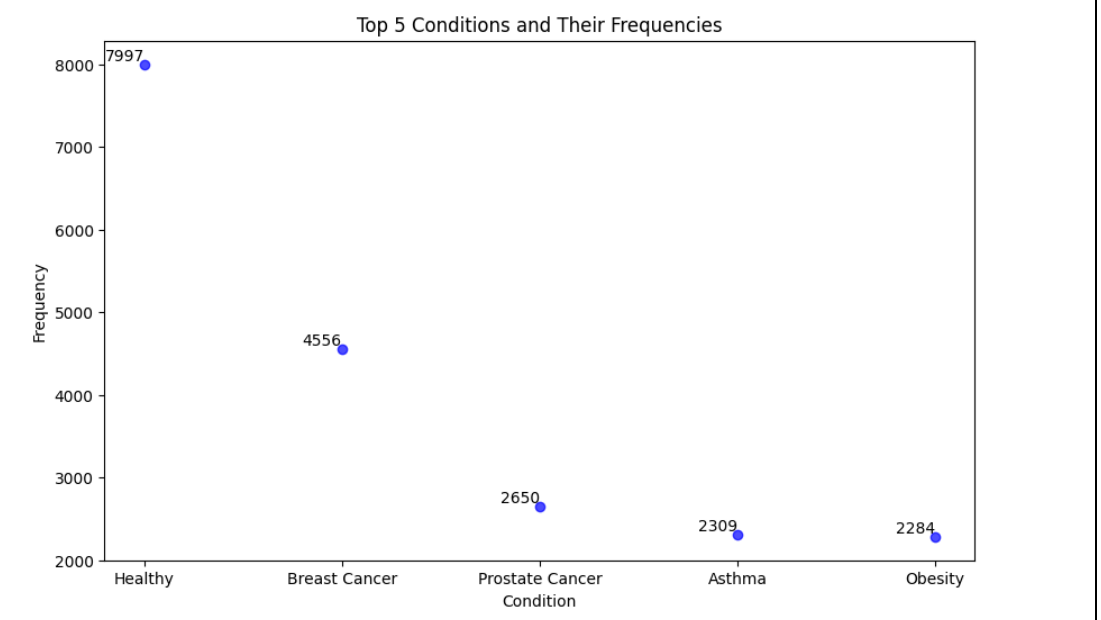
Main Idea : The code groups the clinicaltrial_df by the "Conditions" column, orders the count column in descending order, selects the top 5 most common conditions from the ordered data and display them.

RDD Approach

Directly finding the top 5 conditions from the dataset

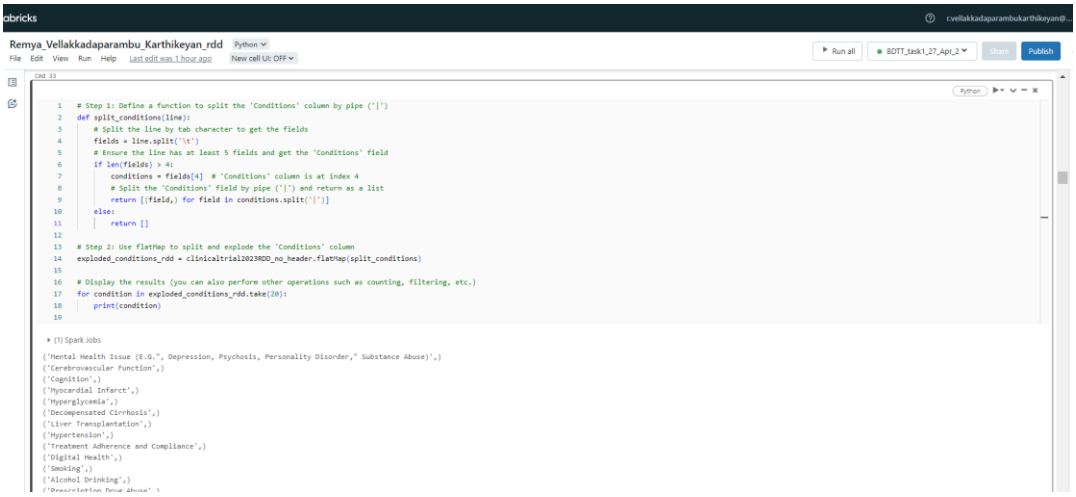


Visualization



Command took 0.40 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:44 PM on BDTT_task1_27_Apr_2

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



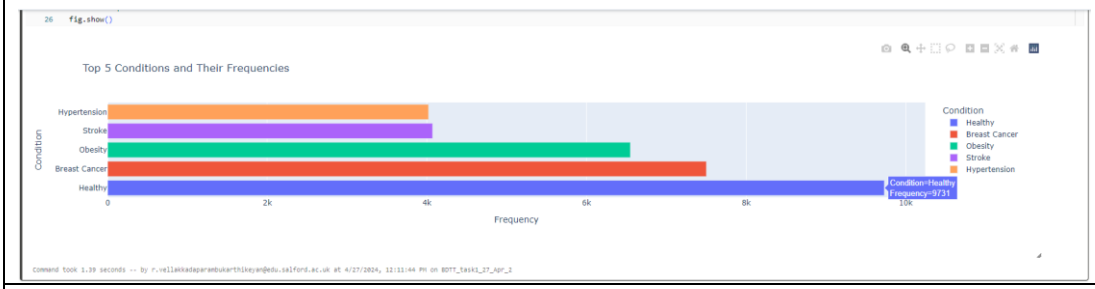
```
# Step 1: Define a function to split the 'Conditions' column by pipe ('|')
def split_conditions(line):
    # Split the line by tab character to get the fields
    fields = line.split('\t')
    # Ensure the line has at least 5 fields and get the 'Conditions' field
    if len(fields) > 4:
        conditions = fields[4]  # 'Conditions' column is at index 4
        # Split the 'Conditions' field by pipe ('|') and return as a list
        return [(field,) for field in conditions.split('|')]
    else:
        return []

# Step 2: Use flatMap to split and explode the 'Conditions' column
exploded_conditions_rdd = clinicaltrial2023RDD_no_header.flatMap(split_conditions)

# Display the results (you can also perform other operations such as counting, filtering, etc.)
for condition in exploded_conditions_rdd.take(20):
    print(condition)
```

```
('Mental Health Issue (E.G.", Depression, Psychosis, Personality Disorder," Substance Abuse)',)
('Cerebrovascular Function',)
('Cognition',)
('Myocardial Infarct',)
('Hyperglycemia',)
('Decompensated Cirrhosis',)
('Liver Transplantation',)
('Hypertension',)
('Treatment Adherence and Compliance',)
('Digital Health',)
('Smoking',)
('Alcohol Drinking',)
```

```
# Step 1: Group the exploded RDD by 'Conditions' and count occurrences
condition_counts_rdd = exploded_conditions_rdd.map(lambda condition: (condition, 1)).reduceByKey(lambda a, b: a + b)

# Step 2: Sort the RDD by count in descending order
condition_counts_sorted_rdd = condition_counts_rdd.sortBy(lambda x: -x[1])

# Step 3: Take the top 5 conditions
top_5_conditions_rdd = condition_counts_sorted_rdd.take(5)

# Display the top 5 conditions with their frequencies
print("Top 5 conditions and their frequencies:")
for condition, count in top_5_conditions_rdd:
    print(f"Condition: {condition}, Count: {count}")
```

```
Top 5 conditions and their frequencies:
Condition: ('Healthy',), Count: 9731
Condition: ('Breast Cancer',), Count: 7502
Condition: ('Obesity',), Count: 6549
Condition: ('Stroke',), Count: 4071
Condition: ('Hypertension',), Count: 4020
```

Visualization



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

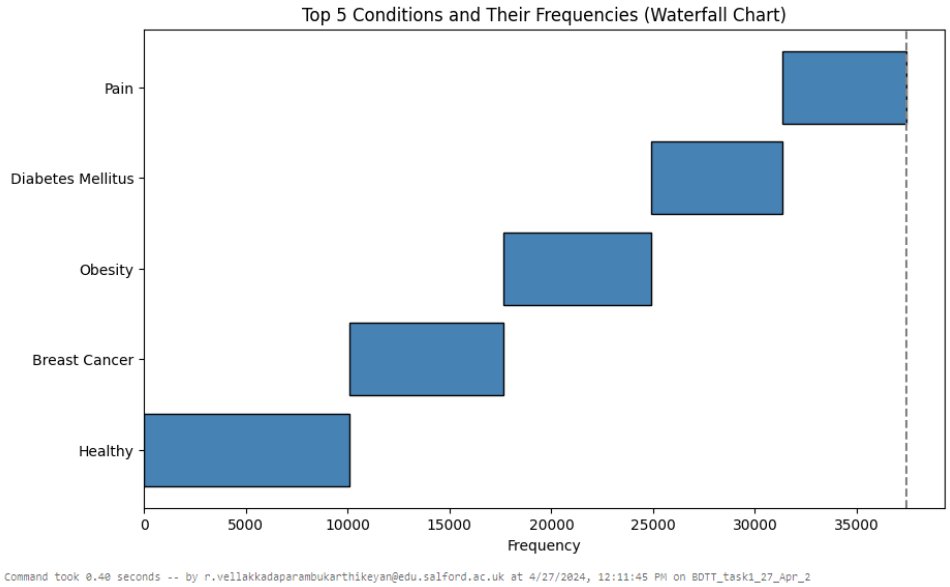
```
from pyspark import SparkContext
from pyspark.sql import SparkSession
import re

# Step 1: Define a function to clean and trim the 'Conditions' field
def clean_conditions(line):
    fields = line.split('\t')
    if len(fields) > 4:  # Ensure there are at least 5 fields
        conditions = fields[4]
        if conditions:
            # Remove quotes and trim leading and trailing whitespaces
            conditions = conditions.replace('"', '').strip()
            # Replace square brackets and parentheses
            conditions = re.sub(r'[\[\]\(\)]', '', conditions)
            # Split the conditions by pipe ('|') and flatten nested conditions
            conditions = re.split(r'[|,]', conditions)
            # Trim leading and trailing whitespaces from each condition
            conditions = [condition.strip() for condition in conditions]
            # Filter out conditions that contain 'e.g.' (case-insensitive)
            conditions = [condition for condition in conditions if 'e.g.' not in condition.lower()]
            return conditions
    return []

# Create an RDD from the cleaned data (assuming clinicaltrial2023RDD_no_header is the original RDD without header)
cleaned_conditions_rdd = clinicaltrial2023RDD_no_header.flatMap(clean_conditions)

# Now, you have a cleaned RDD of conditions after all the filtering and transformations
```

```
# Step 10: Calculate the top 5 conditions with their frequencies in RDD
# First, map each condition to a key-value pair (condition, 1)
condition_counts_rdd = cleaned_conditions_rdd.map(lambda condition: (condition, 1))

# Next, reduce by key to count occurrences of each condition
condition_counts_rdd = condition_counts_rdd.reduceByKey(lambda a, b: a + b)

# Sort the RDD by count in descending order
sorted_condition_counts_rdd = condition_counts_rdd.sortBy(lambda x: -x[1])

# Take the top 5 conditions with the highest frequencies
top_5_conditions_rdd = sorted_condition_counts_rdd.take(5)

# Display the top 5 conditions with their frequencies
print("Top 5 conditions and their frequencies:")
for condition, count in top_5_conditions_rdd:
    print(f"Condition: {condition}, Count: {count}")
```

(3) Spark Jobs

```
Top 5 conditions and their frequencies:
Condition: Healthy, Count: 10102
Condition: Breast Cancer, Count: 7564
Condition: Obesity, Count: 7257
Condition: Diabetes Mellitus, Count: 6438
Condition: Pain, Count: 6103
```

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

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

```sql
WITH exploded_conditions AS (
    -- Step 1: Split the 'Conditions' column by pipe and explode
    SELECT EXPLODE(SPLIT(Conditions, '\\|')) AS Condition
    FROM clinicaltrial_table
),

condition_counts AS (
    -- Step 2: Group the exploded data by 'Condition' and count occurrences
    SELECT Condition, COUNT(*) AS count
    FROM exploded_conditions
    GROUP BY Condition
),

sorted_condition_counts AS (
    -- Step 3: Order the condition counts by count in descending order
    SELECT Condition, count
    FROM condition_counts
    ORDER BY count DESC
)

-- Step 4: Select the top 5 conditions with their frequencies
SELECT *
FROM sorted_condition_counts
LIMIT 5;
```

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 BDTT_TASK1_SQL_1

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

```sql
WITH cleaned_conditions AS (
    -- Step 1: Remove quotes and trim leading and trailing spaces from 'Conditions' column
    SELECT TRIM(REGEXP_REPLACE(Conditions, '"', '')) AS Condition
    FROM clinicaltrial_table
),
no_brackets_conditions AS (
    -- Step 2: Replace square brackets ('[' and ']') and parentheses ('(' and ')') in 'Condition' column
    SELECT REGEXP_REPLACE(Condition, '[\\[\\](){}]', '') AS condition
    FROM cleaned_conditions
),
split_conditions AS (
    -- Step 3 and 4: Split 'Condition' column by pipe ('|') and explode the array
    SELECT EXPLODE(SPLIT(Condition, '\\|')) AS Condition
    FROM no_brackets_conditions
),
split_by_comma_conditions AS (
    -- Step 5 and 6: Split each condition by comma and explode the array
    SELECT EXPLODE(SPLIT(condition, ',')) AS condition
    FROM split_conditions
),
trimmed_conditions AS (
    -- Step 7: Trim leading and trailing whitespaces from 'Condition' column
    SELECT TRIM(condition) AS Condition
    FROM split_by_comma_conditions
),
filtered_conditions AS (
    -- Step 8: Filter out rows with 'e.g' in a case-insensitive manner
    SELECT Condition
    FROM trimmed_conditions
    WHERE LOWER(Condition) NOT LIKE 'e.g%'
),
top_5_conditions AS (
    -- Calculate the top 5 conditions with their frequencies
    SELECT condition, COUNT(*) AS frequency
    FROM filtered_conditions
    GROUP BY condition
    ORDER BY frequency DESC
    LIMIT 5
)
-- Display the top 5 conditions with their frequencies
SELECT condition, frequency
FROM top_5_conditions;
```

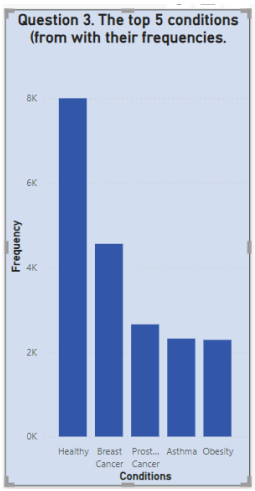
Table

	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_1

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

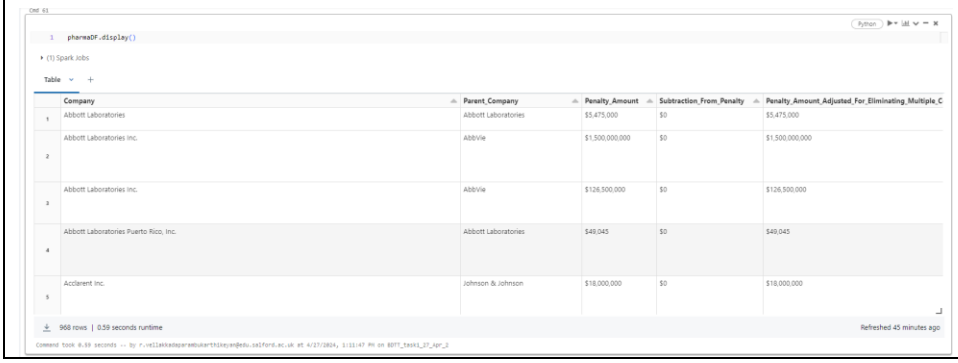
Table 3 (Splitting the conditions column, by taking the e.g conditions out of bracket)	
Conditions	**Count**

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



```python
from pyspark.sql.functions import count, desc
# From parent company column, extracting the list of all pharmaceutical companies
pharmaceutical_companies = [row[0] for row in pharmaDF.select('Parent_Company').distinct().collect()]

# Using clinical trial Sponsor column, filtering out all non pharmaceutical companies
non_pharmaceutical_sponsors = clinicaltrial2023DF.filter(~clinicaltrial2023DF['Sponsor'].isin(pharmaceutical_companies))

# Count the number of clinical trials sponsored by each sponsor and sort in descending order
tenTopSponsors = non_pharmaceutical_sponsors.groupBy('Sponsor') \
                                            .agg(count("*").alias("numberOfTrials")) \
                                            .sort(desc("numberOfTrials"))

# Show the top 10 sponsors
tenTopSponsors.show(10, truncate = False)
```

▸ (4) Spark Jobs

▸ non_pharmaceutical_sponsors: pyspark.sql.dataframe.DataFrame = [Id: string, Study Title: string ... 12 more fields]

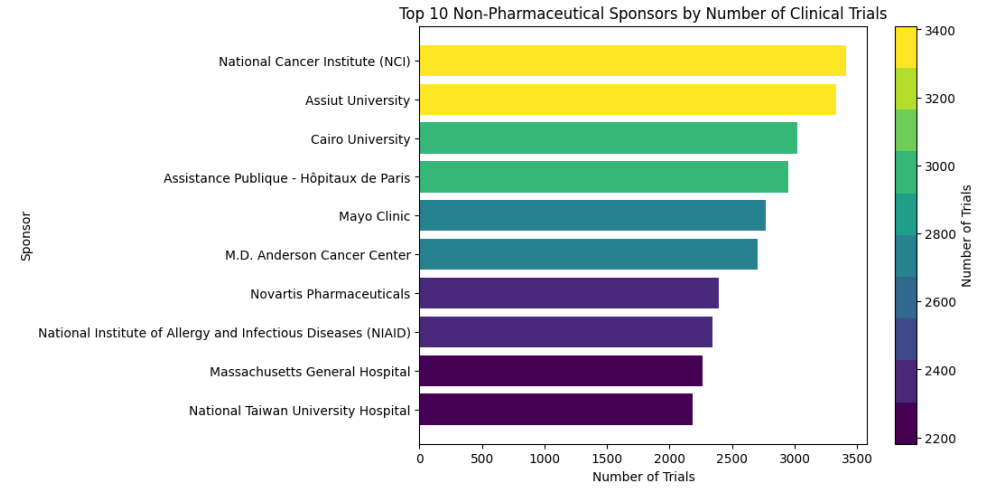
▸ tenTopSponsors: pyspark.sql.dataframe.DataFrame = [Sponsor: string, numberOfTrials: long]

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	2702
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 9.12 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 1:58:40 PM on BDTT_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 BDTT_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

```python
# Step 1: Extract the "Sponsor" column from the RDD
sponsor2023RDD = clinicaltrial2023RDD_no_header.map(lambda line: line.split('\t')[6])

# Step 2: Take the first 5 sponsor rows
first_5_sponsors_rdd = sponsor2023RDD.take(5)

# Step 3: Print the first 5 sponsors
print("First 5 sponsors:")
for sponsor in first_5_sponsors_rdd:
    print(sponsor)
```

(1) Spark Jobs

```
First 5 sponsors:
Sangath
McMaster University
Melbourne Health
Assistance Publique - Hôpitaux de Paris
Istituto Auxologico Italiano
```

Command took 1.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

Cmd 60

```python
# Read the CSV file and create an RDD
import re
pharma_RDD = sc.textFile("/FileStore/tables/" + filerootl + "/" + filerootl + ".csv")
header = pharma_RDD.first()
pharma_RDD = pharma_RDD.filter(lambda row: row != header)

# Split each row based on commas by excluding commas enclosed within double quotes
pharma_RDD = pharma_RDD.map(lambda x: re.split(',(?=(?:[^"]*"[^"]*")*[^"]*$)', x))
```

(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

Cmd 62

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

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

```python
pharma_Parent_RDD.take(10)
```

(1) Spark Jobs

```
['"Abbott Laboratories"',
 '"AbbVie"',
 '"AbbVie"',
 '"Abbott Laboratories"',
 '"Johnson & Johnson"',
 '"Abbott Laboratories"',
 '"Abbott Laboratories"',
 '"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

```python
pharma_companies_group = set(pharma_Parent_RDD.distinct().collect())
```

(1) Spark Jobs

Command took 0.70 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

Cmd 65

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

```
{'Novo Holdings A/S', 'Perrigo', 'Daiichi Sankyo', 'IDEXX Laboratories', 'GlaxoSmithKline', 'CytRx', 'Amphastar Pharmaceuticals', 'SciClone Pharmaceuticals', 'Currax Pharmaceuticals', 'Bausch Health', 'Bristol
-Myers Squibb', 'Zoetis', 'Mallinckrodt', 'Gilead Sciences', 'CSL Limited', 'Clovis Oncology', 'Novartis', 'Johnson & Johnson', 'SELLAS Life Sciences Group', 'Viatris', 'AVEO Pharmaceuticals', 'Elanco Animal H
ealth', 'Pfizer', 'Wockhardt Limited', 'Phibro Animal Health', 'Otsuka Pharmaceutical', 'Jazz Pharmaceuticals', 'Roche', 'Hikma Pharmaceuticals', 'Pacira BioSciences', 'Merck', 'Indivior Inc.', 'Sun Pharmaceut
icals', 'Purdue Pharma', 'Akorn Inc.', 'Biocryst Pharmaceuticals', 'AstraZeneca', 'Amneal Pharmaceuticals', 'United Therapeutics', 'Parexel International', 'Eisai', 'Lannett Co.', 'Apotex Corp.', 'Incyte Cor
p.', 'Organon & Co.', 'Takeda Pharmaceutical', 'Amgen', 'Anika Therapeutics', 'Astellas Pharma', 'Aceto', 'Endo International', 'Alcami', 'Abbott Laboratories', 'Boehringer Ingelheim', 'Merck KGaA (EMD)', 'Cur
ia Inc.', "Dr. Reddy's Laboratories", 'Lundbeck', 'Nutraceutical International Corp.', 'Cambrex', 'Biogen Idec', 'Sanofi', 'Insys Therapeutics', 'KV Pharmaceutical', 'Eli Lilly', 'Amryt Pharma', 'Regeneron Pha
rmaceuticals', 'UCB', 'Taro Pharmaceutical Industries', 'Arbor Pharmaceuticals', 'Teva Pharmaceutical Industries', 'AbbVie'}
```

Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

Cmd 67

```python
# filter out all the sponsors that are not pharmaceutical companies
sponsor_non_pharma_RDD = sponsor2023RDD.filter(lambda x: x not in pharma_Parent_RDD)

# count the number of clinical trials sponsored by each non-pharmaceutical company
sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

# get the top 10 non-pharma sponsors by number of clinical trials sponsored
sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])

# display the corresponding results
for sponsor, count in sponsor_non_pharma_top10_RDD:
    print(sponsor, count)
```

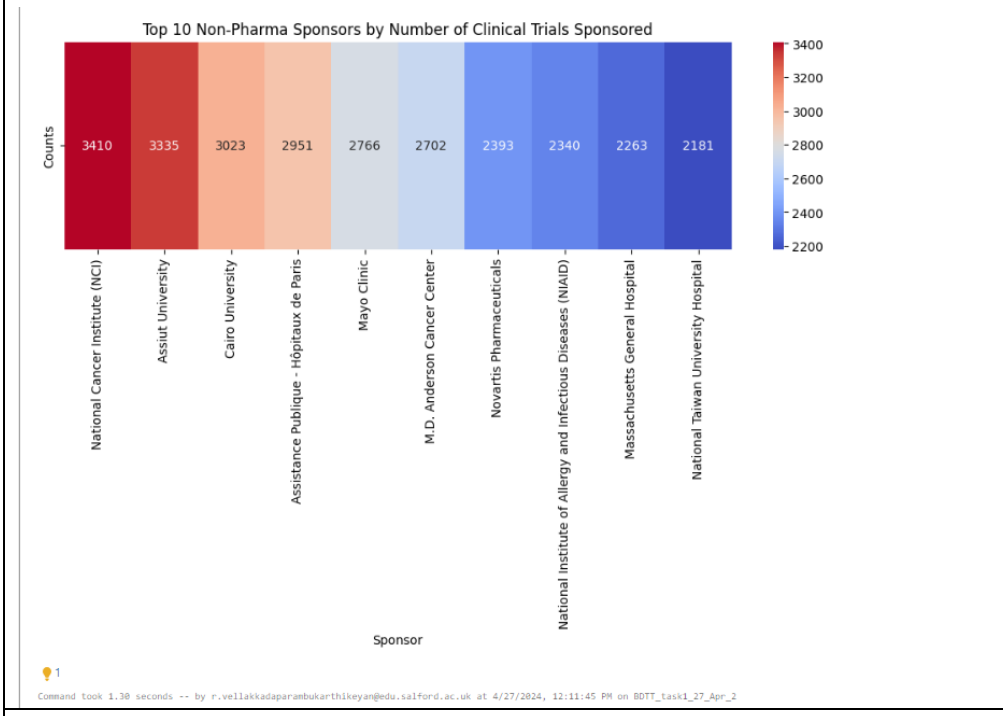
(1) Spark Jobs

```
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
```

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

```
Cmd 68

# Filter out all the sponsors that are not pharmaceutical companies
sponsor_non_pharma_RDD = sponsor2023RDD.filter(lambda x: x not in pharma_Parent_RDD)

# Count the number of clinical trials sponsored by each non-pharmaceutical company
sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

# Get the top 10 non-pharma sponsors by number of clinical trials sponsored
sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])

# Display the corresponding results with header
print(f"{trial_Year}:")
print("+----------------------------------------+-----+")
print("|{:40s}|{:5s}|".format("Sponsor", "count"))
print("+----------------------------------------+-----+")
for sponsor, count in sponsor_non_pharma_top10_RDD:
    print("|{:40s}|{:5d}|".format(sponsor, count))
print("+----------------------------------------+-----+")

▸ (1) Spark Jobs

2023:
+----------------------------------------+-----+
|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|
+----------------------------------------+-----+

Command took 16.29 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



1

Command took 1.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

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

```sql
-- Step 1: Retrieve a list of pharmaceutical companies from pharma_table
WITH pharma_companies AS (
    SELECT DISTINCT Parent_Company
    FROM pharma_table
)

-- Step 2: Identify non-pharmaceutical sponsors and count the number of clinical trials they have sponsored
SELECT c.Sponsor, COUNT(*) AS num_trials
FROM clinicaltrial_table AS c
-- Step 3: Left join with pharma_companies to filter out pharmaceutical sponsors
LEFT JOIN pharma_companies AS p ON c.Sponsor = p.Parent_Company
-- Step 4: Only keep non-pharmaceutical sponsors (i.e., sponsors not in the pharma_companies list)
WHERE p.Parent_Company IS NULL
-- Step 5: Group by sponsor and count the number of clinical trials they have sponsored
GROUP BY c.Sponsor
-- Step 6: Order by the count in descending order and limit the results to the top 10
ORDER BY num_trials DESC
LIMIT 10;
```

(5) Spark Jobs

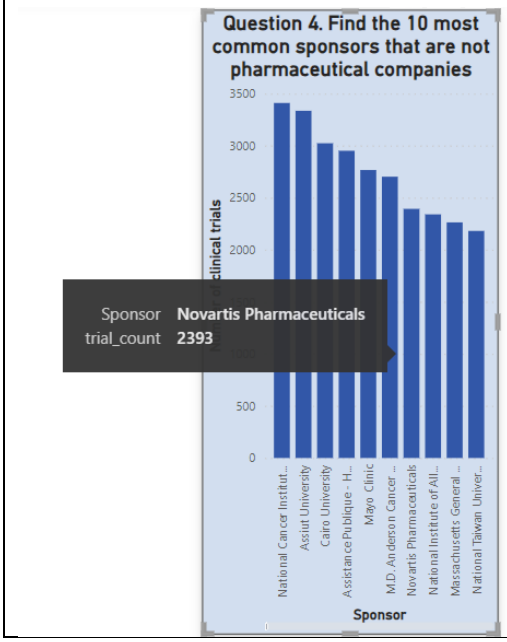
Table

	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 BDTT_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.

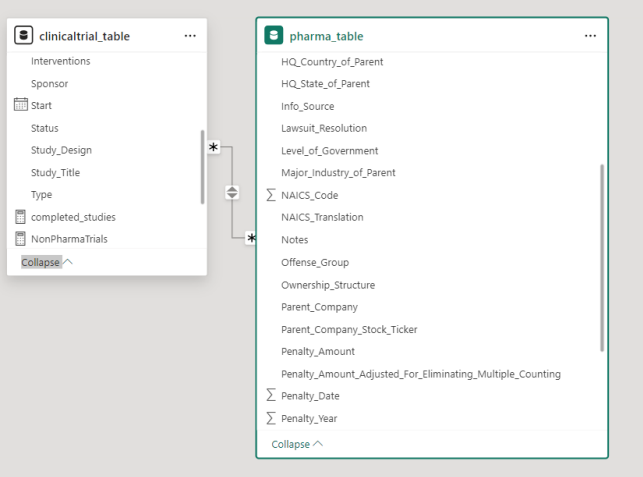


Table 1.4 Discussion of Results

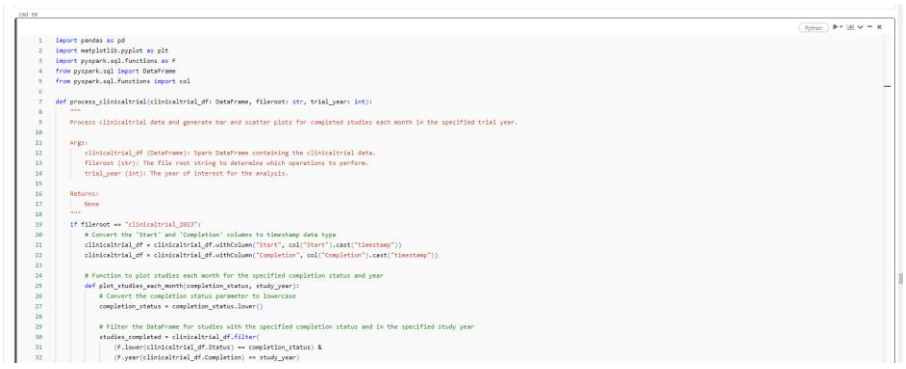
Result	
Sponsor	**count**
National Cancer Institute (NCI)	3410
Assiut University	3335
Cairo University	3023
Assistance Publique - Hôpitaux de Paris	2951
Mayo Clinic	2766

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

Insights: The result demonstrates the significant role of academic and government institutions in sponsoring clinical trials. This insight highlights the importance of collaboration between healthcare institutions, research centers, and universities in advancing medical research and patient care.

1.10 Plot number of completed studies for each month in 2023

Dataframe approach



```
import pandas as pd
import matplotlib.pyplot as plt
import pyspark.sql.functions as f
from pyspark.sql import DataFrame
from pyspark.sql.functions import col

def process_clinicaltrial(clinicaltrial_df: DataFrame, fileroot: str, trial_year: int):
    """
    Process clinicaltrial data and generate bar and scatter plots for completed studies each month in the specified trial year.

    Args:
        clinicaltrial_df (DataFrame): Spark DataFrame containing the clinicaltrial data.
        fileroot (str): The file root string to determine which operations to perform.
        trial_year (int): The year of interest for the analysis.

    Returns:
        None
    """
    if fileroot == "clinicaltrial_2023":
        # Convert the 'Start' and 'Completion' columns to timestamp data type
        clinicaltrial_df = clinicaltrial_df.withColumn("Start", col("Start").cast("timestamp"))
        clinicaltrial_df = clinicaltrial_df.withColumn("Completion", col("Completion").cast("timestamp"))

        # Function to plot studies each month for the specified completion status and year
        def plot_studies_each_month(completion_status, study_year):
            # Convert the completion status parameter to lowercase
            completion_status = completion_status.lower()

            # Filter the DataFrame for studies with the specified completion status and in the specified study year
            studies_completed = clinicaltrial_df.filter(
                (F.lower(clinicaltrial_df.Status) == completion_status) &
                (F.year(clinicaltrial_df.Completion) == study_year)
            )

            # Extract the month from the 'Completion' column
            studies_completed = studies_completed.withColumn(
                "month", F.month(clinicaltrial_df.Completion)
            )

            # Group by the extracted month and count the number of studies
            monthly_basis_results = studies_completed.groupBy("month").count()

            # Display the count of completed studies for each month, ordered by month
            monthly_basis_results.orderBy("month").show()

            # Collect the results to lists for plotting
            month_number = [row.month for row in monthly_basis_results.collect()]
            no_of_completed_studies = [row["count"] for row in monthly_basis_results.collect()]

            return month_number, no_of_completed_studies

        # Call the function with the desired completion status and study year
        month_number, no_of_completed_studies = plot_studies_each_month("Completed", trial_year)

        # Create a bar graph
        plt.figure(figsize=(10, 6))
        bars = plt.bar(month_number, no_of_completed_studies, color='blue')

        # Add counts on top of each bar
        for bar in bars:
            y_val = bar.get_height()  # Get the height of the bar (count)
            plt.text(
                bar.get_x() + bar.get_width() / 2,  # Center the text over the bar
```

```
        no_of_completed_studies = [row["count"] for row in monthly_basis_results.collect()]

        return month_number, no_of_completed_studies

    # Call the function with the desired completion status and study year
    month_number, no_of_completed_studies = plot_studies_each_month("Completed", trial_year)

    # Create a bar graph
    plt.figure(figsize=(10, 6))
    bars = plt.bar(month_number, no_of_completed_studies, color='blue')

    # Add counts on top of each bar
    for bar in bars:
        y_val = bar.get_height()  # Get the height of the bar (count)
        plt.text(
            bar.get_x() + bar.get_width() / 2,  # Center the text over the bar
            y_val,  # Place the text at the top of the bar
            str(y_val),  # Convert count to string
            ha='center',  # Center-align the text
            va='bottom'   # Place the text at the bottom (above the bar)
        )

    # Add labels and title
    plt.xlabel('Month')
    plt.ylabel('Number of Completed Studies')
    plt.title(f'Bar Graph of Completed Studies Each Month in {trial_year}')

    # Display the bar plot
    plt.show()

    # Create a scatter plot
    plt.figure(figsize=(10, 6))
    plt.scatter(month_number, no_of_completed_studies, color='blue', marker='o')
```

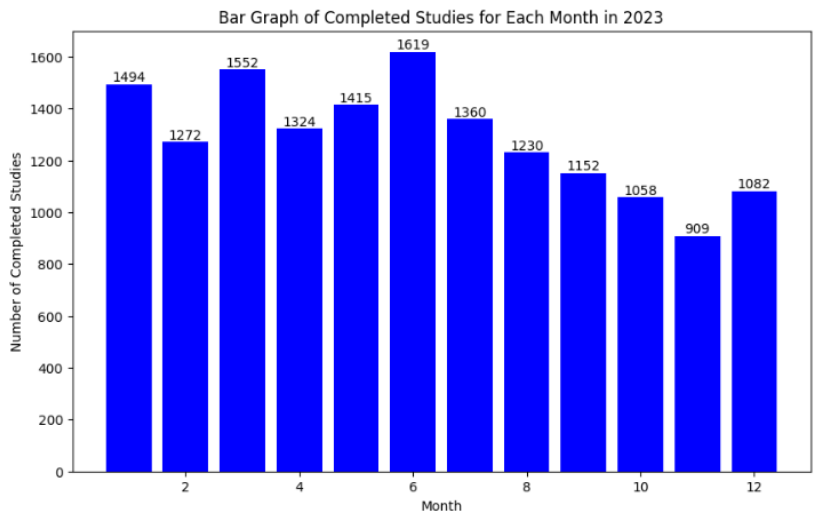
```
    # Display the scatter plot
    plt.show()

elif fileroot in ["clinicaltrial_2020", "clinicaltrial_2021"]:
    # Filter the DataFrame to only include completed studies in the specified trial year
    completed_studies_df = clinicaltrial_df.filter(
        (F.col("Status") == "Completed") &
        (F.year(F.to_date(F.col("Completion"), "MMM yyyy")) == trial_year)
    )

    # Group the data by month and count the number of completed studies
    monthly_completed_studies_df = completed_studies_df.withColumn(
        "month_year",
        F.date_format(F.to_date(F.col("Completion"), "MMM yyyy"), "yyyy-MM")
    ).groupBy("month_year").count()

    # Order the DataFrame by 'month_year' in ascending order
    monthly_completed_studies_df = monthly_completed_studies_df.orderBy("month_year", ascending=True)

    # Show the monthly completed studies in ascending order of months
    monthly_completed_studies_df.show()

    # Convert the DataFrame to a Pandas DataFrame
    monthly_completed_studies_pd_df = monthly_completed_studies_df.toPandas()

    # Plotting the data
    plt.figure(figsize=(10, 6))
    plt.plot(monthly_completed_studies_pd_df['month_year'], monthly_completed_studies_pd_df['count'], marker='o')
    plt.title(f'Number of Completed Studies Each Month in {trial_year}')
    plt.xlabel('Month')
    plt.ylabel('Number of Completed Studies')
    plt.xticks(rotation=45)  # Rotate the x-axis labels for better readability
    plt.grid(True)  # Add grid lines
    plt.show()  # Display the plot


process_clinicaltrial(clinicaltrial_df, fileroot, trial_Year)
```

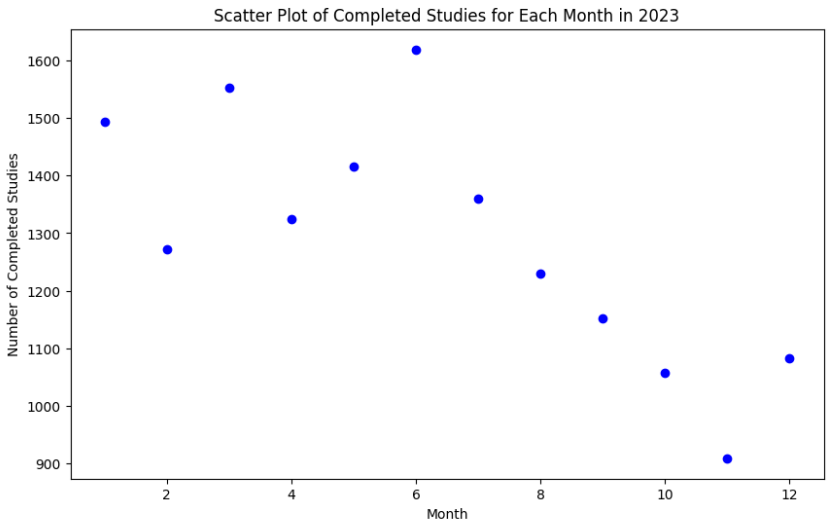
```
122
123  process_clinicaltrial(clinicaltrial_df, fileroot, trial_Year)
```

▸ (5) Spark Jobs

```
+-----+-----+
|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|
+-----+-----+
```

Visualizing the answer





1

Command took 28.19 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 2:21:30 PM on BDTT_task1_27_Apr_2

