OVERVIEW

I implemented this assessment using a Databricks notebook, I have employed the following methods for analysis and visualization in the Notebook:

i RDDS

ii Spark Dataframes

iii Spark SQL

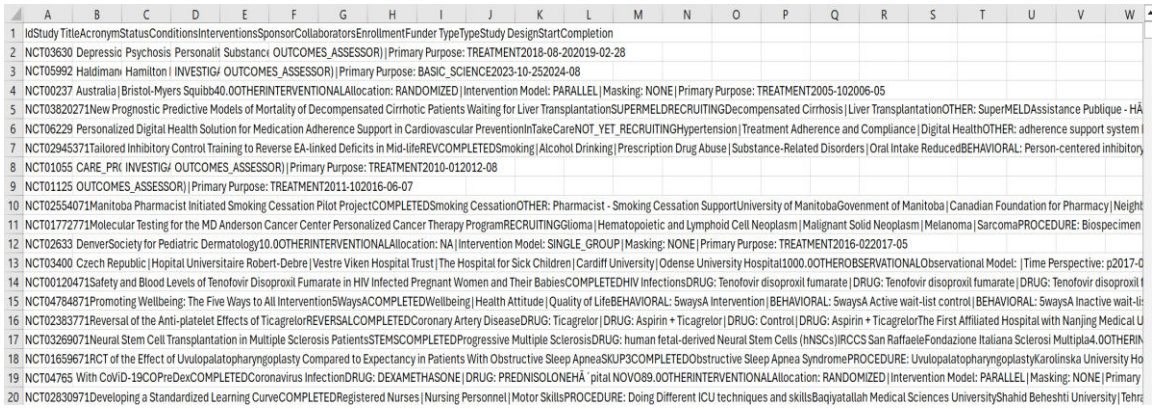
iv MATPLOTLIB (Visualization)

v Machine Learning (Recommender System)

PART 1, DATASETS IMPLEMENTATION IN DATABRICKS USING RDD, DATAFRAME AND SQL

1. **Data Cleaning, Preparation and Importation**

Datasets are already in comma separated values format. Examining the dataset in Excel reveals that it already has excellent separators like "," and ideal delimiters like "|." Within the dataset, rows and columns have valid names. Although it is necessary, more cleaning and transformation might produce results that are more accurate.

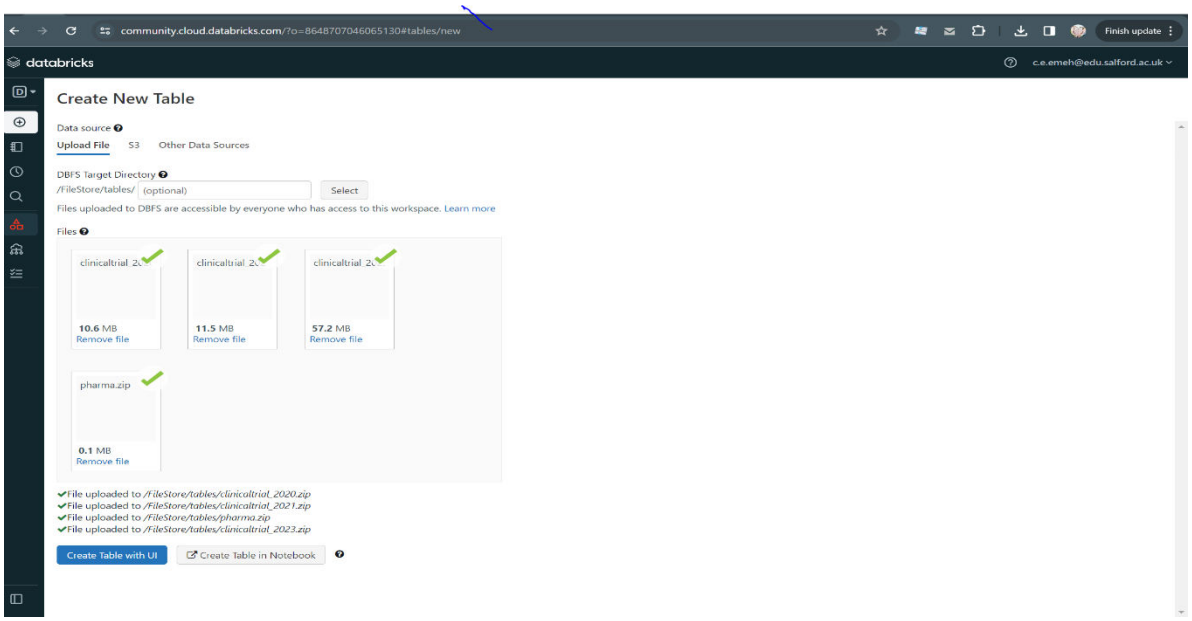


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	IdStudy TitleAcronymStatusConditionsInterventionsSponsorCollaboratorsEnrollmentFunder TypeTypeStudy DesignStartCompletion																						
2	NCT03630 Depressio Psychosis Personalit Substance OUTCOMES_ASSESSOR) | Primary Purpose: TREATMENT2018-08-202019-02-28																						
3	NCT05992 Haldiman Hamilton I INVESTIGA OUTCOMES_ASSESSOR) | Primary Purpose: BASIC_SCIENCE2023-10-252024-08																						
4	NCT00237 Australia | Bristol-Myers Squibb40.00THERINTERVENTIONALAllocation: RANDOMIZED | Intervention Model: PARALLEL | Masking: NONE | Primary Purpose: TREATMENT2005-102006-05																						
5	NCT03820271New Prognostic Predictive Models of Mortality of Decompensated Cirrhotic Patients Waiting for Liver TransplantationSUPERMELDRECRUITINGDecompensated Cirrhosis | Liver TransplantationOTHER: SuperMELDAssistance Publique - HÃ																						
6	NCT06229 Personalized Digital Health Solution for Medication Adherence Support in Cardiovascular PreventionInTakeCareNOT_YET_RECRUITINGHypertension | Treatment Adherence and Compliance | Digital HealthOTHER: adherence support system																						
7	NCT02945371Tailored Inhibitory Control Training to Reverse EA-linked Deficits in Mid-lifeREVCOMPLETEDSmoking | Alcohol Drinking | Prescription Drug Abuse | Substance-Related Disorders | Oral Intake ReducedBEHAVIORAL: Person-centered inhibitory																						
8	NCT01055 CARE_PR(INVESTIGA OUTCOMES_ASSESSOR) | Primary Purpose: TREATMENT2010-012012-08																						
9	NCT01125 OUTCOMES_ASSESSOR) | Primary Purpose: TREATMENT2011-102016-06-07																						
10	NCT02554071Manitoba Pharmacist Initiated Smoking Cessation Pilot ProjectCOMPLETEDSmoking CessationOTHER: Pharmacist - Smoking Cessation SupportUniversity of ManitobaGovenment of Manitoba | Canadian Foundation for Pharmacy | Neight																						
11	NCT01772771Molecular Testing for the MD Anderson Cancer Center Personalized Cancer Therapy ProgramRECRUITINGGlioma | Hematopoietic and Lymphoid Cell Neoplasm | Malignant Solid Neoplasm | Melanoma | SarcomaPROCEDURE: Biospecimen																						
12	NCT02633 DenverSociety for Pediatric Dermatology10.00THERINTERVENTIONALAllocation: NA | Intervention Model: SINGLE_GROUP | Masking: NONE | Primary Purpose: TREATMENT2016-022017-05																						
13	NCT03400 Czech Republic | Hopital Universitaire Robert-Debre | Vestre Viken Hospital Trust | The Hospital for Sick Children | Cardiff University | Odense University Hospital1000.00THEROBSERVATIONALObservational Model: | Time Perspective: p2017-0																						
14	NCT00120471Safety and Blood Levels of Tenofovir Disoproxil Fumarate in HIV Infected Pregnant Women and Their BabiesCOMPLETEDHIV InfectionsDRUG: Tenofovir disoproxil fumarate | DRUG: Tenofovir disoproxil fumarate | DRUG: Tenofovir disoproxil f																						
15	NCT04784871Promoting Wellbeing: The Five Ways to All Intervention5WaysACOMPLETEDWellbeing | Health Attitude | Quality of LifeBEHAVIORAL: 5waysA Intervention | BEHAVIORAL: 5waysA Active wait-list control | BEHAVIORAL: 5waysA Inactive wait-lis																						
16	NCT02383771Reversal of the Anti-platelet Effects of TicagrelorREVERSALCOMPLETEDCoronary Artery DiseaseDRUG: Ticagrelor | DRUG: Aspirin + Ticagrelor | DRUG: Control | DRUG: Aspirin + TicagrelorThe First Affiliated Hospital with Nanjing Medical U																						
17	NCT03269071Neural Stem Cell Transplantation in Multiple Sclerosis PatientsSTEMSCOMPLETEDProgressive Multiple SclerosisDRUG: human fetal-derived Neural Stem Cells (hNSCs)IRCCS San RaffaeleFondazione Italiana Sclerosi Multipla4.00THERIN																						
18	NCT01659671RCT of the Effect of Uvulopalatopharyngoplasty Compared to Expectancy in Patients With Obstructive Sleep ApneaSKUP3COMPLETEDObstructive Sleep Apnea SyndromePROCEDURE: UvulopalatopharyngoplastyKarolinska University Ho																						
19	NCT04765 With CoViD-19COPreDexCOMPLETEDCoronavirus InfectionDRUG: DEXAMETHASONE | DRUG: PREDNISOLONEHÂ´pital NOVO89.00THERINTERVENTIONALAllocation: RANDOMIZED | Intervention Model: PARALLEL | Masking: NONE | Primary																						
20	NCT02830971Developing a Standardized Learning CurveCOMPLETEDRegistered Nurses | Nursing Personnel | Motor SkillsPROCEDURE: Doing Different ICU techniques and skillsBaqiyatallah Medical Sciences UniversityShahid Beheshti University | Tehra																						

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	Company	Parent_Co	Penalty_Ar	Subtractio	Penalty_Ar	Penalty_Ye	Penalty_D	Offense_G	Primary_O	Secondary	Descriptio	Level_of_G	Action_Typ	Agency	Civil
2	Abbott Lat	Abbott Lat	$5,475,00	$0	$5,475,00	2013	20131227	governmer	False Clair	kickbacks	Abbott Lat	federal	agency ac	Justice De	civil
3	Abbott Lat	AbbVie	$1,500,00	$0	$1,500,00	2012	20120507	healthcare	off-label or unapprove		Global Hea	federal	agency ac	Food and D	civil
4	Abbott Lat	AbbVie	$126,500,(	$0	$126,500,(	2010	20101207	governmer	False Claims Act and		Abbott Lat	federal	agency ac	Justice De	civil
5	Abbott Lat	Abbott Lat	$49,045	$0	$49,045	2009	20090305	employme	wage and h	Fair Labor	Standards ,	federal	agency ac	Labor Dep	civil
6	Acclarent	Johnson &	$18,000,0(	$0	$18,000,0(	2016	20160722	governmer	False Claims Act and		California-	federal	agency ac	Justice De	civil
7	Advanced	Abbott Lat	$16,800	$0	$16,800	2004	20040412	employme	labor relations violatio		back pay a	federal	agency ac	National La	civil
8	Advanced	Abbott Lat	$2,950,00(	$0	$2,950,00(	2007	20070702	healthcare	HHS civil n	kickbacks	The HHS In	federal	agency ac	Health & H	civil
9	Advanced	Johnson &	$136,800	$0	$136,800	2014	20140520	environme	environmental violatio		The U.S. Ei	federal	agency ac	Environme	civil
10	Advanced	Johnson &	$1,200,00(	$0	$1,200,00(	2013	20131204	safety-rela	drug or medical equip		Settlemen	federal	agency ac	Food and D	civil
11	Alere San I	Abbott Lat	$10,572	$0	$10,572	2017	20170309	employme	wage and h	Fair Labor	Standards ,	federal	agency ac	Labor Dep	civil
12	Allergan In	AbbVie	$600,000,(	$0	$600,000,(	2010	20100901	healthcare	off-label or unapprove		American	federal	agency ac	Food and D	civil
13	Allergan In	AbbVie	$15,000,0(	$0	$15,000,0(	2017	20170117	financial o	investor protection vi		Allergan In	federal	agency ac	Securities	civil
14	Alpharma I	Pfizer	$42,500,0(	$0	$42,500,0(	2010	20100316	governmer	False Claim	kickbacks	Alpharma I	federal	agency ac	Justice De	civil
15	Alpharma,	Pfizer	$2,500,00(	$0	$2,500,00(	2004	20040812	competitic	price-fixing	consumer	Generic dr	federal	agency ac	Federal Tra	civil
16	American I	Bristol-My	$160,203	$0	$160,203	2004	20041226	employme	wage and h	Fair Labor	Standards ,	federal	agency ac	Labor Dep	civil
17	Amgen Inc	Amgen	$762,000,(	$0	$762,000,(	2012	20121219	healthcare	off-label or unapprove		U.S. Distri	federal	agency ac	Food and D	civil
18	Amgen Inc	Amgen	$24,900,0(	$0	$24,900,0(	2013	20130416	governmer	False Claim	kickbacks	Amgen Inc	federal	agency ac	Justice De	civil
19	Amgen, Inc	Amgen	$40,000	$0	$40,000	2011	20110302	employme	labor relations violatio		back pay a	federal	agency ac	National La	civil
20	Amneal Ph	Amneal Ph	$99,000	$0	$99,000	2015	20150930	employme	employment discrimination			federal	agency ac	Office of F	civil

3.0 Data Loading for Dataframes Implementation.

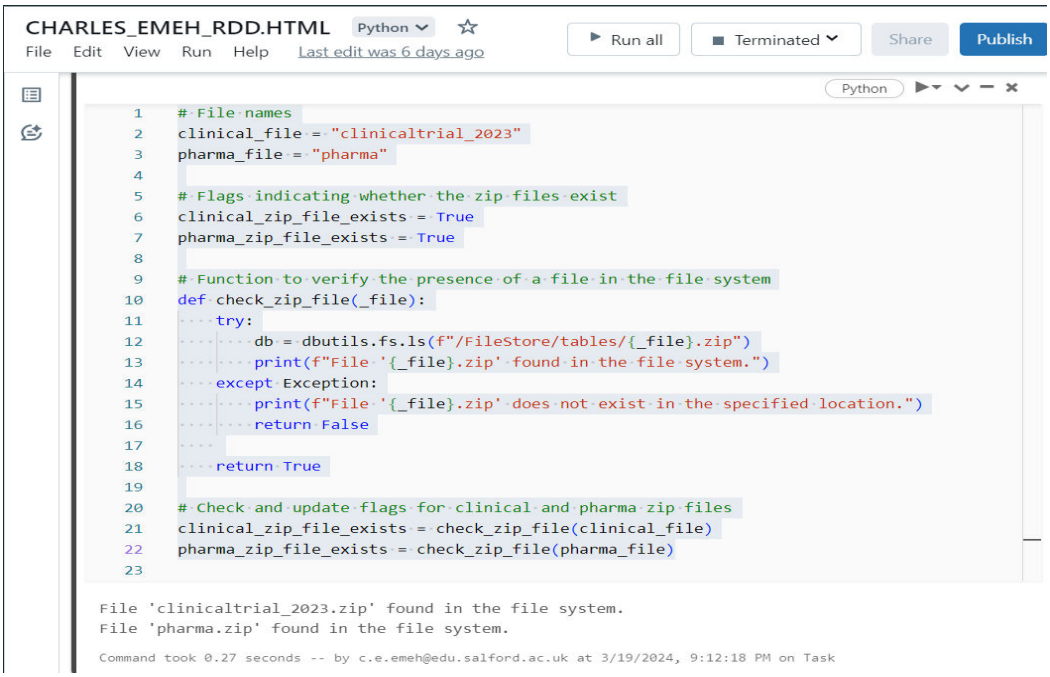
The Dataset in a zip file has been downloaded via blackboard and unpacked within Databricks. Databricks will list all the files and directories located in the **/FileStore/tables/**, and it was displayed in the notebook output. Using the “dbutils.fs.ls("/FileStore/tables/")” command it was helpful exploring the files available in the Databricks environment and accessing them for analysis or processing as also shown in the notebook, the zip files located in the database file system (DBFS) were as follows; clinicaltrial_2020.zip, clinicaltrial_2021.zip, clinicaltrial_2023.zip and pharma.zip.



4.0 Dataset preparation for Databricks implementations.

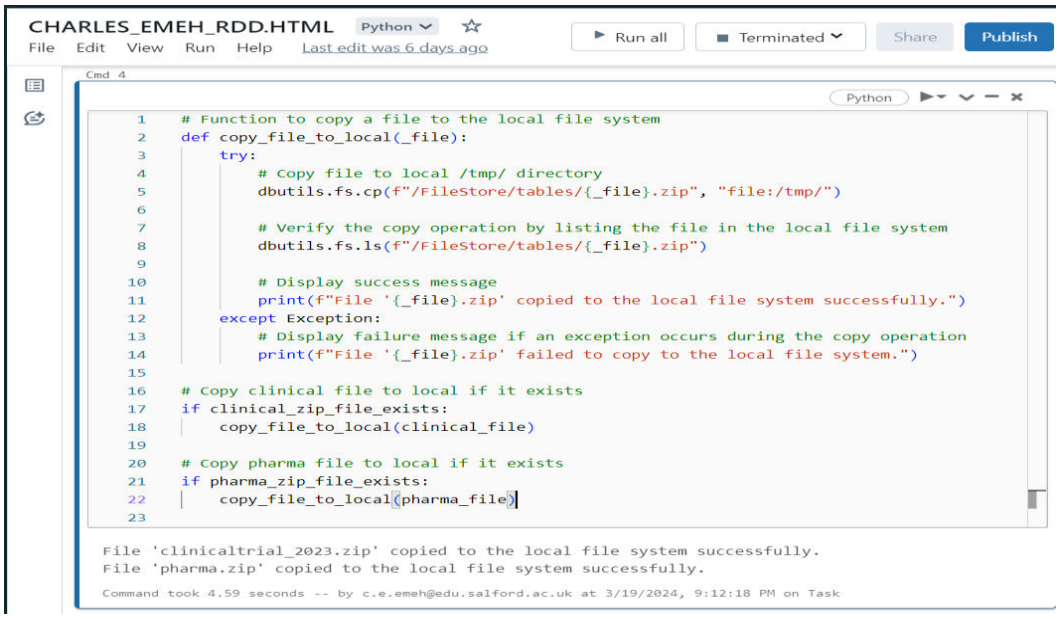
To check if specific zip files exist in the file system and update flags accordingly. It's a way to verify the presence of required files before performing further operations, fig 4 shows clinical_file and pharma_file these variables store the names of two files. In this case, clinical_file is set from "clinicaltrial_2023" and pharma_file is set from "pharma”, clinical_zip_file_exists and pharma_zip_file_exists: These are boolean variables used as flags to indicate whether the zip files for the clinical and pharma data exist. They are initially set to ‘True’. Check_zip_file(_file): This is a function defined to check if a zip file exists in the specified location. It takes a parameter _file, which represents the name of the file to be checked.

I tried to list the files in the specified location using dbutils.fs.ls(). If the file exists, it prints a message indicating that the file is found. If the file does not exist, it catches the exception and prints a message indicating that the file does not exist. Finally, it returns True if the file exists and False if it does not. The function is called twice, once for each file (clinical_file and pharma_file). The flags clinical_zip_file_exists and pharma_zip_file_exists are updated based on whether the respective files exist or not.



4.1 Copy File to the Local File System

The `copy_file_to_local( file)` function is a tool that enables users to copy a file from a specified location in Databricks (DBFS) to the local directory. It uses a try-except block to handle potential errors and verifies the copy operation by listing the copied file in the local file system. The function is invoked twice for each file, ensuring necessary files are available locally for further processing or analysis.



4.2 Defining Environment Variables for DataSets

Important file locations like "clinical_file and pharma_file" or configuration parameters need to be saved as environment variables so that they may be retrieved by other programs operating on the same computer or by my Python program, fig 4.3 shows code snippets with this regard.

Cmd 5

```
# Define environment variables for clinical and pharma files
import os

# Assign file names to environment variables
os.environ['clinical_file'] = clinical_file
os.environ['pharma_file'] = pharma_file

# Indicate successful assignment of environment variables
print("Environment variables 'clinical_file' and 'pharma_file' have been set.")
```

```
Environment variables 'clinical_file' and 'pharma_file' have been set.
```

Command took 0.19 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

5 Unzipping of the Datasets and transfer into the Databricks management system.

Since the datasets were imported in the form of zip files, I unzipped the files into csv. files which are easily accessed into a temporal directory and later transferred into the databricks management system.

```
%sh
unzip -d /tmp "/tmp/$clinical_file.zip"
unzip -d /tmp "/tmp/$pharma_file.zip"
```

```
Archive:  /tmp/clinicaltrial_2023.zip
  inflating: /tmp/clinicaltrial_2020.zip
  inflating: /tmp/clinicaltrial_2021.zip
  inflating: /tmp/clinicaltrial_2023.csv
Archive:  /tmp/pharma.zip
  inflating: /tmp/pharma.csv
```

Command took 3.10 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

Cmd 7

```
def move_file_to_dbfs(_file):
    try:
        # Transfer the file from the local /tmp/ directory to the Databricks File System (DBFS)
        dbutils.fs.mv(f"file:/tmp/{_file}.csv", f"/FileStore/tables/{_file}.csv")

        # Confirm the successful transfer by listing the file in DBFS
        dbutils.fs.ls(f"/FileStore/tables/{_file}.csv")

        # Display a confirmation message for a successful move
        print(f"The file '{_file}.csv' has been successfully moved to DBFS.")
    except Exception:
        # Display an error message if an exception occurs during the transfer
        print(f"Failed to move the file '{_file}.csv' to DBFS.")

# Move the clinical file to DBFS
move_file_to_dbfs(clinical_file)

# Move the pharma file to DBFS
move_file_to_dbfs(pharma_file)

```

```
The file 'clinicaltrial_2023.csv' has been successfully moved to DBFS.
The file 'pharma.csv' has been successfully moved to DBFS.
```

Command took 15.32 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

6. JUSTIFICATION

As was previously said, I have created and assigned new variables to my Files in the databrickks Management Systems to be used in implementing functions in RDD, Dataframe and SQL. The purpose of doing this was to prevent additional variables. An overload occurs with every new variable generated, increasing the amount of memory used by the system. Additionally, since we're using Databricks community edition, it's best to minimize the number of variables to make the programs run more quickly.

7. Creation of Resilient Distributed Dataset (RDD) in Apache Spark to Answer Questions

The RDD was created using Apache Spark's `create_rdd` function, which reads a CSV file parameter `_file` and converts it into RDDs. The text file is read line by line, returning each line as an entry. The RDD is then split and cleaned using delimiter_selector, resulting in the first ten records of clinical and pharma data.

```
3  def create_rdd(_file):
4      RDD = sc.textFile(f"/FileStore/tables/{_file}.csv")
5      return RDD
```

Command took 0.04 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

Cmd 10

```
1  def delimiter_selector(f):
2      if f == "clinicaltrial_2023":
3          return "\t"
4      if f == "clinicaltrial_2021":
5          return "|"
6      if f == "clinicaltrial_2020":
7          return "|"
8      if f == "pharma":
9          return ","
10
11
12 def clean_rdd(RDD, _file):
13     CLEAN_RDD = RDD.map(lambda x: x.split(delimiter_selector(_file))).map(lambda x: [i.replace(",",'').replace('"','') for i in x])
14     return CLEAN_RDD
```

Command took 0.09 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

```
#  Create and clean RDD

CF_RDD_MAIN = create_rdd(clinical_file)
PHARMA_RDD_MAIN = create_rdd(pharma_file)

CF_RDD_CLEAN = clean_rdd(CF_RDD_MAIN, clinical_file)
PHARMA_RDD_CLEAN = clean_rdd(PHARMA_RDD_MAIN, pharma_file)

CF_RDD_CLEAN.take(10)
```

▸ (1) Spark Jobs

```
 'National Institute on Alcohol Abuse and Alcoholism (NIAAA)',
 '439.0',
 'OTHER',
 'INTERVENTIONAL',
 'Allocation: RANDOMIZED|Intervention Model: PARALLEL|Masking: DOUBLE (INVESTIGATOR OUTCOMES_ASSESSOR)|Primary Purpose: TREATMENT',
 '2011-10',
 '2016-06-07'],
['NCT02554071',
 'Manitoba Pharmacist Initiated Smoking Cessation Pilot Project',
 '',
 'COMPLETED',
 'Smoking Cessation',
 'OTHER: Pharmacist - Smoking Cessation Support',
 'University of Manitoba',
 'Govenment of Manitoba|Canadian Foundation for Pharmacy|Neighbourhood Pharmacy Association of Canada',
 '119.0',
 'OTHER',
 'INTERVENTIONAL',
 'Allocation: NA|Intervention Model: SINGLE_GROUP|Masking: NONE|Primary Purpose: SUPPORTIVE_CARE',
 '2014-01',
 '2014-11']]
```

Command took 8.91 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

8. Tasks and Results

8.1 Task 1 in RDD; The number of studies in the dataset

The implementation process has already been described above. I have made the RDD route, given it a new name, and indicated which area of the storage I want the files to reside in as well as which area of the system I want the files to look at. I instructed the system to take the total number of studies in the dataset using the recently developed "distinct().count()" function.

Cmd 12

```
# QUESTION NUMBER ONE: DISTINCT NUMBER OF DATA
# Get the header from the clinical RDD
header = CF_RDD_MAIN.first()

# Count the number of distinct rows excluding the header in the clinical RDD
distinct_row_count = CF_RDD_MAIN.filter(lambda row: row != header).distinct().count()

# Display the result
distinct_row_count
```

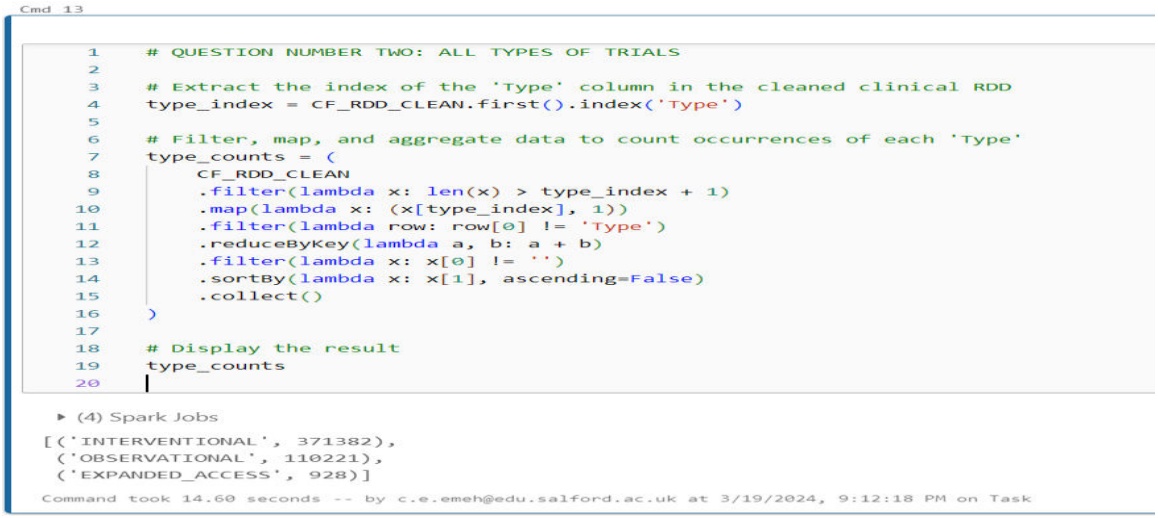
▸ (2) Spark Jobs

```
483422
```

Command took 16.96 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:12:18 PM on Task

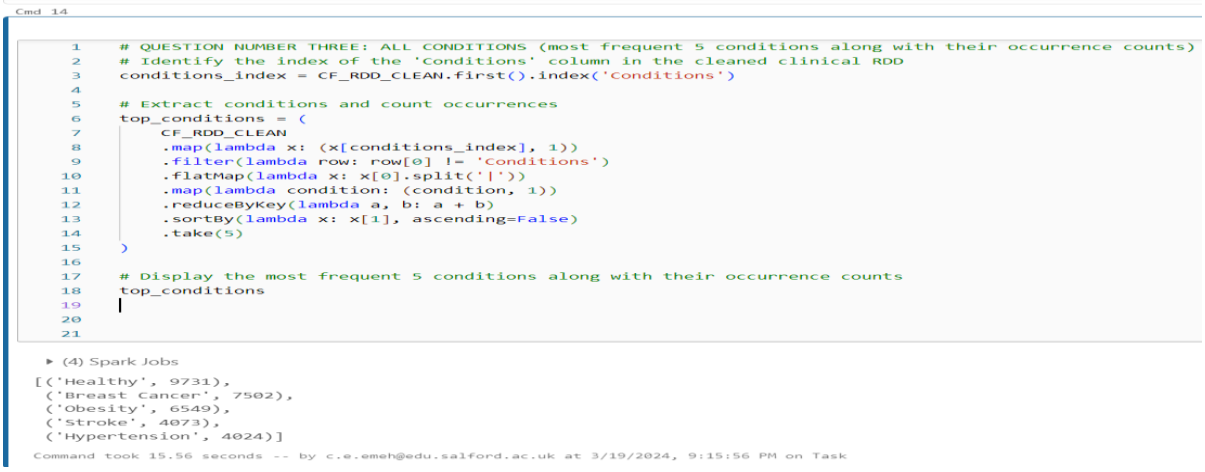
8.1.1 Task 2 in RDD; All the types of studies in the dataset (the frequencies of each type from most frequent)

I took out the 'Type' column index by determining the RDD (Resilient Distributed Dataset) designated ‘CF_RDD_CLEAN’ and the index position of the 'Type' column. The `index('Type’) `method returns the index of the 'Type' column inside the first row of the RDD that is retrieved by the `first()` function (assuming that the row contains column names).
I, then map each row to a key-value pair where the key is the value of the 'Type' column and the value is 1, filtering out rows where the length of the row is larger than the index of the 'Type' column plus one (ensuring the 'Type' column exists in the row).



8.1.2 Task 3 in RDD; The top 5 conditions with their frequencies.

I, obtained the first row of the RDD, and the `index('Conditions')` function is used to return the index of the 'Conditions' column. Every row in the RDD is associated with a tuple {(condition, 1)} by the code segment {(x[conditions_index], 1)}, where {condition} denote the value obtained in the 'Conditions' column, ‘\.filter(lambda) row: row[0]!= 'Conditions')}’:this removed every row when the field 'Conditions' contains the header row, and {.flatMap(x[0] as lambda x).split('|'))} separates every condition into separate ones. Also map(lambda) Multiple conditions may be included in each condition, split by '|'.



8.1.3 Task 4 in RDD; The 10 most common sponsors and along the number of clinical trials they have sponsored.

I used this function to retrieve the index of the 'Sponsor' column in the first row of the cleaned clinical. I pulled the 'Sponsor' data from the clinical RDD by mapping each row to its value in the 'Sponsor' and then, as "Sponsor" more likely denotes the header row, it is filtered out of all rows. I extracted the second element (index 1) from each cleaned row of the pharmaceutical RDD ({PHARMA_RDD_CLEAN}) and then removed any double quotes from the extracted values.