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

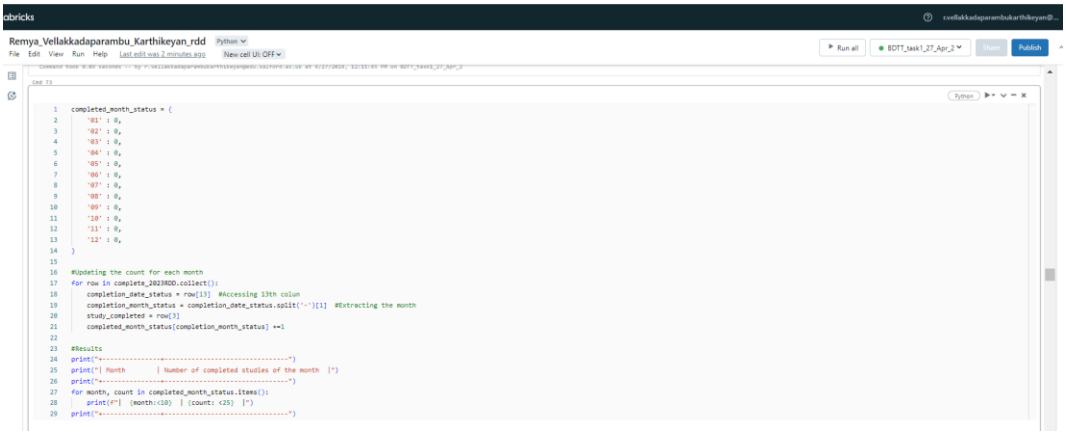
```
split_rdd = clinicaltrial2023RDD.map(lambda line: line.split('\t'))
split_rdd.take(2)
```

```
(1) Spark Jobs
['Collaborators',
 'Enrollment',
 'Funder Type',
 'Type',
 'Study Design',
 'Start',
 'Completion'],
['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)',
 "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',
 '2019-03-26']
```

Command took 1.28 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

```
#filtering to include status and completion year
complete_2023RDD = split_rdd.filter(lambda row:len(row)>13 and row[3] == 'COMPLETED' and row[13].startswith('2023'))
```

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



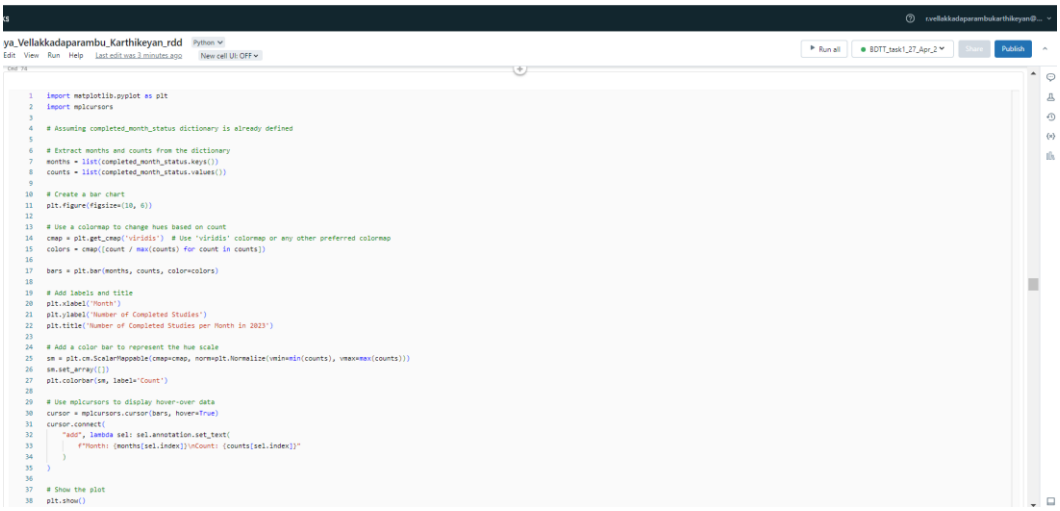
```
completed_month_status = {
    '01' : 0,
    '02' : 0,
    '03' : 0,
    '04' : 0,
    '05' : 0,
    '06' : 0,
    '07' : 0,
    '08' : 0,
    '09' : 0,
    '10' : 0,
    '11' : 0,
    '12' : 0,
}

#Updating the count for each month
for row in complete_2023RDD.collect():
    completion_date_status = row[13]  #Accessing 13th colun
    completion_month_status = completion_date_status.split('-')[1]  #Extracting the month
    study_completed = row[3]
    completed_month_status[completion_month_status] +=1

#Results
print("+---------------+---------------------------------+")
print("| Month         | Number of completed studies of the month  |")
print("+---------------+---------------------------------+")
for month, count in completed_month_status.items():
    print(f"|  {month:<10}  | {count: <25}  |")
print("+---------------+---------------------------------+")
```

▸ (1) Spark Jobs

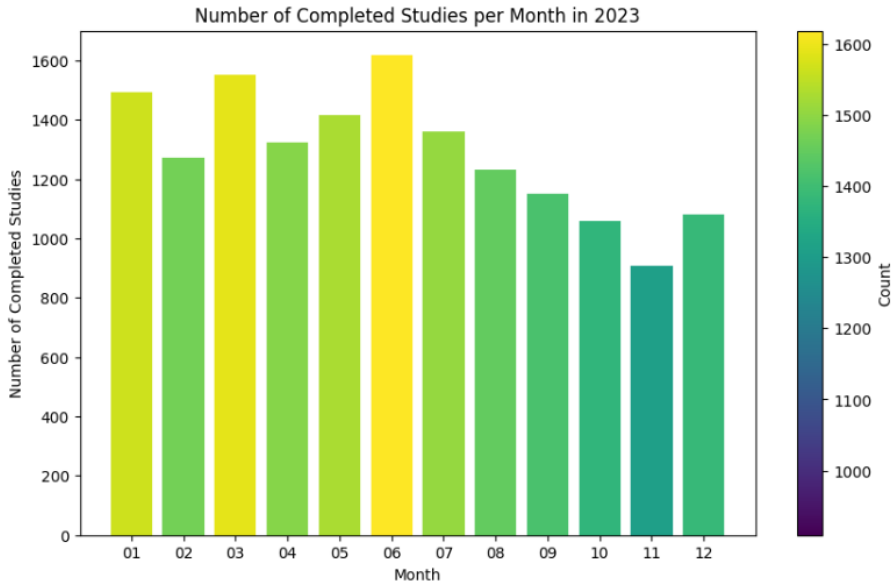
```
+---------------+---------------------------------+
| Month         | Number of completed studies of the month  |
+---------------+---------------------------------+
|  01           | 1494                            |
|  02           | 1272                            |
|  03           | 1552                            |
|  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

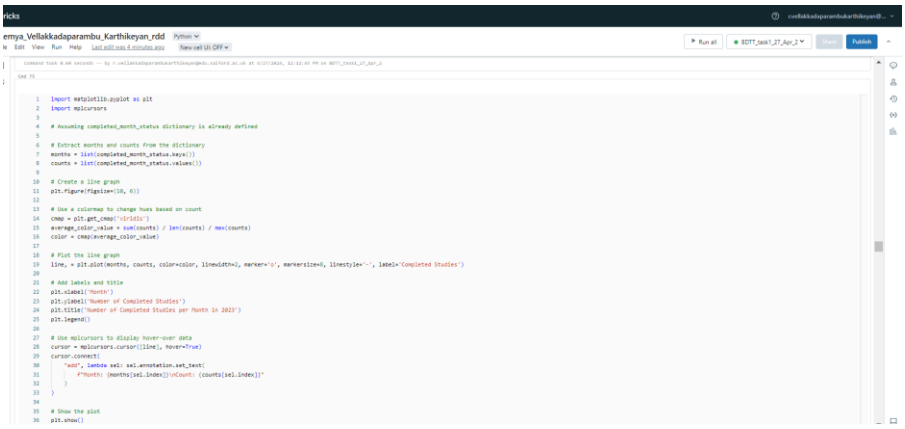
Visualizing the answer



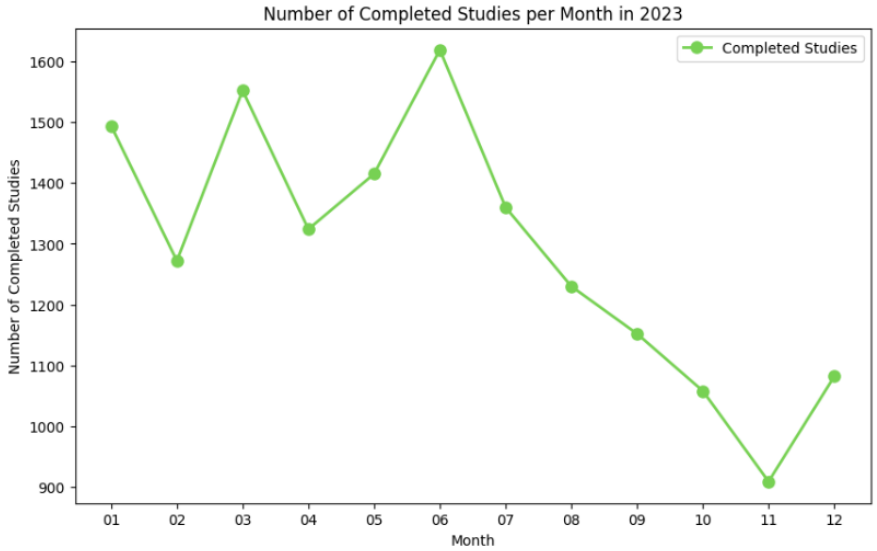
```
import matplotlib.pyplot as plt
import mplcursors

# Assuming completed_month_status dictionary is already defined

# Extract months and counts from the dictionary
months = list(completed_month_status.keys())
counts = list(completed_month_status.values())

# Create a bar chart
plt.figure(figsize=(10, 6))

# Use a colormap to change hues based on count
cmap = plt.get_cmap('viridis')  # Use 'viridis' colormap or any other preferred colormap
colors = cmap([count / max(counts) for count in counts])

bars = plt.bar(months, counts, color=colors)

# Add labels and title
plt.xlabel('Month')
plt.ylabel('Number of Completed Studies')
plt.title('Number of Completed Studies per Month in 2023')

# Add a color bar to represent the hue scale
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min(counts), vmax=max(counts)))
sm.set_array([])
plt.colorbar(sm, label='Count')

# Use mplcursors to display hover-over data
cursor = mplcursors.cursor(bars, hover=True)
cursor.connect(
    "add", lambda sel: sel.annotation.set_text(
        f"Month: {months[sel.index]}\nCount: {counts[sel.index]}"
    )
)

# Show the plot
plt.show()
```



Command took 0.60 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2



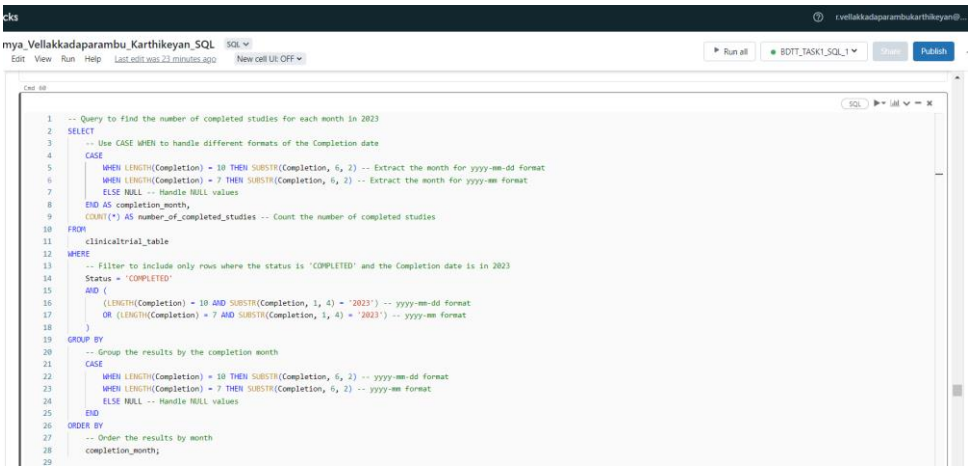
```
import matplotlib.pyplot as plt
import mplcursors

# Assuming completed_month_status dictionary is already defined

# Extract months and counts from the dictionary
months = list(completed_month_status.keys())
counts = list(completed_month_status.values())

# Create a line graph
plt.figure(figsize=(10, 6))

# Use a colormap to change hues based on count
cmap = plt.get_cmap('viridis')
average_color_value = sum(counts) / len(counts) / max(counts)
color = cmap(average_color_value)

# Plot the line graph
line, = plt.plot(months, counts, color=color, linewidth=2, marker='o', markersize=8, linestyle='-', label='Completed Studies')

# Add labels and title
plt.xlabel('Month')
plt.ylabel('Number of Completed Studies')
plt.title('Number of Completed Studies per Month in 2023')
plt.legend()

# Use mplcursors to display hover-over data
cursor = mplcursors.cursor(line, hover=True)
cursor.connect(
    "add", lambda sel: sel.annotation.set_text(
        f"Month: {months[sel.index]}\nCount: {counts[sel.index]}"
    )
)

# Show the plot
plt.show()
```



Command took 0.59 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2

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



```sql
-- Query to find the number of completed studies for each month in 2023
SELECT
    -- Use CASE WHEN to handle different formats of the Completion date
    CASE
        WHEN LENGTH(Completion) = 10 THEN SUBSTR(Completion, 6, 2) -- Extract the month for yyyy-mm-dd format
        WHEN LENGTH(Completion) = 7 THEN SUBSTR(Completion, 6, 2) -- Extract the month for yyyy-mm format
        ELSE NULL -- Handle NULL values
    END AS completion_month,
    COUNT(*) AS number_of_completed_studies -- Count the number of completed studies
FROM
    clinicaltrial_table
WHERE
    -- Filter to include only rows where the status is 'COMPLETED' and the Completion date is in 2023
    Status = 'COMPLETED'
    AND (
        (LENGTH(Completion) = 10 AND SUBSTR(Completion, 1, 4) = '2023') -- yyyy-mm-dd format
        OR (LENGTH(Completion) = 7 AND SUBSTR(Completion, 1, 4) = '2023') -- yyyy-mm format
    )
GROUP BY
    -- Group the results by the completion month
    CASE
        WHEN LENGTH(Completion) = 10 THEN SUBSTR(Completion, 6, 2) -- yyyy-mm-dd format
        WHEN LENGTH(Completion) = 7 THEN SUBSTR(Completion, 6, 2) -- yyyy-mm format
        ELSE NULL -- Handle NULL values
    END
ORDER BY
    -- Order the results by month
    completion_month;
```

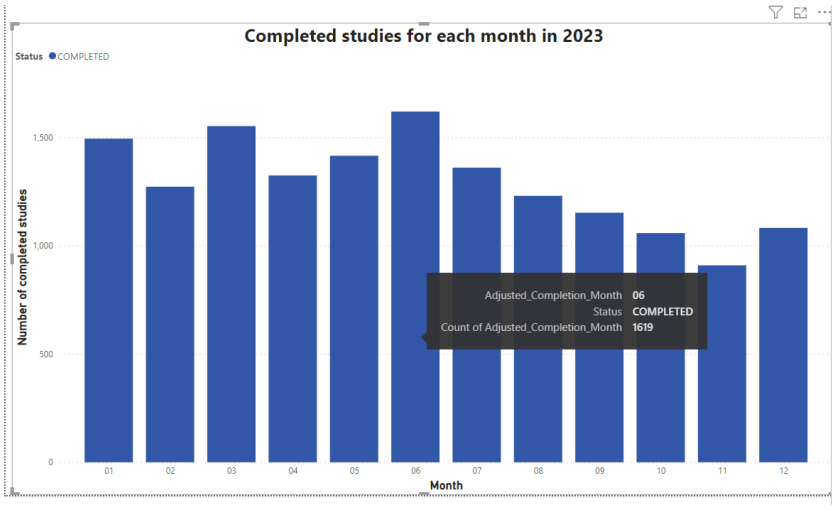
Table

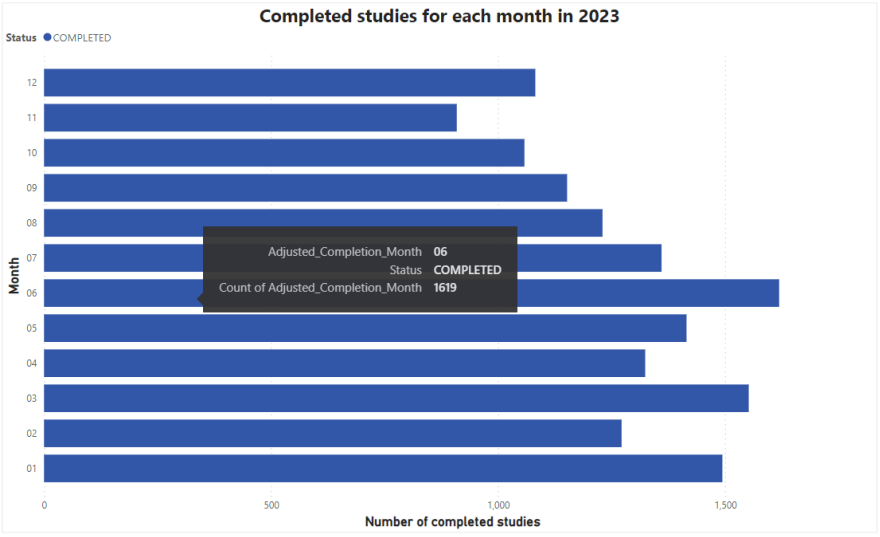
	completion_month	number_of_completed_studies
1	01	1494
2	02	1272
3	03	1552
4	04	1324
5	05	1415
6	06	1619
7	07	1360
8	08	1230
9	09	1152
10	10	1058
11	11	909
12	12	1082

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 BDTT_TASK1_SQL_1

Visualizing the answer in POWER BI



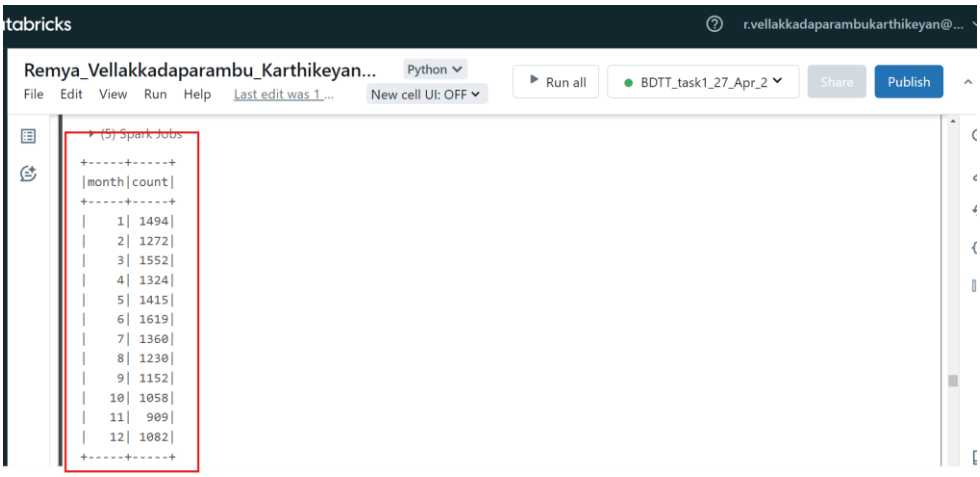


Main Idea: The visualization in Power BI utilized the following calculated columns

```
Adjusted_Completion =
    IF(
        ISBLANK(clinicaltrial_table[Completion]),
        "0000-00-00",
        IF(
            LEN(clinicaltrial_table[Completion]) = 7,
            clinicaltrial_table[Completion] & "-01",
            clinicaltrial_table[Completion]
        )
    )

```

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

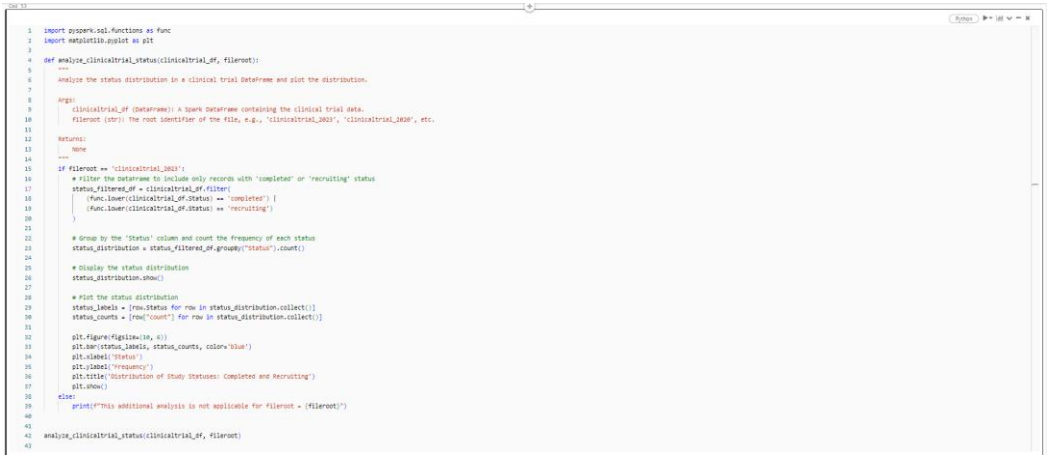
Table 1.5 Discussion of Results

Result

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.

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



```
import pyspark.sql.functions as func
import matplotlib.pyplot as plt

def analyze_clinicaltrial_status(clinicaltrial_df, fileroot):
    """
    Analyze the status distribution in a clinical trial DataFrame and plot the distribution.

    Args:
        clinicaltrial_df (DataFrame): A Spark DataFrame containing the clinical trial data.
        fileroot (str): The root identifier of the file, e.g., 'clinicaltrial_2023', 'clinicaltrial_2020', etc.

    Returns:
        None
    """
    if fileroot == 'clinicaltrial_2023':
        # Filter the DataFrame to include only records with 'completed' or 'recruiting' status
        status_filtered_df = clinicaltrial_df.filter(
            (func.lower(clinicaltrial_df.Status) == 'completed') |
            (func.lower(clinicaltrial_df.Status) == 'recruiting')
        )

        # Group by the 'Status' column and count the frequency of each status
        status_distribution = status_filtered_df.groupBy("Status").count()

        # Display the status distribution
        status_distribution.show()

        # Plot the status distribution
        status_labels = [row.Status for row in status_distribution.collect()]
        status_counts = [row["count"] for row in status_distribution.collect()]

        plt.figure(figsize=(10, 6))
        plt.bar(status_labels, status_counts, color='blue')
        plt.xlabel('Status')
        plt.ylabel('Frequency')
        plt.title('Distribution of Study Statuses: Completed and Recruiting')
        plt.show()
    else:
        print(f"This additional analysis is not applicable for fileroot = {fileroot}")

analyze_clinicaltrial_status(clinicaltrial_df, fileroot)
```

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.

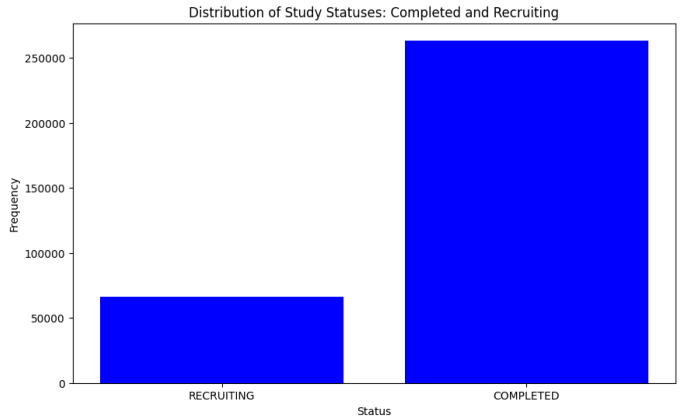
Discussion of Result:

```
▸ (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 BDTT_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:

▸ (3) Spark Jobs



Command took 7.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 2:21:31 PM on BDTT_task1_27_Apr_2

Analysis 2:

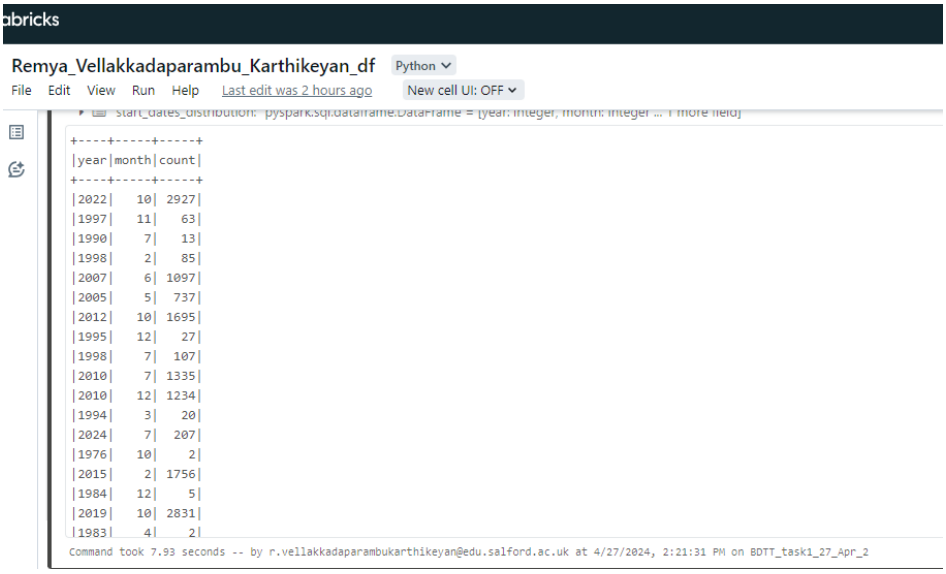
Question : Find the distribution of clinical trial start dates by year and month (2015-2023)

```python
import pyspark.sql.functions as func
import matplotlib.pyplot as plt
import seaborn as sns

def analyze_study_start_dates(clinicaltrial_df, fileroot):
    """
    Analyze and plot the distribution of study start dates in a clinical trial DataFrame.

    Args:
        clinicaltrial_df (DataFrame): A Spark DataFrame containing the clinical trial data.
        fileroot (str): The root identifier of the file, e.g., 'clinicaltrial_2023', 'clinicaltrial_2020', etc.

    Returns:
        None
    """
    if fileroot == 'clinicaltrial_2023':
        # Group the data by year and month of the 'Start' column
        start_dates_distribution = clinicaltrial_df.withColumn("year", func.year("Start")) \
            .withColumn("month", func.month("Start")) \
            .groupBy("year", "month") \
            .count()

        # Display the results
        start_dates_distribution.show()

        # Filter the data to include only the years from 2015 to 2023
        filtered_data = start_dates_distribution.filter(
            (start_dates_distribution.year >= 2015) & (start_dates_distribution.year <= 2023)
        )

        # Create a scatter plot
        plt.figure(figsize=(10, 6))

        # Create a color palette with 12 colors (one for each month)
        colors = sns.color_palette("hsl", 12)

        # Iterate through each month and plot a scatter plot with each month in different colors
        for month in range(1, 13):  # Months from 1 to 12
            # Filter the data for the current month
            monthly_data = filtered_data.filter(filtered_data.month == month)
```

```python
        # Filter the data to include only the years from 2015 to 2023
        filtered_data = start_dates_distribution.filter(
            (start_dates_distribution.year >= 2015) & (start_dates_distribution.year <= 2023)
        )

        # Create a scatter plot
        plt.figure(figsize=(10, 6))

        # Create a color palette with 12 colors (one for each month)
        colors = sns.color_palette("hsl", 12)

        # Iterate through each month and plot a scatter plot with each month in different colors
        for month in range(1, 13):  # Months from 1 to 12
            # Filter the data for the current month
            monthly_data = filtered_data.filter(filtered_data.month == month)

            # Extract year and count for the current month
            years = [row.year for row in monthly_data.collect()]
            counts = [row["count"] for row in monthly_data.collect()]

            # Plot the data points for the current month in a specific color
            plt.scatter(years, counts, color=colors[month - 1], label=f'Month {month}', alpha=0.7)

        # Add labels and title
        plt.xlabel('Year')
        plt.ylabel('Number of Studies Started')
        plt.title('Distribution of Study Start Dates from 2015 to 2023')

        # Add a legend with a reduced font size
        plt.legend(loc='upper left', fontsize='small')

        # Show the plot
        plt.show()
    else:
        print(f"This additional analysis is not applicable for fileroot = {fileroot}")


analyze_study_start_dates(clinicaltrial_df, fileroot)
```

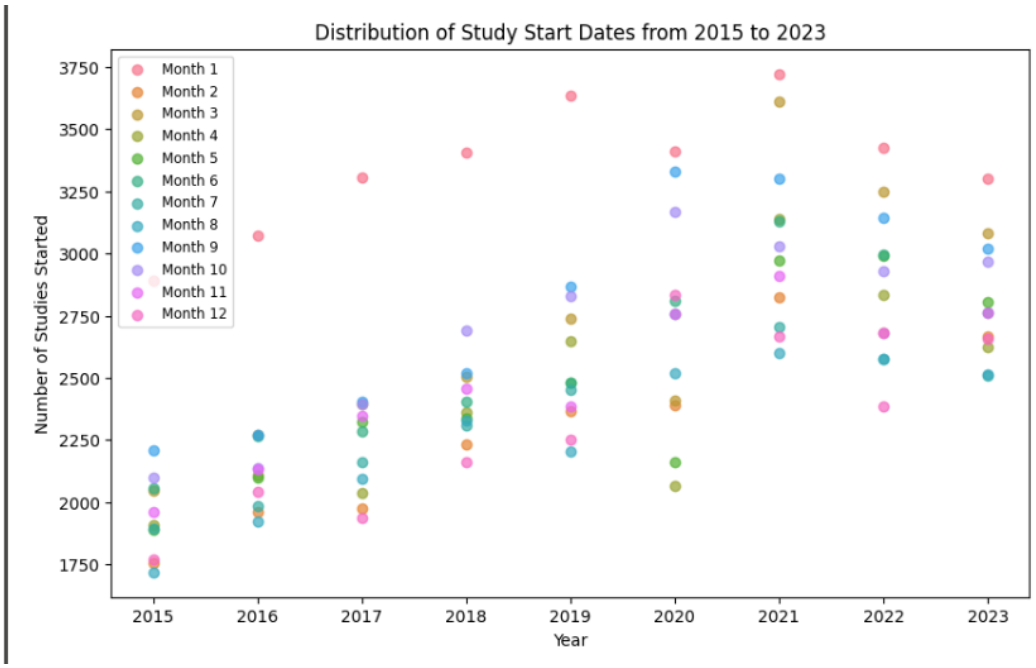
Main Idea: The code begins by extracting the year and month from the 'Start' column of the clinical trial data. Once the year and month columns have been created, the code groups the data based on these columns. After grouping the data, the code counts the number of clinical trials in each group. The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

Discussion of Result:



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?

```
import pyspark.sql.functions as func
import pandas as pd
import plotly.express as px

def analyze_interventions(clinicaltrial_df, fileroot):
    """
    Analyze the top 5 intervention types in a clinical trial DataFrame.

    Args:
        clinicaltrial_df (DataFrame): A Spark DataFrame containing the clinical trial data.
        fileroot (str): The root identifier of the file, e.g., 'clinicaltrial_2023', 'clinicaltrial_2020', etc.

    Returns:
        None
    """
    if fileroot == 'clinicaltrial_2023':
        # Step 1: Clean the 'Interventions' column by replacing all double quotes (") with no space
        cleaned_interventions_df = clinicaltrial_df.withColumn(
            "Cleaned_Interventions",
            func.regexp_replace(clinicaltrial_df["Interventions"], '"', '')
        )

        # Step 2: Extract the word before the colon (:)
        cleaned_interventions_df = cleaned_interventions_df.withColumn(
            "Intervention_Type",
            func.split(cleaned_interventions_df["Cleaned_Interventions"], ":").getItem(0)
        )

        # Step 3: Remove leading and trailing spaces from the intervention type
        cleaned_interventions_df = cleaned_interventions_df.withColumn(
            "Intervention_Type",
            func.trim(cleaned_interventions_df["Intervention_Type"])
        )

        # Step 4: Group by 'Intervention_Type' and count the occurrences, order by count in descending order
        intervention_counts = cleaned_interventions_df.groupBy("Intervention_Type").count().orderBy(func.desc("count"))
```

```
        )

        # Step 3: Remove leading and trailing spaces from the intervention type
        cleaned_interventions_df = cleaned_interventions_df.withColumn(
            "Intervention_Type",
            func.trim(cleaned_interventions_df["Intervention_Type"])
        )

        # Step 4: Group by 'Intervention_Type' and count the occurrences, order by count in descending order
        intervention_counts = cleaned_interventions_df.groupBy("Intervention_Type").count().orderBy(func.desc("count"))

        # Step 5: Display only the top 5 counts
        top_5_intervention_counts = intervention_counts.limit(5).collect()

        # Convert the top 5 intervention counts to a pandas DataFrame
        df = pd.DataFrame({
            'Intervention_Type': [row['Intervention_Type'] for row in top_5_intervention_counts],
            'Count': [row['count'] for row in top_5_intervention_counts]
        })

        # Create an interactive bar chart using Plotly
        fig = px.bar(
            df,
            x='Intervention_Type',
            y='Count',
            color='Count',
            color_continuous_scale='Reds',
            title='Top 5 Intervention Types and Their Counts'
        )

        # Customize the layout for better visibility
        fig.update_layout(
            xaxis_title='Intervention Types',
            yaxis_title='Count'
        )

        # Display the interactive plot
        fig.show()
    else:
        print(f"This additional analysis is not applicable for fileroot = {fileroot}")

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:

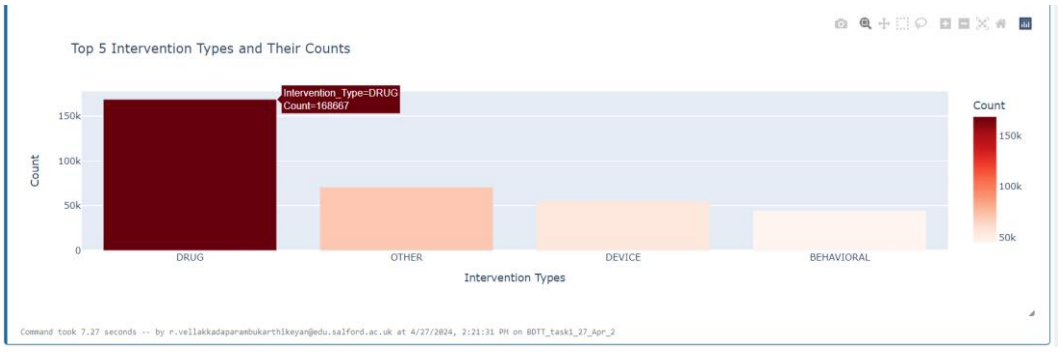
```
70
71   analyze_interventions(clinicaltrial_df, fileroot)
72
```

▸ (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:

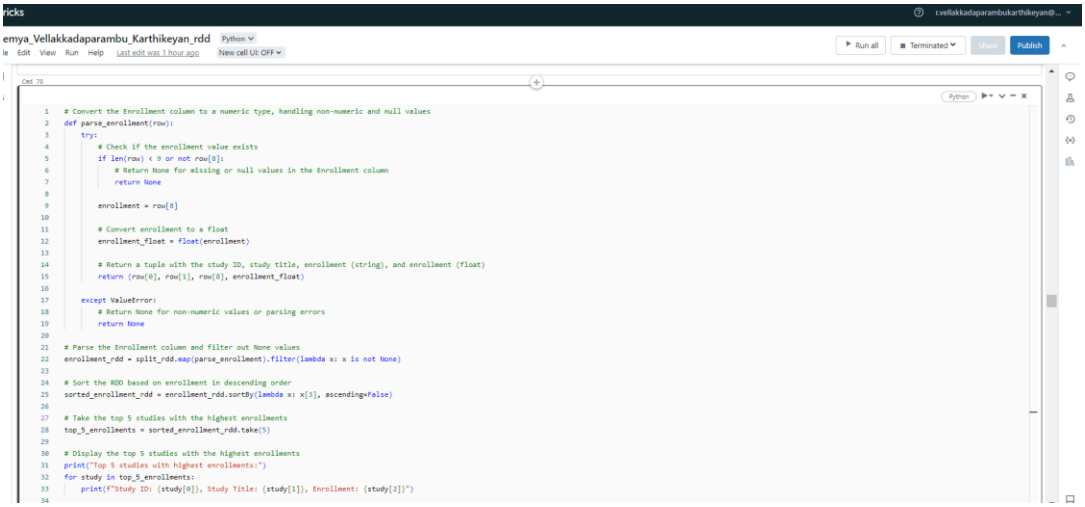


1.12 Additional Analyses and Visualization in RDD Approach

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) < 9 or not row[8]:
            # Return None for missing or null values in the Enrollment column
            return None

        enrollment = row[8]

        # 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[8], 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.

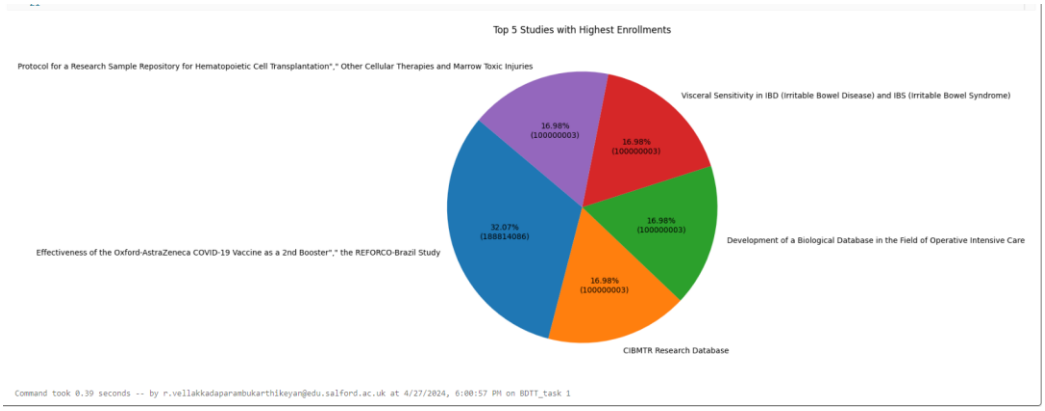
Discussion of Result:

```
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: NCT01166009, Study Title: CIBMTR 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
9999999.0
```

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:

```
Cmd 79

import matplotlib.pyplot as plt

# Extracting data from the top 5 studies with the highest enrollments
study_titles = [study[1] for study in top_5_enrollments]
enrollments = [study[3] for study in top_5_enrollments]

# Calculate the total enrollment of the top 5 studies
total_enrollment = sum(enrollments)

# Create a pie chart
plt.figure(figsize=(10, 8))

# Plotting the pie chart using enrollments as sizes and study titles as labels
plt.pie(enrollments, labels=study_titles, autopct=lambda p: f'{p:.2f}%\n({p * total_enrollment / 100:.0f})', startangle=140)

# Add a title
plt.title('Top 5 Studies with Highest Enrollments')

# Display the plot
plt.show()
```



Command took 0.39 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:57 PM on BDTT_task 1

Analysis 2:

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

```
Cmd 80

# Filter the RDD to include only studies sponsored by "Mayo Clinic"
Mayo_Clinic_studies = split_rdd.filter(lambda row: row[6].strip() == "Mayo Clinic")