Cmd 15

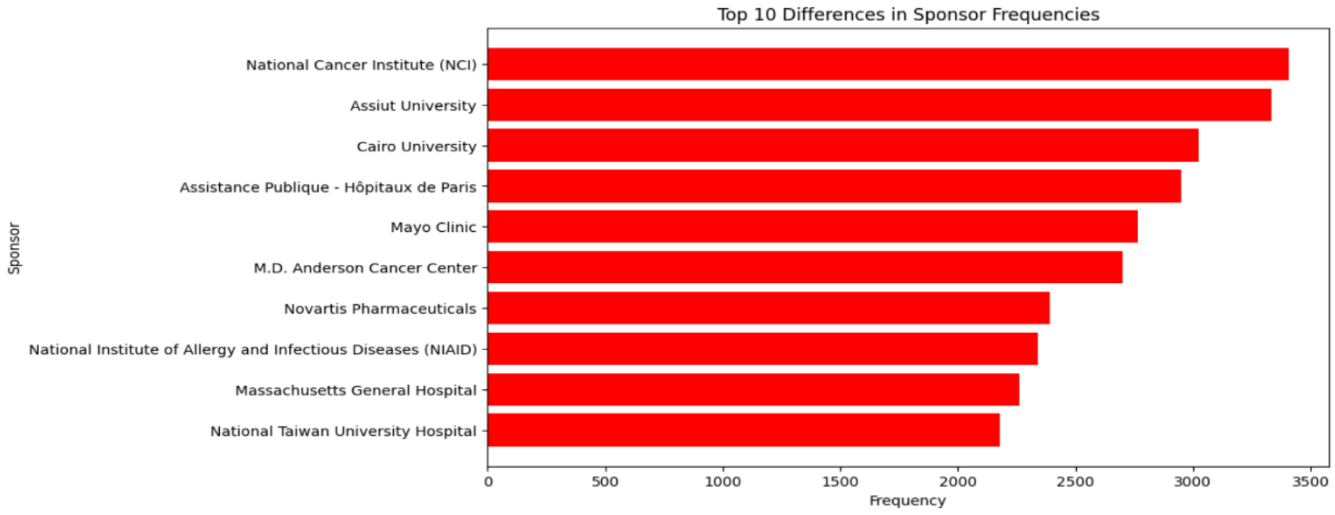
```
# QUESTION NUMBER FOUR,  TOP 10 NON - PHARMA COMPANIES

# Retrieve the index of the 'Sponsor' column in the cleaned clinical RDD
sponsor_col_index = CF_RDD_CLEAN.first().index('Sponsor')

# Extract 'Sponsor' information from the clinical RDD and refine the parent pharmaceutical company data
clinical_sponsors = CF_RDD_CLEAN.map(lambda x: x[sponsor_col_index]).filter(lambda row: row != 'Sponsor')
cleaned_pharm_companies = PHARMA_RDD_CLEAN.map(lambda x: x[1].replace('"', ''))

# Identify discrepancies in sponsors between clinical and pharmaceutical RDDs
sponsor_differences = clinical_sponsors.subtract(cleaned_pharm_companies).map(lambda x: (x, 1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], ascending=False).take(10)

# Display the top 10 differences in sponsor frequencies
sponsor_differences

```

▸ (4) 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 16.31 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:24:30 PM on Task

```
# Extract sponsor names and frequencies from the sponsor differences RDD
sponsors = [item[0] for item in sponsor_differences]
frequencies = [item[1] for item in sponsor_differences]

# Plot the data
plt.figure(figsize=(10, 6))
plt.barh(sponsors, frequencies, color='red')
plt.xlabel('Frequency')
plt.ylabel('Sponsor')
plt.title('Top 10 Differences in Sponsor Frequencies')
plt.gca().invert_yaxis()  # Invert y-axis to display highest frequency at the top
plt.show()

```



Command took 0.40 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/28/2024, 12:12:07 AM on task 1

8.1.4 Task 5 in RDD; The number of completed studies for each month in 2023 and its visualization.

The function stores clean data in a Resilient Distributed Dataset (RDD), maps clinical trial files to years using a vocabulary, and maps numerical month representations to their corresponding abbreviations using a month_selector dictionary. It extracts the month and year from the 'Completion' column, verifies if the year is "2023", and counts the occurrences of each month. The code then uses the'month_selector' dictionary to translate the numerical month representations to their respective abbreviations. The plotting results are visualized using a bar chart, representing the month and the number of studies completed in that month.

```
# Question 5,Number of completed studies for each month in 2023
year = {
    "clinicaltrial_2020": "2020",
    "clinicaltrial_2021": "2021",
    "clinicaltrial_2023": "2023",
}
CF_RDD_CLEAN = CF_RDD_CLEAN.map(lambda x: [i.replace(',', '').replace('"', '') for i in x])

month_selector = {"01": "Jan", "02":"Feb", "03":"Mar", "04":"Apr", "05":"May", "06":"Jun", "07":"Jul", "08":"Aug", "09":"Sept", "10":"Oct", "11":"Nov", "12":"Dec"}


cf_completion_col_index = CF_RDD_CLEAN.first().index('Completion')
cf_status_col_index = CF_RDD_CLEAN.first().index('Status')

CF_RDD_CLEAN.filter(lambda x: len(x) > type_index + 1).map(lambda x: (x[cf_completion_col_index], x[cf_status_col_index])).filter(lambda x: x[0] != 'Completion').filter(lambda x: x[1] == 'COMPLETED' or x[1] == 'Completed').map(lambda x: (x[0][5:7], x[0][0:4])).filter(lambda x: x[1] == "2023").map(lambda x: (x[0], 1)).reduceByKey(lambda a,b: a + b).map(lambda x: (month_selector[x[0]], x[1])).collect()
```

▸ (3) Spark Jobs

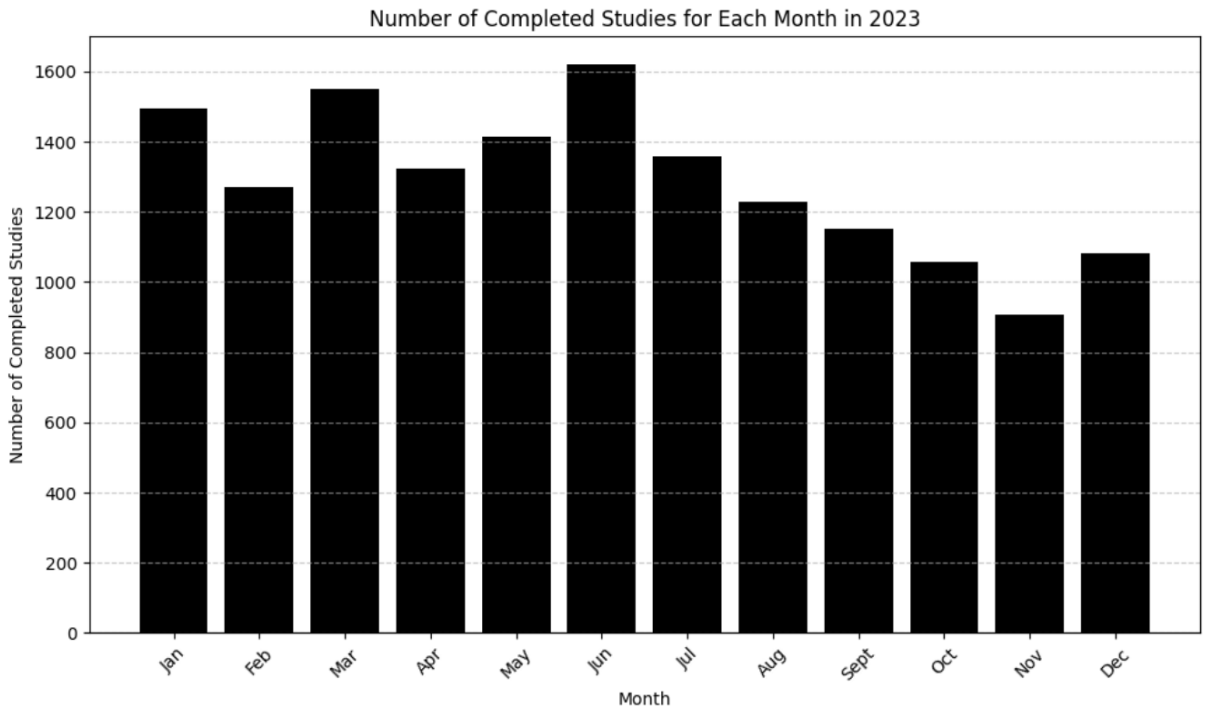
```
[('Feb', 1272),
 ('Oct', 1058),
 ('Jun', 1619),
 ('Jan', 1494),
 ('Apr', 1324),
 ('Aug', 1230),
 ('May', 1415),
 ('Jul', 1360),
 ('Nov', 909),
 ('Sept', 1152),
 ('Dec', 1082),
 ('Mar', 1552)]
```

Command took 13.83 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/19/2024, 9:24:34 PM on Task

Cmd 17

Python

```python
import matplotlib.pyplot as plt

# Data from the RDD
data = [("Jan", 1494), ("Feb", 1272), ("Mar", 1552), ("Apr", 1324), ("May", 1415), ("Jun", 1619), ("Jul", 1360), ("Aug", 1230), ("Sept", 1152), ("Oct", 1058), ("Nov", 909), ("Dec", 1082)]

# Extracting months and counts
months = [item[0] for item in data]
counts = [item[1] for item in data]

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(months, counts, color='black')
plt.xlabel('Month')
plt.ylabel('Number of Completed Studies')
plt.title('Number of Completed Studies for Each Month in 2023')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Command took 0.42 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:32:31 AM on TASK

8.2 Task in DataFrame

Just as the files clinicaltrial_2023 and Pharma are stored in the Databricks File System as clinical_file and pharma_file respectively, the function generates a DataFrame for the clinical file and specifies delimiters for other files. It reads RDD, divides row, adjusts for mismatched column lengths, and creates a DataFrame out of it. Next, a 'index' column is inserted and the columns are renamed to make the df. The 'index' column is eliminated along with the header row.

Cmd 2

```
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Define delimiters for different files
delimiter = {
    "clinicaltrial_2023": "\t",
    "clinicaltrial_2021": "|",
    "clinicaltrial_2020": "|",
    "pharma": ","
}

def create_dataframe(_file):
    if _file == "clinicaltrial_2023":
        # Read RDD, split rows, and handle mismatched column lengths
        rdd = sc.textFile(f"/FileStore/tables/{_file}.csv").map(lambda row: row.split(delimiter[_file]))
        head = rdd.first()
        rdd = rdd.map(lambda row: row + [" " for _ in range(len(head) - len(row))] if len(row) < len(head) else row)

        # Convert RDD to DataFrame, handle column renaming, and add an 'index' column
        df = rdd.toDF()
        first_row = df.first()
        for col in range(0, len(list(first_row))):
            df = df.withColumnRenamed(f"_{col + 1}", list(first_row)[col])
        df = df.withColumn('index', monotonically_increasing_id())

        # Filter out the header row and drop the 'index' column
        return df.filter(~df.index.isin([0])).drop('index')
    else:
        # Read CSV directly for other files
        return spark.read.csv(f"/FileStore/tables/{_file}.csv", sep=delimiter[_file], header=True)

# Create a DataFrame for the clinical file and display the first 50 rows
cf_dataframe = create_dataframe(clinical_file)
cf_dataframe.show(50)
```

▸ (4) Spark Jobs

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

```
2015-09",,,,,,,,,,...|
|"NCT01068171|Developing a Diab...|              |           WITHDRAWN|Diabetic Foot Ulcers|OTHER:
RANDO...|   2010-05|
2012-06",,,,,,,,,,...|
|"NCT04875871|Particle-based Pa...|PARTICLE-PATHY|          RECRUITING|              Cancer|RADIAT
NON_R...|2021-11-11|
2024-12",,,,,,,,,,...|
|"NCT00553371|Follow-up Evaluat...|              |             UNKNOWN|Testicular Germ C...|OTHER:
al Mod...|   2006-04|
",,,,,,,,,,,,,,,,,...|
|"NCT04674371|German Point Prev...|              |           COMPLETED|     Data Collection|OTHER:
al Mod...|2022-05-17|
2022-05-20",,,,,,,,...|
|"NCT02321371|Effect of Goal Di...|              |           COMPLETED|Acute-On-Chronic ...|DRUG:
RANDO...|2014-10-19|
2016-01-31",,,,,,,,...|
+------------+--------------------+--------------+--------------------+--------------------+------
---------+----------+--------------------------------------------------------------------------------
--------------------------------------+
only showing top 50 rows
```

Command took 6.01 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK

8.2.1 Task 1 in DataFrame; The number of studies in the dataset

I used the distinct().count() function to remove any duplicate rows from the DataFrame, keeping only unique rows and using distinct_row_count, the count operation's output, or the number of distinct (unique) rows in the DataFrame, is stored in this variable.

Cmd 3

```
# QUESTION 1
# Count the number of distinct rows in the DataFrame
distinct_row_count = cf_dataframe.distinct().count()

# Print the result
print(f"The number of distinct rows in the DataFrame is: {distinct_row_count}")

```

▸ (3) Spark Jobs

```
The number of distinct rows in the DataFrame is: 483422
```

Command took 17.45 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK

8.2.2 Task 2 in DataFrame; All the types of studies in the dataset (the frequencies of each type from most frequent).

I used the ‘groupBy’ function meaning 'Type'.count() as it uses the values in the 'Type' column to categorize the data in the DataFrame (cf_dataframe). It then tallies the instances of every distinct "Type." This procedure counts the number of times each product type appears in the DataFrame, for instance, if 'Type' represents various product kinds.
While use ‘orderBy('count', ascending=False)’ function to arrange the result in descending order based on the count of occurrences (ascending=False) after counting the instances of each 'Type'. This implies that the result will display the most popular 'Type' first.

Cmd 4

```
# QUESTION NUMBER TWO: ALL TYPES OF TRIALS
# Group the DataFrame by 'Type', count occurrences, and order by count in ascending order
type_counts = cf_dataframe.groupBy('Type').count().orderBy('count', ascending=False)

# Show the all top records based on count
type_counts.show()

```

▸ (2) Spark Jobs

▸ type_counts: pyspark.sql.dataframe.DataFrame = [Type: string, count: long]

```
+---------------+------+
|           Type| count|
+---------------+------+
| INTERVENTIONAL|371382|
|  OBSERVATIONAL|110221|
|EXPANDED_ACCESS|   928|
|               |   889|
|               |     2|
+---------------+------+
```

Command took 13.13 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK

8.2.3 Task 3 in DataFrame; The top 5 conditions with their frequencies.

I imported functions from the `pyspark.sql.functions` module to manipulate data in Spark DataFrames. It defines delimiters using a `conditions_delimiter`, which contains different delimiters for different files and data transformations include adding new columns, exploding, splitting, grouping, counting, ordering, and filtering. Appling these transformations to the `cf_dataframe` DataFrame, splitting and exploding values in the 'Conditions' column, grouping the DataFrame by the 'Conditions' column, counting occurrences of each condition, filtering out empty rows, and sorting the DataFrame by the count of occurrences in descending order. The code then displays the top 5 conditions and their counts in a tabular format.

Cmd 5

```
# QUESTION NUMBER THREE: ALL CONDITIONS (most frequent 5 conditions along with their occurrence counts)

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

# Define delimiters for different files
conditions_delimiter = {
    "clinicaltrial_2023": "\|",
    "clinicaltrial_2021": ",",
    "clinicaltrial_2020": ","
}

# Split and explode the 'Conditions' column, group by conditions, count occurrences, and filter non-empty conditions
conditions_counts = (
    cf_dataframe
    .withColumn('Conditions', explode(split(col('Conditions'), conditions_delimiter[clinical_file])))
    .groupBy('Conditions')
    .count()
    .orderBy('count', ascending=False)
    .filter("Conditions != ''")
)

# Show the top 5 conditions with their counts
conditions_counts.show(5, truncate=False)

```

▸ (2) Spark Jobs

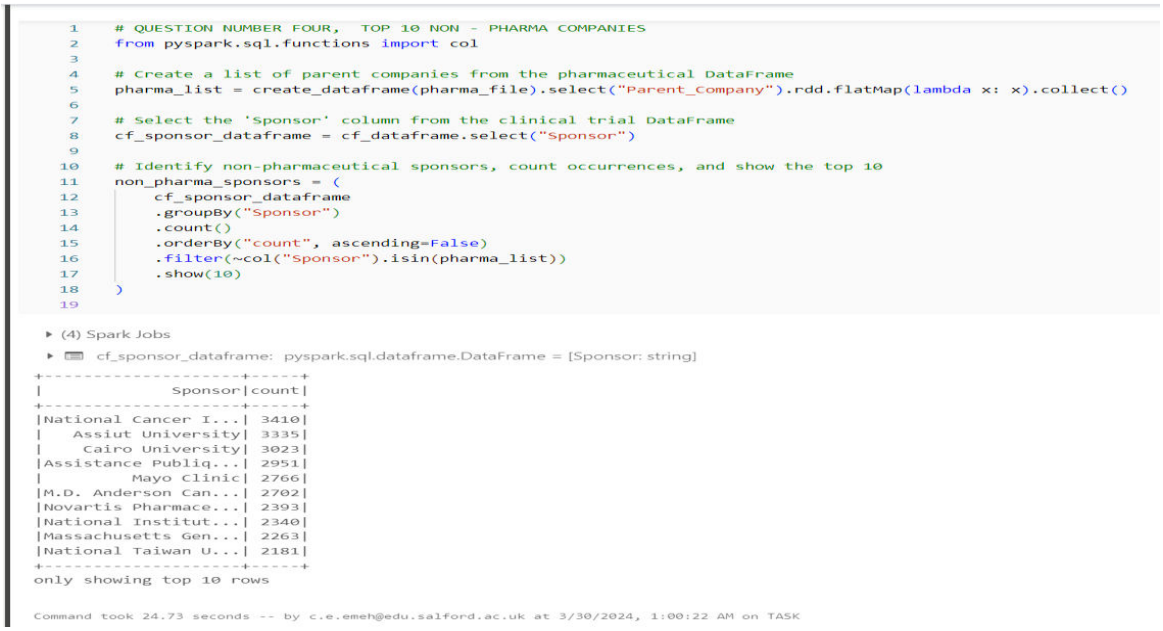
▸ conditions_counts: pyspark.sql.dataframe.DataFrame = [Conditions: string, count: long]

```
+-------------+-----+
|Conditions   |count|
+-------------+-----+
|Healthy      |9731 |
|Breast Cancer|7502 |
|Obesity      |6549 |
|Stroke       |4071 |
|Hypertension |4020 |
+-------------+-----+
only showing top 5 rows
```

Command took 13.28 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK

8.2.4 Task 4 in DataFrame; The 10 most common sponsors and along the number of clinical trials they have sponsored.

To generate a list of parent businesses from a DataFrame containing pharmaceutical data I imported a function named `col} from the `pyspark.sql.functions` module also `.rdd.flatMap(lambda x: x)} is used to transform the DataFrame into an RDD (Resilient Distributed Dataset). Following collection, the data is sent back as a list. One DataFrame containing clinical trial data is chosen, and its 'Sponsor' column is allocated to a new DataFrame named `cf_sponsor_dataframe}. Then grouping the data by the 'Sponsor' column, counts the occurrences of each sponsor, and sorts them in decreasing order to identify non-pharmaceutical sponsors in the clinical trial data.



```
# QUESTION NUMBER FOUR,  TOP 10 NON - PHARMA COMPANIES
from pyspark.sql.functions import col

# Create a list of parent companies from the pharmaceutical DataFrame
pharma_list = create_dataframe(pharma_file).select("Parent_Company").rdd.flatMap(lambda x: x).collect()

# Select the 'Sponsor' column from the clinical trial DataFrame
cf_sponsor_dataframe = cf_dataframe.select("Sponsor")

# Identify non-pharmaceutical sponsors, count occurrences, and show the top 10
non_pharma_sponsors = (
    cf_sponsor_dataframe
    .groupBy("Sponsor")
    .count()
    .orderBy("count", ascending=False)
    .filter(~col("Sponsor").isin(pharma_list))
    .show(10)
)
```

▸ (4) Spark Jobs

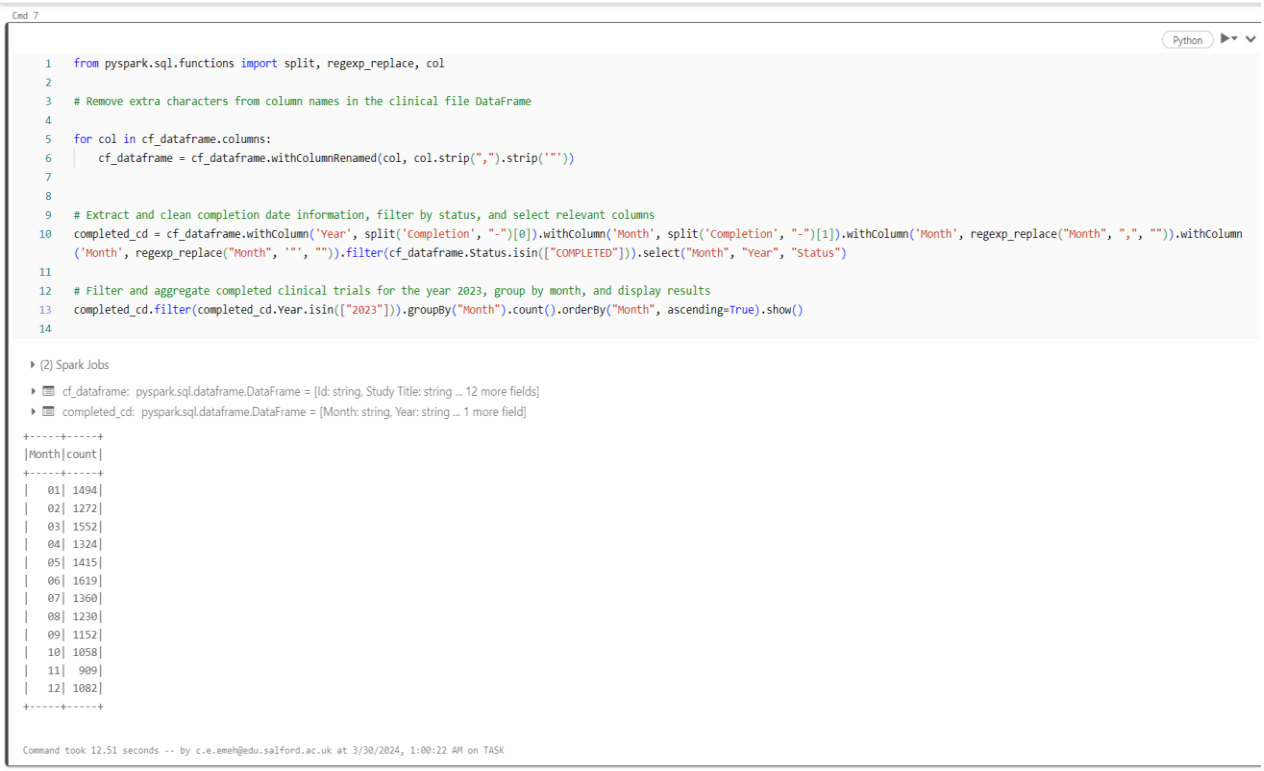
▸ cf_sponsor_dataframe: pyspark.sql.dataframe.DataFrame = [Sponsor: string]

```
+--------------------+-----+
|             Sponsor|count|
+--------------------+-----+
|National Cancer I...| 3410|
|   Assiut University| 3335|
|    Cairo University| 3023|
|Assistance Publiq...| 2951|
|          Mayo Clinic| 2766|
|M.D. Anderson Can...| 2702|
|Novartis Pharmace...| 2393|
|National Institut...| 2340|
|Massachusetts Gen...| 2263|
|National Taiwan U...| 2181|
+--------------------+-----+
only showing top 10 rows
```

Command took 24.73 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK

8.2.5 Task 5 in Dataframe; The number of completed studies for each month in 2023.

The date information is extracted and cleaned from the `Completion` column, the DataFrame is filtered to include only rows with a `Status` of "COMPLETED," and the DataFrame is filtered to include only rows with the year "2023." The data is then grouped by month, the number of completed trials is tallied for each month, and the results are sorted in ascending order by month as shown below.



```
Cmd 7

Python

from pyspark.sql.functions import split, regexp_replace, col

# Remove extra characters from column names in the clinical file DataFrame

for col in cf_dataframe.columns:
    cf_dataframe = cf_dataframe.withColumnRenamed(col, col.strip(",").strip('"'))


# Extract and clean completion date information, filter by status, and select relevant columns
completed_cd = cf_dataframe.withColumn('Year', split('Completion', "-")[0]).withColumn('Month', split('Completion', "-")[1]).withColumn('Month', regexp_replace("Month", ",", "")).withColumn
('Month', regexp_replace("Month", '"', "")).filter(cf_dataframe.Status.isin(["COMPLETED"])).select("Month", "Year", "Status")

# Filter and aggregate completed clinical trials for the year 2023, group by month, and display results
completed_cd.filter(completed_cd.Year.isin(["2023"])).groupBy("Month").count().orderBy("Month", ascending=True).show()

▸ (2) Spark Jobs
▸ cf_dataframe:  pyspark.sql.dataframe.DataFrame = [Id: string, Study Title: string ... 12 more fields]
▸ completed_cd:  pyspark.sql.dataframe.DataFrame = [Month: string, Year: string ... 1 more field]
+-----+-----+
|Month|count|
+-----+-----+
|   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 12.51 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 1:00:22 AM on TASK
```

8.3 Task in SQL (Importing functions for SQL implementation)

I imported the modules `pyspark.sql.functions' and `pyspark.sql.types', which are utilized for manipulating data and creating data types. Delimiters, like tabs and commas, for various file formats are included in the `delimiter` dictionary. To construct a DataFrame from the file, I used the method `create_dataframe(_file)'. The file "clinicaltrial_2023" is an example of how it reads an RDD, separates rows according to the delimiter, manages column lengths that aren't aligned, changes it to a DataFrame, renames columns, adds a 'index' column, extracts the header row. PySpark is an effective tool for converting CSV files kept in a Databricks FileStore into DataFrames.

To construct DataFrames for the clinical and pharma files, the function is invoked twice. This created temporary views for clinical and pharmaceutical data in a DataFrame called `cf_dataframe`. These views act as virtual tables, allowing SQL-like queries.
The `pharma_dataframe.createOrReplaceTempView` creates a similar view for pharma data while filtered data is then grouped by the 'Month' column, counting the number of occurrences of each month. The aggregated data is then sorted in ascending order, ensuring the months are displayed in order.

```python
%python
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Define delimiters for different files
delimiter = {
    "clinicaltrial_2023": "\t",
    "clinicaltrial_2021": "|",
    "clinicaltrial_2020": "|",
    "pharma": ","
}

def create_dataframe(_file):
    if _file == "clinicaltrial_2023":
        # Read RDD, split rows, and handle mismatched column lengths
        rdd = sc.textFile(f"/FileStore/tables/{_file}.csv").map(lambda x: x.rstrip(',').strip('"')).map(lambda row: row.split(delimiter[_file]))
        head = rdd.first()
        rdd = rdd.map(lambda row: row + [" " for _ in range(len(head) - len(row))] if len(row) < len(head) else row)

        # Convert RDD to DataFrame, handle column renaming, and add an 'index' column
        df = rdd.toDF()
        first_row = df.first()
        for col in range(0, len(list(first_row))):
            df = df.withColumnRenamed(f"_{col + 1}", list(first_row)[col])
        df = df.withColumn('index', monotonically_increasing_id())

        # Filter out the header row and drop the 'index' column
        return df.filter(~df.index.isin([0])).drop('index')
    else:
        # Read CSV directly for other files
        return spark.read.csv(f"/FileStore/tables/{_file}.csv", sep=delimiter[_file], header=True)

# Create a DataFrame for the clinical file and display the first 50 rows
cf_dataframe = create_dataframe(clinical_file)
pharma_dataframe = create_dataframe(pharma_file)
cf_dataframe.show(50)
```

▸ (5) Spark Jobs

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

▸ pharma_dataframe: pyspark.sql.dataframe.DataFrame = [Company: string, Parent_Company: string ... 32 more field

```
2015-09",,,,,,,,,,|
|"NCT01068171|Developing a Diab...|              |           WITHDRAWN|Diabetic Foot Ulcers|OTHER:
RANDO...|   2010-05|
2012-06",,,,,,,,,,|
|"NCT04875871|Particle-based Pa...|PARTICLE-PATHY|          RECRUITING|              Cancer|RADIAT
NON_R...|2021-11-11|
2024-12",,,,,,,,,,|
|"NCT00553371|Follow-up Evaluat...|              |             UNKNOWN|Testicular Germ C...|OTHER:
al Mod...|   2006-04|
",,,,,,,,,,,,,,,,,,|
|"NCT04674371|German Point Prev...|              |           COMPLETED|     Data Collection|OTHER:
al Mod...|2022-05-17|
2022-05-20",,,,,,,,|
|"NCT02321371|Effect of Goal Di...|              |           COMPLETED|Acute-On-Chronic ...|DRUG:
RANDO...|2014-10-19|
2016-01-31",,,,,,,,|
+------------+--------------------+--------------+--------------------+--------------------+------
---------+----------+--------------------------------------------------------------------------------
----------------------------------------+
only showing top 50 rows
```

Command took 24.22 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 12:25:48 AM on TASK

8.3.1Retrieving all Columns From the DataSets

I retrieved all columns in the clinical_file_view database using the SQL query '%sql SELECT * FROM clinical_file_view' for clinicaltrial_2023 and '%sql SELECT * FROM pharma_file_view for pharma.