# Count the number of studies that satisfy the condition
num_Mayo_Clinic_studies = Mayo_Clinic_studies.count()

# Print the result
print(f"Number of studies sponsored by Mayo Clinic: {num_Mayo_Clinic_studies}")

# Filter the RDD to include only studies sponsored by "Massachusetts General Hospital"
M_G_studies = split_rdd.filter(lambda row: row[6].strip() == "Massachusetts General Hospital")

# Count the number of studies that satisfy the condition
num_M_G_studies = M_G_studies.count()

# Print the result
print(f"Number of studies sponsored by Massachusetts General Hospital: {num_M_G_studies}")

▸ (2) Spark Jobs
Number of studies sponsored by Mayo Clinic: 2766
Number of studies sponsored by Massachusetts General Hospital: 2263

Command took 28.16 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 12:11:45 PM on BDTT_task1_27_Apr_2
```

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

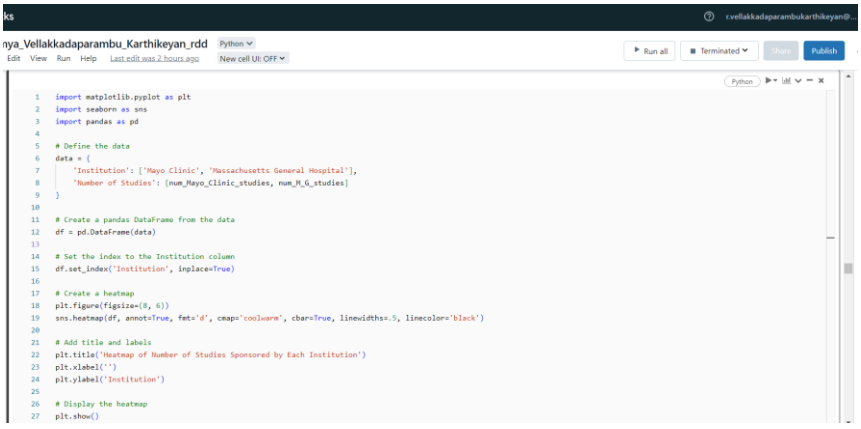
Discussion of Result:

▸ (2) Spark Jobs

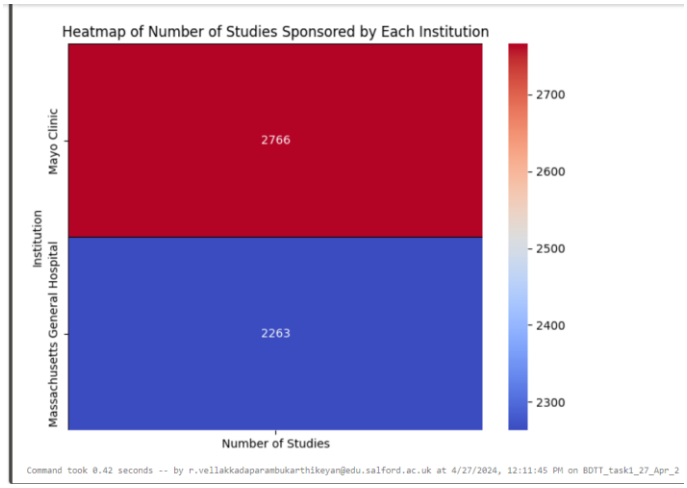
```
Number of studies sponsored by Mayo Clinic: 2766
Number of studies sponsored by Massachusetts General Hospital: 2263
```

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:

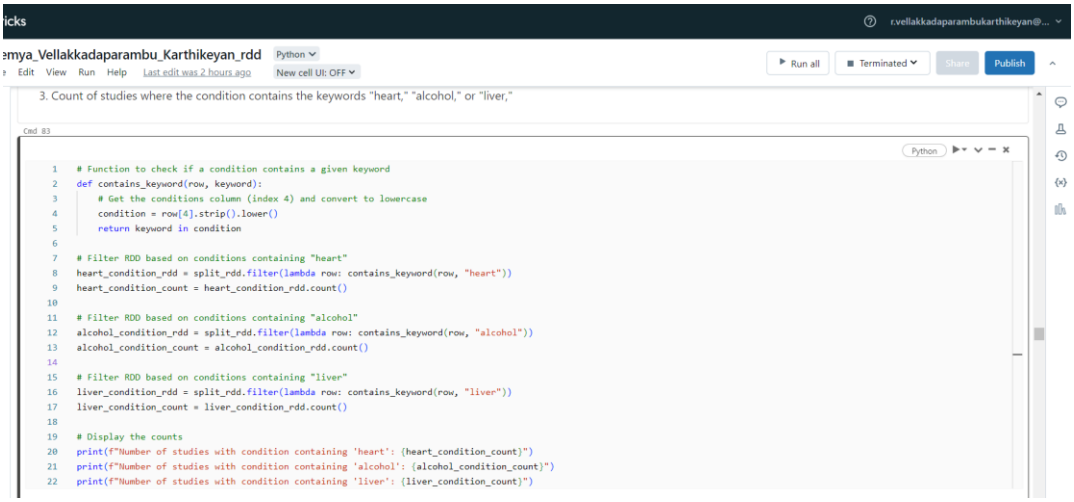


```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Define the data
data = {
    'Institution': ['Mayo Clinic', 'Massachusetts General Hospital'],
    'Number of Studies': [num_Mayo_Clinic_studies, num_M_G_studies]
}

# Create a pandas DataFrame from the data
df = pd.DataFrame(data)

# Set the index to the Institution column
df.set_index('Institution', inplace=True)

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df, annot=True, fmt='d', cmap='coolwarm', cbar=True, linewidths=.5, linecolor='black')

# Add title and labels
plt.title('Heatmap of Number of Studies Sponsored by Each Institution')
plt.xlabel('')
plt.ylabel('Institution')

# Display the heatmap
plt.show()
```



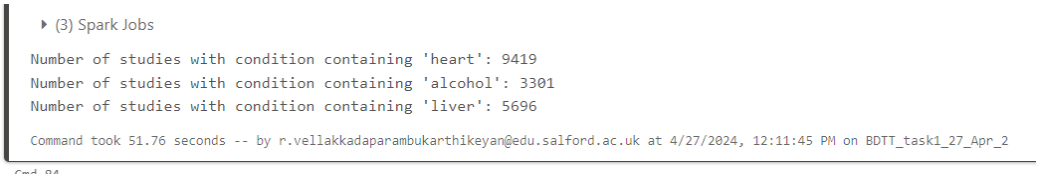
Analysis 3:

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



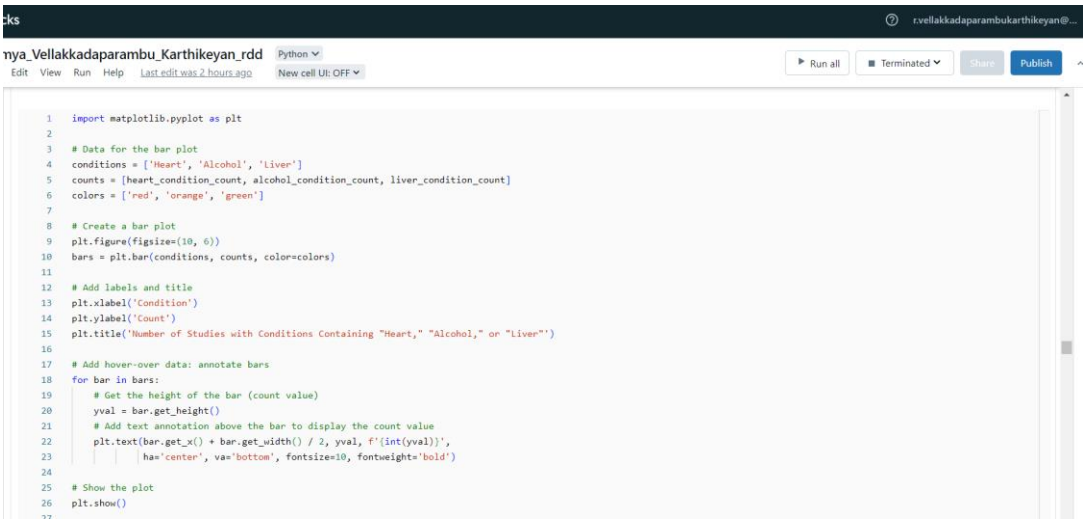
Main Idea: The code filters the RDD to include only studies where the conditions contain the keywords "heart" , “alcohol”, and “liver” and the count of each keyword based conditions are aggregated.

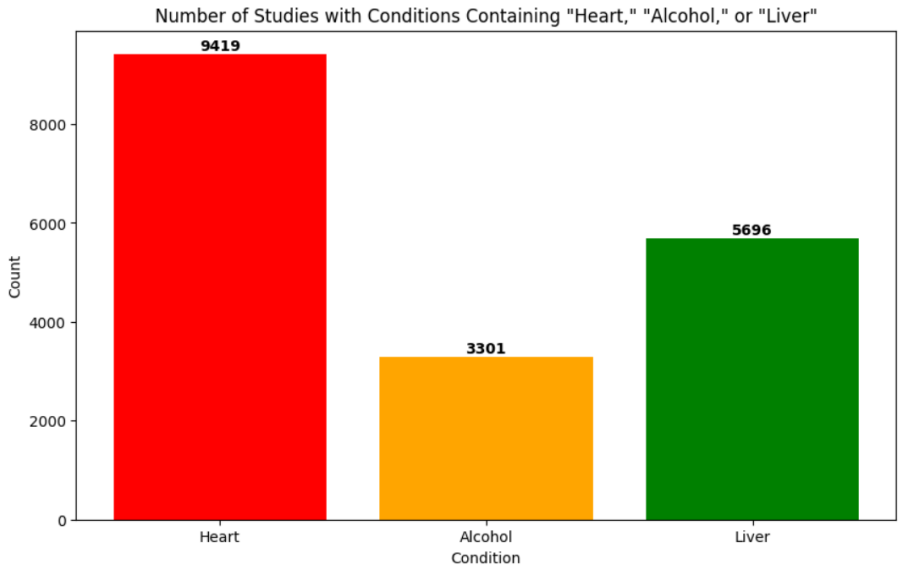
Discussion of Result:



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:

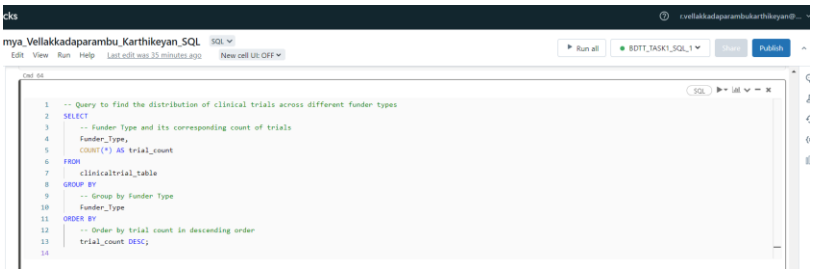




1.13 Additional Analyses and Visualization in Spark SQL Approach

Analysis 1:

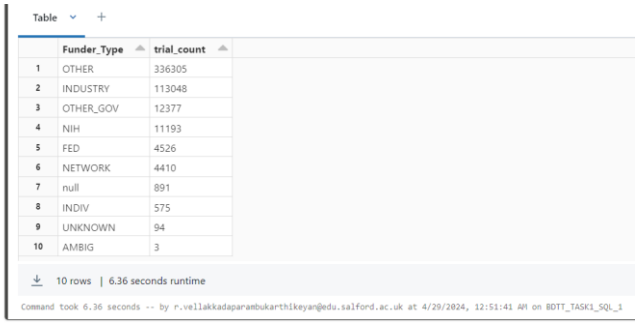
Question: Show the distribution of clinical trials across different funder types



```sql
-- Query to find the distribution of clinical trials across different funder types
SELECT
    -- Funder Type and its corresponding count of trials
    Funder_Type,
    COUNT(*) AS trial_count
FROM
    clinicaltrial_table
GROUP BY
    -- Group by Funder Type
    Funder_Type
ORDER BY
    -- Order by trial count in descending order
    trial_count DESC;
```

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:



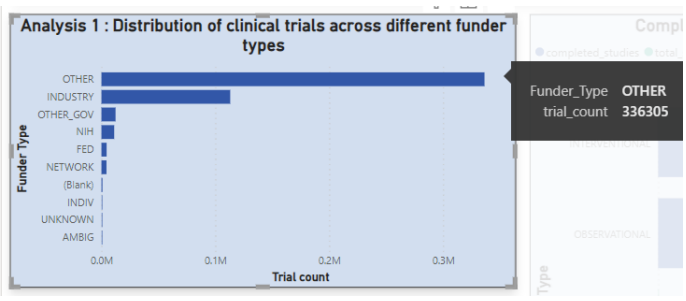
	Funder_Type	trial_count
1	OTHER	336305
2	INDUSTRY	113048
3	OTHER_GOV	12377
4	NIH	11193
5	FED	4526
6	NETWORK	4410
7	null	891
8	INDIV	575
9	UNKNOWN	94
10	AMBIG	3

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 BDTT_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:



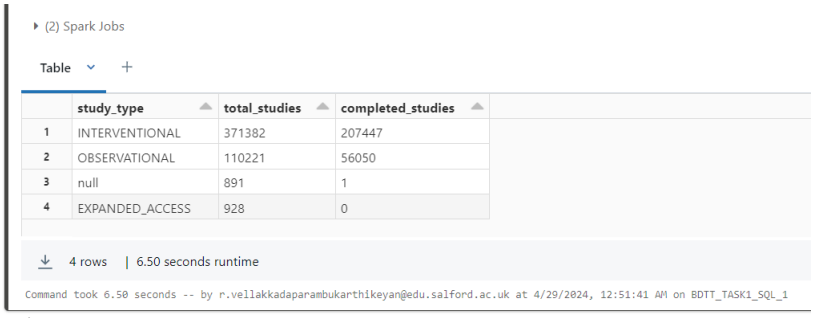
Analysis 2:

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

```
SELECT
    Type AS study_type,
    COUNT(*) AS total_studies,
    SUM(CASE WHEN Status = 'COMPLETED' THEN 1 ELSE 0 END) AS completed_studies
FROM
    clinicaltrial_table
GROUP BY
    Type
ORDER BY
    completed_studies DESC, total_studies DESC;
```

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

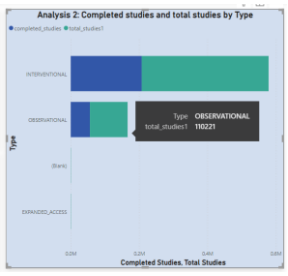
	study_type	total_studies	completed_studies
1	INTERVENTIONAL	371382	207447
2	OBSERVATIONAL	110221	56050
3	null	891	1
4	EXPANDED_ACCESS	928	0

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

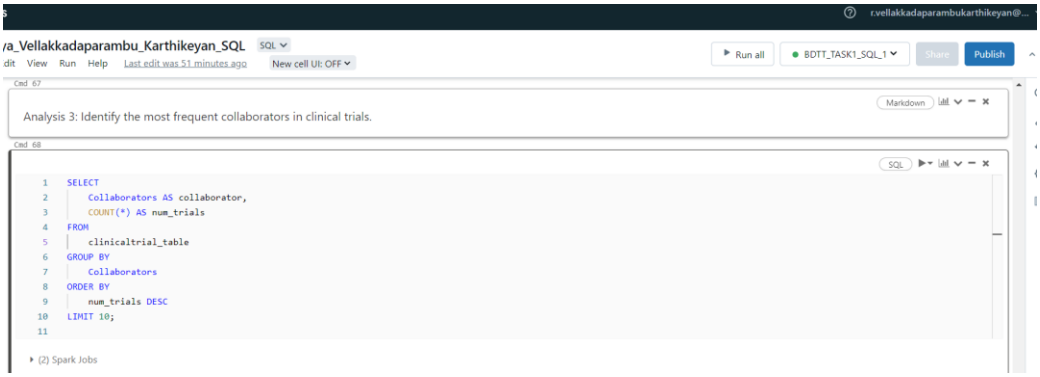
Insights : By ordering the results based on completed studies and total studies, it can identify which study types are more commonly completed, providing valuable insights into trends in research completion across various trial types.

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?



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:

Table

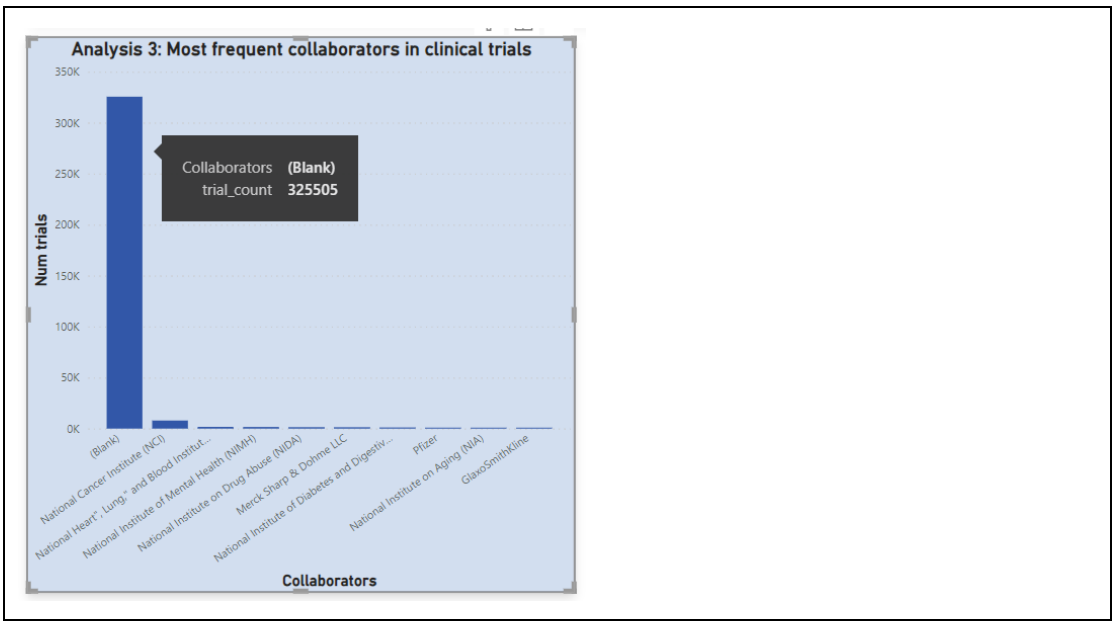
	collaborator	num_trials
1	null	325505
2	National Cancer Institute (NCI)	7918
3	National Heart", Lung," and Blood Institute (NHLBI)	1689
4	National Institute of Mental Health (NIMH)	1590
5	National Institute on Drug Abuse (NIDA)	1394
6	Merck Sharp & Dohme LLC	1334
7	National Institute of Diabetes and 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 BDTT_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.

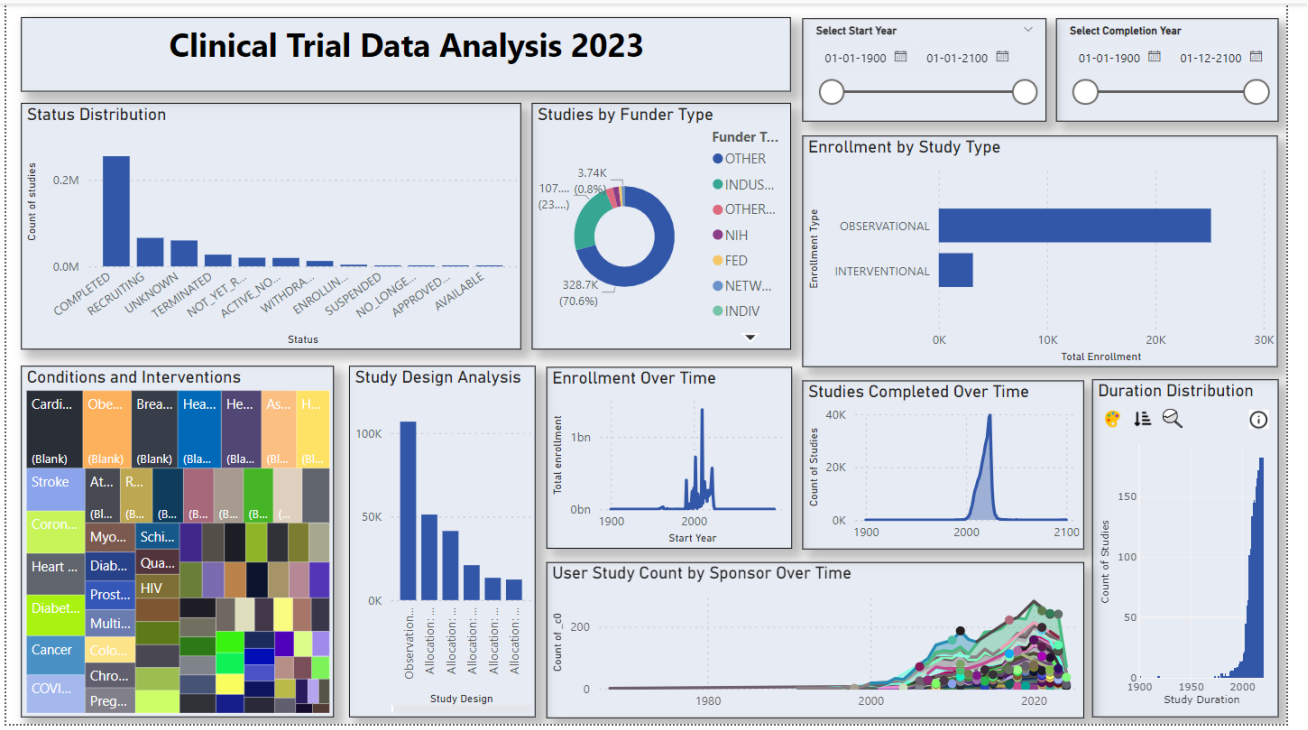
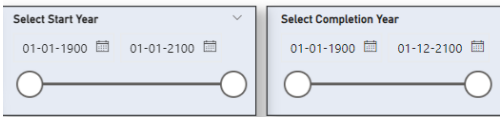
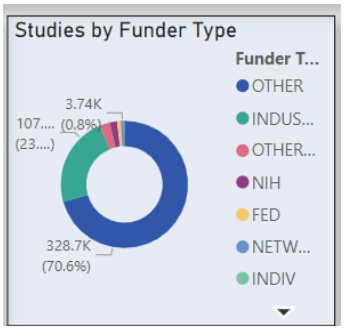
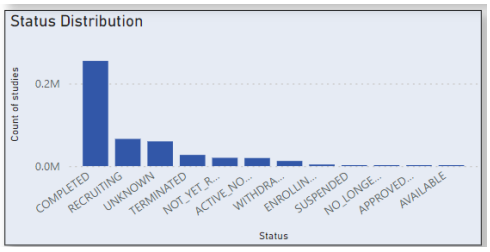
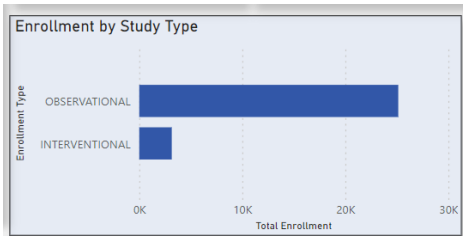
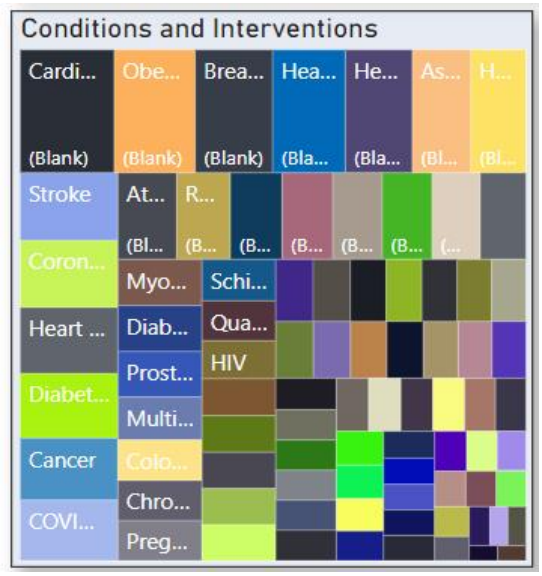
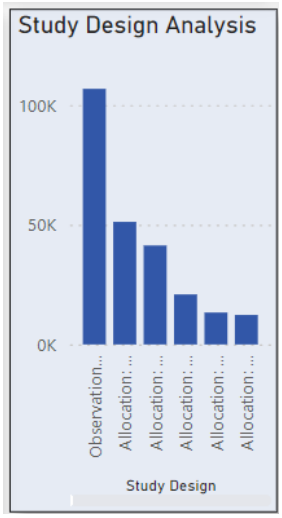
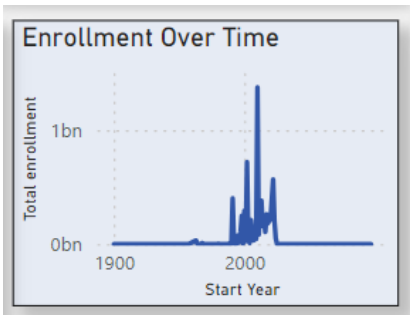
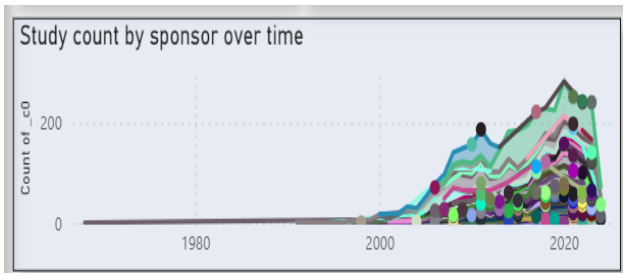
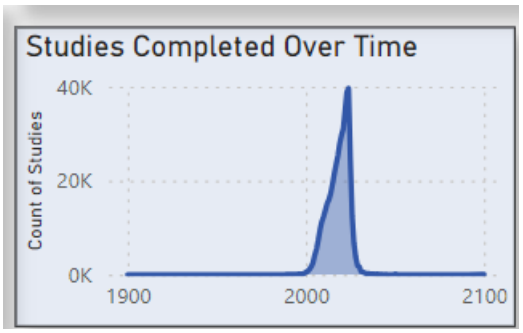
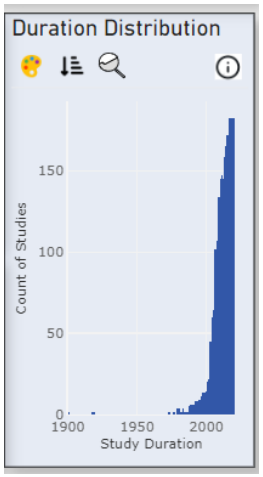


Figure 1.6 Clinical trial 2023 dashboard in Power BI

Table 1.6 Power BI dashboard details

Slider for selection start and completion year 	Donut chart for visualizing studies by funder type 
Stacked column chart for visualizing status distribution 	Stacked bar chart visualizing enrollment by study type 
Treemap showing conditions and interventions 	Stacked column chart showing study design analysis 

Line chart showing enrollment over time 	Stacked area chart showing study count by sponsor over time 
Area chart showing studies over time 	Histogram showing distribution of completion duration 

1.15 Analyses of historical dataset Clinicaltrial_2020 and Clinicaltrial_2021

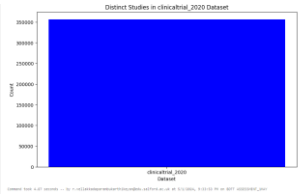
All the notebooks are general and reusable code for different versions of data. Here the results are shown the RDD approach as additional analyses in historical datasets are included in this approach. The same results for question 1- 5 are achieved in Dataframe and Spark approaches as well.

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

Clinicaltrial_2020

```
Cmd 107
Python
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@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1
```

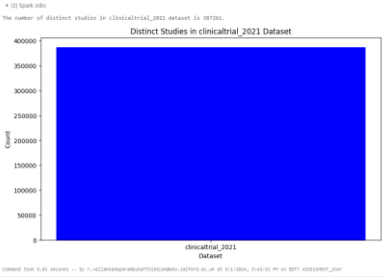
Visualization



Clinicaltrial_2021

```
Cmd 107
Python
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@edu.salford.ac.uk at 4/27/2024, 6:40:12 PM on BDTT_task 1
Cmd 108
```

Visualization

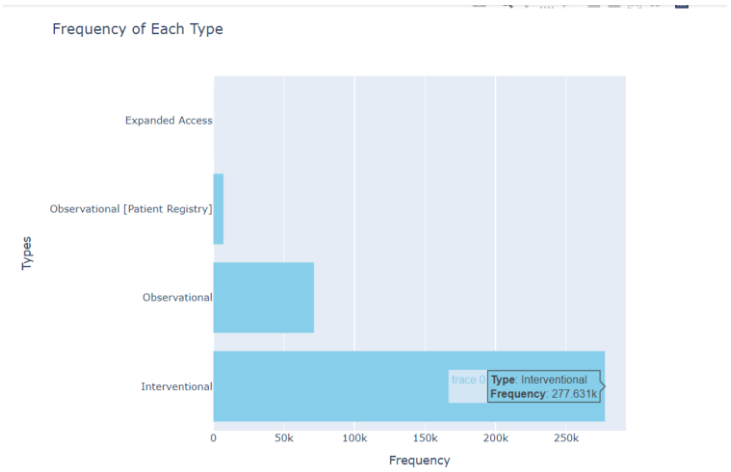


1.15.2 List all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type.

Clinicaltrial_2020

```
Cmd 115
Python
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 277631
Observational 71436
Observational [Patient Registry] 7332
Expanded Access 69
Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1
```

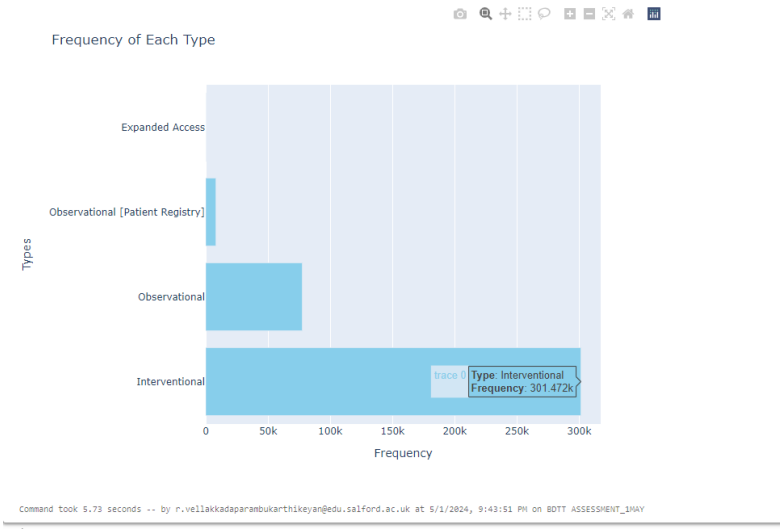
Visualization



Clinicaltrial_2021

```
Cmd 115

# display the obtained result
print("All types with their frequencies:")
for types, count in all_Type:
    print(types, count)

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

Command took 0.09 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:40:12 PM on BDTT_task 1
```

Visualization

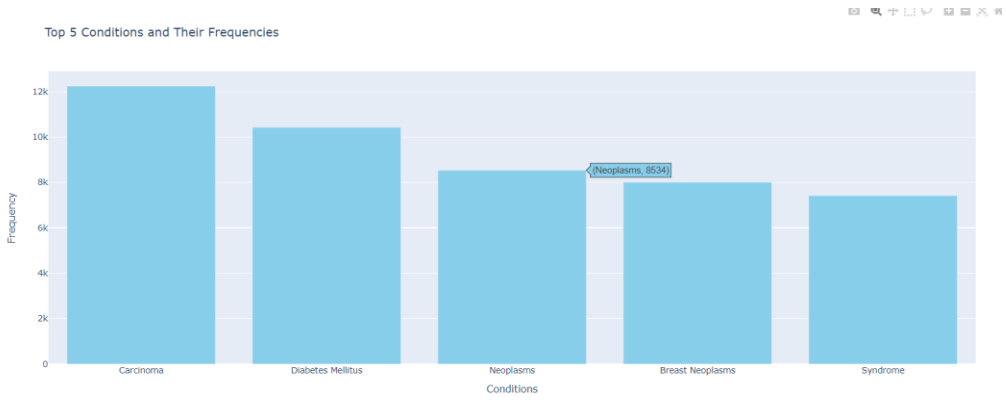


1.15.3 The top 5 conditions (from Conditions) with their frequencies.

Clinicaltrial_2020

```
Cmd 123                                                                                                    Python

# add up the frequencies for each condition and get the top 5 conditions
condition_top5 = condition_frequency.reduceByKey(lambda a, b: a + b).takeOrdered(5, key=lambda x: -x[1])

# display the corresponding result
print("Top 5 conditions and their frequencies:")
for condition, count in condition_top5:
    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 2.91 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1
```

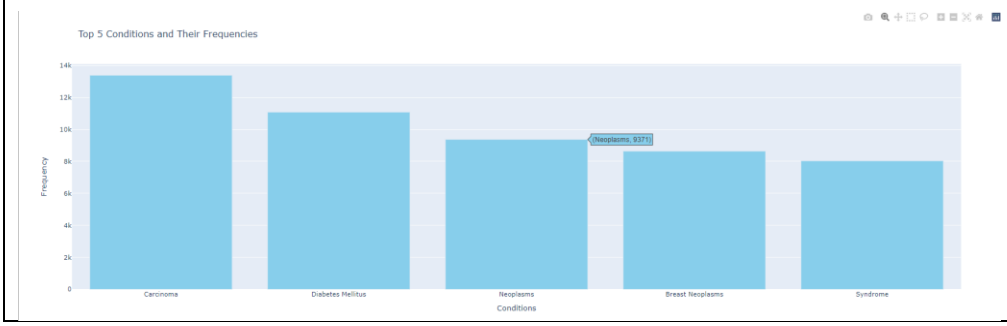
Visualization



Clinicaltrial_2021

```
Cmd 123                                                                                                    Python

# add up the frequencies for each condition and get the top 5 conditions
condition_top5 = condition_frequency.reduceByKey(lambda a, b: a + b).takeOrdered(5, key=lambda x: -x[1])

# display the corresponding result
print("Top 5 conditions and their frequencies:")
for condition, count in condition_top5:
    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 2.90 seconds -- by r.vellakkadaparambukarthikeyan@edu.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

```
# Filter out all the sponsors that are not pharmaceutical companies
sponsor_non_pharma_RDD = sponsor_RDD.filter(lambda x: x not in pharma_Parent_RDD)

# Count the number of clinical trials sponsored by each non-pharmaceutical company
sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

# Get the top 10 non-pharma sponsors by number of clinical trials sponsored
sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])

# Display the corresponding results with header
print(f"{trial_Year2}:")
print("+----------------------------------------+-----+")
print("|{:<40s}|{:5s}|".format("Sponsor", "count"))
print("+----------------------------------------+-----+")
for sponsor, count in sponsor_non_pharma_top10_RDD:
    print("|{:<40s}|{:5d}|".format(sponsor, count))
print("+----------------------------------------+-----+")
```

▸ (1) Spark Jobs

```
2020:
+----------------------------------------+-----+
|Sponsor                                 |count|
+----------------------------------------+-----+
|National Cancer Institute (NCI)         | 3100|
|M.D. Anderson Cancer Center             | 2238|
|Merck Sharp & Dohme Corp.               | 2184|
|Mayo Clinic                             | 2097|
|Assistance Publique - Hôpitaux de Paris | 2043|
|Novartis Pharmaceuticals                | 1962|
|Massachusetts General Hospital          | 1823|
|Assiut University                       | 1806|
|Hoffmann-La Roche                       | 1761|
|National Taiwan University Hospital     | 1720|
+----------------------------------------+-----+
```

Clinicaltrial_2021

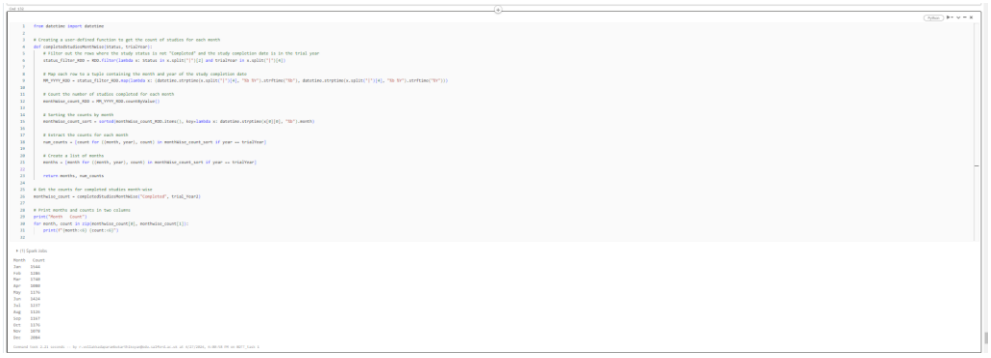
```
# Filter out all the sponsors that are not pharmaceutical companies
sponsor_non_pharma_RDD = sponsor_RDD.filter(lambda x: x not in pharma_Parent_RDD)

# Count the number of clinical trials sponsored by each non-pharmaceutical company
sponsor_non_pharma_count_RDD = sponsor_non_pharma_RDD.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y)

# Get the top 10 non-pharma sponsors by number of clinical trials sponsored
sponsor_non_pharma_top10_RDD = sponsor_non_pharma_count_RDD.takeOrdered(10, key=lambda x: -x[1])

# Display the corresponding results with header
print(f"{trial_Year2}:")
print("+----------------------------------------+-----+")
print("|{:<40s}|{:5s}|".format("Sponsor", "count"))
print("+----------------------------------------+-----+")
for sponsor, count in sponsor_non_pharma_top10_RDD:
    print("|{:<40s}|{:5d}|".format(sponsor, count))
print("+----------------------------------------+-----+")
```

▸ (1) Spark Jobs

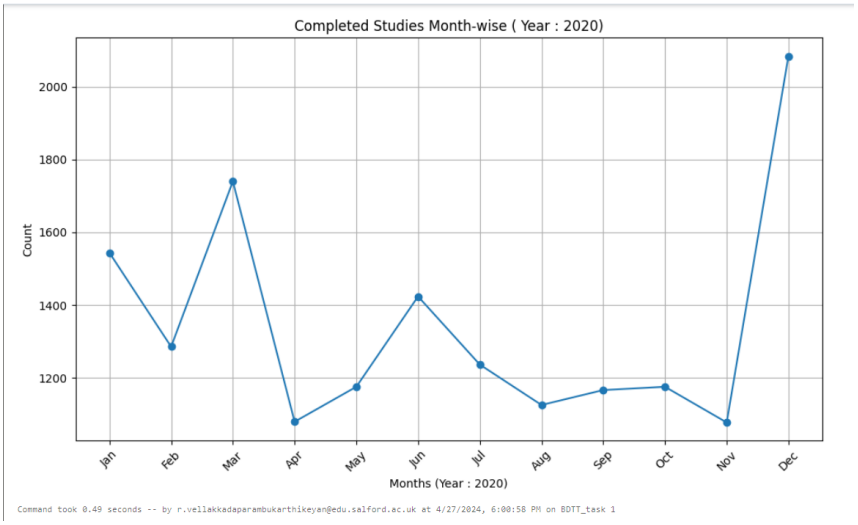
```
2021:
+----------------------------------------+-----+
|Sponsor                                 |count|
+----------------------------------------+-----+
|National Cancer Institute (NCI)         | 3218|
|M.D. Anderson Cancer Center             | 2414|
|Assistance Publique - Hôpitaux de Paris | 2369|
|Mayo Clinic                             | 2308|
|Merck Sharp & Dohme Corp.               | 2243|
|Assiut University                       | 2154|
|Novartis Pharmaceuticals                | 2088|
|Massachusetts General Hospital          | 1971|
|Cairo University                        | 1928|
|Hoffmann-La Roche                       | 1828|
+----------------------------------------+-----+
```