```
%sql
-- Retrieve all columns from the clinical trial data for the year 2023
SELECT * FROM clinical_file_view
```

(1) Spark Jobs

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

Table +

	"Id	Study Title
1	"NCT03630471	Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India
2	"NCT05992571	Oral Ketone Monoester Supplementation and Resting-state Brain Connectivity
3	"NCT00237471	Impact of Tight Glycaemic Control in Acute Myocardial Infarction
4	"NCT03820271	New Prognostic Predictive Models of Mortality of Decompensated Cirrhotic Patients Waiting for Liver Transplantation
5	"NCT06229171	InTake Care: Development and Validation of an Innovative"," Personalized Digital Health Solution for Medication Adherence Support in Cardiovascular Prevention
6	"NCT02945371	Tailored Inhibitory Control Training to Reverse EA-linked Deficits in Mid-life

3,222 rows | Truncated data | 3.82 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[4]`. Learn more

ommand took 3.82 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 12:25:48 AM on TASK

Cmd 5

```
%sql
-- Retrieve all columns from the pharmaceutical data
SELECT * FROM pharma_file_view
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Company: string, Parent_Company: string ... 32 more fields]

Table +

	Company	Parent_Company
1	Abbott Laboratories	Abbott Laboratories
2	Abbott Laboratories Inc.	AbbVie
3	Abbott Laboratories Inc.	AbbVie
4	Abbott Laboratories Puerto Rico, Inc.	Abbott Laboratories
5	Acclarent Inc.	Johnson & Johnson

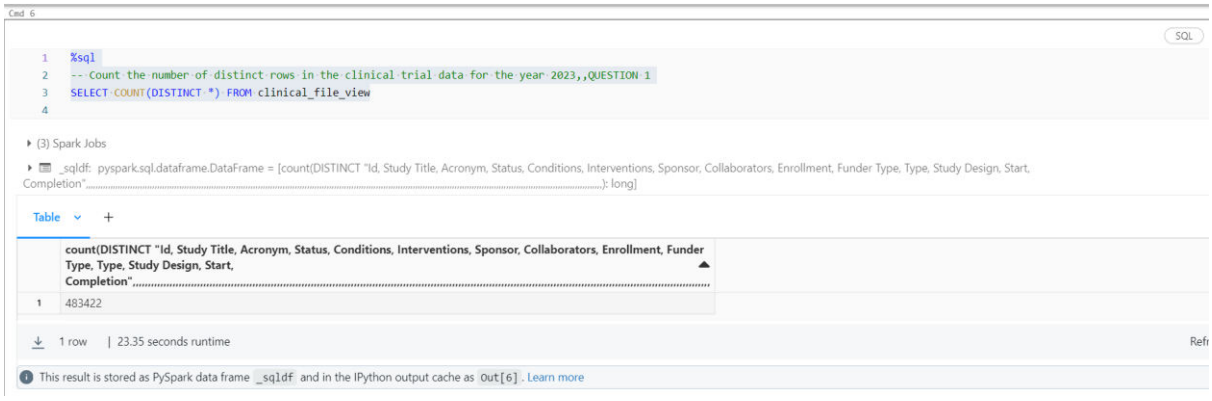
968 rows | 1.36 seconds runtime

This result is stored as PySpark data frame `_sqldf` and in the IPython output cache as `Out[5]`. Learn more

Command took 1.36 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 12:25:48 AM on TASK

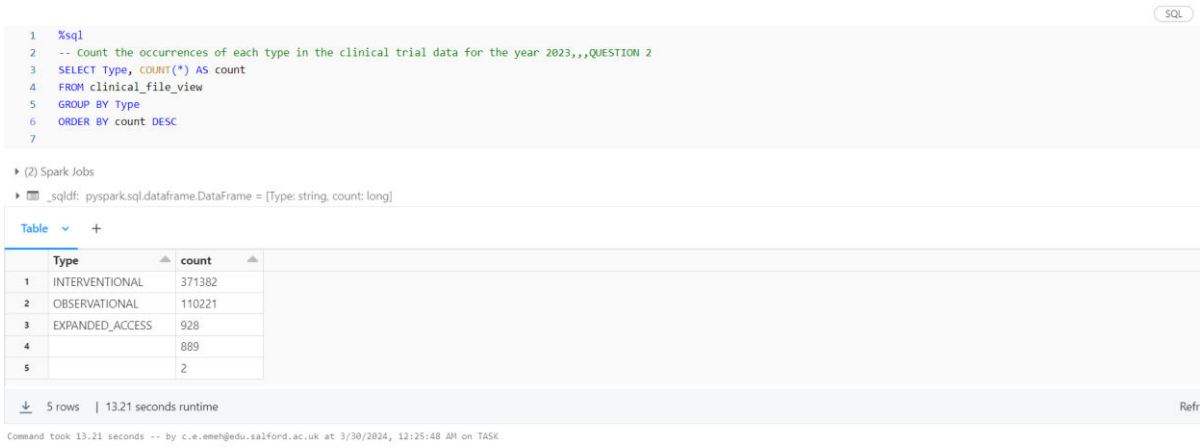
8.3.2 Task1 in SQL: The number of studies in the dataset

I extracted data using feature like the `%sql` directive, which signals that a code cell has SQL queries in it. To include columns or expressions in the query result, I used the `SELECT` command. The number of rows in a result set was counted using the `COUNT` function, either for all rows (*) or based on a particular column or expression. By removing duplicate values before counting, the `DISTINCT keyword makes sure that only distinct values are counted.
In the table, each column is represented by an asterisk (*), counting unique rows in each column. The table or view from which to get data is specified by the `FROM` clinical_file_view command.



8.3.2 Task 2 in SQL: All the types of studies in the dataset (the frequencies of each type from most frequent).

The type and the number of rows will be the two fields that the query chooses from the clinical_file_view table. To determine the total number of rows in each group, I used the COUNT (*) function. The result is renamed to count using the AS count alias. It indicated which table the data will be obtained from. I ensured that entries with the same value are aggregated, the GROUP BY Type clause groups rows according to the values in the Type column. The highest count groups display first using the ORDER BY count DESC order, which arranges the result determined by the count column in descending order.



8.3.3 Task1 in SQL: The top 5 conditions with their frequencies

Generating virtual table named `all conditions, a temporary view created by a query, or replaces it. By dividing the values in the `conditions` column by the pipe character ({|}), it specified the query that is used to fill the `all conditions' view. By splitting the string according to the supplied delimiter, the `split()` generates an array of values. Essentially "exploding" the array into numerous rows, the `explode()` turns each member of the array into a new row. Within all conditions, the SELECT conditions, COUNT(*) AS count ORDER BY Count, GROUP BY conditions, used to count the occurrences of all conditions. Using the `GROUP BY conditions` phrase, the count is aggregated for every distinct condition, and the `ORDER BY count DESC arranges the outcomes in a descending order with most frequent.

8.3.4 Task 4 in SQL; The 10 most common sponsors and along the number of clinical trials they have sponsored.

The pharmaceutical sponsors are removed from the data and a temporary view named `non_pharma_sponsor` is created. I choose the clinical trial information from the `clinical_file_view` table's `Sponsor' column. Sponsors that are not listed in the `Parent_Company` column of the `pharma_file_view` database are excluded by the `WHERE Sponsor NOT IN (SELECT Parent_Company FROM pharma_file_view) ` filter. From the `non_pharma_sponsor` view, separate sponsors are chosen, and the number of instances of each sponsor is counted using the `SELECT Sponsor, COUNT (*) AS count FROM non_pharma_sponsor` command.

SQL

```
%sql
-- Create a temporary view to select non-pharmaceutical sponsors from the clinical trial data for 2023,,,,QUESTION 4
CREATE OR REPLACE TEMP VIEW non_pharma_sponsor AS
    SELECT Sponsor
    FROM clinical_file_view
    WHERE Sponsor NOT IN (SELECT Parent_Company FROM pharma_file_view);

-- Count the occurrences of each non-pharmaceutical sponsor and display the top 10
SELECT Sponsor, COUNT(*) AS count
FROM non_pharma_sponsor
GROUP BY Sponsor
ORDER BY count DESC
LIMIT 10;
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [Sponsor: string, count: long]

Table +

	Sponsor	count
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

10 rows | 15.84 seconds runtime

Command took 15.84 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/30/2024, 12:25:48 AM on TASK

8.3.5 Task 5 in SQL The number of completed studies for each month in 2023.

Extracting the month component of the completion date, I separated the dates into an array of strings using "-" as a delimiter, generating a temporary view ({extractedDates}) containing clinical trial data, and aliases the `Month' column as the result of the split process. To count the occurrences of each month, using the {Count(*)} function. Only completed trials from 2023 are included in the filtered rows based on the status column.
The script then uses a `CASE` statement to translate the numerical month values to their corresponding names and a `SELECT` statement on the temporary view `extractedDates` to convert month numbers to names. The result is allocated the alias {Month}of the `CASE` statement. The `Count` column from the `extractedDates` view is also selected.

```
%sql
create or replace temp view extractedDates as SELECT split(Completion, '-') [1] as Month, count(*) as Count
From clinical_file_view
where clinical_file_view.Status in ('COMPLETED', 'completed') and split(clinical_file_view.Completion, '-') [0] == '2023'
group by Month
order by Month;

Select
CASE Month
        WHEN '01' THEN 'January'
        WHEN '02' THEN 'February'
        WHEN '03' THEN 'March'
        WHEN '04' THEN 'April'
        WHEN '05' THEN 'May'
        WHEN '06' THEN 'June'
        WHEN '07' THEN 'July'
        WHEN '08' THEN 'August'
        WHEN '09' THEN 'September'
        WHEN '10' THEN 'October'
        WHEN '11' THEN 'November'
        WHEN '12' THEN 'December'
        ELSE 'Unknown'
END AS Month, Count from extractedDates
```

▸ (2) Spark Jobs

Table +

	Month	Count
1	January	1494
2	February	1272
3	March	1552
4	April	1324
5	May	1415
6	June	1619

12 rows | 13.67 seconds runtime

Command took 13.67 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/5/2024, 7:50:58 PM on task

8.4 DISCUSSION OF RESULTS.

8.4.1 Task 1: The number of studies in the dataset

The result is shown to be the same across all board. After using different commands in RDD, DataFrame and SQL to take the total count of studies within the dataset, the consistent result across board is 483422.

8.4.2 Task 2; All the types of studies in the dataset (the frequencies of each type from most frequent).

The types of studies in the dataset are.
Interventional, 371382
Observational, 110221
Expanded Access, 928
The most frequently occurring study is 'Interventional studies' with it having taken place 371,382 times.
The least occurring type of study is the Expanded Access studies with it having just 928 entries.

8.4.3 Task 3; The top 5 conditions with their frequencies

The top 5 conditions in this study are.

Healthy, 9731
Breast Cancer, 7502
Obesity, 6549
Stroke, 4073
Hypertension, 4024

Healthy has the highest condition with 9731.
Breast Cancer has the second highest condition with 7502, while Hypertension has the least number of occurrences of 4024 conditions and the results are the same across all board.

8.4.3 Task 4; The 10 most common sponsors and along the number of clinical trials they have sponsored.

The result was consistent across all board.

i National Cancer Institute (NCI), 3410
ii Assiut University, 3335
iii Cairo University, 3023
iv Assistance Publique - Hôpitaux de Paris, 2951
v Mayo Clinic, 2766
vi M.D. Anderson Cancer Center, 2702
vii Novartis Pharmaceuticals, 2393
viii National Institute of Allergy and Infectious Diseases (NIAID), 2340
ix Massachusetts General Hospital, 2263
x National Taiwan University Hospital, 2181

National Cancer Institute has the highest number of trials conducted with 3410. Assiut University has the second highest Number with 3335. National Taiwan University Hospital has the 10th largest number with 2181.

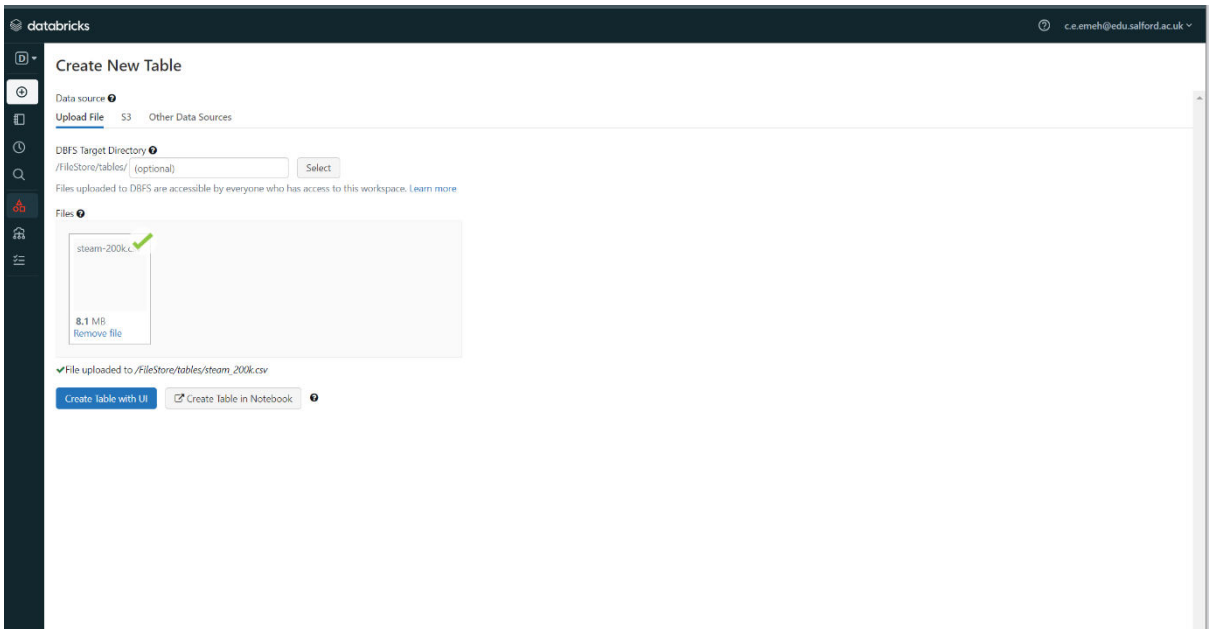
8.4.3 Task 5; The number of completed studies for each month in 2023.

The results are the same across all board, the month with the highest number of completed studies is June, March is a close second, while November has lowest number of completed trial studies.

PART 2 MACHINE LEARNING (RECOMENDER SYSTEM).

9.0 Downloading Dataset into the DataBricks.

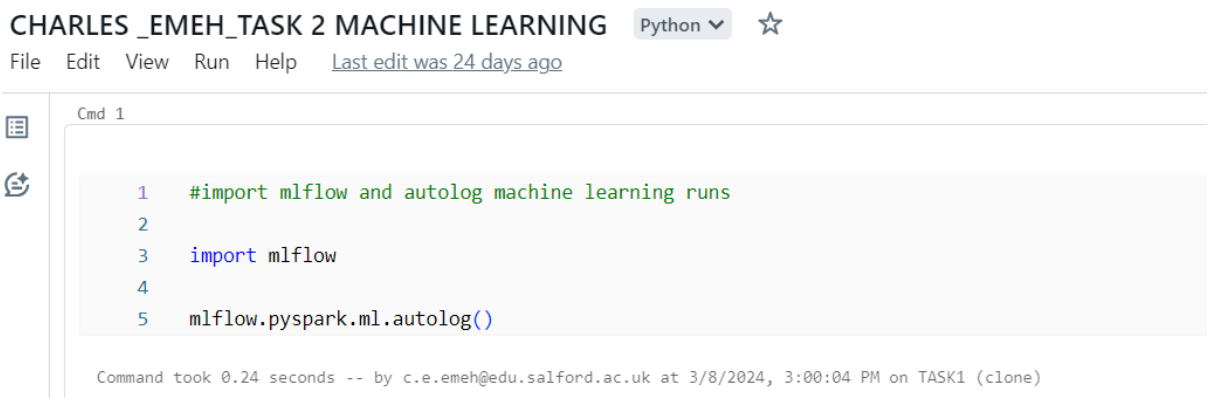
The Dataset in csv file has been downloaded via blackboard and unpacked within Databricks. Datasets are already in columns and separated values format. Examining the dataset in Excel reveals that it already has excellent specified and named column within the dataset, rows and columns have valid names. Although it is necessary, more cleaning and transformation might produce results that are more accurate.



	A	B	C	D
1	1.52E+08	The Elder S	purchase	1
2	1.52E+08	The Elder S	play	273
3	1.52E+08	Fallout 4	purchase	1
4	1.52E+08	Fallout 4	play	87
5	1.52E+08	Spore	purchase	1
6	1.52E+08	Spore	play	14.9
7	1.52E+08	Fallout Ne	purchase	1
8	1.52E+08	Fallout Ne	play	12.1
9	1.52E+08	Left 4 Dea	purchase	1
10	1.52E+08	Left 4 Dea	play	8.9
11	1.52E+08	HuniePop	purchase	1
12	1.52E+08	HuniePop	play	8.5
13	1.52E+08	Path of Exi	purchase	1
14	1.52E+08	Path of Exi	play	8.1
15	1.52E+08	Poly Bridge	purchase	1
16	1.52E+08	Poly Bridge	play	7.5
17	1.52E+08	Left 4 Dea	purchase	1
18	1.52E+08	Left 4 Dea	play	3.3
19	1.52E+08	Team Forti	purchase	1
20	1.52E+08	Team Forti	play	2.8

9.1 Importing PySpark Machine Learning Library

Using an open-source platform for handling the whole machine learning lifecycle, MLflow and PySpark MLlib, a Python API for Apache Spark, I imported this method by `mlflow.pyspark.ml.autolog() permitting automated logging of machine learning runs. Without needing explicit logging calls in the code, MLflow's autologging feature automatically records details about machine learning runs, including parameters, metrics, and models. This makes tracking experiments and model performance easier. The PySpark MLlib function `PySpark ML Autologging` facilitates autologging for machine learning algorithms and Spark processing tools for massive datasets.



9.2 Defining Dataset

I assigned the file pertaining to Steam data to `steam_file` variable, which is defined in this code snippet. The path to the CSV file in the FileStore is then represented by a string that is created as a variable called `file_path'. The `file_head' variable contains the first few lines of the CSV file, which are read by the `dbutils.fs.head()` method. The `print()` function is then used to print the contents of the file, resulting in the notebook output showing the first few lines of the CSV file. With the help of Databricks utilities, this code snippet successfully fetches the path to a CSV file, reads the first few lines, and publishes contents to the notebook output.

9.3 Importing Pyspark Function for Implementation (Creating Dataframe)

I used the codes below to import the `pyspark.sql.types` module's `StructType` and `StructField` classes, which specify a DataFrame's structure. Additionally, it imports every function from the `pyspark.sql.functions module, which is necessary for a number of DataFrame activities, including aggregation and data manipulation.
The name of the CSV file to be read is represented by the argument `file_name', which is taken by the function `create_dataframe(file_name). The directory path where the CSV file is placed and the `file_name parameter are concatenated to create the `file_path. The `spark.read.options().csv()` method is used by the function to read the CSV file into a DataFrame while maintaining comma separation. Next, the DataFrame is given back to CSV file to be read is named `steam_200k}, and this is also specified in the code.

Cmd 3

```
from pyspark.sql.types import StructType, StructField
from pyspark.sql.functions import *

def create_dataframe(file_name):
    # Define the file path
    file_path = f"/FileStore/tables/{file_name}.csv"

    # Read the CSV file into a DataFrame
    df = spark.read.options(delimiter=",").csv(file_path)

    return df

# Specify the file name for the Steam dataset
steam_file = "steam_200k"

# Create a DataFrame for the Steam dataset
steam_dataframe = create_dataframe(steam_file)

# Show the first 100 rows of the DataFrame
steam_dataframe.show(100)

```

▸ (2) Spark Jobs

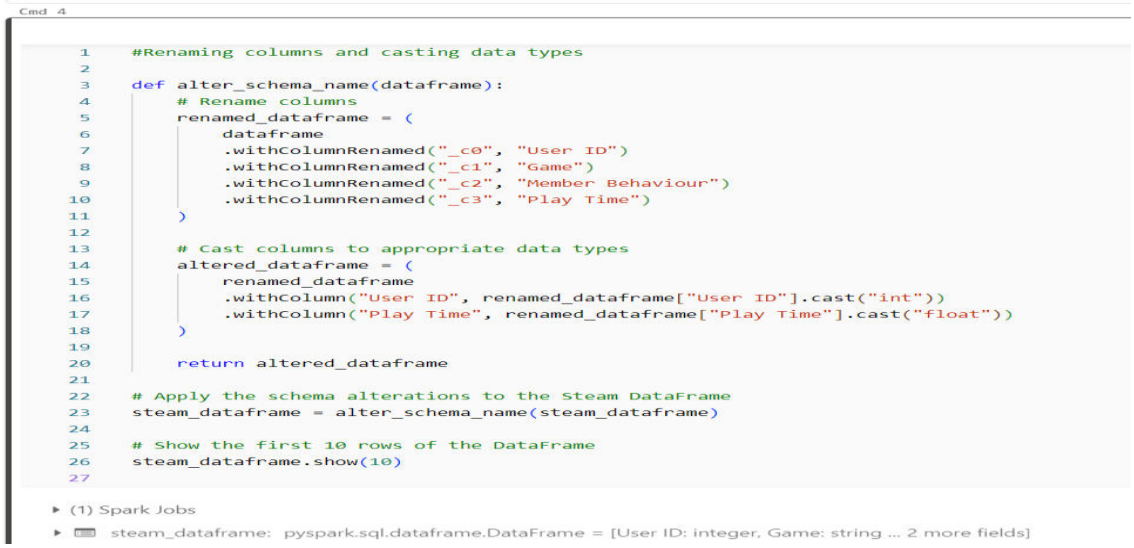
▸ steam_dataframe: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 2 more fields]

```
+---------+--------------------+--------+----+
|      _c0|                 _c1|     _c2| _c3|
+---------+--------------------+--------+----+
|151603712|The Elder Scrolls...|purchase|   1|
|151603712|The Elder Scrolls...|    play| 273|
|151603712|           Fallout 4|purchase|   1|
|151603712|           Fallout 4|    play|  87|
|151603712|               Spore|purchase|   1|
|151603712|               Spore|    play|14.9|
|151603712|   Fallout New Vegas|purchase|   1|
|151603712|   Fallout New Vegas|    play|12.1|
|151603712|       Left 4 Dead 2|purchase|   1|
|151603712|       Left 4 Dead 2|    play| 8.9|
|151603712|            HuniePop|purchase|   1|
|151603712|            HuniePop|    play| 8.5|
|151603712|       Path of Exile|purchase|   1|
|151603712|       Path of Exile|    play| 8.1|
|151603712|         Poly Bridge|purchase|   1|
|151603712|         Poly Bridge|    play| 7.5|
|151603712|         Left 4 Dead|purchase|   1|
|151603712|         Left 4 Dead|    play| 3.3|
```

Command took 7.98 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/8/2024, 3:21:06 PM on TASK1 (clone)

9.4 Renaming columns and casting datatypes.

The Python function `alter_schema_name' was employed to adjust the schema of a DataFrame, particularly referred to as `dataframe'. The `withColumnRenamed` function is used to rename columns; the `withColumn` function is used to cast data types; and the amended DataFrame containing the renamed columns and changed data types is returned. Next, using the `alter_schema_name' function, the updated schema is applied to the original DataFrame ({steam_dataframe}) and assigned to `steam_dataframe'.



```
Cmd 4

#Renaming columns and casting data types

def alter_schema_name(dataframe):
    # Rename columns
    renamed_dataframe = (
        dataframe
        .withColumnRenamed("_c0", "User ID")
        .withColumnRenamed("_c1", "Game")
        .withColumnRenamed("_c2", "Member Behaviour")
        .withColumnRenamed("_c3", "Play Time")
    )

    # Cast columns to appropriate data types
    altered_dataframe = (
        renamed_dataframe
        .withColumn("User ID", renamed_dataframe["User ID"].cast("int"))
        .withColumn("Play Time", renamed_dataframe["Play Time"].cast("float"))
    )

    return altered_dataframe

# Apply the schema alterations to the Steam DataFrame
steam_dataframe = alter_schema_name(steam_dataframe)

# Show the first 10 rows of the DataFrame
steam_dataframe.show(10)

▸ (1) Spark Jobs
▸ steam_dataframe: pyspark.sql.dataframe.DataFrame = [User ID: integer, Game: string ... 2 more fields]
```

```
+---------+--------------------+----------------+---------+
|  User ID|                Game|Member Behaviour|Play Time|
+---------+--------------------+----------------+---------+
|151603712|The Elder Scrolls...|        purchase|      1.0|
|151603712|The Elder Scrolls...|            play|    273.0|
|151603712|          Fallout 4|        purchase|      1.0|
|151603712|          Fallout 4|            play|     87.0|
|151603712|               Spore|        purchase|      1.0|
|151603712|               Spore|            play|     14.9|
|151603712| Fallout New Vegas|        purchase|      1.0|
|151603712| Fallout New Vegas|            play|     12.1|
|151603712|      Left 4 Dead 2|        purchase|      1.0|
|151603712|      Left 4 Dead 2|            play|      8.9|
+---------+--------------------+----------------+---------+
only showing top 10 rows

Command took 1.36 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/8/2024, 3:32:22 PM on TASK1 (clone)
```

9.5 Task 1: Total Number of distinct records

I imported a function in Apache Spark called `steam_dataframe.distinct().count()' eliminates duplicate rows from the `steam_dataframe' and creates a new DataFrame with unique rows. The original DataFrame's number of unique rows is counted using this approach. While the `steam_dataframe.count()` function counts the number of unique rows in the original DataFrame, the `steam_dataframe.distinct()` method creates a new DataFrame with unique rows.

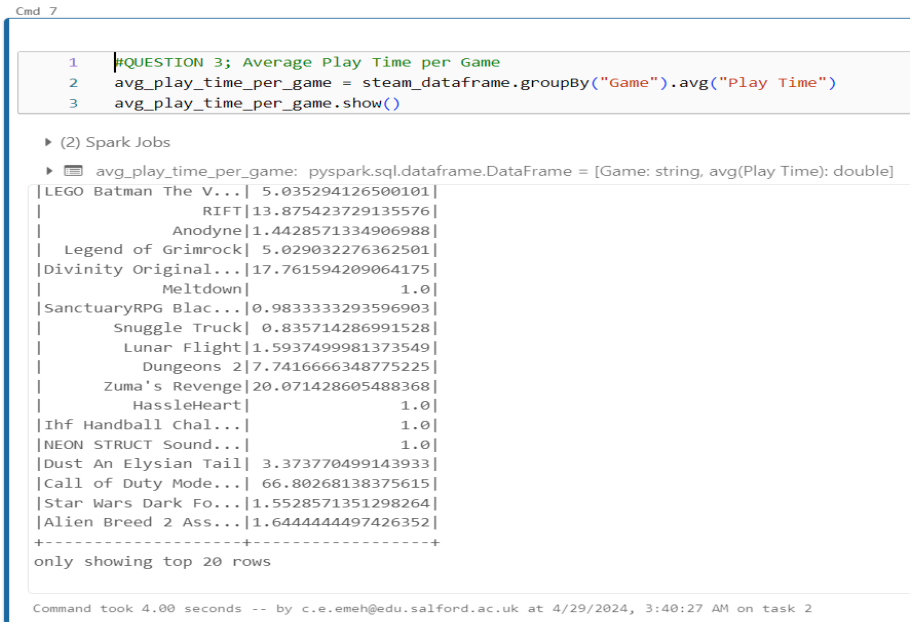
```
#QUESTION 1: Number of distinct records

steam_dataframe.distinct().count()

▸ (3) Spark Jobs
199293

Command took 6.74 seconds -- by c.e.emeh@edu.salford.ac.uk at 3/8/2024, 3:59:53 PM on TASK1 (clone)
```

9.5.1 Task 2: Percentage of Purchased vs played games

By counting the number of rows in the DataFrame {steam_dataframe}, the code determines the total number of records. The total number of purchases is obtained by filtering the DataFrame to only include rows where the "Member Behaviour" column equals "purchase". To get the total number of plays, the same procedure is used to filter the DataFrame so that only rows with "play" in the "Member Behaviour" column are included. By dividing the total number by the "Member Behaviour" column and multiplying the result by 100, the percentage of purchases is determined. By dividing the "play_count" by the "Total_records" column and multiplying the result by 100, one can determine the percentage of plays.

9.5.2 Task 3: Average Play Time per Game

By classifying the data according to game titles, the code determines the average play time for each group of games in the dataset. For every group, the "Play Time" column's mean value is calculated. Next, the average play time for each game is shown by displaying the average play time per game.

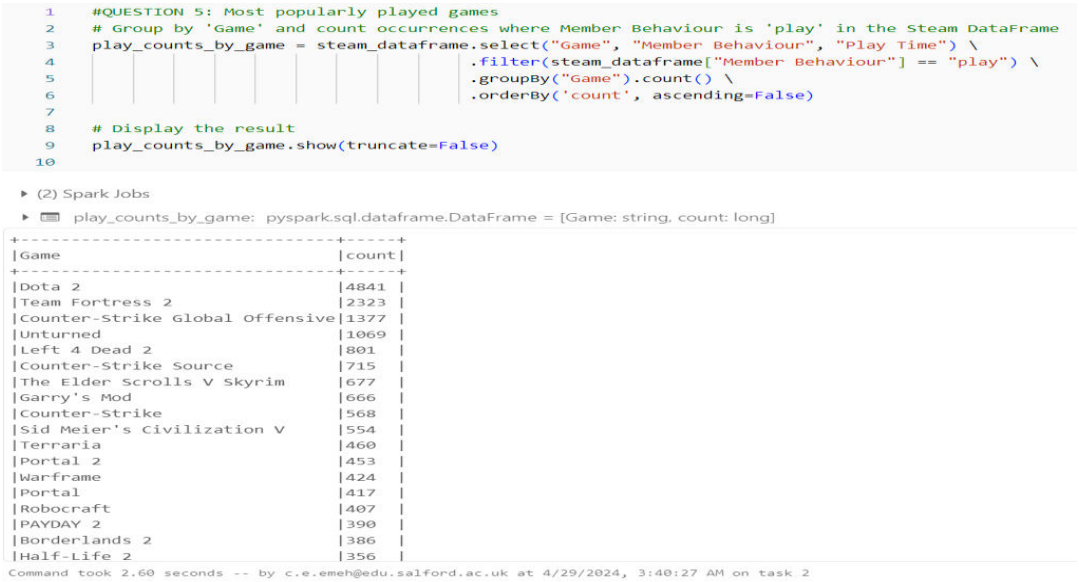


9.5.3 Task 4: Total Number of Games Played

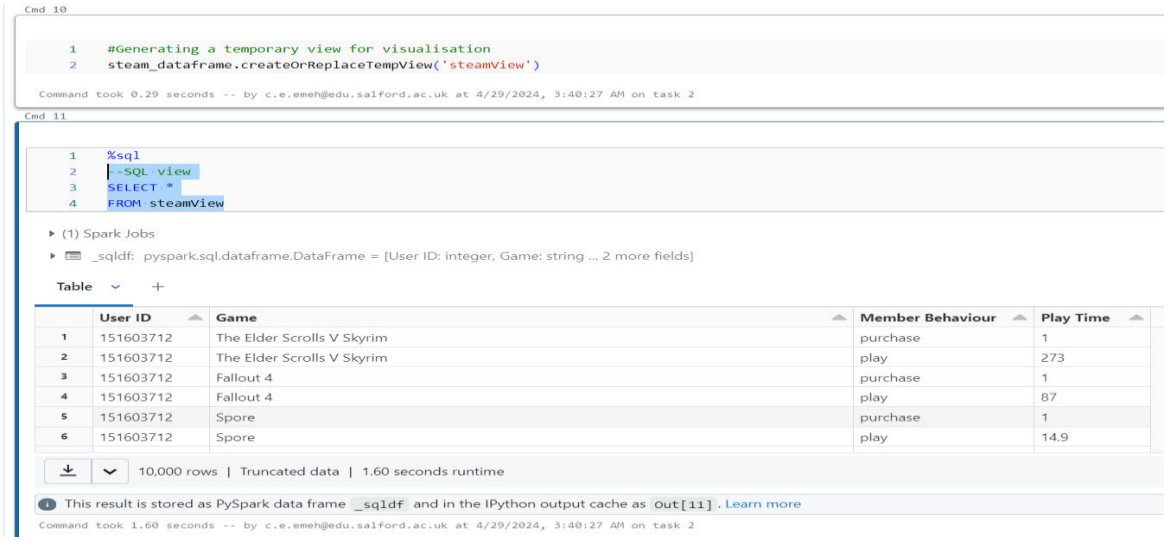
The code snippets create groups with all rows having the same value by grouping the DataFrame {steam_dataframe` according to the "Game" column. The longest play time for each game is determined by calculating the maximum value of the "Play Time" column within each group. The game with the longest play time appears first in the results, which are arranged by maximum play time in descending order. Then it shows the play time that is the shortest. These little pieces of code assist in determining which game in the {steam_dataframe} DataFrame has the longest and shortest play times.

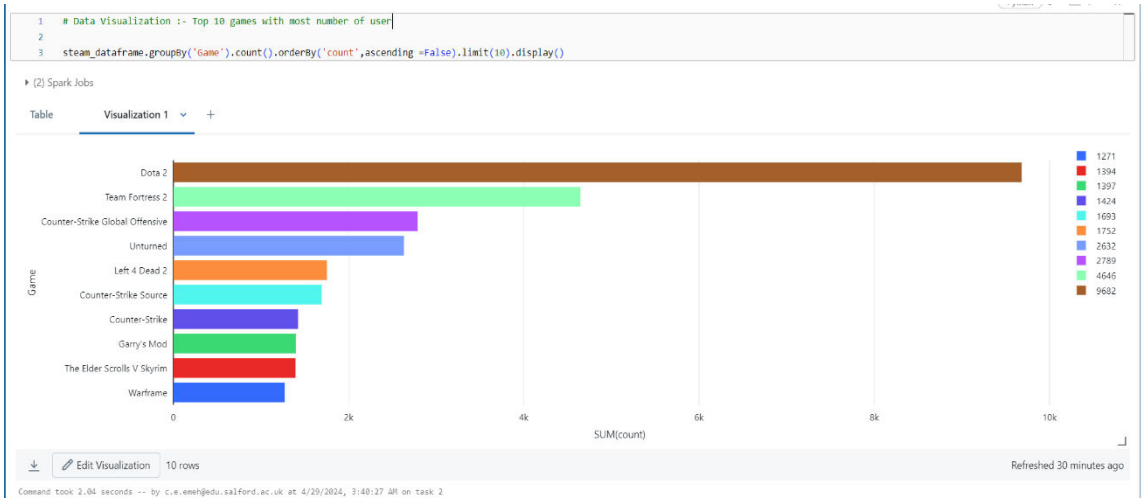
9.5.4 Task 5: Most Popular Played Games and its visualization

'Game' grouping, Member Behaviour filtering, 'Count()' method, 'OrderBy('count', ascending=False)' function, and 'Result presentation' are the five stages of the code. Only rows where the value of the 'Member Behaviour' column equals 'play' are filtered out of the filtered DataFrame, which is grouped by the 'Game' column. The number of rows in each group is counted by the 'Count()' method, which shows how many times each game is played during play sessions. The DataFrame is arranged in descending order by the 'OrderBy('count', ascending=False)' function, which guarantees that the games with the highest number of 'play' occurrences are shown first.



Use the Spark DataFrame API's `steam_dataframe.createOrReplaceTempView('steamView')} method to create a temporary view from a DataFrame. This view is transient and available only during the SparkSession.
Next, to run a SQL query on the temporary view. All columns from the temporary view `steamView` are retrieved using the query `SELECT * FROM steamView}, which provides a tabular representation of the complete dataset.





9.5.5 Task 6: Most Popular Games Purchased

I chose three columns—"Game," "Member Behaviour," and "Play Time" from the "steam_dataframe". The DataFrame is filtered such that only rows with the value "purchase" in the "Member Behaviour" column are included. Next, the "Game" column is used to group the filtered DataFrame and count how many times each game appears. The games with the greatest purchase counts are shown first in the decreasing order of the aggregated DataFrame. Using the `show()` method, the final DataFrame, {purchase_counts_by_game}, is shown, guaranteeing that every cell's information is presented in its entirety without being truncated.