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

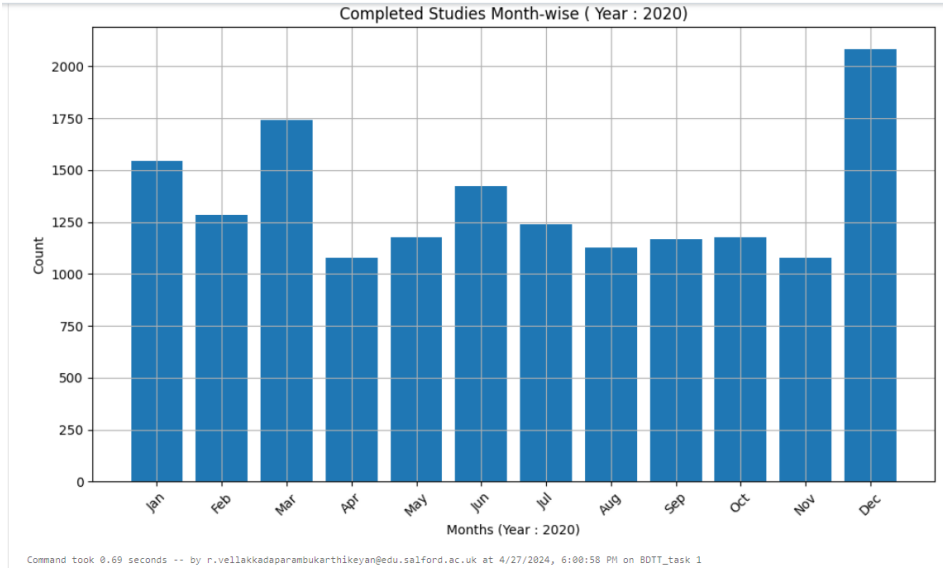
Clinicaltrial_2020



Visualization



Command took 0.49 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1



Command took 0.69 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1

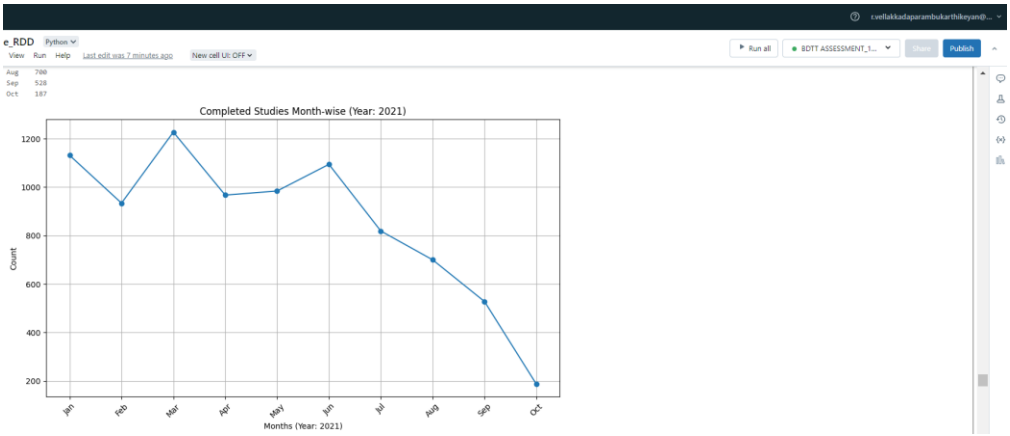
Clinicaltrial_2021

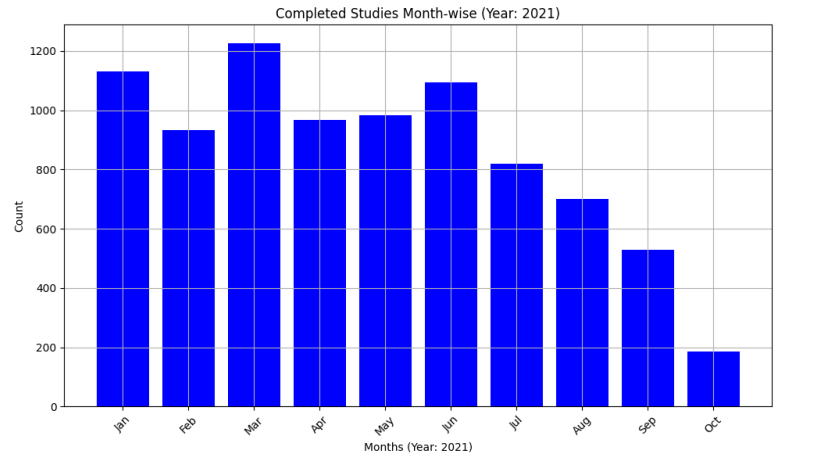
```
    status_filter_RDD = RDD.filter(lambda x: Status in x.split("|")[2] and trialYear in x.split("|")[4])

    # Map each row to a tuple containing the month and year of the study completion date
    MM_YYYY_RDD = status_filter_RDD.map(lambda x: (datetime.strptime(x.split("|")[4], "%b %Y").strftime("%b"), datetime.strptime(x.split("|")[4], "%b %Y").strftime("%Y")))

    # Count the number of studies completed for each month
    monthWise_count_RDD = MM_YYYY_RDD.countByValue()

    # Sorting the counts by month
    monthWise_count_sort = sorted(monthWise_count_RDD.items(), key=lambda x: datetime.strptime(x[0][0], "%b").month)

    # Extract the counts for each month
    num_counts = [count for ((month, year), count) in monthWise_count_sort if year == trialYear]

    # Create a list of months
    months = [month for ((month, year), count) in monthWise_count_sort if year == trialYear]

    return months, num_counts

# Get the counts for completed studies month-wise
monthwise_count = completedStudiesMonthWise("Completed", trial_Year2)

# Print months and counts in two columns
print("Month   Count")
for month, count in zip(monthwise_count[0], monthwise_count[1]):
    print(f"{month:<8} {count:<6}")
```

▸ (1) Spark Jobs

```
Month   Count
Jan     1131
Feb     934
Mar     1227
Apr     967
May     984
Jun     1094
Jul     819
Aug     700
Sep     528
Oct     187
```

Visualization





Command took 2.70 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 5/1/2024, 9:43:51 PM on BDTT ASSESSMENT_1MAY

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

# Define the user-defined function to list studies started in a specific year
def studies_started_in_year(data_rdd, year):
    # Filter RDD to select only the rows where the study start year matches the input year
    studies_year = data_rdd.filter(lambda line: year in line.split("|")[3])

    # Extract study ID, sponsor, and start date from each row
    study_info = studies_year.map(lambda line: (line.split("|")[0], line.split("|")[1], line.split("|")[3]))

    # Return the study information
    return study_info

# Input the year
input_year = "2005"  # Example: Input year as a string

# Call the function with the input year and retrieve the study information
studies_year_info = studies_started_in_year(RDD, input_year)

# Print the study information
for study_id, sponsor, start_date in studies_year_info.collect():
    print(f"Study ID: {study_id}, Sponsor: {sponsor}, Start Date: {start_date}")

▸ (1) Spark Jobs
```

▸ (1) Spark Jobs

Study ID	Sponsor	Start Date
NCT02758028	The University of Hong Kong	Aug 2005
NCT02754024	Werner Anderl	Sep 2005
NCT02750644	Joaquin de Haro, M.D.	Jan 2005
NCT00170053	Mayo Clinic	Jun 2005
NCT00179647	Celgene Corporation	Sep 2005
NCT00174785	Sanofi	Jun 2005
NCT00175058	TherOx	Aug 2005
NCT00173875	National Taiwan University Hospital	Mar 2005
NCT00179933	Ann & Robert H Lurie Children's Hospital of Chicago	Sep 2005
NCT00172653	National Taiwan University Hospital	Aug 2005
NCT00176475	University of Medicine and Dentistry of New Jersey	Jan 2005
NCT00178087	University of Pittsburgh	Aug 2005
NCT00178503	The University of Texas Health Science Center, Houston	Sep 2005
NCT00176644	University of Medicine and Dentistry of New Jersey	May 2005
NCT00174070	National Taiwan University Hospital	Aug 2005
NCT00172640	National Taiwan University Hospital	Jan 2005
NCT00170508	National Institute of Allergy and Infectious Diseases (NIAID)	Jun 2005
NCT00172718	National Taiwan University Hospital	Mar 2005
NCT00177112	University of Minnesota	Jun 2005

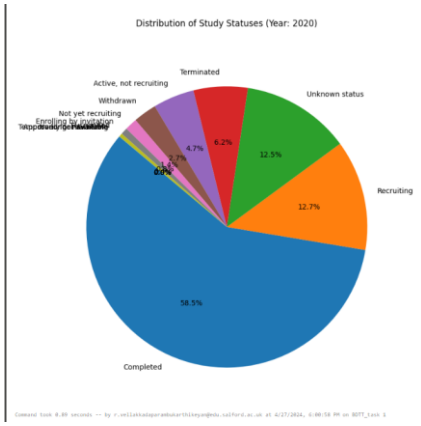
Command took 2.43 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/27/2024, 6:00:58 PM on BDTT_task 1

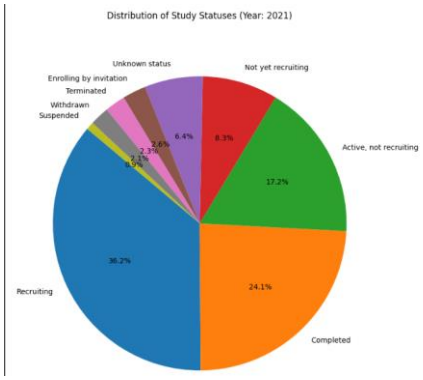
▸ (1) Spark Jobs

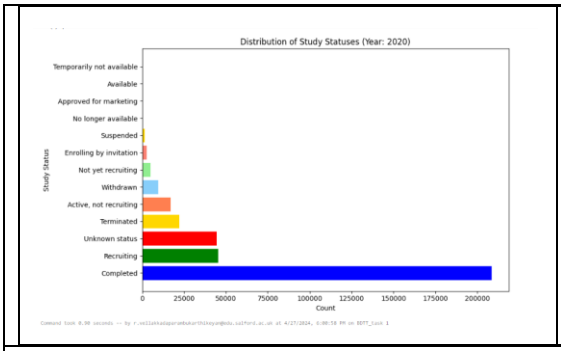
Study ID	Sponsor	Start Date
NCT03095404	McHaster University	Sep 2021
NCT03098225	University of Pisa	Jan 2021
NCT03097796	Pulmotect, Inc.	Dec 2021
NCT03090048	Manchanda Medical Clinic	Apr 2021
NCT03924310	Martin Berli	Apr 2021
NCT03925194	Heidelberg University	Oct 2021
NCT03923907	Chinese University of Hong Kong	Oct 2021
NCT03928639	National Heart, Lung, and Blood Institute (NHLBI)	Nov 2021
NCT03924778	H. Lee Moffitt Cancer Center and Research Institute	Sep 2021
NCT03926819	Hillhurst Biopharmaceuticals, Inc.	May 2021
NCT03921515	National Institute of Allergy and Infectious Diseases (NIAID)	Nov 2021
NCT03928275	Carman Giacomantonio	Sep 2021
NCT03927066	Mayo Clinic	May 2021
NCT03925077	University of Alabama at Birmingham	Feb 2021
NCT03924063	Balgrist University Hospital	May 2021
NCT03929289	Hospices Civils de Lyon	Apr 2021
NCT03920462	Laboratoire TIMC-IMAG	Mar 2021
NCT03926520	Brent Forester	Jan 2021
NCT03927625	Chris Lascola, MD	Dec 2021

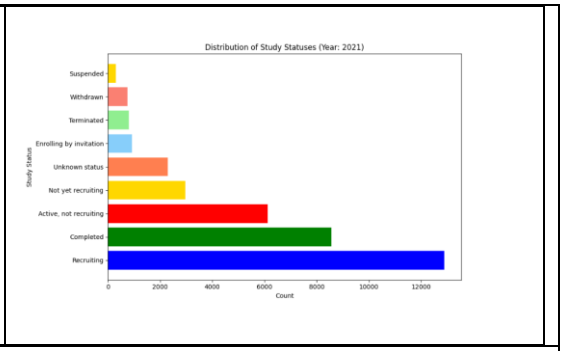
Command took 2.01 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 5/1/2024, 9:43:51 PM on BDTT ASSES

Distribution of Study Statuses

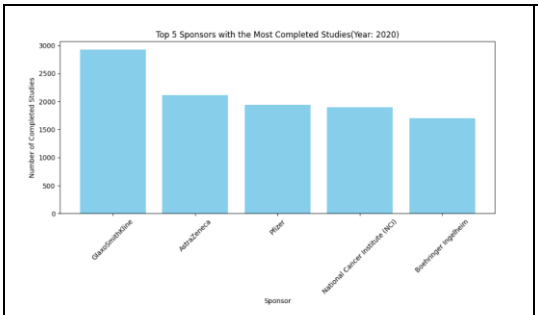


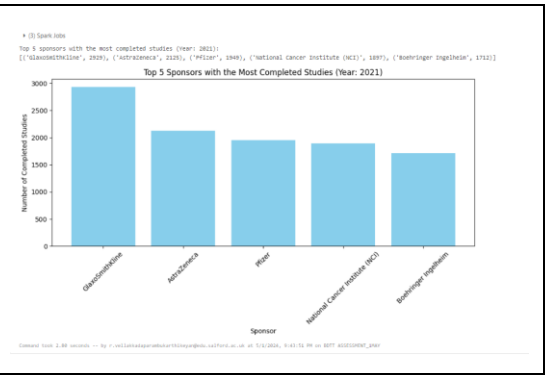






Top 5 sponsors with the most completed studies





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 but 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 in 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 implemntation 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).

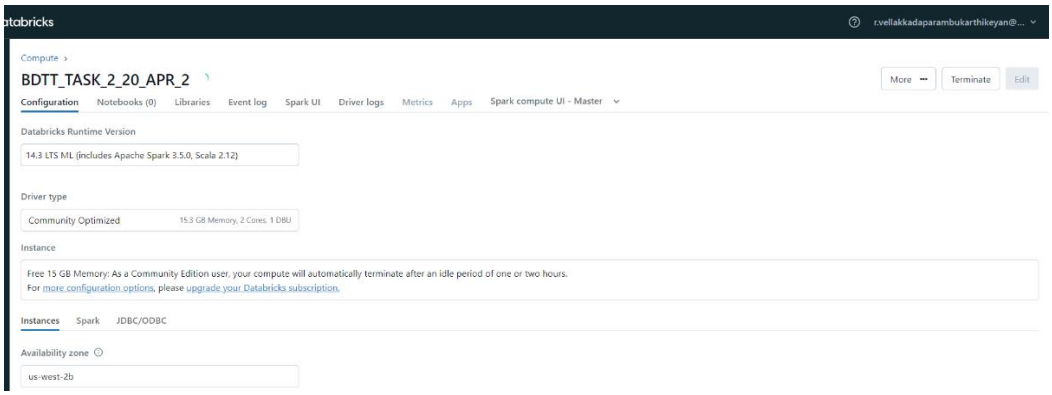
- Upload or import your data to the Databricks environment (e.g., from your local machine, cloud storage, or a database).

- Create a notebook or a job where you will write your ML code.

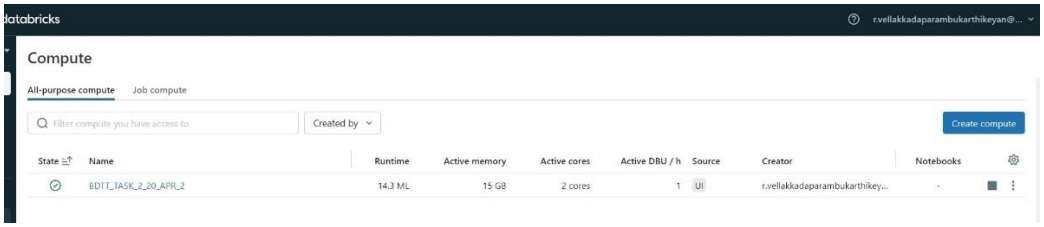
- In your notebook, import the necessary packages and libraries for the task (e.g., **pyspark**, **mlflow**, etc.).

Figure 2.1 shows the related figures

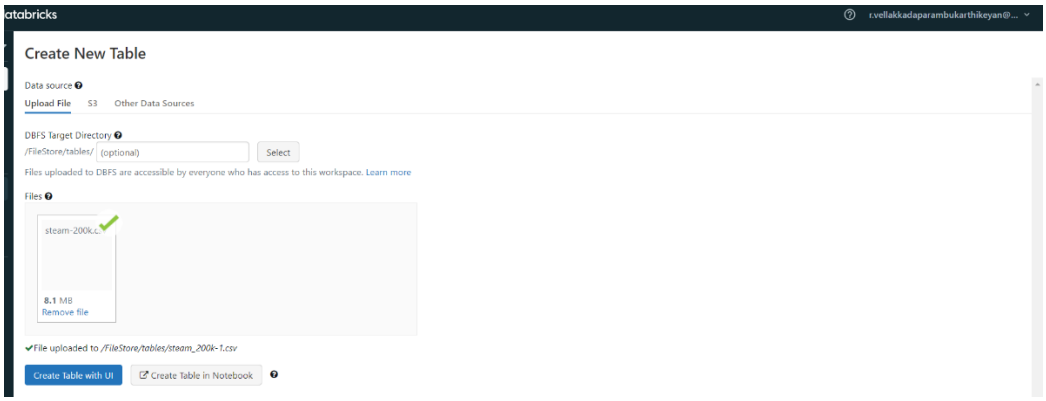
Creating cluster



Cluster details



Uploading .csv file



Creating notebook

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

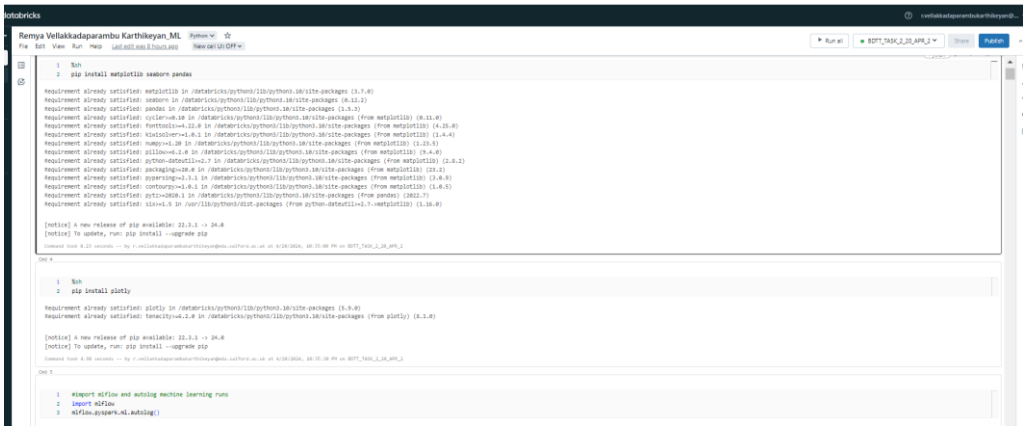


Figure 2.1 Setting up databricks environment for building recommender system

2.3 Data Management

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

Highlighting **steam-200k.csv** in DBFS file system

Converting CSV file to Spark dataframe and adding headers  Cmd 7 Python 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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:00:44 AM on BDTT_TASK_2_20_APR_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 Python 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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:00:44 AM on BDTT_TASK_2_20_APR_1
Removing rows with missing values and display of head 20 rows

```
# Remove rows with any missing values (NaN) from the DataFrame and then display the first 20 rows
steam = steam.dropna()
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|HuniePop                  |purchase|1.0  |
|151603712|HuniePop                  |play    |8.5  |
|151603712|Path of Exile             |purchase|1.0  |
|151603712|Path of Exile             |play    |8.1  |
|151603712|Poly Bridge               |purchase|1.0  |
|151603712|Poly Bridge               |play    |7.5  |
|151603712|Left 4 Dead               |purchase|1.0  |
|151603712|Left 4 Dead               |play    |3.3  |
```

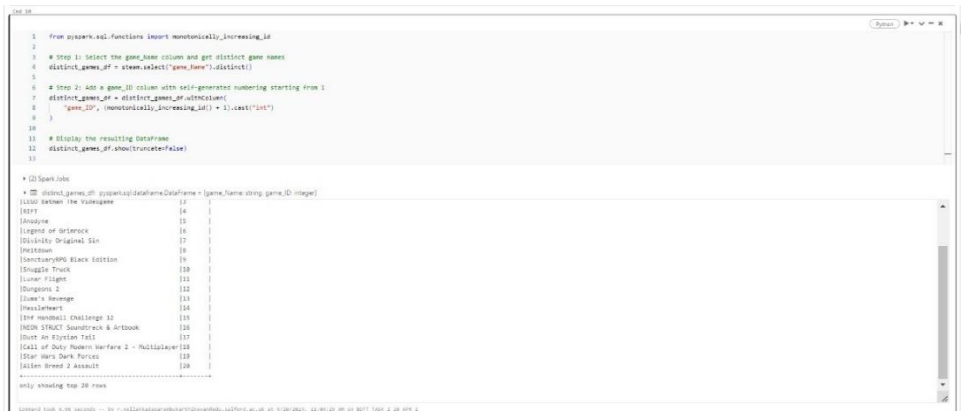
Command took 1.71 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:01:56 AM on BDTT_TASK_2_20_APR_1

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) ia 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

```
# Get the count of distinct games in the distinct_games_df DataFrame
distinct_game_count = distinct_games_df.count()

# Display the count of distinct games
print(f"Total number of distinct games in distinct_games_df: {distinct_game_count}")
```

```
(3) Spark Jobs
Total number of distinct games in distinct_games_df: 5155
Command took 3.75 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:06:56 AM on BDTT_TASK_2_20_APR_1
```

Last 20 game names

```
# Get the last 20 rows of the DataFrame
last_20_rows = distinct_games_df.tail(20)

# Display the last 20 rows
for row in last_20_rows:
    print(row)
```

```
(2) Spark Jobs
Row(game_Name='Samantha Swift and the Golden Touch', game_ID=5136)
Row(game_Name='Nancy Drew Lights, Camera, Curses!', game_ID=5137)
Row(game_Name='TIS - Player Profiles Empire - Alhadance', game_ID=5138)
Row(game_Name='TIS - Player Profiles MVMtmtb - HEIN', game_ID=5139)
Row(game_Name='Habitat', game_ID=5140)
Row(game_Name='ePic Character Generator', game_ID=5141)
Row(game_Name='Vex', game_ID=5142)
Row(game_Name='TFAG Accessorie Pack', game_ID=5143)
Row(game_Name='TIS - Player Profiles Newbee - Nu', game_ID=5144)
Row(game_Name="Ferrum's Secrets where is grandpa?", game_ID=5145)
Row(game_Name='GAMEBAC WARRIORS 4-IT', game_ID=5146)
Row(game_Name='Blitzkrieg 2 Anthology', game_ID=5147)
Row(game_Name='Family Guy Back to the Multiverse', game_ID=5148)
Row(game_Name='Ford Racing 2', game_ID=5149)
Row(game_Name='Sandmason', game_ID=5150)
Row(game_Name='Circuits', game_ID=5151)
Row(game_Name='Lineage II The Chaotic Throne', game_ID=5152)
Row(game_Name='GameMaker Studio Standard', game_ID=5153)
Row(game_Name='Shark Attack Deathmatch 2', game_ID=5154)
Row(game_Name="Dragon's Prophet Prov Starter Pack", game_ID=5155)
Command took 1.29 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/19/2024, 8:50:38 PM on BDTT_TASK_19_APR_2024
```

Merging distinct games_df and steam df

```
from pyspark.sql.functions import desc, sum, count

# Join the steam and distinct_games_df DataFrames on the game name (game_Name)
joined_df = steam.join(distinct_games_df, steam["game_Name"] == distinct_games_df["game_Name"], "inner")
joined_df.show(truncate=False)
```

```
(3) Spark Jobs
joined_df: pyspark.sql.dataframe.DataFrame = [ID: integer, game_Name: string ... 4 more fields]
+---------+--------------------------+----------+-----+--------------------------+-------+
|ID       |game_Name                 |mem_Type  |value|game_Name                 |game_ID|
+---------+--------------------------+----------+-----+--------------------------+-------+
|151603712|The Elder Scrolls V Skyrim|purchase  |1.0  |The Elder Scrolls V Skyrim|2610   |
|151603712|The Elder Scrolls V Skyrim|play      |273.0|The Elder Scrolls V Skyrim|2610   |
|151603712|Fallout 4                 |purchase  |1.0  |Fallout 4                 |411    |
|151603712|Fallout 4                 |play      |87.0 |Fallout 4                 |411    |
|151603712|Spore                     |purchase  |1.0  |Spore                     |3869   |
|151603712|Spore                     |play      |14.9 |Spore                     |3869   |
|151603712|Fallout New Vegas         |purchase  |1.0  |Fallout New Vegas         |3821   |
|151603712|Fallout New Vegas         |play      |12.1 |Fallout New Vegas         |3821   |
|151603712|Left 4 Dead 2             |purchase  |1.0  |Left 4 Dead 2             |70     |
|151603712|Left 4 Dead 2             |play      |8.9  |Left 4 Dead 2             |70     |
|151603712|HuniePop                  |purchase  |1.0  |HuniePop                  |3341   |
|151603712|HuniePop                  |play      |8.5  |HuniePop                  |3341   |
|151603712|Path of Exile             |purchase  |1.0  |Path of Exile             |1563   |
|151603712|Path of Exile             |play      |8.1  |Path of Exile             |1563   |
|151603712|Poly Bridge               |purchase  |1.0  |Poly Bridge               |4117   |
|151603712|Poly Bridge               |play      |7.5  |Poly Bridge               |4117   |
|151603712|Left 4 Dead               |purchase  |1.0  |Left 4 Dead               |1993   |
|151603712|Left 4 Dead               |play      |3.3  |Left 4 Dead               |1993   |
Command took 4.09 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/19/2024, 11:39:41 AM on BDTT_TASK_2_20_APR_1
```

Deleting redundant game name column

```
# Join the steam and distinct_games_df DataFrames on the game name (game_Name)
joined_df = steam.join(distinct_games_df, steam["game_Name"] == distinct_games_df["game_Name"], "inner")

# Drop the game_Name column from the joined DataFrame
joined_df = joined_df.drop(steam["game_Name"])

# Display the joined DataFrame without the game_Name column repetition
joined_df.show(truncate=False)
```

```
(3) Spark Jobs
joined_df: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
+---------+--------+-----+--------------------------+-------+
|ID       |mem_Type|value|game_Name                 |game_ID|
+---------+--------+-----+--------------------------+-------+
|151603712|purchase|1.0  |The Elder Scrolls V Skyrim|2610   |
|151603712|play    |273.0|The Elder Scrolls V Skyrim|2610   |
|151603712|purchase|1.0  |Fallout 4                 |411    |
|151603712|play    |87.0 |Fallout 4                 |411    |
|151603712|purchase|1.0  |Spore                     |3869   |
|151603712|play    |14.9 |Spore                     |3869   |
|151603712|purchase|1.0  |Fallout New Vegas         |3821   |
|151603712|play    |12.1 |Fallout New Vegas         |3821   |
|151603712|purchase|1.0  |Left 4 Dead 2             |70     |
|151603712|play    |8.9  |Left 4 Dead 2             |70     |
|151603712|purchase|1.0  |HuniePop                  |3341   |
|151603712|play    |8.5  |HuniePop                  |3341   |
|151603712|purchase|1.0  |Path of Exile             |1563   |
|151603712|play    |8.1  |Path of Exile             |1563   |
|151603712|purchase|1.0  |Poly Bridge               |4117   |
|151603712|play    |7.5  |Poly Bridge               |4117   |
|151603712|purchase|1.0  |Left 4 Dead               |1993   |
|151603712|play    |3.3  |Left 4 Dead               |1993   |
Command took 2.25 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:40:12 AM on BDTT_TASK_2_20_APR_1
```

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 Python 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 BDTT_TASK_2_20_APR_1
Total rows in steam df  Python 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 2.90 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:17:04 AM on BDTT_TASK_2_20_APR_1
Unique user behavior types  Cmd 16 Python 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 BDTT_TASK_2_20_APR_1

Unique values of game name

```
print("\nUnique values in the game_Name column:")
steam.select("game_Name").distinct().show(truncate=False)
```

(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 |
|HuniePop                           |
|The Banner Saga                    |
|Grand Theft Auto IV                |
|Dead Island Epidemic               |
|Marvel Heroes 2015                 |
|Eldevin                            |
+-----------------------------------+
Command took 1.88 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:18:49 AM on BDTT_TASK_2_20_APR_1
```

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

Cmd 18

```
# Group by mem_Type and count the number of rows in each group
print("\nGroup by mem_Type and count the number of rows in each group:")
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@edu.salford.ac.uk at 4/20/2024, 11:20:00 AM on BDTT_TASK_2_20_APR_1
```

Users with *purchase* behavior

Cmd 19

```
# Filter the DataFrame to include only rows where mem_Type is "purchase"
purchase_df = steam.filter(steam.mem_Type == "purchase")
purchase_df.show()
```

(1) Spark Jobs

purchase_df: pyspark.sql.dataframe.DataFrame
ID: integer
game_Name: string
mem_Type: string
value: double

```
+---------+--------------------+--------+-----+
|       ID|           game_Name|mem_Type|value|
+---------+--------------------+--------+-----+
|151603712|The Elder Scrolls...|purchase|  1.0|
|151603712|            Fallout 4|purchase|  1.0|
|151603712|              Spore|purchase|  1.0|
|151603712|Fallout New Vegas|purchase|  1.0|
|151603712|         Left 4 Dead 2|purchase|  1.0|
|151603712|             HuniePop|purchase|  1.0|
|151603712|        Path of Exile|purchase|  1.0|
|151603712|          Poly Bridge|purchase|  1.0|
|151603712|          Left 4 Dead|purchase|  1.0|
|151603712|       Team Fortress 2|purchase|  1.0|
|151603712|          Tomb Raider|purchase|  1.0|
|151603712|      The Banner Saga|purchase|  1.0|
|151603712|Dead Island Epidemic|purchase|  1.0|
|151603712|    BioShock Infinite|purchase|  1.0|
|151603712|Dragon Age Origin...|purchase|  1.0|
|151603712|Fallout 3 - Game ...|purchase|  1.0|
|151603712|SEGA Genesis & Me...|purchase|  1.0|
|151603712| Grand Theft Auto IV|purchase|  1.0|
+---------+--------------------+--------+-----+
Command took 1.13 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:21:41 AM on BDTT_TASK_2_20_APR_1
```

Game name and their count of occurrences

```
# Group the filtered DataFrame by game_Name and count the occurrences
purchase_count_df = purchase_df.groupBy("game_Name").count()

purchase_count_df.show(truncate = False)
```

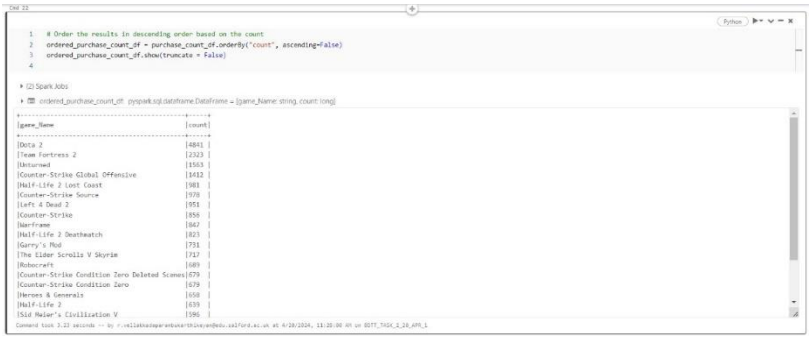
(2) Spark Jobs

purchase_count_df: pyspark.sql.dataframe.DataFrame = [game_Name: string, count: long]

game_Name	count
Dota 2	4841
METAL GEAR SOLID V THE PHANTOM PAIN	68
LEGO Batman The Videogame	11
RIFT	161
Anodyne	11
Legend of Grimrock	58
Divinity Original Sin	71
Meltdown	4
SanctuaryRPG Black Edition	5
Snuggle Truck	9
Lunar Flight	10
Dungeons 2	6
Zuma's Revenge	4
HexxaHeart	1
Ihf Handball Challenge 12	1
NEON STRUCT Soundtrack & Artbook	2
Dust An Elysian Tail	76
Call of Duty Modern Warfare 2 - Multiplayer	343

Command took 1.99 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:23:30 AM on BDTT_TASK_2_20_APR_1

Games and count in descending order



```
# Order the results in descending order based on the count
ordered_purchase_count_df = purchase_count_df.orderBy("count", ascending=False)
ordered_purchase_count_df.show(truncate = False)
```

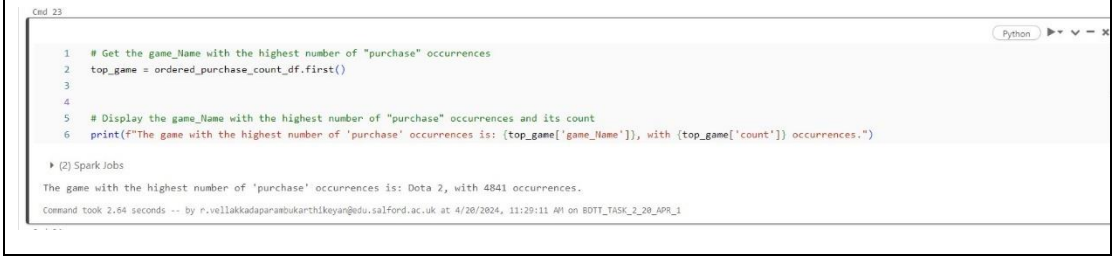
(2) Spark Jobs

ordered_purchase_count_df: pyspark.sql.dataframe.DataFrame = [game_Name: string, count: long]

game_Name	count
Dota 2	4841
Team Fortress 2	2323
Unturned	1563
Counter-Strike Global Offensive	1412
Half-Life 2 Lost Coast	981
Counter-Strike Source	978
Left 4 Dead 2	951
Counter-Strike	856
Warframe	847
Half-Life 2 Deathmatch	823
Garry's Mod	731
The Elder Scrolls V Skyrim	717
Robocraft	689
Counter-Strike Condition Zero Deleted Scenes	679
Counter-Strike Condition Zero	679
Heroes & Generals	658
Half-Life 2	639
Sid Meier's Civilization V	596

Command took 2.22 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:28:00 AM on BDTT_TASK_2_20_APR_1

Game with highest number of purchase occurrences



Cmd 23

```
# Get the game_Name with the highest number of "purchase" occurrences
top_game = ordered_purchase_count_df.first()


# Display the game_Name with the highest number of "purchase" occurrences and its count
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_20_APR_1

Games with '*Play*' behavior

```
from pyspark.sql.functions import sum

# Filter the DataFrame to include only rows where mem_Type is "play"
play_df = steam.filter(steam.mem_Type == "play")
play_df.show(truncate = False)
```

(1) Spark Jobs

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	Spore	play	14.9
151603712	Fallout New Vegas	play	12.1
151603712	Left 4 Dead 2	play	8.9
151603712	HuniePop	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	Tomb Raider	play	2.5
151603712	The Banner Saga	play	2.0
151603712	Dead Island Epidemic	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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:30:29 AM on BDTT_TASK_2_20_APR_1

User and cumulative play time

```
from pyspark.sql.functions import sum, round

# Group the filtered DataFrame by ID and calculate the cumulative play time (sum of 'value' column)
cumulative_play_time_df = play_df.groupBy("ID").agg(sum("value").alias("cumulative_play_time"))

# Round the cumulative_play_time column to two decimal places
cumulative_play_time_df = cumulative_play_time_df.withColumn("cumulative_play_time", round(cumulative_play_time_df["cumulative_play_time"], 2))

# Display the DataFrame with cumulative play time rounded to two decimal places
cumulative_play_time_df.show(truncate=False)
```

(2) Spark Jobs

cumulative_play_time_df: pyspark.sql.dataframe.DataFrame = [ID: integer, cumulative_play_time: double]

ID	cumulative_play_time
16167221	1017.0
166705920	697.0
244878837	6.0
99992274	6.3
174415183	25.6
156156544	5.3
152861732	871.8
171911285	32.0
128412180	1.2
74557142	0.1
197279094	23.0
108219790	35.2
208061820	23.9
61502266	5.0
138455695	112.0
35264447	125.4
187877855	31.2
148546383	116.0

Command took 3.28 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:31:30 AM on BDTT_TASK_2_20_APR_1

Users with highest play time

```
# Arrange the cumulative_play_time_df DataFrame in descending order of cumulative play time
cumulative_play_time_df = cumulative_play_time_df.orderBy("cumulative_play_time", ascending=False)

# Display the DataFrame with cumulative play time rounded to two decimal places, ordered in descending order
cumulative_play_time_df.show(truncate=False)
```

(2) Spark Jobs

cumulative_play_time_df: pyspark.sql.dataframe.DataFrame = [ID: integer, cumulative_play_time: double]

ID	cumulative_play_time
77017395	11754.0
10599862	11651.7
100630947	10853.2
26702388	10470.1
153382649	9640.0
43684432	9546.3
48790067	9427.2
52731290	9417.6
42935819	8172.9
14544587	8137.1
52167955	7836.7
130882834	7801.9
57433226	7741.5
63015403	7190.4
47063596	7161.0
67694595	7152.1
17017968	7108.5
58818751	6929.9

Command took 2.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:32:48 AM on BDTT_TASK_2_20_APR_1

Top 5 user based on playtime

```
Cmd 27

# Take the top 5 IDs from the ordered DataFrame
top_5_ids_df = cumulative_play_time_df.limit(5)

# Display the top 5 IDs with their corresponding cumulative play time
top_5_ids_df.show(truncate=False)

▸ (2) Spark Jobs
▸ top_5_ids_df: pyspark.sql.dataframe.DataFrame = [ID: integer, cumulative_play_time: double]

+---------+-------------------+
|ID       |cumulative_play_time|
+---------+-------------------+
|73017395 |11754.0            |
|10599862 |11651.7            |
|100630947|10853.2            |
|26762388 |10470.1            |
|153382649|9640.0             |
+---------+-------------------+

Command took 2.35 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:36:53 AM on BDTT_TASK_2_20_APR_1
```

Frequency of occurrence of game IDs

```
Cmd 33

# Distribution of games
game_distribution = joined_df.groupBy("game_ID").count().orderBy(desc("count")).toPandas()
print("\nTop 10 most frequent games:")
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     3690   1424
7     2670   1397
8     2610   1394
9     2220   1271

Command took 6.09 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:47:57 AM on BDTT_TASK_2_20_APR_1
```

Top 10 frequent games