```
# #QUESTION 6: Most popularly purchased games
# Count occurrences where Member Behaviour is 'purchase' in the Steam DataFrame, grouped by 'Game'
purchase_counts_by_game = steam_dataframe.select("Game", "Member Behaviour", "Play Time") \
                                         .filter(steam_dataframe["Member Behaviour"] == "purchase") \
                                         .groupBy("Game").count() \
                                         .orderBy('count', ascending=False)

# Display the result
purchase_counts_by_game.show(truncate=False)
```

▸ (2) Spark Jobs

▸ purchase_counts_by_game: pyspark.sql.dataframe.DataFrame = [Game: string, count: long]

Game	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 1.68 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/29/2024, 3:40:27 AM on task 2

9.5.6 Task 7: Total Games with the Highest Number of Play time (hours)

Choosing three columns from the DataFrame: "Game", "Member Behaviour", and "Play Time". The code filtered the DataFrame to include only rows where the "Member Behaviour" column value is equal to "play". The DataFrame is then grouped by the "Game" column and the "Play Time" values for each game are summed up within each group to calculate the total playtime for each game. The resulting DataFrame is ordered by the summed playtime column in descending order, with games with the highest total playtime appearing first.

```
#QUESTION 7: Games with the highest number of play time in hours
# Sum 'Play Time' where Member Behaviour is 'play' in the Steam DataFrame, grouped by 'Game'
playtime_sum_by_game = steam_dataframe.select("Game", "Member Behaviour", "Play Time") \
                                      .filter(steam_dataframe["Member Behaviour"] == "play") \
                                      .groupBy("Game").sum("Play Time") \
                                      .orderBy('sum(Play Time)', ascending=False)

# Display the result
playtime_sum_by_game.show(truncate=False)
```

▸ (2) Spark Jobs

▸ playtime_sum_by_game: pyspark.sql.dataframe.DataFrame = [Game: string, sum(Play Time): double]

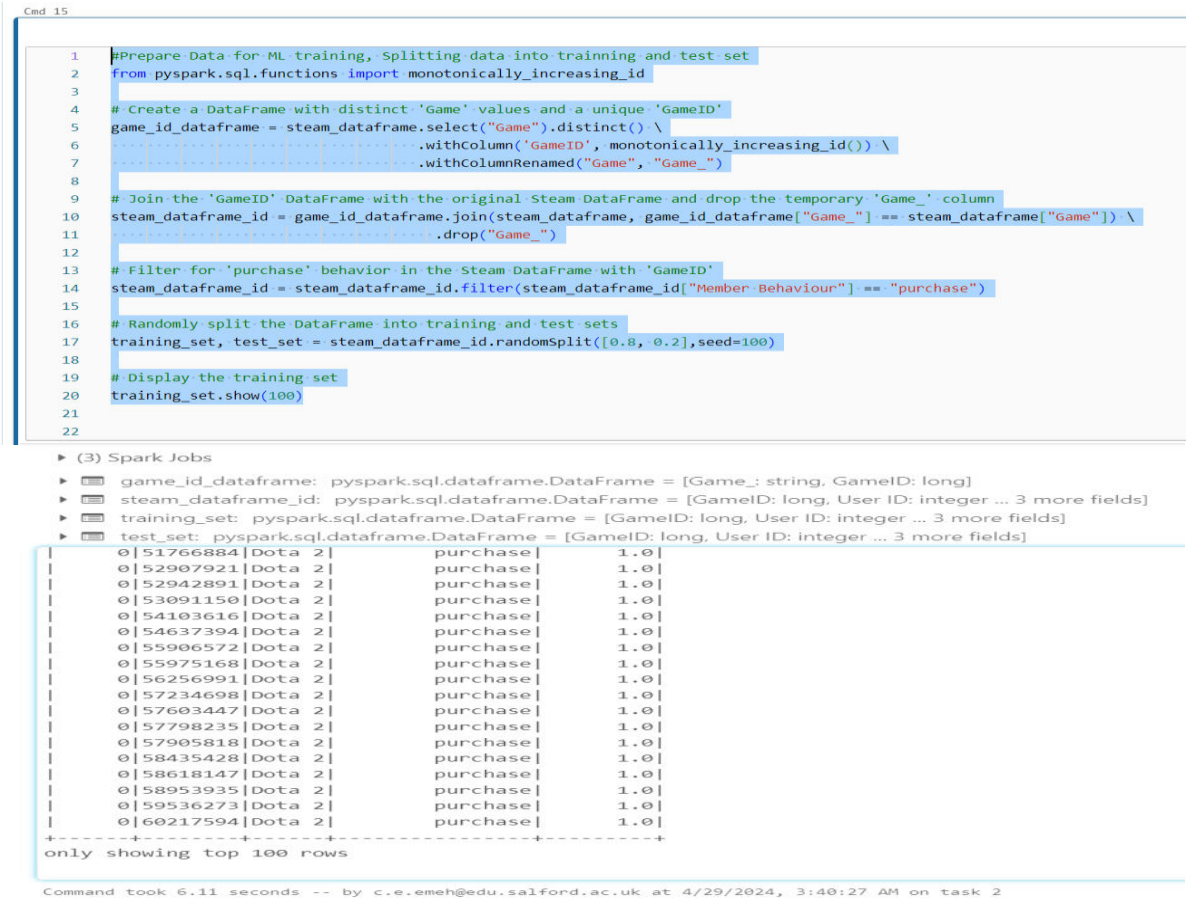
Game	sum(Play Time)
Team Fortress 2	173673.30000534654
Counter-Strike	134261.1000032574
Sid Meier's Civilization V	99821.30000032485
Counter-Strike Source	96075.4999980852
The Elder Scrolls V Skyrim	70889.30000342429
Garry's Mod	49725.300001084805
Call of Duty Modern Warfare 2 - Multiplayer	42009.8999973014
Left 4 Dead 2	33596.70000024885
Football Manager 2013	32308.599999904633
Football Manager 2012	30845.80000168085
Football Manager 2014	30574.800000548363
Terraria	29951.80000526458
Warframe	27074.599995546043
Football Manager 2015	24283.10000061989
Arma 3	24055.700000062585
Grand Theft Auto V	22956.699999585748
Borderlands 2	22667.900001987815
Empire Total War	21030.300002798438

only showing top 20 rows

Command took 3.19 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/3/2024, 8:30:25 AM on task

9.6 Preparing Data for Machine Learning Training (For Purchased Behavior)

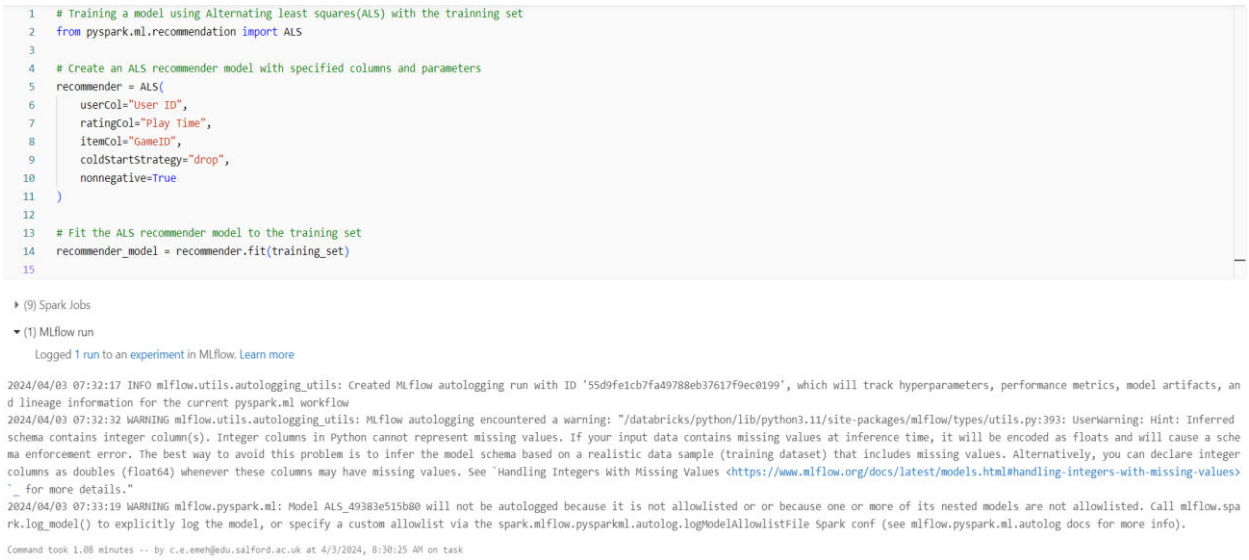
The `pyspark.sql` module is used to create the DataFrame, which has distinct game names and IDs. Unique IDs are produced by the `distinct()` function, whereas unique IDs are produced by the `monotonically_increasing_id()` function. A distinct GameID is appended to every row of the original DataFrame by joining it with the `game_id_dataframe`. Next, the `steam_dataframe_id} is filtered so that only rows with 'purchase' in the 'Member Behaviour' column are included. Using the `randomSplit()` function, the DataFrame is then divided into training and test sets at a ratio of 80:20. By giving each game a unique identifier, screening the data for purchase behavior, and dividing it into training and test sets for machine learning, these steps prepare the data for training.



```
Cmd 15

#Prepare Data for ML training, Splitting data into trainning and test set
from pyspark.sql.functions import monotonically_increasing_id

# Create a DataFrame with distinct 'Game' values and a unique 'GameID'
game_id_dataframe = steam_dataframe.select("Game").distinct() \
                                   .withColumn('GameID', monotonically_increasing_id()) \
                                   .withColumnRenamed("Game", "Game_")

# Join the 'GameID' DataFrame with the original Steam DataFrame and drop the temporary 'Game_' column
steam_dataframe_id = game_id_dataframe.join(steam_dataframe, game_id_dataframe["Game_"] == steam_dataframe["Game"]) \
                                      .drop("Game_")

# Filter for 'purchase' behavior in the Steam DataFrame with 'GameID'
steam_dataframe_id = steam_dataframe_id.filter(steam_dataframe_id["Member Behaviour"] == "purchase")

# Randomly split the DataFrame into training and test sets
training_set, test_set = steam_dataframe_id.randomSplit([0.8, 0.2],seed=100)

# Display the training set
training_set.show(100)

▸ (3) Spark Jobs
▸ game_id_dataframe: pyspark.sql.dataframe.DataFrame = [Game_: string, GameID: long]
▸ steam_dataframe_id: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
▸ training_set: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
▸ test_set: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
|      0|51766884|Dota 2|          purchase|       1.0|
|      0|52907921|Dota 2|          purchase|       1.0|
|      0|52942891|Dota 2|          purchase|       1.0|
|      0|53091150|Dota 2|          purchase|       1.0|
|      0|54103616|Dota 2|          purchase|       1.0|
|      0|54637394|Dota 2|          purchase|       1.0|
|      0|55906572|Dota 2|          purchase|       1.0|
|      0|55975168|Dota 2|          purchase|       1.0|
|      0|56256991|Dota 2|          purchase|       1.0|
|      0|57234698|Dota 2|          purchase|       1.0|
|      0|57603447|Dota 2|          purchase|       1.0|
|      0|57798235|Dota 2|          purchase|       1.0|
|      0|57905818|Dota 2|          purchase|       1.0|
|      0|58435428|Dota 2|          purchase|       1.0|
|      0|58618147|Dota 2|          purchase|       1.0|
|      0|58953935|Dota 2|          purchase|       1.0|
|      0|59536273|Dota 2|          purchase|       1.0|
|      0|60217594|Dota 2|          purchase|       1.0|
+-------+--------+------+------------------+----------+
only showing top 100 rows