```
Cmd 34

# Distribution of games
game_distribution = joined_df.groupBy("game_Name").count().orderBy(desc("count")).toPandas()
print("\nTop 10 most frequent games:")
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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:49:12 AM on BDTT_TASK_2_20_APR_1
```

Top 10 games based on play time

```
# Top games in terms of play time
top_games_play_time = joined_df.filter(joined_df["mem_Type"] == "play")\
                               .groupBy("game_Name")\
                               .agg(sum("value").alias("total_play_time"))\
                               .orderBy(desc("total_play_time"))\
                               .toPandas()

print("\nTop 10 games by play time:")
print(top_games_play_time.head(10))
```

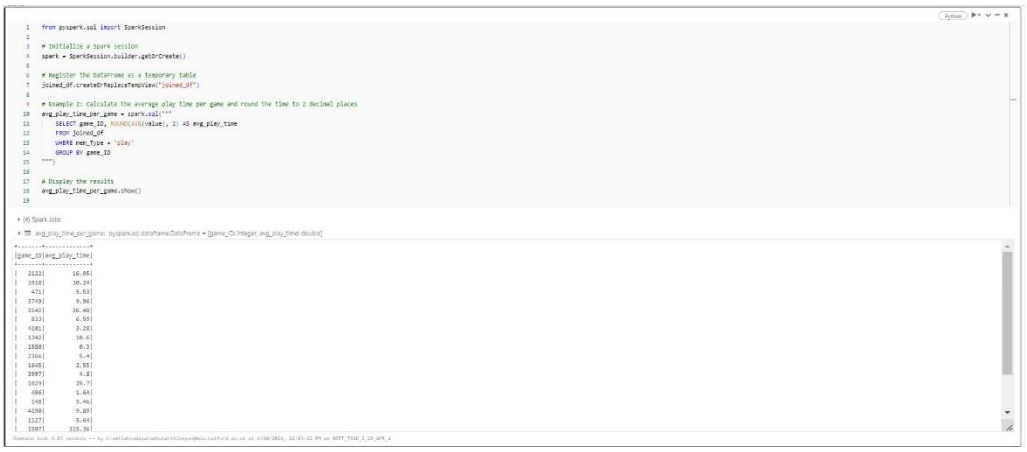
(6) Spark Jobs

Top 10 games by play time:

	game_Name	total_play_time
0	Dota 2	981684.6
1	Counter-Strike Global Offensive	322771.6
2	Team Fortress 2	173673.8
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	70889.3
7	Garry's Mod	49725.3
8	Call of Duty Modern Warfare 2 - Multiplayer	42009.9
9	Left 4 Dead 2	33596.7

Command took 5.08 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:50:48 AM on BDTT_TASK_2_2B_APR_1

Game IDs and average play time



```
from pyspark.sql import SparkSession

# Initialize a Spark session
spark = SparkSession.builder.getOrCreate()

# Register the DataFrame as a temporary table
joined_df.createOrReplaceTempView("joined_df")

# Example 2: Calculate the average play time per game and round the time to 2 decimal places
avg_play_time_per_game = spark.sql("""
    SELECT game_ID, ROUND(AVG(value), 2) AS avg_play_time
    FROM joined_df
    WHERE mem_Type = 'play'
    GROUP BY game_ID
""")

# Display the results
avg_play_time_per_game.show()
```

(4) Spark Jobs

avg_play_time_per_game: pyspark.sql.dataframe.DataFrame = [game_ID: integer, avg_play_time: double]

game_ID	avg_play_time
2122	16.95
3918	30.29
471	5.53
3749	9.96
2142	26.48
833	6.59
4101	3.28
1342	58.6
1828	0.3
2366	5.4
1645	2.55
1997	6.2
1829	25.7
496	1.64
148	5.96
4198	9.89
1127	5.64
1907	335.36

Command took 3.83 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 11:51:42 PM on BDTT_TASK_2_2B_APR_1

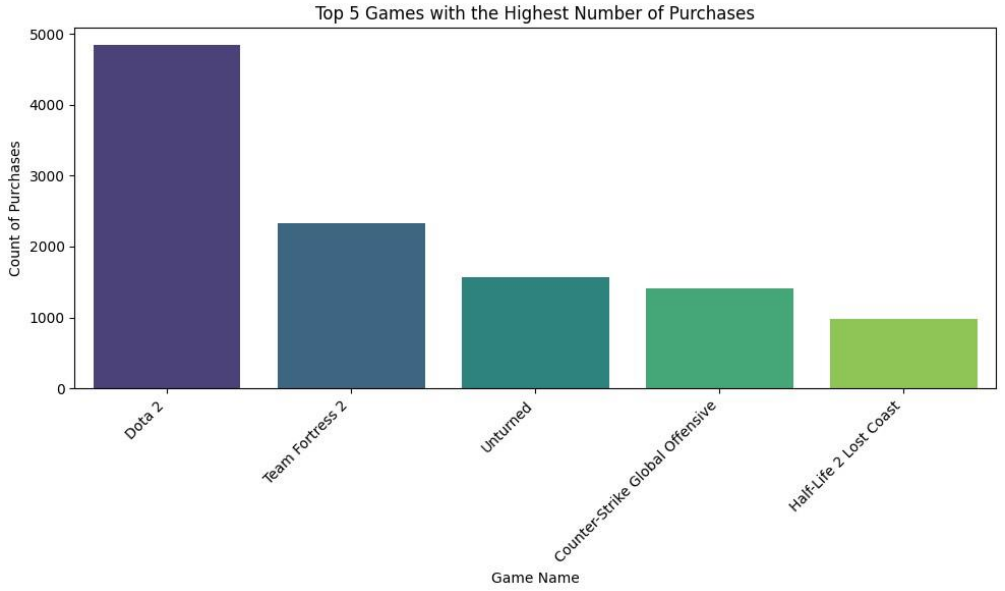
2.5 Visualization on Dataframes using Python Matplotlib, and Plotly

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

Table 2.4 Visualization using Python Matplotlib and Plotly

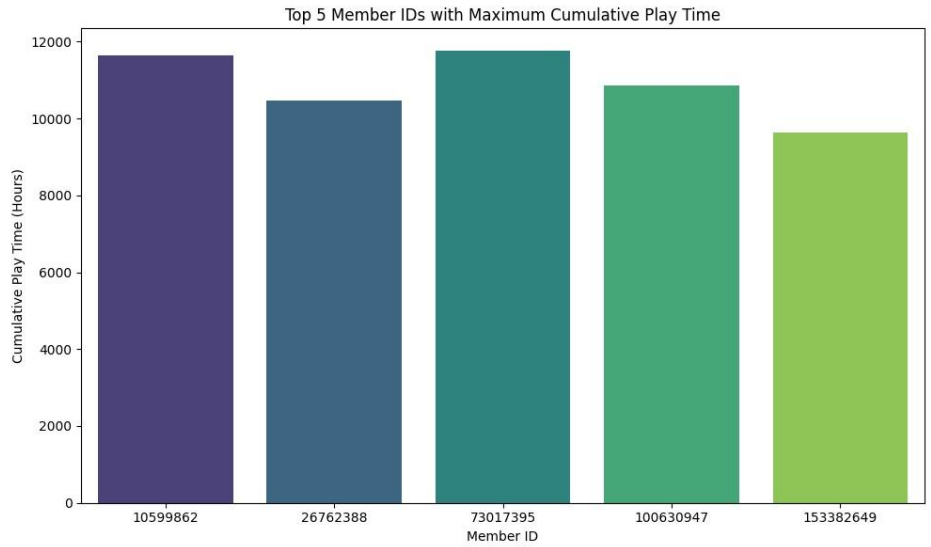
Top 5 games with highest number of purchases

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Convert the purchase_count_df DataFrame to a Pandas DataFrame
purchase_count_pandas_df = purchase_count_df.toPandas()

# Sort the DataFrame in descending order based on the count
purchase_count_pandas_df = purchase_count_pandas_df.sort_values(by='count', ascending=False)

# Select the top 5 rows (top 5 games with the highest number of purchases)
top_5_games = purchase_count_pandas_df.head(5)

# Plot a vertical bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='game_Name', y='count', data=top_5_games, palette='viridis')

# Set plot labels and title
plt.xlabel('Game Name')
plt.ylabel('Count of Purchases')
plt.title('Top 5 Games with the Highest Number of Purchases')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.tight_layout()
plt.show()
```



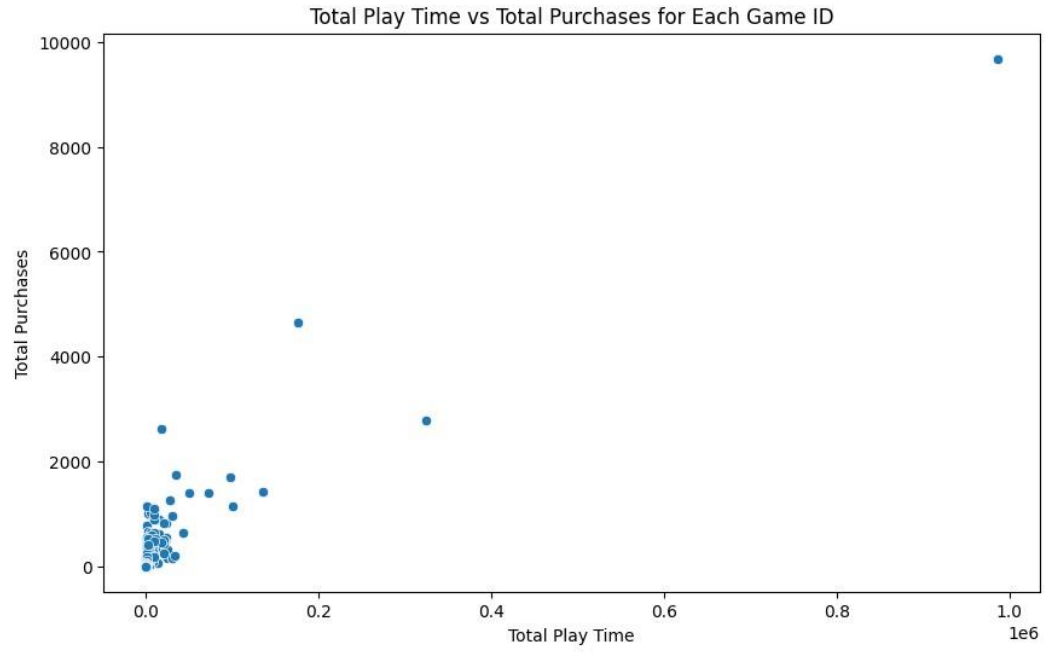
Top 5 users with maximum cumulative play time

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Convert the top 5 IDs DataFrame to a Pandas DataFrame
top_5_ids_pandas_df = top_5_ids_df.toPandas()

# Plot a horizontal bar chart
plt.figure(figsize=(10, 6))
sns.barplot(y='cumulative_play_time', x='ID', data=top_5_ids_pandas_df, palette='viridis')

# Set plot labels and title
plt.xlabel('Member ID')
plt.ylabel('Cumulative Play Time (Hours)')
plt.title('Top 5 Member IDs with Maximum Cumulative Play Time')

# Show the plot
plt.tight_layout()
plt.show()
```



Play Time versus Total Purchases

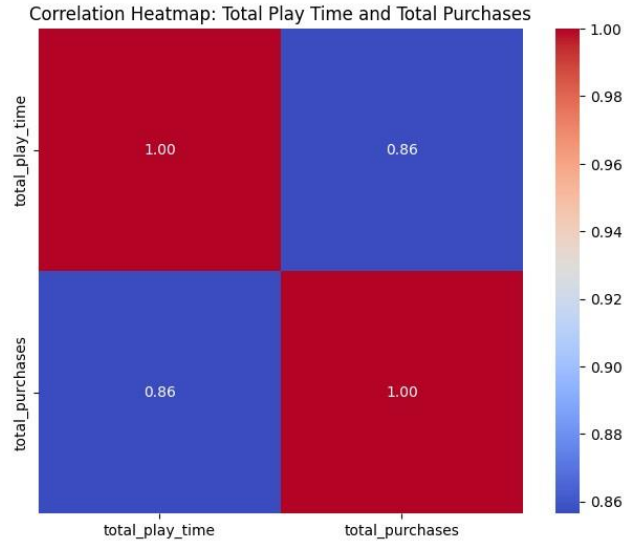
```
Cmd 36                                                                                    Python
# Visualize the relationship between total play time and total purchases for each game
play_vs_purchases_df = joined_df.groupBy("game_ID")\
                                .agg(sum("value").alias("total_play_time"), count("ID").alias("total_purchases"))

# Convert to Pandas for plotting
play_vs_purchases_df_pandas = play_vs_purchases_df.toPandas()

# Plot the relationship between total play time and total purchases
plt.figure(figsize=(10, 6))
sns.scatterplot(x='total_play_time', y='total_purchases', data=play_vs_purchases_df_pandas)
plt.xlabel("Total Play Time")
plt.ylabel("Total Purchases")
plt.title("Total Play Time vs Total Purchases for Each Game ID")
plt.show()
```



Correlation Heat map: Total Play time and Total Purchases

Cmd 37

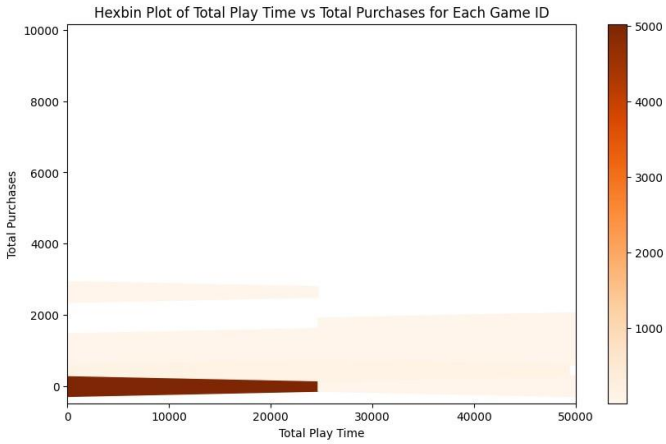
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from pyspark.sql.functions import sum, count

# Aggregate the data by game name and calculate total play time and total purchases
game_aggregated_df = joined_df.groupBy("game_Name")\
                              .agg(sum("value").alias("total_play_time"), count("ID").alias("total_purchases"))

# Convert the aggregated DataFrame to a Pandas DataFrame
game_aggregated_df_pandas = game_aggregated_df.toPandas()

# Calculate the correlation matrix between total play time and total purchases
correlation_matrix = game_aggregated_df_pandas[['total_play_time', 'total_purchases']].corr()

# Plot the correlation heat map
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True, fmt='.2f')
plt.title("Correlation Heatmap: Total Play Time and Total Purchases")
plt.show()

```



Total Play time versus Total Purchases for Each Game ID

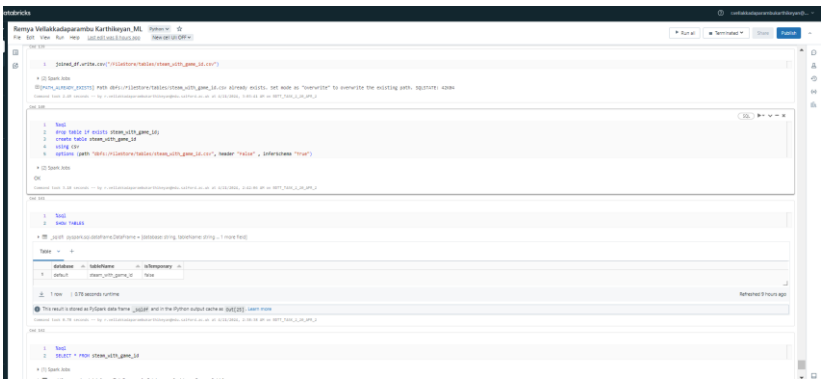
Cmd 38

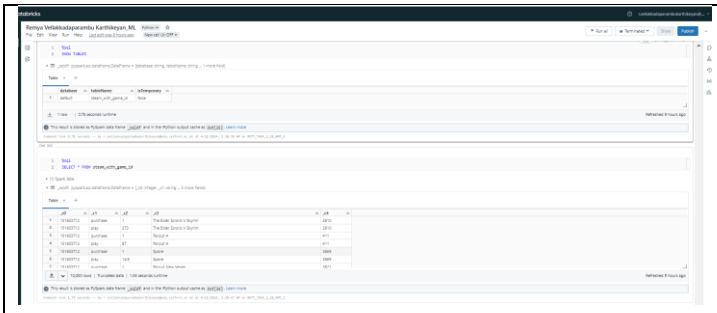
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Convert to Pandas for plotting
play_vs_purchases_df_pandas = play_vs_purchases_df.toPandas()

# Create a hexbin plot to visualize the relationship between total play time and total purchases
plt.figure(figsize=(10, 6))
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)

# Add a color bar to the plot for reference
plt.colorbar(hexbin_plot)

# Add labels and title
plt.xlabel("Total Play Time")
plt.ylabel("Total Purchases")
plt.title("Hexbin Plot of Total Play Time vs Total Purchases for Each Game ID")

# Set the x-axis range to display only up to 50,000
plt.xlim(0, 50000)

# Show the plot
plt.show()

```



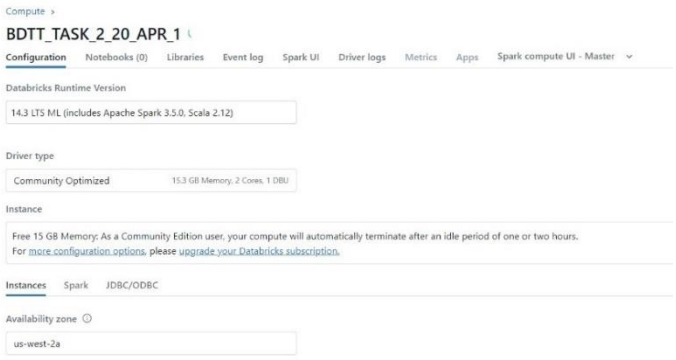
2.6 Visualization on Dataframes using Power BI

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

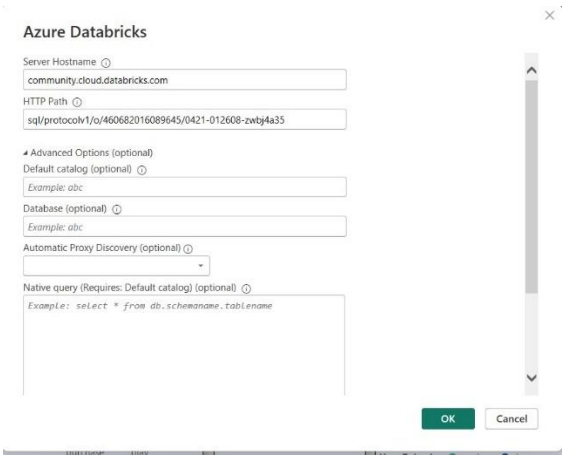
Saving the joined_df as csv file and then to spark SQL Delta table 
Table is saved in default database



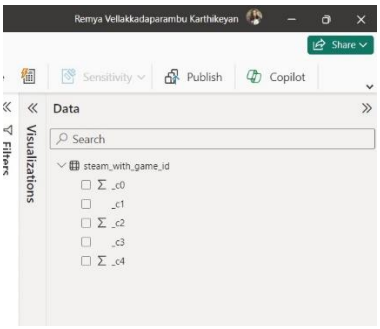
Getting the server hostname and http path from ML cluster



Copying the authentification values to the databricks interface in Power BI



The table loaded to the Power BI



Viewing sample data loaded to Power BI

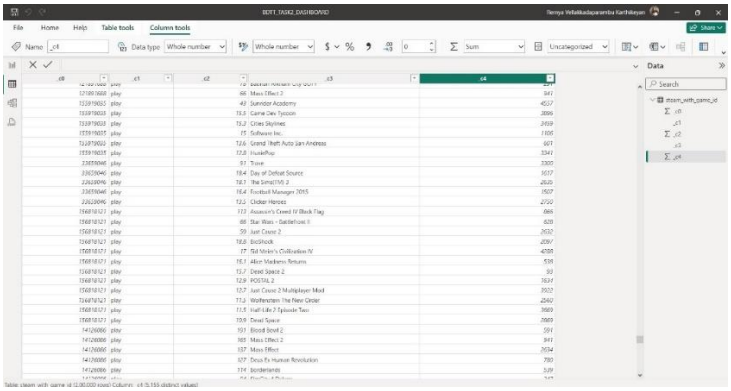


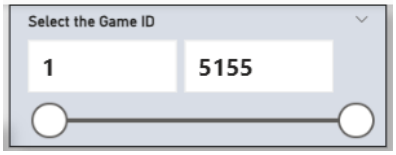
Figure 2.2 Loading data from databricks to Power BI

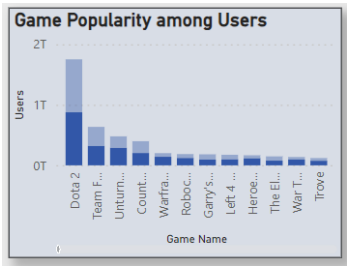
Figure 2.3 shows the dashboard created using Power BI on the data uploaded from Databricks

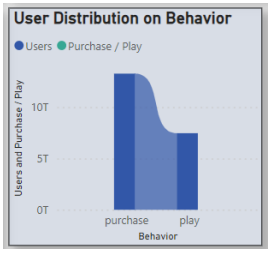
Figure 2.3 dashboard build on data loaded from databricks to Power BI

Analyzing each visulaization closer in Table 2.4.

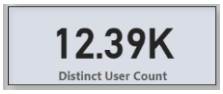
Table 2.4 Close up view of visualization using PowerBI

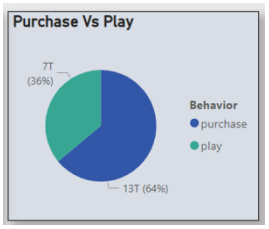
<table>
<tr>
<td> 

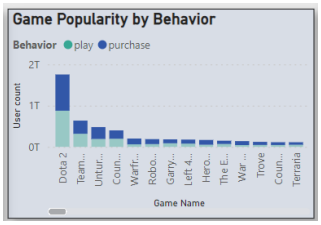
Slider to select game ID</td>
<td colspan="2"> 

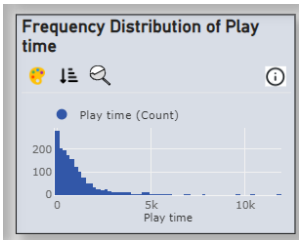
Popular games among users based on behavior (clustered column chart)</td>
</tr>
<tr>
<td rowspan="2"> 

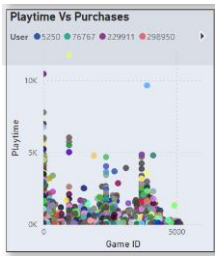
Ribbon chart shwoing user distribution based on behavior</td>
<td> 

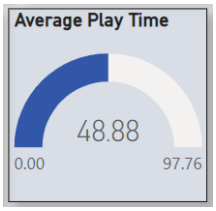
Total games count (card)</td>
<td> 

Total user count (card)</td>
</tr>
<tr>
<td colspan="2"></td>
</tr>
<tr>
<td> 

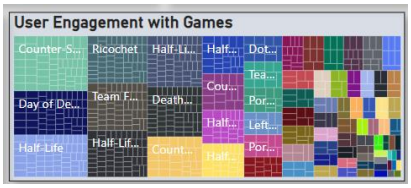
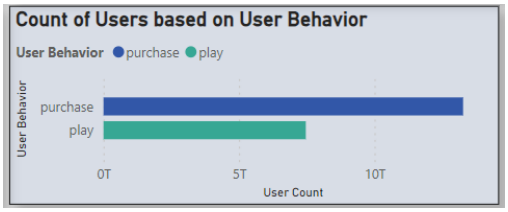
Pie chart showing purchase and play proportion</td>
<td colspan="2"> 

Stacked column chart showing popular games based on behavior</td>
</tr>
<tr>
<td> 

Histogram showing frequency distribution of playtime</td>
<td> 

Scatter plot showing playtime vs purchases of the games</td>
<td> 

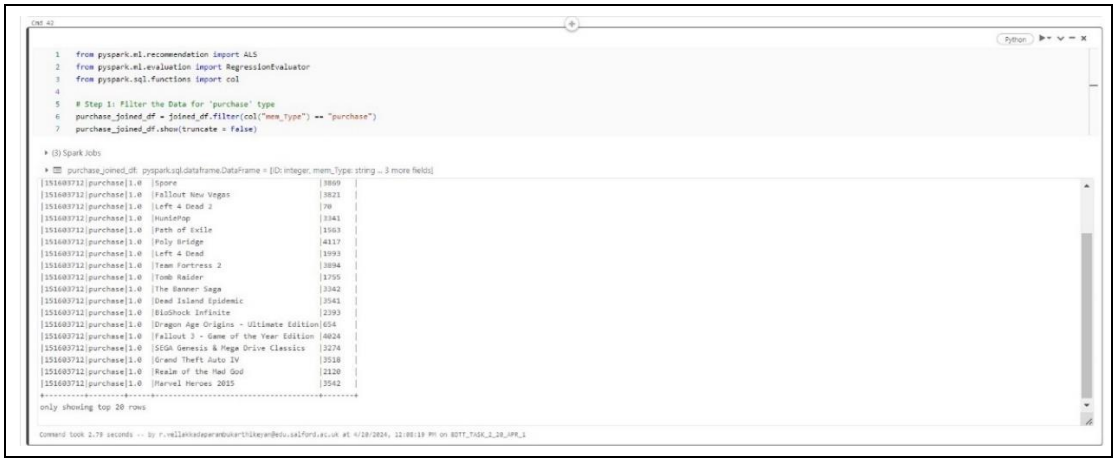
Gauge showing average playtime</td>
</tr>
</table>

	
Treemap showing user engagement	Stacked bar chart showing user count based on behavior

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



```
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col

# Step 1: Filter the Data for 'purchase' type
purchase_joined_df = joined_df.filter(col("mem_type") == "purchase")
purchase_joined_df.show(truncate = false)
```

```
(3) Spark Jobs
purchase_joined_df: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
|151603712|purchase|1.0|Spore                                    |3869|
|151603712|purchase|1.0|Fallout New Vegas                        |3821|
|151603712|purchase|1.0|Left 4 Dead 2                            |70  |
|151603712|purchase|1.0|HuniePop                                 |3341|
|151603712|purchase|1.0|Path of Exile                            |1563|
|151603712|purchase|1.0|Poly Bridge                              |4117|
|151603712|purchase|1.0|Left 4 Dead                              |1993|
|151603712|purchase|1.0|Team Fortress 2                          |3894|
|151603712|purchase|1.0|Tomb Raider                              |1795|
|151603712|purchase|1.0|The Banner Saga                          |3342|
|151603712|purchase|1.0|Dead Island Epidemic                     |3541|
|151603712|purchase|1.0|BioShock Infinite                        |2393|
|151603712|purchase|1.0|Dragon Age Origins - Ultimate Edition    |654 |
|151603712|purchase|1.0|Fallout 3 - Game of the Year Edition     |4024|
|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|
|151603712|purchase|1.0|Marvel Heroes 2015                       |3542|
only showing top 20 rows
Command took 2.79 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:08:19 PM on BDTT_TASK_2_20_APR_1
```

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

```
(training,test) = purchase_joined_df.randomSplit([0.8,0.2],seed = 100)
```

```
training: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
test: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
Command took 0.15 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:16:03 PM on BDTT_TASK_2_20_APR_1
```

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 metrics, 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, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (training dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details."
2024/04/20 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 allowlisted. Call mlflow.spark.log_model() to explicitly log the model, or specify a custom allowlist via the spark.mlflow.pysparkml.autolog.logModelAllowlistFile Spark conf (see mlflow.pyspark.ml.autolog docs for more info).

Command took 29.36 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:16:41 PM on BDTT_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
▸ predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

|    ID|mem_Type|value|                game_Name|game_ID|prediction|
|  5250|purchase|  1.0|Counter-Strike So...|     45|0.97753596|
|  5250|purchase|  1.0|Half-Life 2 Death...|    443| 0.9784805|
|  5250|purchase|  1.0|Half-Life Blue Shift|    446|0.97952604|
|  5250|purchase|  1.0|Team Fortress Cla...|   3747| 0.9805962|
| 76767|purchase|  1.0|Arma 2 Operation ...|   2562|0.99100848|
| 76767|purchase|  1.0|Call of Duty Mode...|   3851|0.95813698|
| 76767|purchase|  1.0|       Counter-Strike|   3690|0.91379315|
| 76767|purchase|  1.0|Counter-Strike Gl...|   2354|0.94207656|
| 76767|purchase|  1.0|         Day of Defeat|   1997| 0.9180697|
| 76767|purchase|  1.0|  Deathmatch Classic|   1972|0.91533568|
| 76767|purchase|  1.0|Half-Life Blue Shift|    446| 0.9454339|
| 76767|purchase|  1.0|Half-Life Opposin...|   1528| 0.9466351|
| 76787|purchase|  1.0|                Thief 2|   1532| 1.0119815|
|103360|purchase|  1.0|Counter-Strike Co...|   3600|0.95582018|
|103360|purchase|  1.0|Counter-Strike Co...|   2044|0.85071225|
|103360|purchase|  1.0|             Ricochet|    629| 0.9652074|
|181212|purchase|  1.0|         Half-Life 2|    346|  1.026925|
|181212|purchase|  1.0|Team Fortress Cla...|   3747| 1.0027679|

Command took 0.36 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:24:35 PM on BDTT_TASK_2_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
▸ predictions_sorted: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

|    ID|mem_Type|value|           game_Name|game_ID|prediction|
|309404240|purchase|  1.0|           Unturned|   3439| 0.9521162|
|309262400|purchase|  1.0|A.V.A - Alliance ...|   1879| 0.9600036|
|309262440|purchase|  1.0|     Team Fortress 2|   3894|0.87179154|
|309188905|purchase|  1.0|           Unturned|   3439| 0.9862124|
|309052991|purchase|  1.0|          Brawlhalla|    299| 1.0741915|
|309052991|purchase|  1.0|          Happy Wars|   2511| 1.0346134|
|308595132|purchase|  1.0|Counter-Strike Ne...|   1860| 0.8632668|
|308553033|purchase|  1.0|           theHunter|   4352| 1.0010802|
|308468736|purchase|  1.0|Metal War Online ...|   2400| 0.7742528|
|308305300|purchase|  1.0|The Lord of the R...|    835|0.99574447|
|307888442|purchase|  1.0|Metal War Online ...|   2400| 0.7390567|
|307351018|purchase|  1.0|  Marvel Heroes 2015|   3542| 0.8495149|
|306993682|purchase|  1.0|   Heroes & Generals|   1942|0.92803645|
|306993682|purchase|  1.0|             Warface|   1505| 0.9383248|
|306690580|purchase|  1.0|The Elder Scrolls...|     43|0.99141794|
|306617101|purchase|  1.0|           Unturned|   3439|0.91121715|
|306547522|purchase|  1.0|     Team Fortress 2|   3894|0.98988236|
|306547522|purchase|  1.0|             Warframe|   2220| 0.9974419|

Command took 9.48 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:25:54 PM on BDTT_TASK_2_20_APR_1
```

Predictions listed by ordering game Id in ascending order

```
Cmd 47

# Order the predictions DataFrame in ascending order of game_ID
ordered_predictions = predictions.orderBy("game_ID", ascending=True)

# Display the ordered DataFrame
ordered_predictions.show(truncate=False)
```

(5) Spark Jobs

ordered_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

ID	mem_Type	value	game_Name	game_ID	prediction
43160799	purchase	1.0	Dota 2	1	0.8325339
43360645	purchase	1.0	Dota 2	1	0.83884656
75689918	purchase	1.0	Dota 2	1	0.91335434
76787637	purchase	1.0	Dota 2	1	0.78468263
994489	purchase	1.0	Dota 2	1	0.8891226
43684632	purchase	1.0	Dota 2	1	0.8200301
44321815	purchase	1.0	Dota 2	1	0.8233659
37490443	purchase	1.0	Dota 2	1	0.8579175
12144171	purchase	1.0	Dota 2	1	0.82463825
44866715	purchase	1.0	Dota 2	1	0.91352737
47384303	purchase	1.0	Dota 2	1	0.78100336
298950	purchase	1.0	Dota 2	1	0.79777396
17530772	purchase	1.0	Dota 2	1	0.8120209
45974860	purchase	1.0	Dota 2	1	0.8255135
47457723	purchase	1.0	Dota 2	1	0.7585415
4834220	purchase	1.0	Dota 2	1	0.87821746
27579751	purchase	1.0	Dota 2	1	0.8998766
46534663	purchase	1.0	Dota 2	1	0.9181451

Command took 0.91 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:27:17 PM on BDTT_TASK_2_20_APR_1

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

```
from pyspark.sql.functions import desc, asc

# Order the predictions DataFrame based on ID in descending order and game_ID in ascending order
ordered_predictions = predictions.orderBy(desc("ID"), asc("game_ID"))

# Show the ordered predictions
ordered_predictions.show()
```

(5) Spark Jobs

ordered_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

ID	mem_Type	value	game_Name	game_ID	prediction
309404240	purchase	1.0	Unturned	3439	0.9521162
309262440	purchase	1.0	A.V.A - Alliance ...	1879	0.9600036
309262440	purchase	1.0	Team Fortress 2	3894	0.87179154
309188905	purchase	1.0	Unturned	3439	0.9862124
309052991	purchase	1.0	Brawlhalla	299	1.0741915
309052591	purchase	1.0	Happy Wars	2511	1.0346134
308695132	purchase	1.0	Counter-Strike Ne...	1860	0.8632668
308653033	purchase	1.0	theHunter	4352	1.0010802
308468736	purchase	1.0	Metal War Online ...	2400	0.7742528
308305300	purchase	1.0	The Lord of the R...	835	0.99574447
307688442	purchase	1.0	Metal War Online ...	2400	0.7390567
307351018	purchase	1.0	Marvel Heroes 2015	3542	0.8495149
306993682	purchase	1.0	Warface	1505	0.9383248
306993682	purchase	1.0	Heroes & Generals	1942	0.92803645
306690580	purchase	1.0	The Elder Scrolls...	43	0.99141794
306617101	purchase	1.0	Unturned	3439	0.91121715
306547522	purchase	1.0	Warframe	2220	0.9974415
306547522	purchase	1.0	Team Fortress 2	3894	0.98980236

Command took 8.13 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:37:25 PM on BDTT_TASK_2_20_APR_1

Evaluating RMSE and MAE

Cmd 49

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName = "rmse",labelCol = "value",predictionCol = "prediction")
rmse = evaluator.evaluate(predictions)
print( " Root mean square error is = %g" %rmse)
# Create a RegressionEvaluator for MAE
evaluator_mae = RegressionEvaluator(metricName="mae", labelCol="value", predictionCol="prediction")
mae = evaluator_mae.evaluate(predictions)
print(f"Mean absolute error (MAE) is: {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 BDTT_TASK_2_20_APR_1

Predictions ordered in descending order

```
# Order the predictions DataFrame based on prediction in descending order
ordered_predictions = predictions.orderBy("prediction", ascending=False)

# Show the ordered predictions
ordered_predictions.show(truncate=False)
```

▸ (5) Spark Jobs

▸ ordered_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

ID	mem_Type	value	game_Name	game_ID	prediction
89005236	purchase	1.0	Star Trek D-A-C	1842	1.6195754
14544587	purchase	1.0	Star Trek D-A-C	1842	1.5327085
197949629	purchase	1.0	Might & Magic Heroes Online	775	1.3611796
203650036	purchase	1.0	Might & Magic Heroes Online	775	1.252822
18604016	purchase	1.0	Football Manager 2012	3773	1.2432587
202894728	purchase	1.0	Destination Sol	4130	1.2335975
40080833	purchase	1.0	RollerCoaster Tycoon Deluxe	4123	1.2223594
10599862	purchase	1.0	Pro Evolution Soccer 2015	1224	1.2104812
305130721	purchase	1.0	Might & Magic Heroes Online	775	1.1860325
42849279	purchase	1.0	Football Manager 2011	233	1.1849642
118693656	purchase	1.0	A Virus Named TOM	516	1.1748672
175450855	purchase	1.0	Praetorians	1080	1.1743008
243664576	purchase	1.0	Sheltered	171	1.158559
74764496	purchase	1.0	Assetto Corsa	1932	1.1546407
44866715	purchase	1.0	Transformers Fall of Cybertron	3693	1.1545993
179377677	purchase	1.0	Enclave	4353	1.140118
113423671	purchase	1.0	Metro 2033	3434	1.1396382
130758906	purchase	1.0	Super Killer Hornet Resurrection	1260	1.1395532

Command took 7.56 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:41:26 PM on BDTT_TASK_2_20_APR_1

Predictions ordered in ascending order

Cmd 51

```
# Drop rows with missing values
predictions = predictions.dropna()

# Order the predictions DataFrame based on prediction in descending order
ordered_predictions = predictions.orderBy("prediction", ascending=False)

# Show the ordered predictions
ordered_predictions.show(truncate=False)
```

▸ (5) Spark Jobs

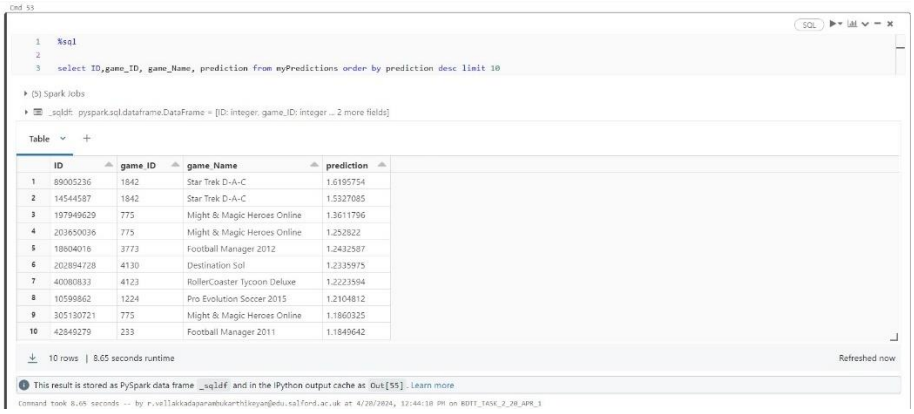
▸ predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

▸ ordered_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

ID	mem_Type	value	game_Name	game_ID	prediction
89005236	purchase	1.0	Star Trek D-A-C	1842	1.6195754
14544587	purchase	1.0	Star Trek D-A-C	1842	1.5327085
197949629	purchase	1.0	Might & Magic Heroes Online	775	1.3611796
203650036	purchase	1.0	Might & Magic Heroes Online	775	1.252822
18604016	purchase	1.0	Football Manager 2012	3773	1.2432587
202894728	purchase	1.0	Destination Sol	4130	1.2335975
40080833	purchase	1.0	RollerCoaster Tycoon Deluxe	4123	1.2223594
10599862	purchase	1.0	Pro Evolution Soccer 2015	1224	1.2104812
305130721	purchase	1.0	Might & Magic Heroes Online	775	1.1860325
42849279	purchase	1.0	Football Manager 2011	233	1.1849642
118693656	purchase	1.0	A Virus Named TOM	516	1.1748672
175450855	purchase	1.0	Praetorians	1080	1.1743008
243664576	purchase	1.0	Sheltered	171	1.158559
74764496	purchase	1.0	Assetto Corsa	1932	1.1546407
44866715	purchase	1.0	Transformers Fall of Cybertron	3693	1.1545993
179377677	purchase	1.0	Enclave	4353	1.140118
113423671	purchase	1.0	Metro 2033	3434	1.1396382
130758906	purchase	1.0	Super Killer Hornet Resurrection	1260	1.1395532

Command took 7.71 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:42:22 PM on BDTT_TASK_2_20_APR_1

SQL queries on predictions based on temporary view



```
%sql

select ID,game_ID, game_Name, prediction from myPredictions 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	89005236	1842	Star Trek D-A-C	1.6195754
2	14544587	1842	Star Trek D-A-C	1.5327085
3	197949629	775	Might & Magic Heroes Online	1.3611796
4	203650036	775	Might & Magic Heroes Online	1.252822
5	18604016	3773	Football Manager 2012	1.2432587
6	202894728	4130	Destination Sol	1.2335975
7	40080833	4123	RollerCoaster Tycoon Deluxe	1.2223594
8	10599862	1224	Pro Evolution Soccer 2015	1.2104812
9	305130721	775	Might & Magic Heroes Online	1.1860325
10	42849279	233	Football Manager 2011	1.1849642

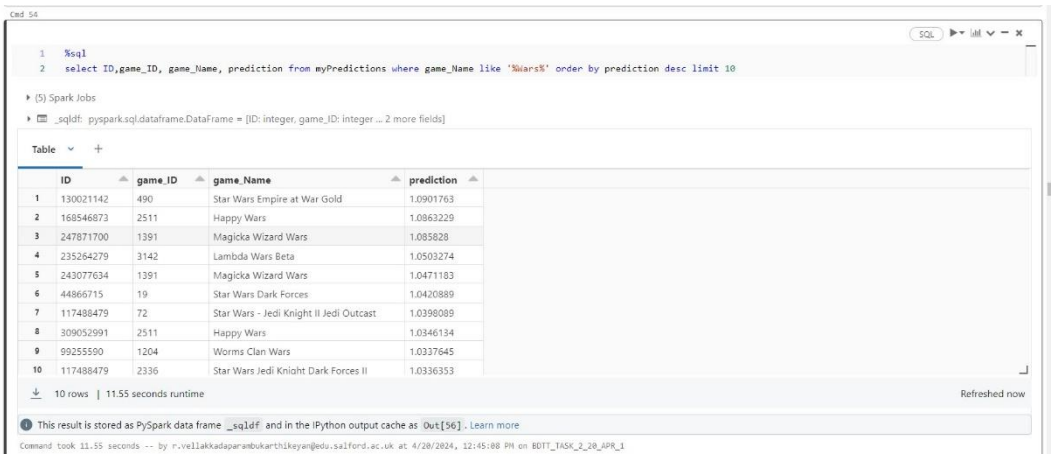
10 rows | 8.65 seconds runtime

Refreshed now

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[55]`. Learn more

Command took 8.65 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:44:10 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



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

Table

	ID	game_ID	game_Name	prediction
1	130021142	490	Star Wars Empire at War Gold	1.0901763
2	168546873	2511	Happy Wars	1.0863229
3	247871700	1391	Magicka Wizard Wars	1.085828
4	235264279	3142	Lambda Wars Beta	1.0503274
5	243077634	1391	Magicka Wizard Wars	1.0471183
6	44866715	19	Star Wars Dark Forces	1.0420889
7	117488479	72	Star Wars - Jedi Knight II Jedi Outcast	1.0398089
8	309052991	2511	Happy Wars	1.0346134
9	99255590	1204	Worms Clan Wars	1.0337645
10	117488479	2336	Star Wars Jedi Knight Dark Forces II	1.0336353

10 rows | 11.55 seconds runtime

Refreshed now

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[56]`. Learn more

Command took 11.55 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:45:08 PM on BDTT_TASK_2_20_APR_1

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

Cmd 55

```
%sql
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	24	Robotex	1.0216267
2	58977564	18	Call of Duty Modern Warfare 2 - Multiplayer	1.0179827
3	63263519	18	Call of Duty Modern Warfare 2 - Multiplayer	1.0179724
4	232207754	24	Robotex	1.0093927
5	86912006	24	Robotex	1.0091228
6	102825821	24	Robotex	1.0047473
7	57798235	18	Call of Duty Modern Warfare 2 - Multiplayer	1.0031015
8	54999247	18	Call of Duty Modern Warfare 2 - Multiplayer	1.0024006
9	21861124	18	Call of Duty Modern Warfare 2 - Multiplayer	1.0017794
10	71704418	18	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, 8:30:30 PM on BDTT_TASK2_19_APR_2024

Function giving recommendation filtered by game name

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

Command took 0.06 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:51:52 PM on BDTT_TASK_2_20_APR_1

Cmd 59

```
recommendationsByGameName('Duty')
```

(5) Spark Jobs

```
+---------+--------+-----+-----------------------------------------------+-------+----------+
|ID       |mem_Type|value|game_Name                                      |game_ID|prediction|
+---------+--------+-----+-----------------------------------------------+-------+----------+
|302916505|purchase|1.0  |Call of Duty World at War                      |2379   |1.0478029 |
|2643609  |purchase|1.0  |Commandos Beyond the Call of Duty              |583    |1.0333916 |
|92103372 |purchase|1.0  |Call of Duty Modern Warfare 3 - Multiplayer|2029   |1.0239547 |
|18036199 |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                            |2049   |1.014595  |
|67349307 |purchase|1.0  |Commandos Beyond the Call of Duty              |583    |1.0132535 |
|109458201|purchase|1.0  |Call of Duty World at War                      |2379   |1.0120603 |
+---------+--------+-----+-----------------------------------------------+-------+----------+
```

Command took 7.31 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:49:29 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by game ID

Cmd 60

```
#Function give recommendatons filtered by game id
def recommendationsByGameID(game_IDs):
    # Filter ordered predictions based on provided game_IDs
    filtered_predictions = ordered_predictions.filter(ordered_predictions.game_ID.isin(game_IDs))

    # Order by prediction in descending order and limit to top 10 results
    top_recommendations = filtered_predictions.orderBy("prediction", ascending=False).limit(10)

    # Show the top recommendations without truncation
    top_recommendations.show(truncate=False)

    # Return the DataFrame containing the top recommendations
    return top_recommendations
```

Command took 0.10 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:51:44 PM on BDTT_TASK_2_20_APR_1

Cmd 61

```
recommendationsByGameID(1)
```

(5) Spark Jobs

```
+---------+--------+-----+---------+-------+----------+
|ID       |mem_Type|value|game_Name|game_ID|prediction|
+---------+--------+-----+---------+-------+----------+
|232082615|purchase|1.0  |Dota 2   |1      |0.9867617 |
|262217260|purchase|1.0  |Dota 2   |1      |0.9867617 |
|235290543|purchase|1.0  |Dota 2   |1      |0.9867617 |
|197732882|purchase|1.0  |Dota 2   |1      |0.9734962 |
|188366796|purchase|1.0  |Dota 2   |1      |0.9734962 |
|237581229|purchase|1.0  |Dota 2   |1      |0.9734962 |
|270059336|purchase|1.0  |Dota 2   |1      |0.96982646|
|205339310|purchase|1.0  |Dota 2   |1      |0.96187985|
|112134832|purchase|1.0  |Dota 2   |1      |0.96187985|
|215832158|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 8.15 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:51:58 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by user ID

Cmd 62

```
#Function give recommendatons filtered by user id
def recommendationsByUserID(ID):
    # Filter ordered predictions based on provided user ID
    user_recommendations = ordered_predictions.filter(ordered_predictions.ID == ID)

    # Order the filtered results by prediction in descending order
    ordered_user_recommendations = user_recommendations.orderBy("prediction", ascending=False)

    # Show all the recommendations for the user ID without truncation
    ordered_user_recommendations.show(truncate=False)

    # Return the DataFrame containing the recommendations for the user ID
    return ordered_user_recommendations
```

Command took 0.06 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:53:42 PM on BDTT_TASK_2_20_APR_1

Cmd 63

```
recommendationsByUserID(59945701)
```

(5) Spark Jobs

```
+--------+--------+-----+-----------------------------------+-------+----------+
|ID      |mem_Type|value|game_Name                          |game_ID|prediction|
+--------+--------+-----+-----------------------------------+-------+----------+
|59945701|purchase|1.0  |Psychonauts                        |1967   |1.0081882 |
|59945701|purchase|1.0  |Cities in Motion 2                 |2731   |1.006336  |
|59945701|purchase|1.0  |Orcs Must Die!                     |3213   |1.0043743 |
|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   |0.9993608 |
|59945701|purchase|1.0  |Fallout New Vegas Dead Money       |3497   |0.9976061 |
|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@edu.salford.ac.uk at 4/20/2024, 12:55:16 PM on BDTT_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

```
userRecs.show(truncate=False)
```

▸ (2) Spark Jobs

```
+--------+----------------------------------------------------------------------------------------------------------------------------------
-----------------+
|ID      |recommendations
|
+--------+----------------------------------------------------------------------------------------------------------------------------------
-----------------+
|76767   |[{4905, 1.5527669}, {1842, 1.5277522}, {1829, 1.2875135}, {2523, 1.1803228}, {4228, 1.1749731}, {775, 1.1496066}, {518, 1.1320826}, {4094, 1.1023601}, {3609, 1.0937335}, {4
661, 1.0923998}] |
|144736  |[{4905, 1.7227709}, {1842, 1.7155082}, {1829, 1.3530054}, {518, 1.2394433}, {2523, 1.2377787}, {4228, 1.2035427}, {775, 1.1891276}, {951, 1.1683924}, {2957, 1.1680532}, {33
92, 1.1634471}]  |
|229911  |[{4905, 1.6113563}, {1842, 1.6015139}, {1829, 1.309021}, {4228, 1.2287848}, {2523, 1.2110366}, {775, 1.19924}, {518, 1.1802309}, {2957, 1.1410441}, {3110, 1.1392164}, {360
9, 1.1381712}]     |
|835015  |[{1842, 1.7261893}, {4905, 1.7173665}, {1829, 1.3506272}, {2523, 1.234631}, {518, 1.234618}, {4228, 1.1937861}, {775, 1.1763262}, {2957, 1.1623528}, {3392, 1.1609262}, {95
1, 1.1587856}]     |
|948368  |[{4905, 1.6697531}, {1842, 1.5981476}, {1829, 1.2944885}, {4228, 1.2025548}, {518, 1.1951764}, {775, 1.1813927}, {2523, 1.1749778}, {4661, 1.1508467}, {951, 1.1468489}, {36
09, 1.1318731}]  |
|975449  |[{1842, 1.7232509}, {4905, 1.5228887}, {1829, 1.3044542}, {2523, 1.1808691}, {518, 1.1193247}, {3392, 1.0912535}, {2957, 1.0755268}, {4094, 1.0619423}, {3110, 1.0530982},
{3437, 1.0524395}]|
|1268792 |[{4905, 1.720392}, {1842, 1.7194513}, {1829, 1.3480662}, {518, 1.2338754}, {2523, 1.2318735}, {4228, 1.1968924}, {775, 1.1799517}, {951, 1.160999}, {2957, 1.1603899}, {339
2, 1.1591319}]     |
|2531540 |[{4905, 1.6629276}, {1842, 1.6524205}, {1829, 1.2555833}, {775, 1.2145339}, {518, 1.203613}, {4228, 1.1838813}, {2523, 1.1808993}, {3609, 1.1664426}, {3110, 1.1594594}, {29
```

Command took 31.45 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 12:57:22 PM on BDTT_TASK_2_20_APR_1

Recommendations for specific user

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

# Function to get game recommendations for a specific user ID and join with game details
def gameRecommendationsForUser(user_ID):
    # Filter the recommendations for the specific user ID from userRecs
    user_recommendations = userRecs.filter(col("ID") == user_ID)

    # Explode the recommendations column to work with each recommendation separately
    user_recommendations_exploded = user_recommendations.select("ID", explode("recommendations").alias("recommendation"))

    # Join the exploded recommendations with the distinct games DataFrame
    # Use the game_ID from the recommendations and the distinct_games_df DataFrame to join
    recommended_games = user_recommendations_exploded.join(distinct_games_df,
                                                           user_recommendations_exploded["recommendation.game_ID"] == distinct_games_df["game_ID"],
                                                           how="inner")\
                                                      .select("game_ID", "game_Name", "recommendation.rating")\
                                                      .orderBy(col("recommendation.rating").desc())

    # Display the recommended games with their ratings (predictions)
    recommended_games.show(truncate=False)

    return recommended_games

# Call the function for a specific user ID (e.g., user ID = 1)
gameRecommendationsForUser(76767)

```

Cmd 67

```python
gameRecommendationsForUser(109323647)

```

▸ (4) Spark Jobs

```
+-------+-----------------------------------------+---------+
|game_ID|game_Name                                |rating   |
+-------+-----------------------------------------+---------+
|1842   |Star Trek D-A-C                          |1.8329778|
|4905   |Duke Nukem Forever                       |1.6623802|
|1829   |Construction-Simulator 2015              |1.4040481|
|2523   |Construction Simulator 2015 Vertical Skyline|1.2796733|
|518    |Cities in Motion                         |1.2415711|
|4228   |The LEGO Movie - Videogame               |1.2242478|
|3392   |SONIC THE HEDGEHOG 4 Episode II          |1.2024618|
|3110   |Railroad Tycoon 2 Platinum               |1.1917119|
|4006   |Freedom Fall                             |1.1873692|
|2957   |Tank Universal                           |1.187072 |
+-------+-----------------------------------------+---------+

DataFrame[game_ID: int, game_Name: string, rating: float]
```

Command took 25.75 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/19/2024, 8:30:30 PM on BDTT_TASK2_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

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

Step 3: Train the ALS model

MaxIter = 5, regParam = 0.01

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

```
# Use the trained model (play_model) to generate predictions on the test data (test_play)
play_predictions = play_model.transform(test_play).dropna()

# Display the predictions without truncation
play_predictions.show()
```

(5) Spark Jobs

play_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

```
+------+--------+-----+--------------------+-------+----------+
|    ID|mem_Type|value|           game_Name|game_ID|prediction|
+------+--------+-----+--------------------+-------+----------+
|  5250|    play| 13.6|            Portal 2|   1018|  30.695309|
| 76767|    play|  3.4|Thief Deadly Shadows|   3444|  15.534629|
| 76767|    play| 12.5|Call of Duty Blac...|   2308|  607.75946|
| 76767|    play| 25.0|Counter-Strike So...|     45| -347.70978|
| 76767|    play|365.0|      Counter-Strike|   3690| -483.53235|
|298950|    play|  0.1|Deus Ex Game of t...|   3077|  7.0383816|
|298950|    play|  0.2|Dragon Age Origin...|    654|  361.30627|
|298950|    play|  0.2|Euro Truck Simula...|   3079|  -53.969936|
|298950|    play|  0.2|Octodad Dadliest ...|   1019| -0.83700967|
|298950|    play|  0.2|         PROTOTYPE 2|   3870|  43.145355|
|298950|    play|  0.2|Scribblenauts Unl...|   2564|  27.161074|
|298950|    play|  0.2|      Shadow Warrior|   1330|  13.162838|
|298950|    play|  0.4|          Insurgency|   2444|  10.508054|
|298950|    play|  0.4|Wasteland 2 Direc...|   1208|  4.8681283|
|298950|    play|  0.5|      Counter-Strike|   3690|  466.48828|
|298950|    play|  0.6|Age of Empires II...|    246| -28.974716|
|298950|    play|  0.7|      Alpha Protocol|     48|  5.0887914|
|298950|    play|  0.8|   Surgeon Simulator|   4354|  4.9575592|
+------+--------+-----+--------------------+-------+----------+
Command took 6.04 seconds -- by r.vellakkadeparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:32:30 PM on BDTT_TASK_2_20_APR_1
```

Step 5: Explore some of the resulting recommendations

Displaying sorted predictions

```
from pyspark.sql.functions import desc

# Order the play_predictions DataFrame in descending order of ID
play_predictions_sorted = play_predictions.orderBy(desc("ID"))

# Display the sorted DataFrame without truncation
play_predictions_sorted.show()
```

(5) Spark Jobs

play_predictions_sorted: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

```
|308760273|    play|  0.6|      Team Fortress 2|   3894|     5.48493|
|308091132|    play|  1.5|          Brawlhalla|    299|-0.02836595|
|308668736|    play|  0.7|         Magic Duels|   3282|-0.01731492|
|307915393|    play| 10.1|Counter-Strike Gl...|   2354|  -0.7895508|
|307674833|    play|  0.4|      Codename CURE|    633| 0.024094163|
|306971738|    play|  0.3|Aveyond 3 2 Gates...|   3735| 0.05308074|
|306971738|    play|  0.5|            Lineage II|    173|  2.041676 6|
|306547522|    play|  0.5|      Team Fortress 2|   3894|  2.3096203|
|306042038|    play|  0.1|Red Crucible Fire...|   3924| 0.03777231|
|304971849|    play|  0.2|Dungeon Defenders II|   1533|   3.1792054|
|304153788|    play|  7.0|      Team Fortress 2|   3894|   2.8980768|
|304224083|    play|  0.2|            Unturned|   3459|   0.4173023|
|304037281|    play|  0.2|          Carpe Diem|   3254|  0.13795693|
|303789064|    play|  0.5|           Undertale|   3979|   -8.555284|
|303467300|    play|  2.5|       Assetto Corsa|   1932|   -0.5633089|
|303326013|    play|  0.2|             WARMODE|   1360|  0.15216118|
|303129589|    play|  0.9|Age of Mythology ...|   2773|  0.94343334|
|303007171|    play|  0.7|         Half-Life 2|    346|-0.03820603|
+---------+--------+-----+--------------------+-------+----------+
only showing top 20 rows
```

Predictions listed by ordering game Id in ascending order

```
# Order the play_predictions DataFrame in ascending order of game_ID
ordered_play_predictions = play_predictions.orderBy("game_ID", ascending=True)

# Display the ordered DataFrame without truncation
ordered_play_predictions.show(truncate=False)
```

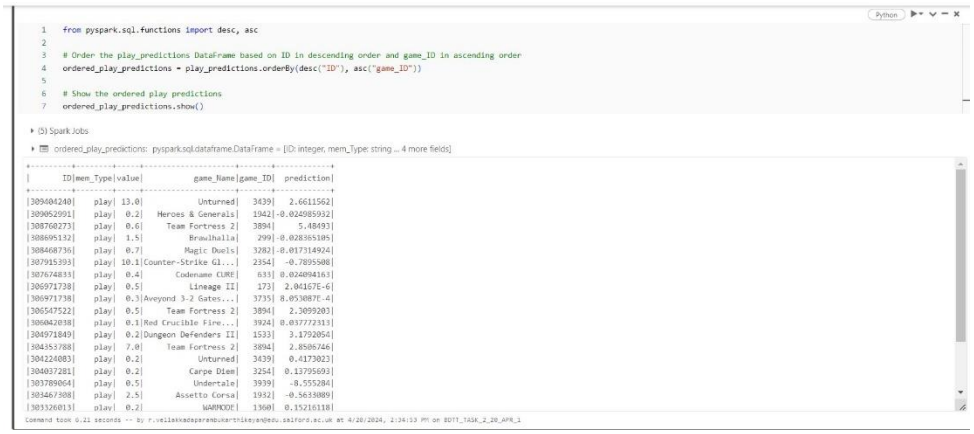
(5) Spark Jobs

ordered_play_predictions: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

```
+---------+--------+-----+---------+-------+----------+
|ID       |mem_Type|value|game_Name|game_ID|prediction|
+---------+--------+-----+---------+-------+----------+
|58893462 |play    |0.2  |Dota 2   |1      |4.5671167 |
|30280104 |play    |1.5  |Dota 2   |1      |-3.1818922|
|86287245 |play    |2.6  |Dota 2   |1      |-4.1594954|
|43360645 |play    |2390.0|Dota 2  |1      |675.4644  |
|59234901 |play    |860.0|Dota 2   |1      |92.27808  |
|32467994 |play    |11.8 |Dota 2   |1      |101.93029 |
|36557643 |play    |160.0|Dota 2   |1      |-17.724874|
|4824107  |play    |91.0 |Dota 2   |1      |-1289.0221|
|59025286 |play    |7.6  |Dota 2   |1      |44.35505  |
|8585433  |play    |33.0 |Dota 2   |1      |378.38928 |
|51822361 |play    |329.0|Dota 2   |1      |-12.014576|
|38465050 |play    |0.2  |Dota 2   |1      |-4.4240103|
|61772065 |play    |22.0 |Dota 2   |1      |201.91684 |
|8865447  |play    |1.3  |Dota 2   |1      |0.4852721 |
|19062398 |play    |1319.0|Dota 2  |1      |-29.375963|
|12529679 |play    |0.5  |Dota 2   |1      |559.71967 |
|61806811 |play    |1.8  |Dota 2   |1      |156.96729 |
|35701646 |play    |764.0|Dota 2   |1      |599.95795 |
```

Command took 6.57 seconds -- by r.vellakkadeparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:34:04 PM on BDTT_TASK_2_20_APR_1

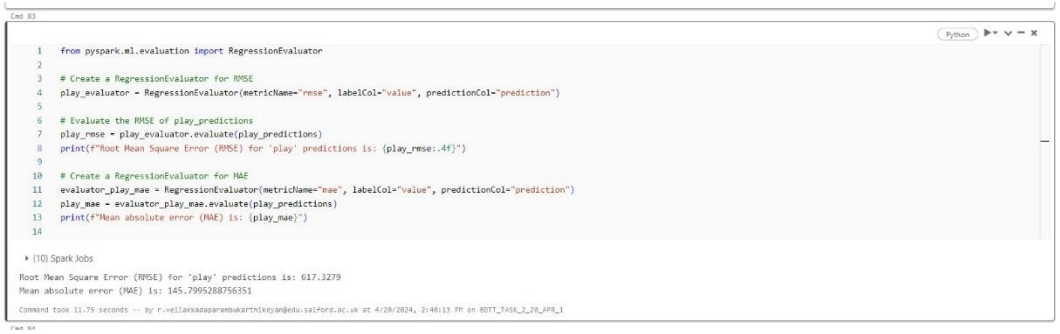
Ordering the predictions based on user ID (descending) and game ID (ascending)



```
from pyspark.sql.functions import desc, asc

# Order the play_predictions DataFrame based on ID in descending order and game_ID in ascending order
ordered_play_predictions = play_predictions.orderBy(desc("ID"), asc("game_ID"))

# Show the ordered play predictions
ordered_play_predictions.show()
```

```
(5) Spark Jobs
ordered_play_predictions:  pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+---------+--------+-----+--------------------+-------+------------+
|       ID|mem_Type|value|           game_Name|game_ID|  prediction|
+---------+--------+-----+--------------------+-------+------------+
|309404240|    play| 13.0|            Unturned|   3439|   2.6611562|
|309052991|    play|  0.2|   Heroes & Generals|   1942|-0.024985932|
|308760273|    play|  0.6|      Team Fortress 2|   3894|     5.48493|
|308695132|    play|  1.5|           Brawlhalla|    299|-0.028365105|
|308468736|    play|  0.7|         Magic Duels|   3282|-0.017314924|
|307915393|    play| 10.1|Counter-Strike Gl...|   2354|  -0.7895508|
|307674833|    play|  0.4|        Codename CURE|    833| 0.024094163|
|306971738|    play|  0.5|          Lineage II|    173|  2.04167E-6|
|306971738|    play|  0.3|Aveyond 3-2 Gates...|   3735| 8.053087E-4|
|306547522|    play|  0.5|      Team Fortress 2|   3894|   2.3099203|
|306042038|    play|  0.1|Red Crucible Fire...|   3924| 0.037772313|
|304971849|    play|  0.2|Dungeon Defenders II|   1533|   3.1792054|
|304157788|    play|  7.0|      Team Fortress 2|   3894|   2.8505746|
|304124083|    play|  0.2|            Unturned|   3439|  0.4173023|
|304037281|    play|  0.2|           Carpe Diem|   3254|  0.13795693|
|303789064|    play|  0.5|           Undertale|   3939|   -8.555284|
|303467308|    play|  2.5|       Assetto Corsa|   1932|  -0.5633089|
|303326013|    play|  0.2|            WARMODE|   1360|  0.15216118|
+---------+--------+-----+--------------------+-------+------------+
Command took 0.21 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:34:13 PM on BDTT_TASK_2_20_APR_1
```

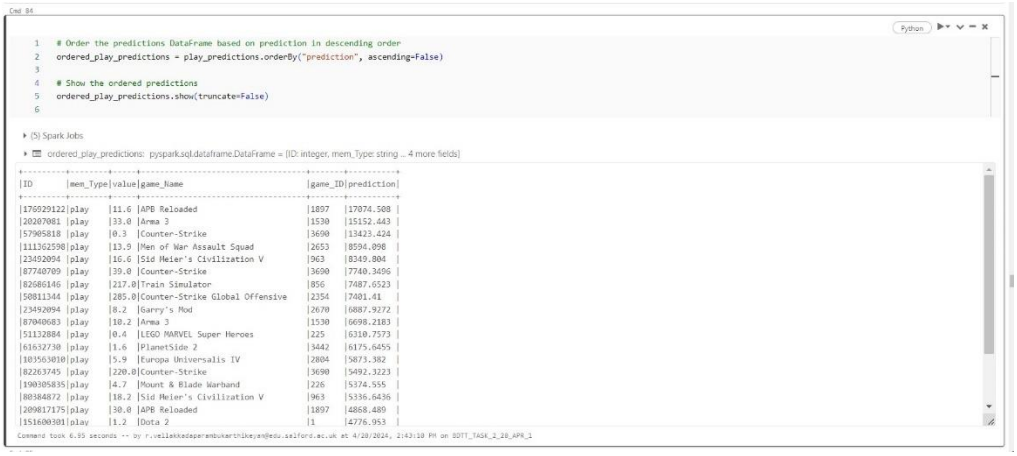
Evaluating RMSE and MAE



```
from pyspark.ml.evaluation import RegressionEvaluator

# Create a RegressionEvaluator for RMSE
play_evaluator = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")

# Evaluate the RMSE of play_predictions
play_rmse = play_evaluator.evaluate(play_predictions)
print(f"Root Mean Square Error (RMSE) for 'play' predictions is: {play_rmse:.4f}")

# Create a RegressionEvaluator for MAE
evaluator_play_mae = RegressionEvaluator(metricName="mae", labelCol="value", predictionCol="prediction")
play_mae = evaluator_play_mae.evaluate(play_predictions)
print(f"Mean absolute error (MAE) is: {play_mae}")
```

```
(10) Spark Jobs
Root Mean Square Error (RMSE) for 'play' predictions is: 617.3279
Mean absolute error (MAE) is: 145.7995288756351
Command took 11.75 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:48:13 PM on BDTT_TASK_2_20_APR_1
```

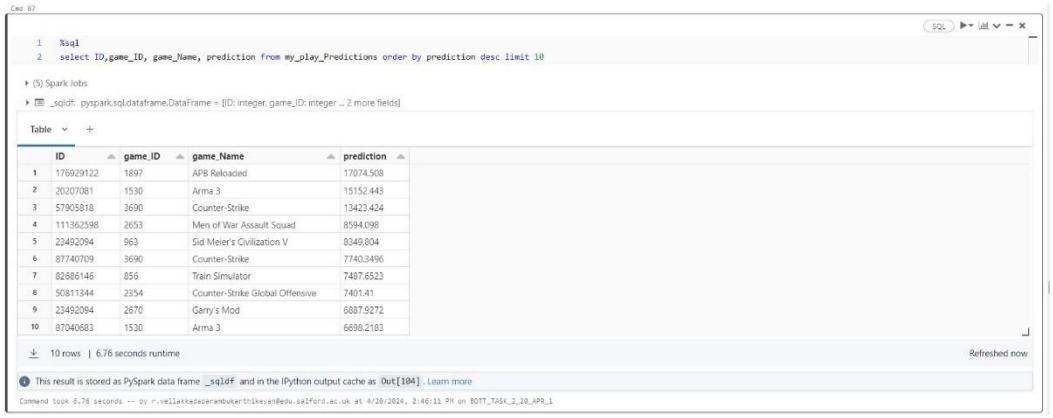
Predictions ordered in descending order



```
# Order the predictions DataFrame based on prediction in descending order
ordered_play_predictions = play_predictions.orderBy("prediction", ascending=False)

# Show the ordered predictions
ordered_play_predictions.show(truncate=False)
```

```
(5) Spark Jobs
ordered_play_predictions:  pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+---------+--------+-----+-----------------------------------------+-------+----------+
|ID       |mem_Type|value|game_Name                                |game_ID|prediction|
+---------+--------+-----+-----------------------------------------+-------+----------+
|176929122|play    |11.6 |APB Reloaded                             |1897   |17074.508 |
|20207081 |play    |33.0 |Arma 3                                   |1530   |15152.443 |
|57905818 |play    |0.3  |Counter-Strike                           |3690   |13423.424 |
|111302590|play    |13.9 |Men of War Assault Squad                 |2653   |8594.098  |
|23492094 |play    |16.6 |Sid Meier's Civilization V               |963    |8349.804  |
|87740709 |play    |39.0 |Counter-Strike                           |3690   |7740.3496 |
|82686146 |play    |217.0|Train Simulator                          |856    |7487.6523 |
|50811344 |play    |285.0|Counter-Strike Global Offensive          |2354   |7401.41   |
|23492094 |play    |8.2  |Garry's Mod                              |2670   |6887.9272 |
|87040683 |play    |10.2 |Arma 3                                   |1530   |6698.2183 |
|51132884 |play    |0.4  |LEGO MARVEL Super Heroes                 |225    |6310.7573 |
|61632730 |play    |1.6  |PlanetSide 2                             |3442   |6175.6455 |
|103563010|play    |5.9  |Europa Universalis IV                    |2804   |5873.382  |
|82263745 |play    |220.0|Counter-Strike                           |3690   |5492.3223 |
|190305835|play    |4.7  |Mount & Blade Warband                    |226    |5374.555  |
|80384872 |play    |18.2 |Sid Meier's Civilization V               |963    |5336.6436 |
|209817175|play    |30.0 |APB Reloaded                             |1897   |4868.489  |
|151600301|play    |1.2  |Dota 2                                   |1      |4776.953  |
+---------+--------+-----+-----------------------------------------+-------+----------+
Command took 6.95 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:43:10 PM on BDTT_TASK_2_20_APR_1
```

SQL queries on predictions based on temporary view



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

(5) Spark Jobs

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

Table

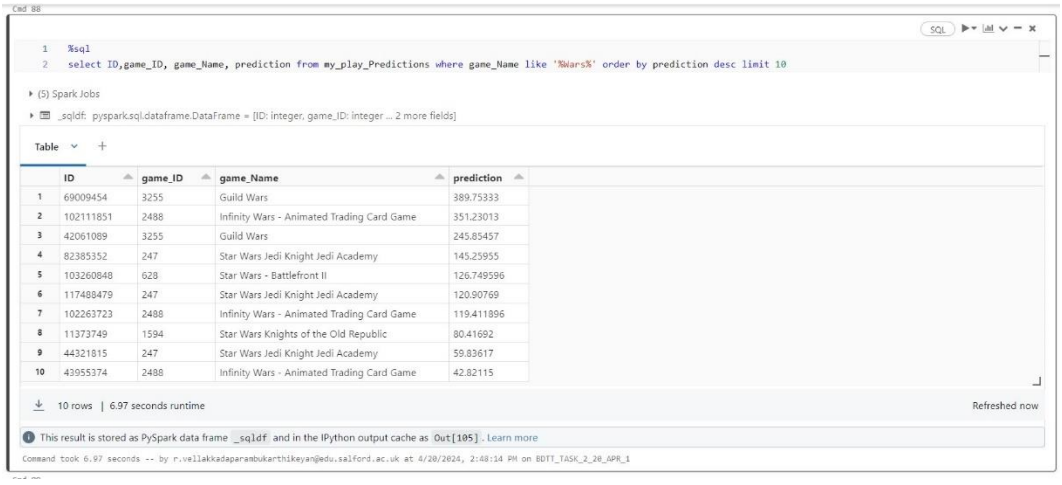
	ID	game_ID	game_Name	prediction
1	176929122	1897	APB Reloaded	17074.508
2	20207081	1530	Arma 3	15152.443
3	57905818	3690	Counter-Strike	13423.424
4	111362598	2653	Men of War Assault Squad	8594.098
5	23492094	963	Sid Meier's Civilization V	8349.804
6	87740709	3690	Counter-Strike	7740.3496
7	82686146	856	Train Simulator	7487.6523
8	50811344	2354	Counter-Strike Global Offensive	7401.41
9	23492094	2670	Garry's Mod	6887.9272
10	87040683	1530	Arma 3	6698.2183

10 rows | 6.76 seconds runtime — Refreshed now

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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 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



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

Table

	ID	game_ID	game_Name	prediction
1	69009454	3255	Guild Wars	389.75333
2	102111851	2488	Infinity Wars - Animated Trading Card Game	351.23013
3	42061089	3255	Guild Wars	245.85457
4	82385352	247	Star Wars Jedi Knight Jedi Academy	145.25955
5	103260848	628	Star Wars - Battlefront II	126.749596
6	117488479	247	Star Wars Jedi Knight Jedi Academy	120.90769
7	102263723	2488	Infinity Wars - Animated Trading Card Game	119.411896
8	11373749	1594	Star Wars Knights of the Old Republic	80.41692
9	44321815	247	Star Wars Jedi Knight Jedi Academy	59.83617
10	43955374	2488	Infinity Wars - Animated Trading Card Game	42.82115

10 rows | 6.97 seconds runtime — Refreshed now

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.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 2:48:14 PM on BDTT_TASK_2_20_APR_1

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

Cmd 89

```
%sql
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	18	Call of Duty Modern Warfare 2 - Multiplayer	3615.7046
2	43908860	18	Call of Duty Modern Warfare 2 - Multiplayer	1941.437
3	61189155	18	Call of Duty Modern Warfare 2 - Multiplayer	1614.936
4	125017535	18	Call of Duty Modern Warfare 2 - Multiplayer	1547.8528
5	74347105	18	Call of Duty Modern Warfare 2 - Multiplayer	1097.4738
6	78309377	18	Call of Duty Modern Warfare 2 - Multiplayer	597.69934
7	55334302	18	Call of Duty Modern Warfare 2 - Multiplayer	515.8308
8	56436989	18	Call of Duty Modern Warfare 2 - Multiplayer	483.5675
9	82352462	18	Call of Duty Modern Warfare 2 - Multiplayer	418.8179
10	55751308	18	Call of Duty Modern Warfare 2 - Multiplayer	299.90842

10 rows | 6.50 seconds runtime Refreshed now

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

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

Command took 0.07 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

```
recommendationsByGamePlayName('Duty')
```

(5) Spark Jobs

```
+---------+--------+-----+----------------------------------------+-------+----------+
|ID       |mem_Type|value|game_Name                               |game_ID|prediction|
+---------+--------+-----+----------------------------------------+-------+----------+
|37422528 |play    |79.0 |Call of Duty Modern Warfare 2 - Multiplayer|18     |3615.7046 |
|93189271 |play    |102.0|Call of Duty Modern Warfare 3 - Multiplayer|2029   |2190.523  |
|43908860 |play    |155.0|Call of Duty Modern Warfare 2 - Multiplayer|18     |1941.437  |
|29753085 |play    |0.1  |Call of Duty Modern Warfare 3 - Multiplayer|2029   |1799.5476 |
|8776918  |play    |46.0 |Call of Duty Modern Warfare 3 - Multiplayer|2029   |1663.0079 |
|61189155 |play    |227.0|Call of Duty Modern Warfare 2 - Multiplayer|18     |1614.936  |
|75134679 |play    |82.0 |Call of Duty Modern Warfare 3 - Multiplayer|2029   |1565.4069 |
|125017535|play    |6.4  |Call of Duty Modern Warfare 2 - Multiplayer|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     |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

```python
#Function to give recommendation filtered by game id
def recommendationsByGamePlayID(game_IDs):
    # Filter ordered predictions based on provided game_IDs
    filtered_play_predictions = ordered_play_predictions.filter(ordered_play_predictions.game_ID.isin(game_IDs))

    # Order by prediction in descending order and limit to top 10 results
    top_recommendations_play = filtered_play_predictions.orderBy("prediction", ascending=False).limit(10)

    # Show the top recommendations without truncation
    top_recommendations_play.show(truncate=False)

    # Return the DataFrame containing the top recommendations
    return top_recommendations_play
```

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_APR_1

Cmd 95

```python
recommendationsByGamePlayID(1)
```

▸ (5) Spark Jobs

ID	mem_Type	value	game_Name	game_ID	prediction
151600301	play	1.2	Dota 2	1	4776.953
79319075	play	29.0	Dota 2	1	1024.8347
99096740	play	1704.0	Dota 2	1	914.5463
21535912	play	8.0	Dota 2	1	878.619
78053539	play	8.2	Dota 2	1	874.91876
83975567	play	318.0	Dota 2	1	868.5405
104721942	play	3.3	Dota 2	1	809.60004
134223421	play	0.3	Dota 2	1	729.785
43360645	play	2390.0	Dota 2	1	675.4644
44321815	play	1054.0	Dota 2	1	665.3171

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

Command took 7.01 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:02:01 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by user ID

Cmd 96

```python
#Function to give recommendation filtered by user id
def recommendationsByUserPlayID(ID):
    # Filter ordered predictions based on provided user ID
    user_recommendations_play = ordered_play_predictions.filter(ordered_play_predictions.ID == ID)

    # Order the filtered results by prediction in descending order
    ordered_user_recommendations_play = user_recommendations_play.orderBy("prediction", ascending=False)

    # Show all the recommendations for the user ID without truncation
    ordered_user_recommendations_play.show(truncate=False)

    # Return the DataFrame containing the recommendations for the user ID
    return ordered_user_recommendations_play

```

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_APR_1

Cmd 97

```python
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	29.130709

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_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 99

```
userRecs_play.show(truncate=False)
```

▸ (2) Spark Jobs

```
+--------+---------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------+
|ID      |recommendations
|
+--------+---------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------+
|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
3, 1103.4972}]          |
|229911  |[{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
90, 2582.9548}]         |
|948368  |[{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
1, 69.65848}]           |
|975449  |[{3698, 241.05455}, {2295, 210.02835}, {3177, 192.05493}, {3203, 171.73044}, {1586, 140.03279}, {1390, 104.18551}, {2085, 103.49003}, {2106, 93.82693}, {2830, 89.28183},
{312, 85.370804}]       |
|1268792 |[{3103, 0.5887631}, {2342, 0.56960773}, {3690, 0.49999994}, {2804, 0.4796917}, {3177, 0.46284655}, {290, 0.31627083}, {427, 0.30924162}, {2295, 0.3053782}, {1241, 0.292914
48}, {3255, 0.26125038}]|
|2531540 |[{3203, 1571.4849}, {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
87.53107}]              |
|2753525 |[{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,
324.94928}]             |
|3450426 |[{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
```

Command took 22.18 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:04:22 PM on BDTT_TASK_2_20_APR_1

Recommendations for specific user

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

# Function to get game recommendations for a specific user ID and join with game details
def gameRecommendationsForUserPlay(user_ID):
    # Filter the recommendations for the specific user ID from userRecs
    user_recommendationsPlay = userRecs_play.filter(col("ID") == user_ID)

    # Explode the recommendations column to work with each recommendation separately
    user_recommendationsPlay_exploded = user_recommendationsPlay.select("ID", explode("recommendations").alias("recommendation"))

    # Join the exploded recommendations with the distinct games DataFrame
    # Use the game_ID from the recommendations and the distinct_games_df DataFrame to join
    recommended_gamesPlay = user_recommendationsPlay_exploded.join(distinct_games_df,
                                                   user_recommendationsPlay_exploded["recommendation.game_ID"] == distinct_games_df["game_ID"],
                                                   how="inner")\
                                                 .select("game_ID", "game_Name", "recommendation.rating")\
                                                 .orderBy(col("recommendation.rating").desc())

    # Display the recommended games with their ratings (predictions)
    recommended_gamesPlay.show(truncate=False)

    return recommended_gamesPlay

# Call the function for a specific user ID (e.g., user ID = 1)
gameRecommendationsForUserPlay(76767)
```

▸ (4) Spark Jobs

game_ID	game_Name	rating
2203	Football Manager 2010	4199.766
3255	Guild Wars	1953.0667
3698	FINAL FANTASY XIV A Realm Reborn	1605.7485
356	GameMaker Studio	1575.394
1451	Might & Magic Heroes VI	1459.061
2085	F1 2013	1327.3253
2588	America's Army Proving Grounds	1316.5726
4355	Supreme Commander Forged Alliance	1239.2446
3203	Sigils of Elohim	1154.2031
233	Football Manager 2011	1103.4972

DataFrame[game_ID: int, game_Name: string, rating: float]

Cmd 101

```
gameRecommendationsForUserPlay(109323647)
```

▸ (4) Spark Jobs

game_ID	game_Name	rating
2295	X-Plane 10 Global - 64 Bit	4650.6436
4125	Football Manager 2014	3426.2288
2342	Baldur's Gate Enhanced Edition	3129.1882
1507	Football Manager 2015	1866.6298
3177	Marvel Puzzle Quest	1725.3738
312	OMSI 2	1520.0271
1530	Arma 3	1469.7559
4678	NOBUNAGA'S AMBITION Kakushin with Power Up Kit	1329.8258
1118	Nobunaga's Ambition Souzou with Power Up Kit	1268.7926
3255	Guild Wars	1179.1804

DataFrame[game_ID: int, game_Name: string, rating: float]

Command took 26.21 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 3:14:31 PM on BDTT_TASK_2_20_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

```
from pyspark.sql.functions import col

# Step 1: Filter the data for 'play' type
purchase_play_df = joined_df.filter(col("mem_Type").isin(["purchase", "play"]))

# Display the play_joined_df DataFrame
purchase_play_df.show(truncate=False)
```

```
(3) Spark Jobs
purchase_play_df: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
+---------+--------+-----+--------------------+-------+
|ID       |mem_Type|value|game_Name           |game_ID|
+---------+--------+-----+--------------------+-------+
|151603712|purchase|1.0  |The Elder Scrolls V Skyrim|2610 |
|151603712|play    |273.0|The Elder Scrolls V Skyrim|2610 |
|151603712|purchase|1.0  |Fallout 4           |411    |
|151603712|play    |87.0 |Fallout 4           |411    |
|151603712|purchase|1.0  |Spore               |3869   |
|151603712|play    |14.9 |Spore               |3869   |
|151603712|purchase|1.0  |Fallout New Vegas   |3821   |
|151603712|play    |17.1 |Fallout New Vegas   |3821   |
|151603712|purchase|1.0  |Left 4 Dead 2       |70     |
|151603712|play    |8.9  |Left 4 Dead 2       |70     |
|151603712|purchase|1.0  |HuniePop            |3341   |
|151603712|play    |8.5  |HuniePop            |3341   |
|151603712|purchase|1.0  |Path of Exile       |1563   |
|151603712|play    |8.1  |Path of Exile       |1563   |
|151603712|purchase|1.0  |Poly Bridge         |4117   |
|151603712|play    |7.5  |Poly Bridge         |4117   |
|151603712|purchase|1.0  |Left 4 Dead         |1993   |
|151603712|play    |3.3  |Left 4 Dead         |1993   |
Command took 2.34 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:07:54 PM on BDTT_TASK_2_20_APR_1
```

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

```
Cmd 109

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

```
train_purchase_play: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
test_purchase_play: pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 3 more fields]
Command took 0.08 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:08:44 PM on BDTT_TASK_2_20_APR_1
```

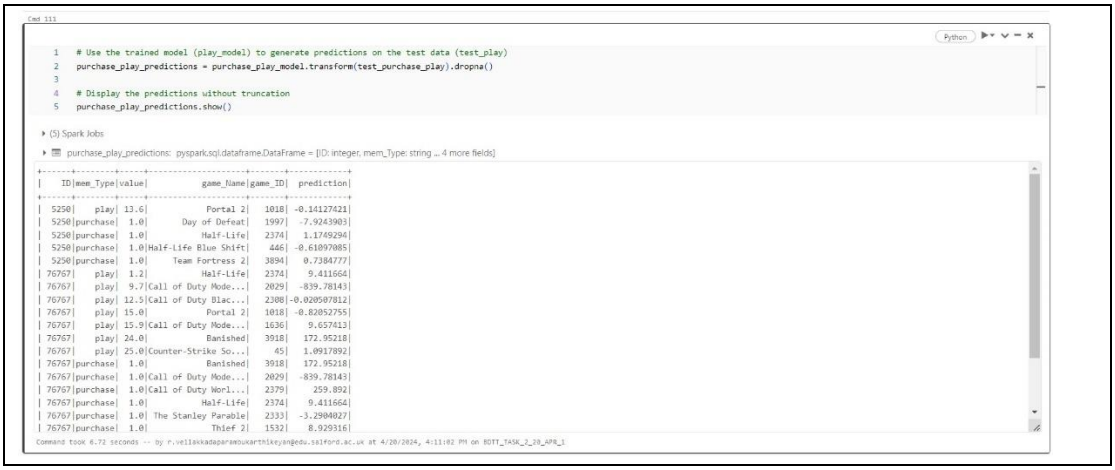
Step 3: Train the ALS model

```
Cmd 110

from pyspark.ml.recommendation import ALS

# Create an ALS model with specified parameters
als_purchase_play = ALS(maxIter=5, regParam=0.01, userCol="ID", itemCol="game_ID", ratingCol="value", seed=100)

# Fit the model to the training data for the 'play' type
purchase_play_model = als_purchase_play.fit(train_purchase_play)
```

```
(9) Spark Jobs
(1) MLflow run
Logged 1 run to an experiment in MLflow. Learn more
2024/04/20 15:09:15 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID 'b113adcc929c482f8b6614fd/1da7f', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/04/20 15:09:16 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, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (training dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details."
2024/04/20 15:09:33 WARNING mlflow.pyspark.ml: Model ALS_144ecf067abb will not be autologged because it is not allowlisted or because one or more of its nested models are not allowlisted. Call mlflow.spark.log_model() to explicitly log the model, or specify a custom allowlist via the spark.mlflow.pysparkml.autolog.logModelAllowlistFile Spark conf (see mlflow.pyspark.ml.autolog docs for more info).
Command took 19.26 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:09:14 PM on BDTT_TASK_2_20_APR_1
```

MaxIter = 5, regParam = 0.01

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

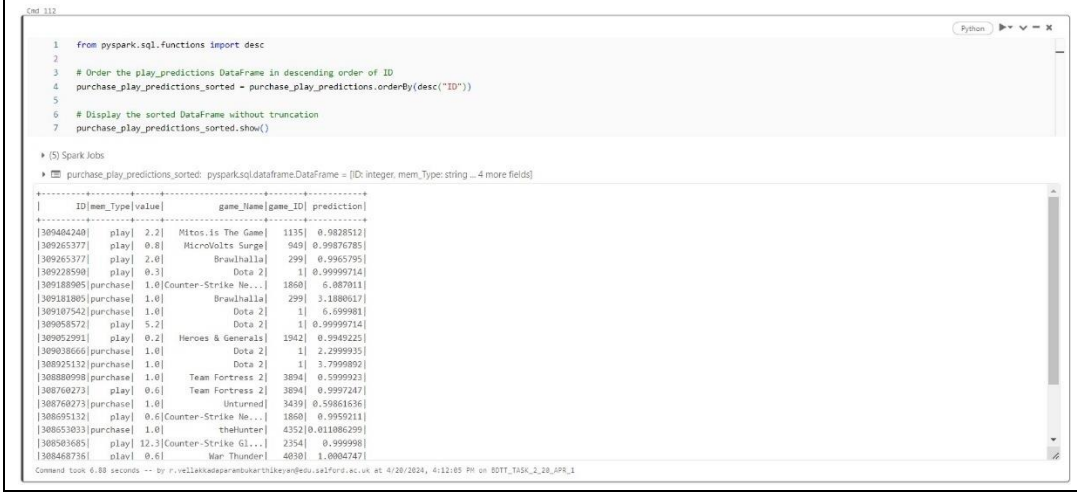


```
# Use the trained model (play_model) to generate predictions on the test data (test_play)
purchase_play_predictions = purchase_play_model.transform(test_purchase_play).dropna()

# Display the predictions without truncation
purchase_play_predictions.show()
```

```
(5) Spark Jobs
purchase_play_predictions:  pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+------+--------+-----+--------------------+-------+------------+
|    ID|mem_Type|value|           game_Name|game_ID|  prediction|
+------+--------+-----+--------------------+-------+------------+
|  5250|    play| 13.6|            Portal 2|   1018| -0.14127421|
|  5250|purchase|  1.0|        Day of Defeat|   1997|  -7.9243903|
|  5250|purchase|  1.0|           Half-Life|   2374|   1.1749294|
|  5250|purchase|  1.0|Half-Life Blue Shift|    446| -8.61097085|
|  5250|purchase|  1.0|     Team Fortress 2|   3894|   0.7384777|
| 76767|    play|  1.2|           Half-Life|   2374|    9.411664|
| 76767|    play|  9.7|Call of Duty Mode...|   2029|  -839.78143|
| 76767|    play| 12.5|Call of Duty Blac...|   2308|-0.020507812|
| 76767|    play| 15.0|            Portal 2|   1018| -0.82052755|
| 76767|    play| 15.9|Call of Duty Mode...|   1636|    9.657413|
| 76767|    play| 24.0|            Banished|   3918|   172.95218|
| 76767|    play| 25.0|Counter-Strike So...|     45|   1.0917892|
| 76767|purchase|  1.0|            Banished|   3918|   172.95218|
| 76767|purchase|  1.0|Call of Duty Mode...|   2029|  -839.78143|
| 76767|purchase|  1.0|Call of Duty Worl...|   2379|     259.892|
| 76767|purchase|  1.0|           Half-Life|   2374|    9.411664|
| 76767|purchase|  1.0| The Stanley Parable|   2333|  -3.2904027|
| 76767|purchase|  1.0|              Thief 2|   1532|    8.929316|
Command took 6.72 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:11:02 PM on BDTT_TASK_2_20_APR_1
```

Step 5: Explore some of the resulting recommendations

Displaying sorted predictions



```
from pyspark.sql.functions import desc

# Order the play_predictions DataFrame in descending order of ID
purchase_play_predictions_sorted = purchase_play_predictions.orderBy(desc("ID"))

# Display the sorted DataFrame without truncation
purchase_play_predictions_sorted.show()
```

```
(5) Spark Jobs
purchase_play_predictions_sorted:  pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+---------+--------+-----+--------------------+-------+-----------+
|       ID|mem_Type|value|           game_Name|game_ID| prediction|
+---------+--------+-----+--------------------+-------+-----------+
|309404240|    play|  2.2|   Mitos.is The Game|   1135|  0.9828512|
|309265377|    play|  0.8|    MicroVolts Surge|    949| 0.99876785|
|309265377|    play|  2.0|          Brawlhalla|    299|  0.9965795|
|309228590|    play|  0.3|              Dota 2|      1| 0.99999714|
|309188905|purchase|  1.0|Counter-Strike Ne...|   1860|   6.087011|
|309181805|purchase|  1.0|          Brawlhalla|    299|  3.1880617|
|309107542|purchase|  1.0|              Dota 2|      1|   6.699981|
|309058572|    play|  5.2|              Dota 2|      1| 0.99999714|
|309052991|    play|  0.2|   Heroes & Generals|   1942|  0.9949225|
|309038666|purchase|  1.0|              Dota 2|      1|  2.2999935|
|308925132|purchase|  1.0|              Dota 2|      1|  3.7990892|
|308880098|purchase|  1.0|     Team Fortress 2|   3894|  0.5999923|
|308760273|    play|  0.6|     Team Fortress 2|   3894|  0.9997247|
|308760273|purchase|  1.0|           Unturned|   3439| 0.59861636|
|308695132|    play|  0.6|Counter-Strike Ne...|   1860|  0.9959211|
|308653033|purchase|  1.0|          theHunter|   4352|0.011086299|
|308503685|    play| 12.3|Counter-Strike Gl...|   2354|   0.999998|
|308468736|    play|  0.6|         War Thunder|   4030|  1.0004747|
Command took 6.88 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:12:05 PM on BDTT_TASK_2_20_APR_1
```

Predictions listed by ordering game Id in ascending order

```
# Order the play_predictions DataFrame in ascending order of game_ID
ordered_purchase_play_predictions = purchase_play_predictions.orderBy("game_ID", ascending=True)

# Display the ordered DataFrame without truncation
ordered_purchase_play_predictions.show(truncate=False)
```

```
(5) Spark Jobs
ordered_purchase_play_predictions:  pyspark.sql.dataframe.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]
+--------+--------+-----+---------+-------+----------+
|ID      |mem_Type|value|game_Name|game_ID|prediction|
+--------+--------+-----+---------+-------+----------+
|14551726|purchase|1.0  |Dota 2   |1      |2.2948184 |
|11373749|play    |2.2  |Dota 2   |1      |55.909546 |
|31669242|purchase|1.0  |Dota 2   |1      |47.167937 |
|9128105 |purchase|1.0  |Dota 2   |1      |-5.861085 |
|2110581 |play    |20.0 |Dota 2   |1      |1.6726432 |
|1950243 |purchase|1.0  |Dota 2   |1      |8.939796  |
|2643009 |purchase|1.0  |Dota 2   |1      |10.091018 |
|2083767 |play    |3.1  |Dota 2   |1      |-3.4864745|
|17530772|play    |1.7  |Dota 2   |1      |56.485104 |
|15221799|purchase|1.0  |Dota 2   |1      |1854.7972 |
|10218563|play    |19.1 |Dota 2   |1      |56.041916 |
|14544587|purchase|1.0  |Dota 2   |1      |94.97991  |
|32847364|purchase|1.0  |Dota 2   |1      |18.163968 |
|4824107 |purchase|1.0  |Dota 2   |1      |91.17673  |
|9245216 |play    |15.9 |Dota 2   |1      |49.801353 |
|3527485 |play    |495.0|Dota 2   |1      |176.05247 |
|17530772|purchase|1.0  |Dota 2   |1      |56.485104 |
|1140577 |play    |2443.0|Dota 2  |1      |-51.377816|
Command took 7.14 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:12:51 PM on BDTT_TASK_2_20_APR_1
```

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

```
Cmd 114
Python
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.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

+---------+--------+-----+--------------------+-------+----------+
|       ID|mem_Type|value|           game_Name|game_ID|prediction|
+---------+--------+-----+--------------------+-------+----------+
|309404240|    play|  2.2|  Mitos.is The Game|   1135|  0.9828512|
|309265377|    play|  2.0|          Brawlhalla|    299|  0.9965795|
|309265377|    play|  0.8|    MicroVolts Surge|    949| 0.99876785|
|309228590|    play|  0.3|              Dota 2|      1| 0.99999714|
|309188905|purchase|  1.0|Counter-Strike Ne...|   1860|   6.087011|
|309181805|purchase|  1.0|          Brawlhalla|    299|  3.1880617|
|309107542|purchase|  1.0|              Dota 2|      1|   6.699981|
|309058572|    play|  5.2|              Dota 2|      1| 0.99999714|
|309052991|    play|  0.2|  Heroes & Generals|   1942|  0.9949225|
|309038866|purchase|  1.0|              Dota 2|      1|  2.2999935|
|308925132|purchase|  1.0|              Dota 2|      1|  3.7999892|
|308880998|purchase|  1.0|     Team Fortress 2|   3894|  0.5999923|
|308760273|purchase|  1.0|            Unturned|   3439| 0.59861636|
|308760273|    play|  0.6|     Team Fortress 2|   3894|  0.9997247|
|308695132|    play|  0.6|Counter-Strike Ne...|   1860|  0.9959211|
|308653033|purchase|  1.0|           theHunter|   4352|0.011086299|
|308503685|    play| 12.3|Counter-Strike Gl...|   2354|   0.999998|
|308468736|    play|  0.7|         Magic Duels|   3282|  1.0007724|
+---------+--------+-----+--------------------+-------+----------+
Command took 7.04 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:13:34 PM on BDTT_TASK_2_20_APR_1
```

Evaluating RMSE and MAE

```
Cmd 115
Python
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: 207.9078
Root Mean Square Error (RMSE) for 'purchase' and play' predictions is: 39.5405
Command took 12.46 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:16:48 PM on BDTT_TASK_2_20_APR_1
```

Predictions ordered in descending order

```
Cmd 116
Python
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.DataFrame = [ID: integer, mem_Type: string ... 4 more fields]

+---------+--------+-----+--------------------------------+-------+----------+
|ID       |mem_Type|value|game_Name                       |game_ID|prediction|
+---------+--------+-----+--------------------------------+-------+----------+
|153382649|purchase|1.0  |Team Fortress 2                 |3894   |9639.875  |
|180753437|play    |1.4  |GameMaker Studio                |356    |6475.676  |
|180753437|purchase|1.0  |GameMaker Studio                |356    |6475.676  |
|81054122 |play    |1317.0|Football Manager 2014          |4125   |6101.131  |
|81054122 |purchase|1.0  |Football Manager 2014           |4125   |6101.131  |
|70487810 |purchase|1.0  |Sid Meier's Civilization V      |963    |6012.9277 |
|101414179|purchase|1.0  |Dota 2                          |1      |5981.9834 |
|93950791 |purchase|1.0  |Sid Meier's Civilization V      |963    |5719.9917 |
|98308239 |purchase|1.0  |Dota 2                          |1      |5141.942  |
|120725727|purchase|1.0  |Dota 2                          |1      |4693.739  |
|57905818 |purchase|1.0  |Counter-Strike Source           |45     |4404.141  |
|92914917 |purchase|1.0  |STORM Frontline Nation          |947    |4073.4326 |
|141706846|purchase|1.0  |Dota 2                          |1      |4066.9893 |
|22088241 |play    |1626.0|Counter-Strike                 |3690   |4010.8364 |
|22088241 |purchase|1.0  |Counter-Strike                  |3690   |4010.8364 |
|119937035|purchase|1.0  |Garry's Mod                     |2670   |3665.7163 |
|87740709 |purchase|1.0  |Dota 2                          |1      |3355.488  |
|99077905 |purchase|1.0  |Dota 2                          |1      |3289.636  |
Command took 5.77 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:17:40 PM on BDTT_TASK_2_20_APR_1
```

SQL queries on predictions based on temporary view

```
%sql
select ID,game_ID, game_Name, prediction from my_purchase_play_Predictions 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	153382649	3894	Team Fortress 2	9639.875
2	180753437	356	GameMaker Studio	6475.676
3	180753437	356	GameMaker Studio	6475.676
4	81054122	4125	Football Manager 2014	6101.131
5	81054122	4125	Football Manager 2014	6101.131
6	70487610	963	Sid Meier's Civilization V	6012.9277
7	101414179	1	Dota 2	5981.9834

10 rows | 7.53 seconds runtime

Refreshed now

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[142]`. Learn more

Command took 7.53 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:20:58 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

```
%sql
select ID,game_ID, game_Name, prediction from my_purchase_play_Predictions where game_Name like '%Wars%' 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	51822361	3255	Guild Wars	165.87782
2	176223416	1594	Star Wars Knights of the Old Republic	162.67662
3	26762388	1594	Star Wars Knights of the Old Republic	55.265408
4	101656933	1594	Star Wars Knights of the Old Republic	47.519855
5	101656933	1594	Star Wars Knights of the Old Republic	47.519855
6	34177747	1594	Star Wars Knights of the Old Republic	38.568573
7	86168617	1594	Star Wars Knights of the Old Republic	35.940487

10 rows | 6.30 seconds runtime

Refreshed now

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[143]`. Learn more

Command took 6.30 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:21:44 PM on BDTT_TASK_2_20_APR_1

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

Cmd 121

```
%sql
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	18	Call of Duty Modern Warfare 2 - Multiplayer	1193.99
2	55763300	18	Call of Duty Modern Warfare 2 - Multiplayer	1137.974
3	21861124	18	Call of Duty Modern Warfare 2 - Multiplayer	967.4204
4	53621752	18	Call of Duty Modern Warfare 2 - Multiplayer	940.9795
5	58724834	18	Call of Duty Modern Warfare 2 - Multiplayer	824.99365
6	24204083	18	Call of Duty Modern Warfare 2 - Multiplayer	739.83997
7	112849197	18	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@edu.salford.ac.uk at 4/20/2024, 4:22:35 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by game name

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

Command took 0.08 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:25:23 PM on BDTT_TASK_2_20_APR_1

Cmd 125

```
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  |
|104132805|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  |
|58640737 |purchase|1.0  |Call of Duty Modern Warfare 3 - Multiplayer|2029   |958.3881  |
|53621752 |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@edu.salford.ac.uk at 4/20/2024, 4:25:31 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by game ID

Cmd 126

```
#Function to get recommendation filtered by game ID
def recommendationsByGamePurchasePlayID(game_IDs):
    # Filter ordered predictions based on provided game_IDs
    filtered_purchase_play_predictions = ordered_purchase_play_predictions.filter(ordered_purchase_play_predictions.game_ID.isin(game_IDs))

    # Order by prediction in descending order and limit to top 10 results
    top_recommendations_purchase_play = filtered_purchase_play_predictions.orderBy("prediction", ascending=False).limit(10)

    # Show the top recommendations without truncation
    top_recommendations_purchase_play.show(truncate=False)

    # Return the DataFrame containing the top recommendations
    return top_recommendations_purchase_play
```

Command took 0.06 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:26:54 PM on BDTT_TASK_2_20_APR_1

Cmd 127

```
recommendationsByGamePlayID(1)
```

▸ (5) Spark Jobs

```
+---------+--------+-----+---------+-------+----------+
|ID       |mem_Type|value|game_Name|game_ID|prediction|
+---------+--------+-----+---------+-------+----------+
|151600301|play    |1.2  |Dota 2   |1      |4776.953  |
|79319075 |play    |29.0 |Dota 2   |1      |1024.8347 |
|99096740 |play    |1704.0|Dota 2  |1      |914.5463  |
|21535912 |play    |8.0  |Dota 2   |1      |878.619   |
|78053539 |play    |8.2  |Dota 2   |1      |874.91876 |
|83975567 |play    |318.0|Dota 2   |1      |868.5405  |
|104721942|play    |3.3  |Dota 2   |1      |809.60004 |
|134223421|play    |0.3  |Dota 2   |1      |729.785   |
|43360645 |play    |2390.0|Dota 2  |1      |675.4644  |
|44321815 |play    |1054.0|Dota 2  |1      |665.3171  |
+---------+--------+-----+---------+-------+----------+
```

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

Command took 5.91 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:27:27 PM on BDTT_TASK_2_20_APR_1

Function giving recommendation filtered by user ID

Cmd 128

```
#Function to get recommendation filtered by user ID
def recommendationsByUserPurchasePlayID(ID):
    # Filter ordered predictions based on provided user ID
    user_recommendations_purchase_play = ordered_purchase_play_predictions.filter(ordered_purchase_play_predictions.ID == ID)

    # Order the filtered results by prediction in descending order
    ordered_user_recommendations_purchase_play = user_recommendations_purchase_play.orderBy("prediction", ascending=False)

    # Show all the recommendations for the user ID without truncation
    ordered_user_recommendations_purchase_play.show(truncate=False)

    # Return the DataFrame containing the recommendations for the user ID
    return ordered_user_recommendations_purchase_play
```

Command took 8.11 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:28:20 PM on BDTT_TASK_2_20_APR_1

Cmd 129

```
recommendationsByUserPurchasePlayID(43908860)
```

▸ (5) Spark Jobs

```
+--------+--------+-----+-----------------------------------------+-------+----------+
|ID      |mem_Type|value|game_Name                                |game_ID|prediction|
+--------+--------+-----+-----------------------------------------+-------+----------+
|43908860|purchase|1.0  |Portal 2                                 |1018   |9.263415  |
|43908860|purchase|1.0  |Half-Life 2 Episode Two                  |3669   |3.6330867 |
|43908860|play    |6.0  |AdVenture Capitalist                     |4031   |1.1460283 |
|43908860|play    |155.0|Call of Duty Modern Warfare 2 - Multiplayer|18     |0.7091713 |
|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, 4:29:00 PM on BDTT_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 131

```
userRecs_purchase_play.show(truncate=False)
```

▸ (2) Spark Jobs

```
+--------+----------------------------------------------------------------------------------------------------------------------------------------------------------
------------------+
|ID      |recommendations
|
+--------+----------------------------------------------------------------------------------------------------------------------------------------------------------
------------------+
|76767   |[{1586, 6225.5273}, {52, 5013.1094}, {226, 4556.9277}, {2203, 3625.5723}, {1390, 2522.3167}, {312, 2481.7205}, {356, 2222.3284}, {3842, 2191.7627}, {1224, 2144.9688}, {275
0, 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}, {4096, 202.04451}, {153
0, 201.69911}]    |
|229911  |[{947, 2936.0073}, {1113, 2716.5212}, {3302, 2280.1013}, {185, 1699.7037}, {1118, 1488.9478}, {3437, 1454.61}, {2085, 1293.1644}, {4792, 1238.7642}, {3177, 1224.5564}, {40
96, 1133.2081}]   |
|835015  |[{3773, 703.8117}, {352, 688.76685}, {4125, 671.99084}, {2203, 637.8292}, {947, 461.16022}, {233, 414.97903}, {1586, 330.3361}, {1118, 295.10806}, {1530, 259.33594}, {343
7, 255.42953}]     |
|948368  |[{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}, {31
77, 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,
47.38927}]       |
|1268792 |[{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, 160.94197}, {1291, 15
7.60779}]         |
|2531540 |[{1224, 1933.53}, {2203, 1674.6844}, {1586, 959.8768}, {1780, 667.7689}, {3602, 621.7649}, {2830, 618.2033}, {1241, 610.1472}, {1897, 556.0581}, {2540, 529.6347}, {233, 50
```

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

Cmd 132

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

# Function to get game recommendations for a specific user ID and join with game details
def gameRecommendationsForUserPurchasePlay(user_ID):
    # Filter the recommendations for the specific user ID from userRecs
    user_recommendationspurchase_Play = userRecs_purchase_play.filter(col("ID") == user_ID)

    # Explode the recommendations column to work with each recommendation separately
    user_recommendationspurchase_Play_exploded = user_recommendationspurchase_Play.select("ID", explode("recommendations").alias("recommendation"))

    # Join the exploded recommendations with the distinct games DataFrame
    # Use the game_ID from the recommendations and the distinct_games_df DataFrame to join
    recommended_gamespurchase_Play = user_recommendationspurchase_Play_exploded.join(distinct_games_df,
                                                user_recommendationspurchase_Play_exploded["recommendation.game_ID"] == distinct_games_df["game_ID"],
                                                how="inner")\
                                            .select("game_ID", "game_Name", "recommendation.rating")\
                                            .orderBy(col("recommendation.rating").desc())

    # Display the recommended games with their ratings (predictions)
    recommended_gamespurchase_Play.show(truncate=False)

    return recommended_gamespurchase_Play

# Call the function for a specific user ID (e.g., user ID = 1)
gameRecommendationsForUserPurchasePlay(76767)
```

▸ (4) Spark Jobs

game_ID	game_Name	rating
1586	Out of the Park Baseball 16	6225.5273
52	Elite Dangerous	5013.1094
226	Mount & Blade Warband	4556.9277
2203	Football Manager 2010	3625.5723
1390	Out of the Park Baseball 15	2522.3167
312	OMSI 2	2481.7205
356	GameMaker Studio	2222.3284
3842	Perpetuum	2191.7627
1224	Pro Evolution Soccer 2015	2144.9688
2750	Clicker Heroes	2134.2842

DataFrame[game_ID: int, game_Name: string, rating: float]

Command took 27.76 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:31:34 PM on BDTT_TASK_2_20_APR_1

Cmd 133

```python
gameRecommendationsForUserPurchasePlay(109323647)
```

▸ (4) Spark Jobs

game_ID	game_Name	rating
856	Train Simulator	713.81866
947	STORM Frontline Nation	577.9968
52	Elite Dangerous	541.4552
1530	Arma 3	461.02936
356	GameMaker Studio	381.5714
1	Dota 2	364.99796
3103	Baldur's Gate II Enhanced Edition	353.08707
1586	Out of the Park Baseball 16	309.0387
352	Football Manager 2013	289.37628
1241	Prime World	271.5041

DataFrame[game_ID: int, game_Name: string, rating: float]

Command took 28.32 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/20/2024, 4:33:51 PM on BDTT_TASK_2_20_APR_1

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.

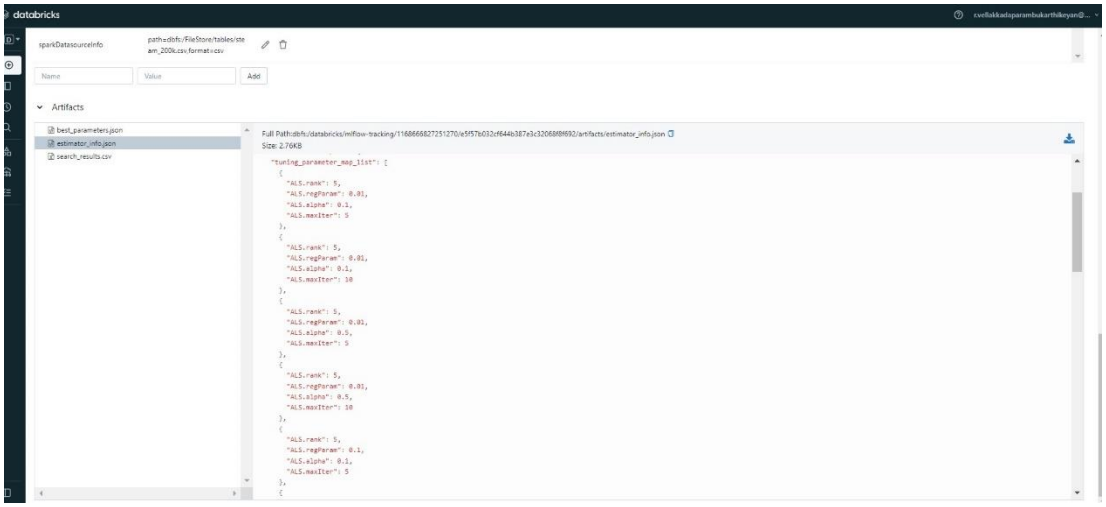


Figure 2.4 Hyperparameter tuning considerations in the recommender system design

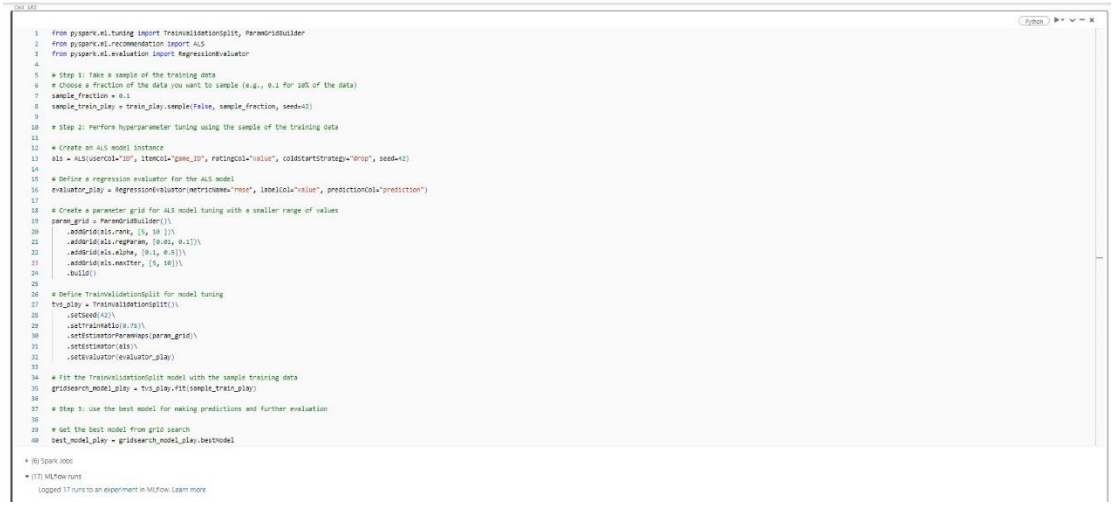
2.10.1 Selection of Hyperparameters in Recommender System based on '*Purchase*' User Behavior

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 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.

```python
from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Step 1: Take a sample of the training data
# Choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
sample_fraction = 0.1
sample_training = training.sample(False, sample_fraction, seed=42)

# Step 2: Perform hyperparameter tuning using the sample of the training data

# Create an ALS model instance
als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)

# Define a regression evaluator for the ALS model
evaluator = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")

# Create a parameter grid for ALS model tuning with a smaller range of values
param_grid = ParamGridBuilder()\
    .addGrid(als.rank, [5, 10 ])\
    .addGrid(als.regParam, [0.01, 0.1])\
    .addGrid(als.alpha, [0.1, 0.5])\
    .addGrid(als.maxIter, [5, 10])\
    .build()

# Define TrainValidationSplit for model tuning
tvs = TrainValidationSplit()\
    .setSeed(42)\
    .setTrainRatio(0.75)\
    .setEstimatorParamMaps(param_grid)\
    .setEstimator(als)\
    .setEvaluator(evaluator)

# Fit the TrainValidationSplit model with the sample training data
gridsearch_model = tvs.fit(sample_training)

# Step 3: Use the best model for making predictions and further evaluation

# Get the best model from grid search
best_model = gridsearch_model.bestModel

```

Figure 2.5 Hyperparameter tuning in recommender system based on '*Purchase*' user behaviour.

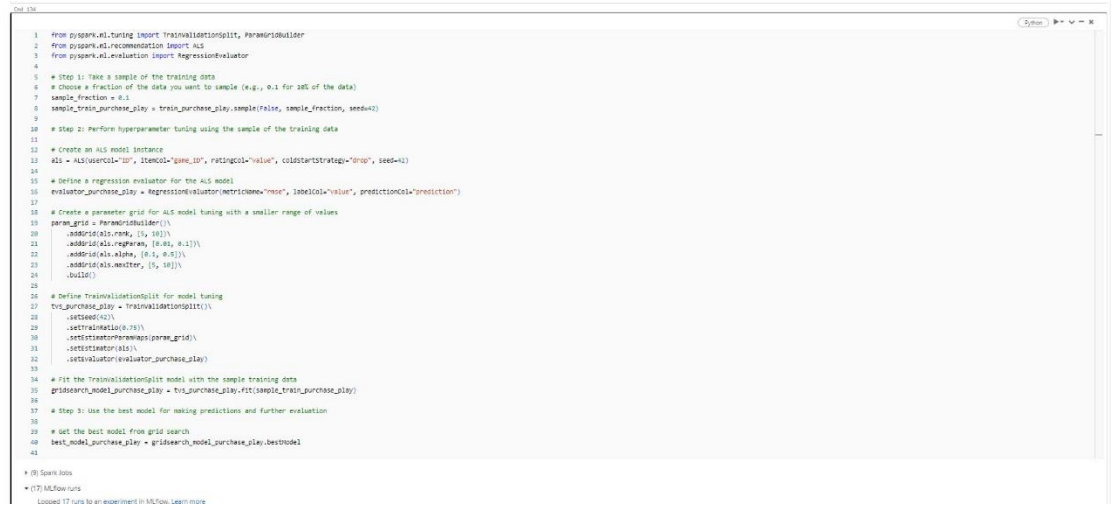
2.10.2 Selection of Hyperparameters in Recommender System based on '*Play*' User Behavior



```
from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Step 1: Take a sample of the training data
# Choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
sample_fraction = 0.1
sample_train_play = train_play.sample(False, sample_fraction, seed=42)

# Step 2: Perform hyperparameter tuning using the sample of the training data

# Create an ALS model instance
als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)

# Define a regression evaluator for the ALS model
evaluator_play = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")

# Create a parameter grid for ALS model tuning with a smaller range of values
param_grid = ParamGridBuilder()\
    .addGrid(als.rank, [5, 10 ])\
    .addGrid(als.regParam, [0.01, 0.1])\
    .addGrid(als.alpha, [0.1, 0.5])\
    .addGrid(als.maxIter, [5, 10])\
    .build()

# Define TrainValidationSplit for model tuning
tvs_play = TrainValidationSplit()\
    .setSeed(42)\
    .setTrainRatio(0.75)\
    .setEstimatorParamMaps(param_grid)\
    .setEstimator(als)\
    .setEvaluator(evaluator_play)

# Fit the TrainValidationSplit model with the sample training data
gridsearch_model_play = tvs_play.fit(sample_train_play)

# Step 3: Use the best model for making predictions and further evaluation

# Get the best model from grid search
best_model_play = gridsearch_model_play.bestModel
```

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



```
from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Step 1: Take a sample of the training data
# Choose a fraction of the data you want to sample (e.g., 0.1 for 10% of the data)
sample_fraction = 0.1
sample_train_purchase_play = train_purchase_play.sample(False, sample_fraction, seed=42)

# Step 2: Perform hyperparameter tuning using the sample of the training data

# Create an ALS model instance
als = ALS(userCol="ID", itemCol="game_ID", ratingCol="value", coldStartStrategy="drop", seed=42)

# Define a regression evaluator for the ALS model
evaluator_purchase_play = RegressionEvaluator(metricName="rmse", labelCol="value", predictionCol="prediction")

# Create a parameter grid for ALS model tuning with a smaller range of values
param_grid = ParamGridBuilder()\
    .addGrid(als.rank, [5, 10])\
    .addGrid(als.regParam, [0.01, 0.1])\
    .addGrid(als.alpha, [0.1, 0.5])\
    .addGrid(als.maxIter, [5, 10])\
    .build()

# Define TrainValidationSplit for model tuning
tvs_purchase_play = TrainValidationSplit()\
    .setSeed(42)\
    .setTrainRatio(0.75)\
    .setEstimatorParamMaps(param_grid)\
    .setEstimator(als)\
    .setEvaluator(evaluator_purchase_play)

# Fit the TrainValidationSplit model with the sample training data
gridsearch_model_purchase_play = tvs_purchase_play.fit(sample_train_purchase_play)

# Step 3: Use the best model for making predictions and further evaluation

# Get the best model from grid search
best_model_purchase_play = gridsearch_model_purchase_play.bestModel
```

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.

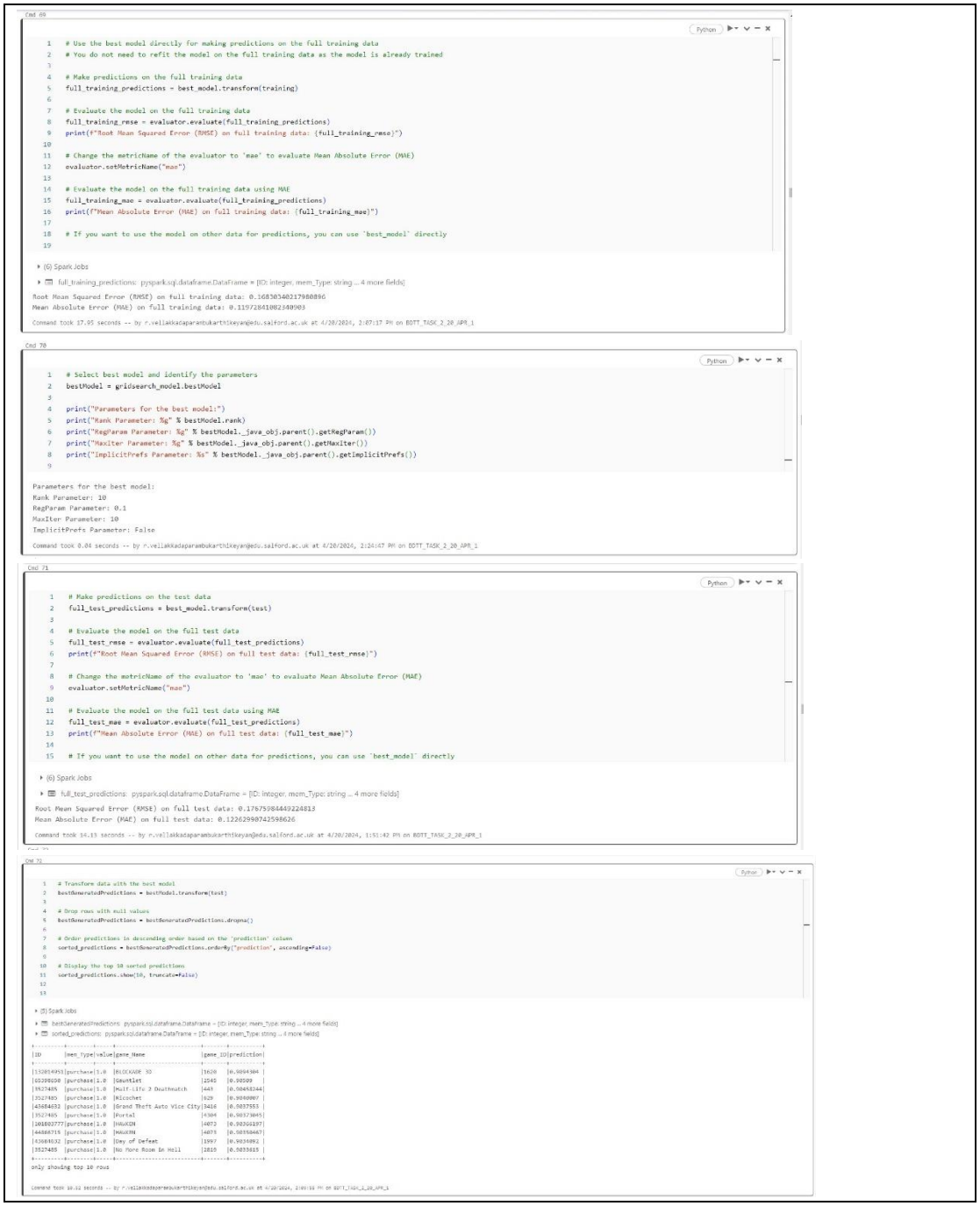


Figure 2.8 Model training and evaluation metrics of '*purchase*' based recommender system

2.11.2 Model training and Evaluation of Recommender System based on '*Play*' Behavior

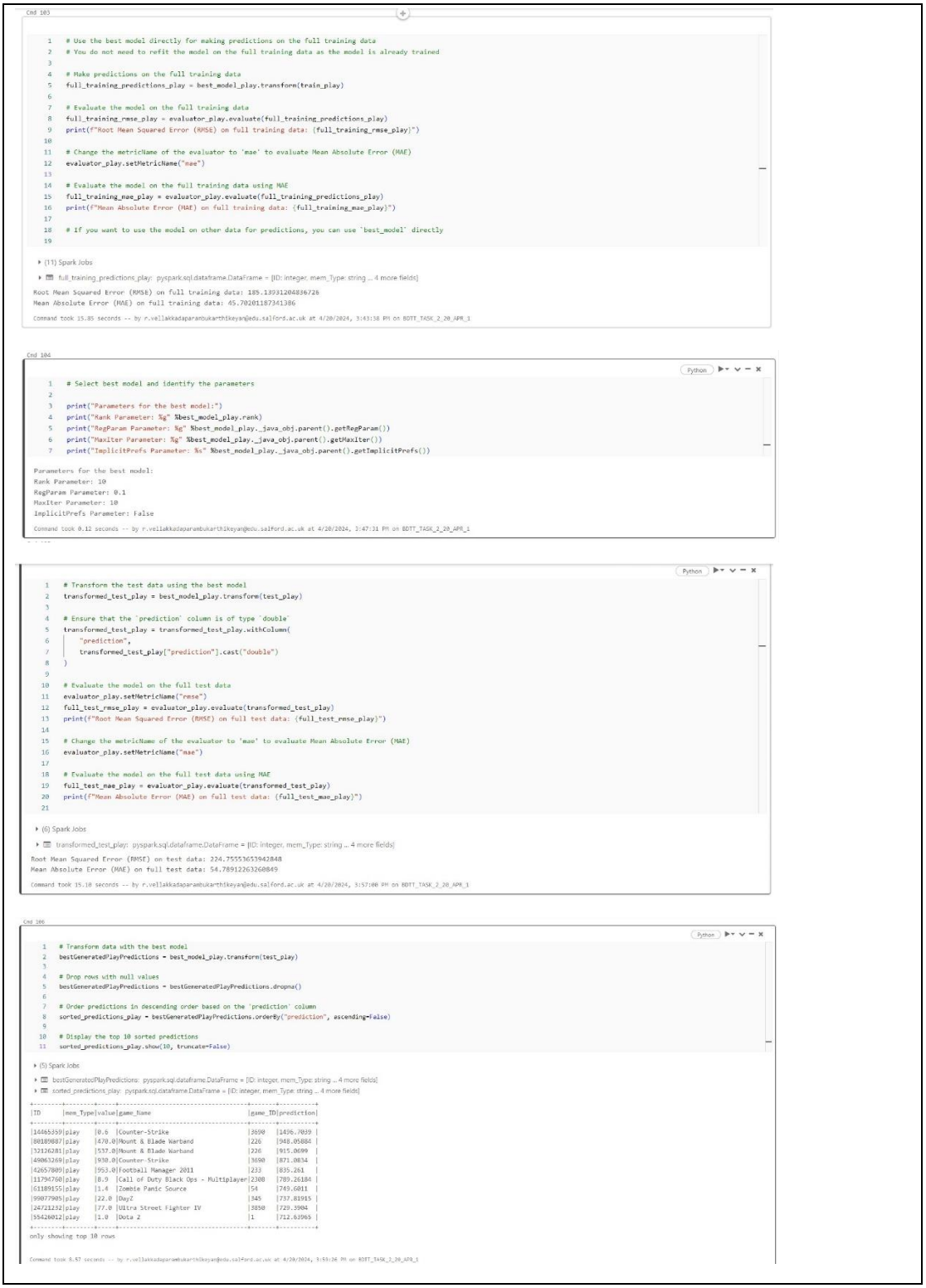


Figure 2.9 Model training and evaluation metrics of '*play*' based recommender system

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

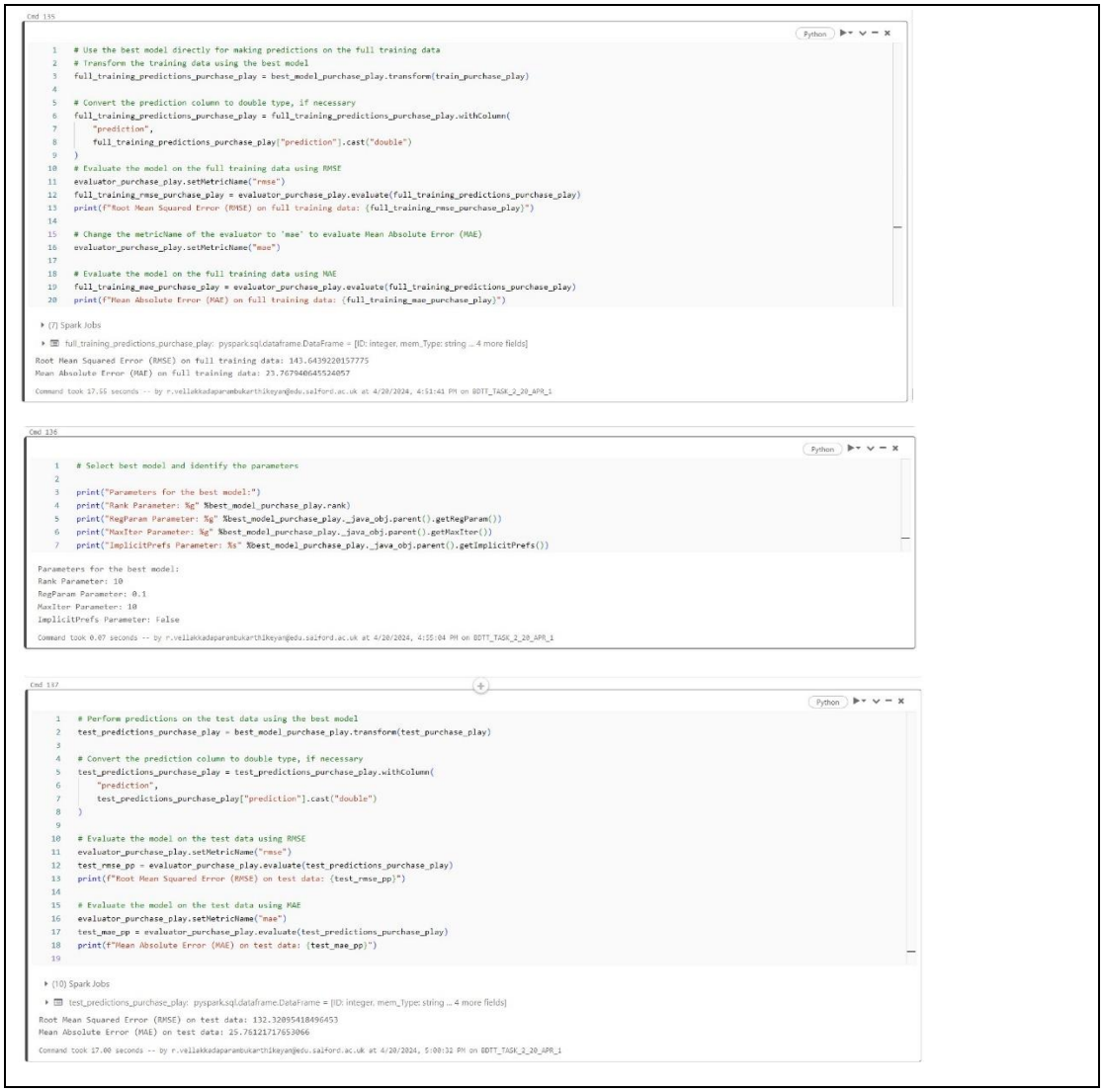


Figure 2.10 Model training and evaluation metrics of '*purchase'* and '*play*' based recommender system

2.12 MLflow Experiment Tracking

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

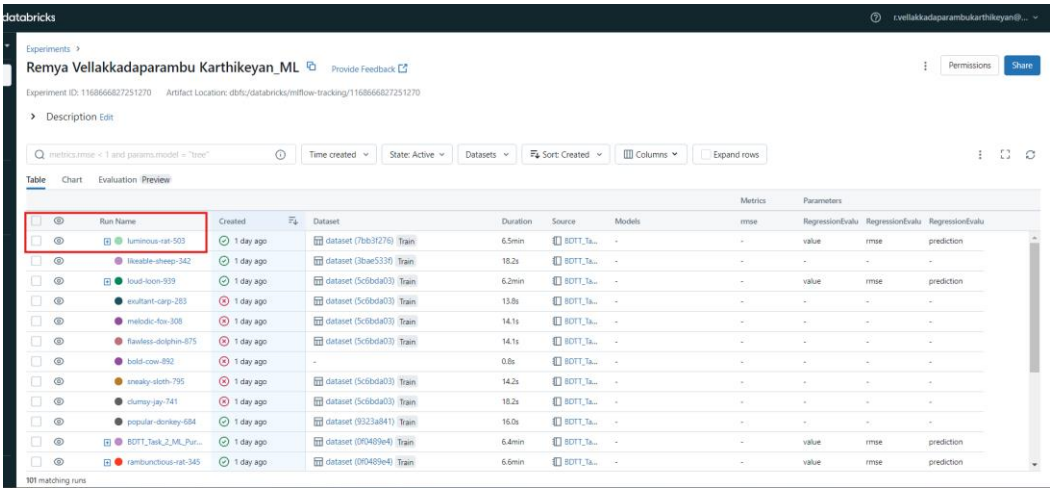


Figure 2.11 Experiment Overview (last run name enclosed in red rectangle box)

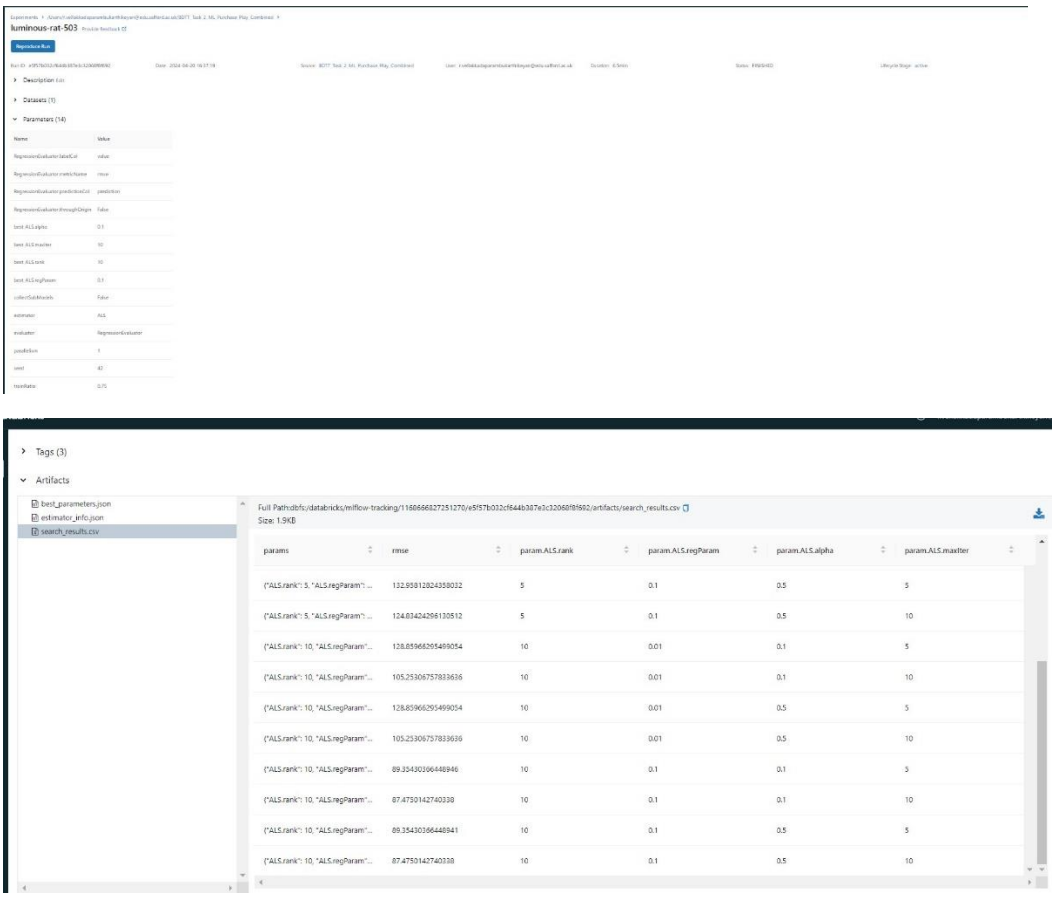


Figure 2.12 Experiment run details (hyperparameter and evaluation metrics)

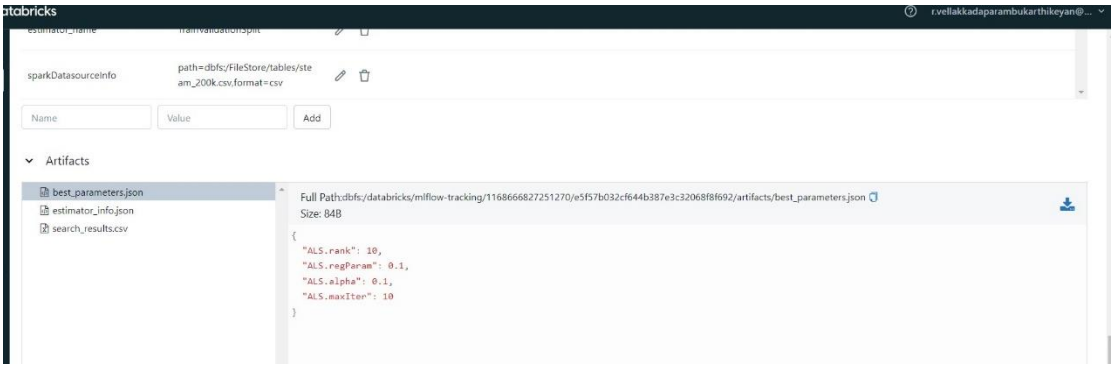
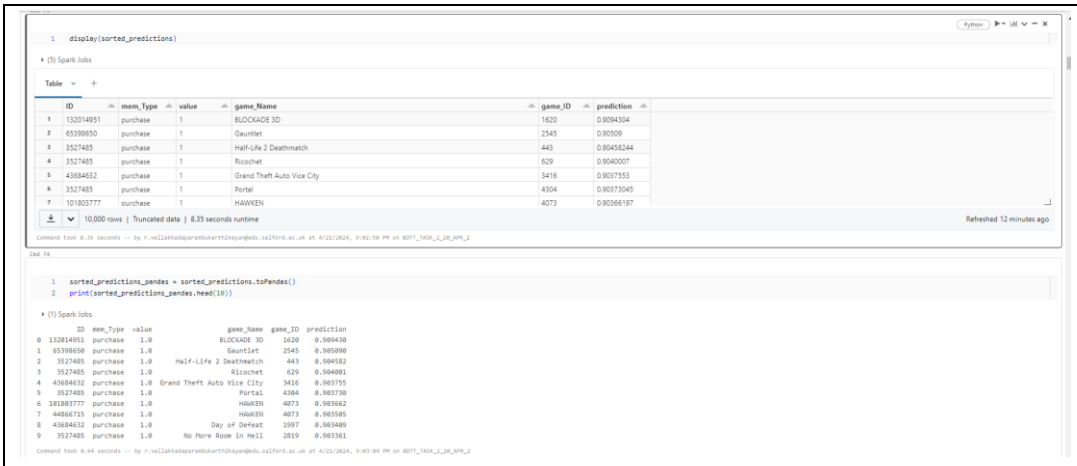


Figure 2.13 Best hyperparameters

2.13 Recommender System based on '*Purchase*' Behavior - Prediction Visualization

Figure 2.14 shows the predictions of the recommender system based on '*Purchase*' behavior



```
display(sorted_predictions)
```

(5) Spark Jobs

Table

	ID	mem_Type	value	game_Name	game_ID	prediction
1	132014951	purchase	1	BLOCKADE 3D	1620	0.9094304
2	65398650	purchase	1	Gauntlet	2545	0.90509
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	101803777	purchase	1	HAWKEN	4073	0.90366197

10,000 rows | Truncated data | 8.35 seconds runtime — Refreshed 12 minutes ago

Command took 8.35 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/21/2024, 9:02:50 PM on BDTT_TASK_2_20_APR_2

Cmd 74

```
sorted_predictions_pandas = sorted_predictions.toPandas()
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	65398650	purchase	1.0	Gauntlet	2545	0.905090
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	101803777	purchase	1.0	HAWKEN	4073	0.903662
7	44866715	purchase	1.0	HAWKEN	4073	0.903585
8	43684632	purchase	1.0	Day of Defeat	1997	0.903409
9	3527485	purchase	1.0	No More Room in Hell	2819	0.903361

Command took 0.64 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/21/2024, 9:03:04 PM on BDTT_TASK_2_20_APR_2

Top 10 recommendations

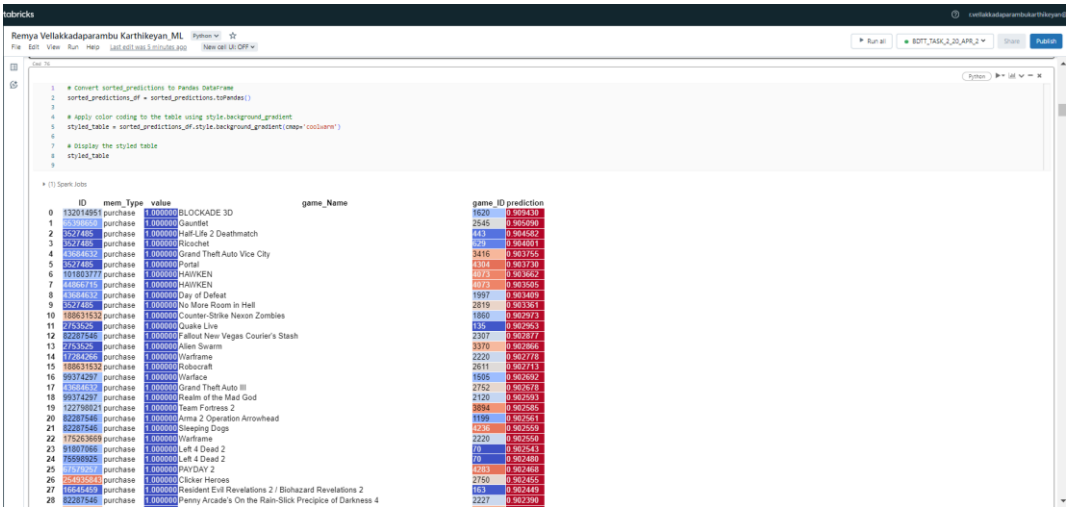
```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert to Pandas DataFrame for plotting
sorted_predictions_pandas = sorted_predictions.toPandas()

# Plotting top 10 predictions as a bar chart
sns.barplot(data=sorted_predictions_pandas.head(10), x='game_ID', y='prediction')
plt.xlabel('Game ID')
plt.ylabel('Prediction')
plt.title('Top 10 Predictions')
plt.show()
```

(1) Spark Jobs

Command took 1.23 seconds -- by r.vellakkadaparambukarthikeyan@edu.salford.ac.uk at 4/21/2024, 9:03:12 PM on BDTT_TASK_2_20_APR_2

Cmd 76

Color coded result visualization (removed from notebook)



databricks

Remya Vellakkadaparambu Karthikeyan_ML

File Edit View Run Help — Last edit was 5 minutes ago — New cell UI: OFF

Run all — BDTT_TASK_2_20_APR_2 — Share — Publish

```
# Convert sorted_predictions to Pandas DataFrame
sorted_predictions_df = sorted_predictions.toPandas()

# Apply color coding to the table using style.background_gradient
styled_table = sorted_predictions_df.style.background_gradient(cmap='coolwarm')

# Display the styled table
styled_table
```

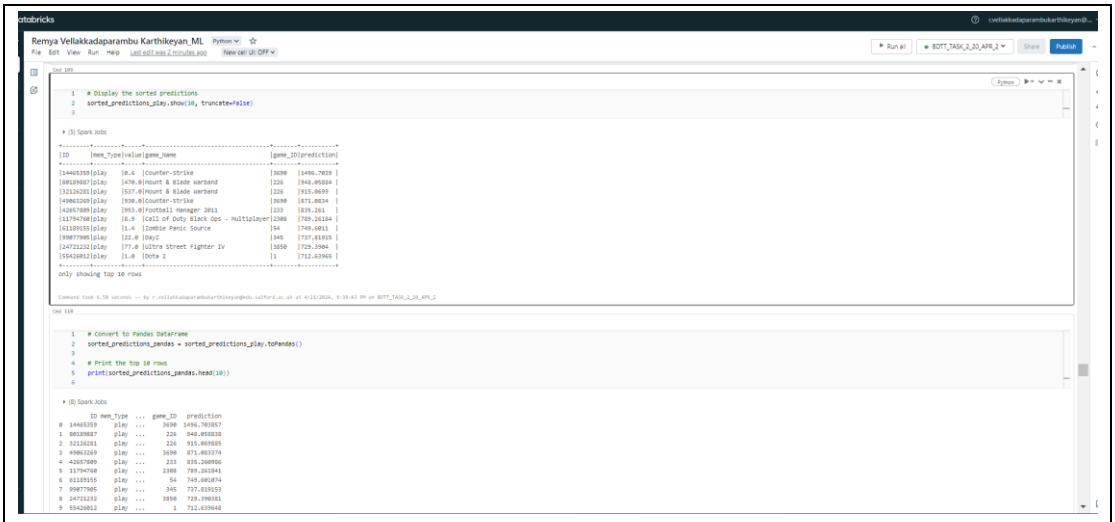
(1) Spark Jobs

	ID	mem_Type	value	game_Name	game_ID	prediction
0	132014951	purchase	1.000000	BLOCKADE 3D	1620	0.909430
1	65398650	purchase	1.000000	Gauntlet	2545	0.905090
2	3527485	purchase	1.000000	Half-Life 2 Deathmatch	443	0.904582
3	3527485	purchase	1.000000	Ricochet	629	0.904001
4	43684632	purchase	1.000000	Grand Theft Auto Vice City	3416	0.903755
5	3527485	purchase	1.000000	Portal	4304	0.903730
6	101803777	purchase	1.000000	HAWKEN	4073	0.903662
7	44866715	purchase	1.000000	HAWKEN	4073	0.903585
8	43684632	purchase	1.000000	Day of Defeat	1997	0.903409
9	3527485	purchase	1.000000	No More Room in Hell	2819	0.903361
10	188631532	purchase	1.000000	Counter-Strike Nexon Zombies	1860	0.902973
11	2753525	purchase	1.000000	Quake Live	135	0.902953
12	82287546	purchase	1.000000	Fallout New Vegas Courier's Stash	2307	0.902877
13	2753525	purchase	1.000000	Alien Swarm	3370	0.902866
14	17284266	purchase	1.000000	Warframe	2220	0.902778
15	188631532	purchase	1.000000	Robocraft	2611	0.902713
16	99374297	purchase	1.000000	Warface	1505	0.902692
17	43684632	purchase	1.000000	Grand Theft Auto III	2752	0.902678
18	99374297	purchase	1.000000	Realm of the Mad God	2120	0.902593
19	122798021	purchase	1.000000	Team Fortress 2	3894	0.902585
20	82287546	purchase	1.000000	Arma 2 Operation Arrowhead	1199	0.902561
21	82287546	purchase	1.000000	Sleeping Dogs	4236	0.902559
22	175263669	purchase	1.000000	Warframe	2220	0.902550
23	91807066	purchase	1.000000	Left 4 Dead 2	70	0.902543
24	75598925	purchase	1.000000	Left 4 Dead 2	70	0.902488
25	57578201	purchase	1.000000	PAYDAY 2	4283	0.902468
26	25493564	purchase	1.000000	Clicker Heroes	2750	0.902455
27	16645463	purchase	1.000000	Resident Evil Revelations 2 / Biohazard Revelations 2	163	0.902449
28	82287546	purchase	1.000000	Penny Arcade's On the Rain-Slick Precipice of Darkness 4	2227	0.902390

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



```
# Display the sorted predictions
sorted_predictions_play.show(10, truncate=False)
```

```
only showing top 10 rows
```

```
# Convert to Pandas DataFrame
sorted_predictions_pandas = sorted_predictions_play.toPandas()

# Print the top 10 rows
print(sorted_predictions_pandas.head(10))
```

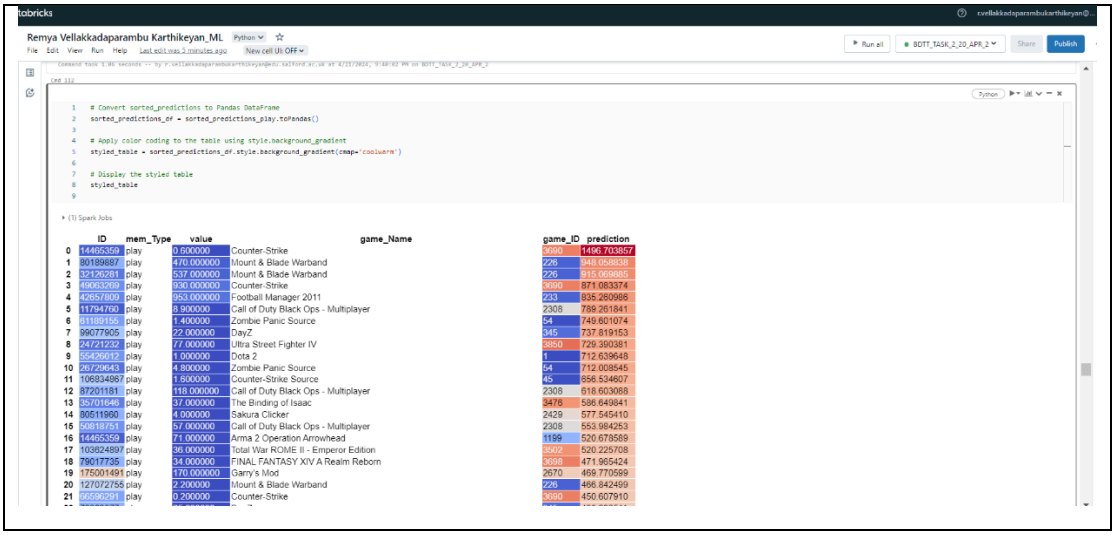
```
        ID mem_type  ...  game_ID  prediction
0 14465959     play  ...     3690  1496.781857
1 80189887     play  ...      226   948.058838
2 32126281     play  ...      226   915.869885
3 49863269     play  ...     3690   871.883374
4 42657899     play  ...      233   835.260986
5 11794768     play  ...     2308   789.261841
6 61189155     play  ...       54   749.681874
7 99877905     play  ...      345   737.819153
8 24721232     play  ...     3850   729.398301
9 55420812     play  ...        1   712.639648
```

Top 10 recommendations

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert to Pandas DataFrame for plotting
sorted_predictions_pandas = sorted_predictions_play.toPandas()

# Plotting top 10 predictions as a bar chart
sns.barplot(data=sorted_predictions_pandas.head(10), x='game_ID', y='prediction')
plt.xlabel('Game ID')
plt.ylabel('Prediction')
plt.title('Top 10 Predictions')
plt.show()
```

(1) Spark Jobs

Color coded result visualization (removed from notebook)

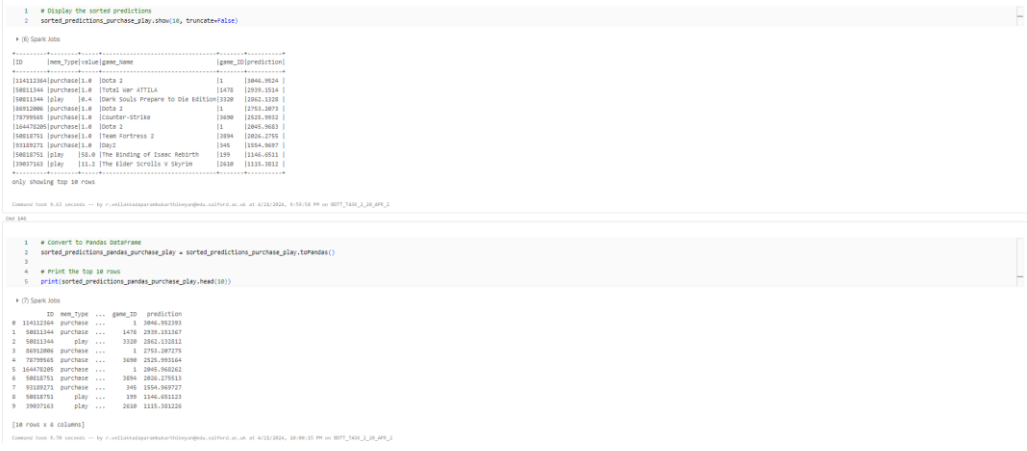


	ID	mem_Type	value	game_Name	game_ID	prediction
0	14465359	play	0.600000	Counter-Strike	3690	1496.703857
1	80189887	play	470.000000	Mount & Blade Warband	226	948.058838
2	32126281	play	537.000000	Mount & Blade Warband	226	915.069885
3	49063269	play	930.000000	Counter-Strike	3690	871.083374
4	42657809	play	953.000000	Football Manager 2011	233	835.260986
5	11794760	play	8.900000	Call of Duty Black Ops - Multiplayer	2308	789.261841
6	61189355	play	1.400000	Zombie Panic Source	54	749.601074
7	99077905	play	22.000000	DayZ	345	737.819153
8	24721228	play	77.000000	Ultra Street Fighter IV	3850	729.360381
9	55426012	play	1.000000	Dota 2	1	712.639648
10	26729643	play	4.800000	Zombie Panic Source	54	712.008545
11	106834987	play	1.800000	Counter-Strike Source	45	656.534607
12	87201181	play	118.000000	Call of Duty Black Ops - Multiplayer	2308	618.603088
13	35701846	play	37.000000	The Binding of Isaac	3478	586.649841
14	80611990	play	4.000000	Sakura Clicker	2429	577.545410
15	50818751	play	57.000000	Call of Duty Black Ops - Multiplayer	2308	553.984253
16	14465359	play	71.000000	Arma 2 Operation Arrowhead	1199	520.678589
17	10362489	play	36.000000	Total War ROME II - Emperor Edition	3502	520.225708
18	79017735	play	34.000000	FINAL FANTASY XIV A Realm Reborn	3698	471.965424
19	175001491	play	170.000000	Garry's Mod	2670	469.770599
20	127072758	play	2.200000	Mount & Blade Warband	226	466.842499
21	66596291	play	0.200000	Counter-Strike	3690	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

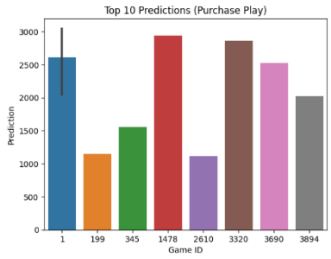
Figure 2.16 shows the predictions of the recommender system based on '*Purchase*' and '*Play*' behavior



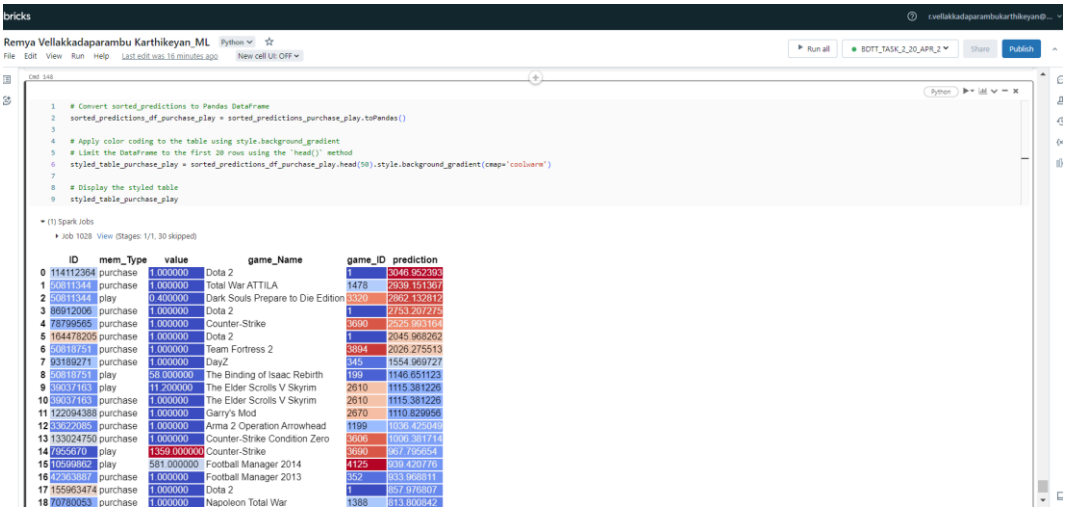
Top 10 recommendations

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert to Pandas DataFrame for plotting
sorted_predictions_pandas_purchase_play = sorted_predictions_purchase_play.toPandas()

# Plotting top 10 predictions as a bar chart
sns.barplot(data=sorted_predictions_pandas_purchase_play.head(10), x='game_ID', y='prediction')
plt.xlabel('Game ID')
plt.ylabel('Prediction')
plt.title('Top 10 Predictions (Purchase Play)')
plt.show()
```

▸ (1) Spark Jobs



Color coded result visualization (removed from notebook)



	ID	mem_Type	value	game_Name	game_ID	prediction
0	114112364	purchase	1.000000	Dota 2	1	3046.952393
1	50811344	purchase	1.000000	Total War ATTILA	1478	2939.151367
2	50811448	play	0.400000	Dark Souls Prepare to Die Edition	3320	2862.132812
3	86912006	purchase	1.000000	Dota 2	1	2753.207275
4	78799565	purchase	1.000000	Counter-Strike	3690	2525.963164
5	164478205	purchase	1.000000	Dota 2	1	2045.968262
6	50818751	purchase	1.000000	Team Fortress 2	3894	2026.275513
7	93189271	purchase	1.000000	DayZ	345	1554.969727
8	50818751	play	58.000000	The Binding of Isaac Rebirth	199	1146.651123
9	39037163	play	11.200000	The Elder Scrolls V Skyrim	2610	1115.381226
10	39037163	purchase	1.000000	The Elder Scrolls V Skyrim	2610	1115.381226
11	122094388	purchase	1.000000	Garry's Mod	2670	1110.829956
12	83672085	purchase	1.000000	Arma 2 Operation Arrowhead	1199	1036.425049
13	13902479	purchase	1.000000	Counter-Strike Condition Zero	3600	1000.381714
14	7955670	play	1359.000000	Counter-Strike	3690	967.705994
15	10599862	play	581.000000	Football Manager 2014	4125	939.420776
16	42363887	purchase	1.000000	Football Manager 2013	352	933.966811
17	155963474	purchase	1.000000	Dota 2	1	857.976807
18	70780053	purchase	1.000000	Napoleon Total War	1388	813.800842

Figure 2.16 Predictions of the recommender system based on '*Purchase*' and '*Play*' behavior

2.16 Results and Discussion

The three recommemder 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.