Command took 6.11 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/29/2024, 3:40:27 AM on task 2
```

9.7 Training Model using Alternating Least Squares (ALS)

Spark MLlib is used for training recommendation systems, generating an ALS recommender model. The model requires configuration of parameters like userCol, ratingCol, itemCol, coldStartStrategy, and nonnegative=True. The 'drop' technique ensures non-negative ratings, while the 'fit' technique fits the model to the training set, enabling objects to learn latent characteristics.



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

# Create an ALS recommender model with specified columns and parameters
recommender = ALS(
    userCol="User ID",
    ratingCol="Play Time",
    itemCol="GameID",
    coldStartStrategy="drop",
    nonnegative=True
)

# Fit the ALS recommender model to the training set
recommender_model = recommender.fit(training_set)

```

▸ (9) Spark Jobs

▾ (1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

```
2024/04/03 07:32:17 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '55d9fe1cb7fa49788eb37617f9ec0199', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/04/03 07:32:32 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/databricks/python/lib/python3.11/site-packages/mlflow/types/utils.py:393: UserWarning: Hint: Inferred schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference time, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (training dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details."
2024/04/03 07:33:19 WARNING mlflow.pyspark.ml: Model ALS_49383e515b80 will not be autologged because it is not allowlisted or or because one or more of its nested models are not allowlisted. Call mlflow.spark.log_model() to explicitly log the model, or specify a custom allowlist via the spark.mlflow.pysparkml.autolog.logModelAllowlistFile Spark conf (see mlflow.pyspark.ml.autolog docs for more info).
```

Command took 1.08 minutes -- by c.e.emeh@edu.salford.ac.uk at 4/3/2024, 8:30:25 AM on task

9.8 Testing and Generation of Predictions using ALS Recommender Model

I generated predictions by applying a trained recommender model to a test dataset. The results are shown in tabular form. Through training data, the model discovers patterns in user-item interactions and forecasts ratings or preferences for items in the test set that users have not yet interacted with. Users can examine the anticipated ratings or preferences for the test set items in this way.

```
#Testing the recommendation system by predicting game play time vs the actual play time
# Generate predictions using the trained ALS recommender model on the test set
predictions = recommender_model.transform(test_set)

# Display the predictions
predictions.show(100)

```

▸ (6) Spark Jobs

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

```
|      3| 80128229|                 RIFT|             play|       3.0|    248.7106|
|      3|140825164|                 RIFT|             play|       1.2|    8.611023|
|      5| 24721232|  Legend of Grimrock|             play|       0.8|    27.91111|
|      6| 44472980|Divinity Original...|             play|      80.0|   56.366043|
|      6| 48984158|Divinity Original...|             play|       1.1|   786.10565|
|      6| 55426012|Divinity Original...|             play|       9.3|    33.56179|
|      6| 77378306|Divinity Original...|             play|       0.8|    7.654256|
|      6| 94110492|Divinity Original...|             play|       2.0|     13.7794|
|      6|106986812|Divinity Original...|             play|       4.1|     8.17012|
|      6|157080495|Divinity Original...|             play|       5.0|    65.16048|
|     16|  2259650|Dust An Elysian Tail|             play|       0.1| 0.75524026|
|     16| 30425578|Dust An Elysian Tail|             play|      51.0|     8.08618|
|     16| 42061089|Dust An Elysian Tail|             play|       2.6|    7.454772|
|     16| 57603447|Dust An Elysian Tail|             play|       1.0|    9.357597|
|     16| 65398650|Dust An Elysian Tail|             play|      12.3|    2.568655|
|     16|106238372|Dust An Elysian Tail|             play|       1.2|    8.139328|
|     17| 11161178|Call of Duty Mode...|             play|      32.0|   166.90099|
|     17| 11373749|Call of Duty Mode...|             play|       7.7|   46.788105|
+-------+---------+--------------------+-----------------+----------+------------+
only showing top 100 rows
```

Command took 11.14 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/3/2024, 8:30:25 AM on task

9.9 ALS Recommender Model Predictions for a Single User

By using the `recommender_model, I generated three components for a single_user DataFrame as the input data and applies the learned model to it. Based on the user's interactions with the objects in the dataset, the model produces suggestions for them;

1. Choosing the DataFrame {steam_dataframe_id} and its "User ID" column.
2. Applying the criteria {"User ID" == 246258933} to filter the DataFrame.
3. Assigning the variable {single_user} to the filtered DataFrame.

```
# Testing the recommendation system on a single user
single_user = steam_dataframe_id.filter(steam_dataframe_id["User ID"] == 246258933)
single_recommendation = recommender_model.transform(single_user)
single_recommendation.show()

```

▸ (6) Spark Jobs

▸ single_user: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]

▸ single_recommendation: pyspark.sql.dataframe.DataFrame = [GameID: integer, User ID: integer ... 4 more fields]

```
+------+---------+--------------------+-----------------+----------+----------+
|GameID|  User ID|                Game|Member Behaviour|Play Time|prediction|
+------+---------+--------------------+-----------------+----------+----------+
|    17|246258933|Call of Duty Mode...|             play|      37.0| 36.751076|
|  2634|246258933|      The Sims(TM) 3|             play|      12.2| 12.272305|
|  3893|246258933|     Team Fortress 2|             play|       5.1| 5.0290155|
|  3438|246258933|            Unturned|             play|       2.8|  3.699242|
|  3850|246258933|Call of Duty Mode...|             play|       1.5| 1.9760013|
|  2818|246258933|No More Room in Hell|             play|       1.4|0.72651356|
|  4283|246258933|               SMITE|             play|       0.1| 1.4382925|
|  3299|246258933|               Trove|             play|       0.1| 5.9648905|
+------+---------+--------------------+-----------------+----------+----------+
```

1

Command took 10.48 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/3/2024, 8:30:25 AM on task

10.0 Checking the effectiveness of the model using Root Mean Square Error (Purchased Behavior)

To gauge the effectiveness of machine learning models, the code segment imports evaluation metrics from the `pyspark.ml.evaluation} module. After that, it makes predictions for the test set based on user behavior by employing a recommender model. After initializing, the regression evaluator computes the root mean square error (RMSE) pertaining to the discrepancy between the predicted and actual labels. By applying the evaluator to the predictions, the `reg_evaluator` is utilized to assess the model. Next, the `%g} format specifier is used to print the RMSE to the console.

```
from pyspark.ml.evaluation import RegressionEvaluator, RankingEvaluator, BinaryClassificationEvaluator
```

```
Command took 0.08 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/29/2024, 3:40:27 AM on task 2
```

```
Cmd 20
```

```
predictions = recommender_model.transform(test_set)

# Regression evaluator
reg_evaluator = RegressionEvaluator(metricName="rmse", labelCol="Play Time", predictionCol="prediction")
rmse = reg_evaluator.evaluate(predictions)
print('Root Mean Square Error is %g ' %rmse)
```

```
▸ (6) Spark Jobs
▾ predictions: pyspark.sql.dataframe.DataFrame
    GameID: integer
    User ID: integer
    Game: string
    Member Behaviour: string
    Play Time: float
    prediction: float

Root Mean Square Error is 0.100701
```

```
Command took 13.91 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/29/2024, 3:40:27 AM on task 2
```

11.0 Preparing Data for Machine Learning Training (For Played Behavior)

```
#Prepare Data for ML training, Splitting data into trainning and test set
from pyspark.sql.functions import monotonically_increasing_id

# Create a DataFrame with distinct 'Game' values and a unique 'GameID'
game_id_dataframe = steam_dataframe.select("Game").distinct() \
                                   .withColumn('GameID', monotonically_increasing_id()) \
                                   .withColumnRenamed("Game", "Game_")

# Join the 'GameID' DataFrame with the original Steam DataFrame and drop the temporary 'Game_' column
steam_dataframe_id = game_id_dataframe.join(steam_dataframe, game_id_dataframe["Game_"] == steam_dataframe["Game"]) \
                                      .drop("Game_")

# Filter for 'play' behavior in the Steam DataFrame with 'GameID'
steam_dataframe_id = steam_dataframe_id.filter(steam_dataframe_id["Member Behaviour"] == "play")

# Randomly split the DataFrame into training and test sets
training_set, test_set = steam_dataframe_id.randomSplit([0.8, 0.2],seed=100)

# Display the training set
training_set.show(100)
```

```
▸ (3) Spark Jobs
▸ game_id_dataframe: pyspark.sql.dataframe.DataFrame = [Game_: string, GameID: long]
▸ steam_dataframe_id: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
▸ training_set: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
▸ test_set: pyspark.sql.dataframe.DataFrame = [GameID: long, User ID: integer ... 3 more fields]
|      0|51766884|Dota 2|              play|       1.7|
|      0|52907921|Dota 2|              play|      29.0|
|      0|52942891|Dota 2|              play|       0.3|
|      0|53091150|Dota 2|              play|      10.6|
|      0|54103616|Dota 2|              play|       3.1|
|      0|54637394|Dota 2|              play|       5.5|
|      0|55906572|Dota 2|              play|       2.0|
|      0|55975168|Dota 2|              play|       0.4|
|      0|56256991|Dota 2|              play|     723.0|
|      0|57234698|Dota 2|              play|       2.6|
|      0|57603447|Dota 2|              play|    1727.0|
|      0|57798235|Dota 2|              play|    1840.0|
|      0|57905818|Dota 2|              play|       0.6|
|      0|58435428|Dota 2|              play|       5.4|
|      0|58618147|Dota 2|              play|      34.0|
|      0|58953935|Dota 2|              play|       0.5|
|      0|59536273|Dota 2|              play|    2412.0|
|      0|60217594|Dota 2|              play|      17.7|
+-------+--------+------+------------------+----------+
only showing top 100 rows
```

```
Command took 2.71 seconds -- by c.e.emeh@edu.salford.ac.uk at 4/29/2024, 3:40:27 AM on task 2
```

12 OBSERVATIONS

A RMSE value is a crucial measure of a model's accuracy in predicting the target variable. It indicates the model's predictions are accurate, with minimal deviation from the true values. A low RMSE (0.100701) suggests excellent performance, while a high RMSE (233.752) indicates significant deviation from the actual values, indicating less accuracy and potential variability or bias and suggests further refinement.

13 Generate Personalized Recommendations for all Users Based on their Interactions with Items in a Dataset.

The collaborative filtering recommender model is a crucial tool in recommendation systems, allowing users to receive personalized recommendations based on their interactions with a dataset. The model gathers user preferences and suggests products like those they have liked or interacted with. Users can access the recommendations generated for each user in the resulting data structure, 'userRecs', also shown below.

14 Recommendation for a Specific User, their Games and Ratings.

The code uses PySpark SQL functions to analyze recommendation data in a DataFrame. It filters rows based on the specified ID in the "User ID" column, flattens the "recommendations" column, and selects the "GameID" and "rating" columns from the exploded recommendations. The resulting DataFrame is then joined with the "game_id_dataframe" DataFrame, ensuring the contents remain intact.

```
from pyspark.sql.functions import explode

userRecs.where(userRecs["User ID"] == 246258933).select("recommendations")\
.withColumn("recommendations", explode("recommendations"))\
.select("recommendations.GameID", "recommendations.rating") \
.join(game_id_dataframe, ["GameID"])\
.show(truncate=False)
```

▸ (5) Spark Jobs

GameID	rating	Game_
3430	0.9008629	Shadow Harvest Phantom Ops
617	0.9008056	The Amazing Spider-Man
2905	0.900684	The Cursed Crusade
1975	0.90064424	Tom Clancy's Ghost Recon Phantoms - NA Recon Starter Pack
4904	0.9004845	Duke Nukem Forever
663	0.9004452	Defend Your Life
0	0.9002001	Dota 2
3086	0.9000455	NBA 2K9
4340	0.900005	Sphere III Enchanted World
3263	0.8998284	Combat Monsters

Command took 36.70 seconds -- by c.e.emeh@edu.salford.ac.uk at 5/1/2024, 10:57:25 PM on task

15 Creating Parameter Grid with Different Combinations of Hyperparameter Values

The PySpark library is utilized for hyperparameter tuning in the Alternating Least Squares (ALS) model, utilizing parameters like user ID, item ID, rating, coldStartStrategy, and seed to ensure accurate tuning and prevent overfitting in the model.

```
from pyspark.ml.tuning import ParamGridBuilder

als = ALS(maxIter=10, userCol="User ID", itemCol="GameID", ratingCol="Play Time", coldStartStrategy="drop", seed=100)

# Create a parameter grid

parameters = ParamGridBuilder()\
.addGrid(als.rank, [5, 10, 15])\
.addGrid(als.regParam, [0.001, 0.005, 0.01, 0.05, 0.1])\
.build()
```

Command took 0.13 seconds -- by c.e.emeh@edu.salford.ac.uk at 5/1/2024, 11:01:10 PM on task

16 The Train Validation Split for Hyperparameter Tuning of the ALS Algorithm

The ALS model's performance is optimized using an evaluator, parameter grid, and RMSE metric. The ALS algorithm is used in the parameter grid, representing dictionaries, and the training data ratio is set to 0.75.

```
from pyspark.ml.tuning import TrainValidationSplit

# Define TrainValidationSplit

tvs = TrainValidationSplit()\
.setSeed(100)\
.setTrainRatio(0.75)\
.setEstimatorParamMaps(parameters)\
.setEstimator(als)\
.setEvaluator(reg_evaluator)
```

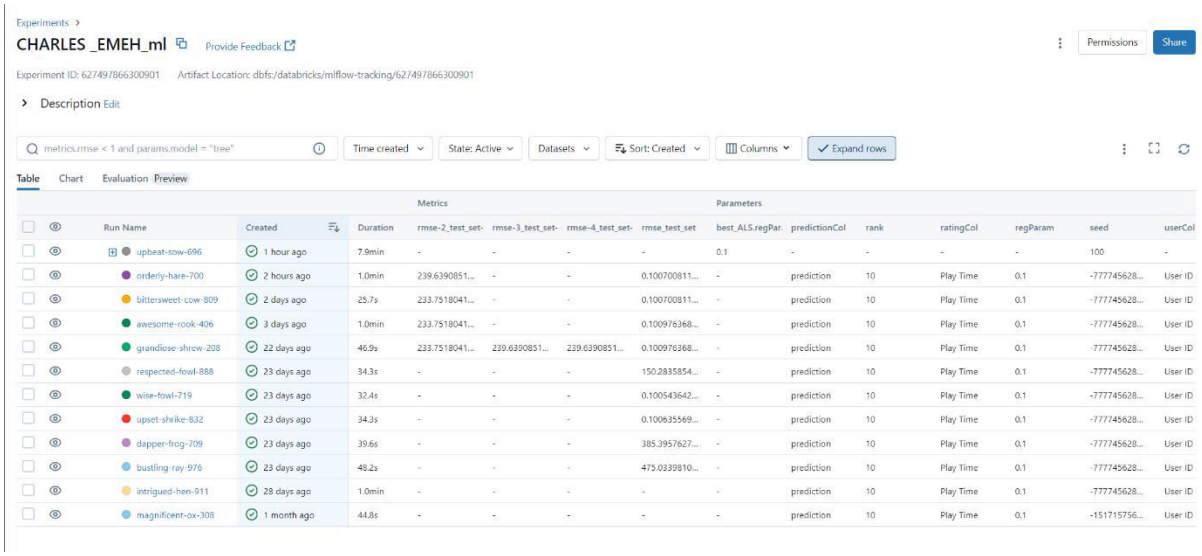
Command took 0.12 seconds -- by c.e.emeh@edu.salford.ac.uk at 5/1/2024, 11:02:33 PM on task

17 Machine Learning Hyperparameters Tuning.

Machine learning (ML) utilizes grid search for tuning hyperparameters before training. This technique optimizes model performance by dividing datasets into training and validation sets and storing results for analysis in the grid search Model variable.

```
1  gridsearchModel = tvs.fit(test_set)

▸ (9) Spark Jobs
▾ (16) MLflow runs
    Logged 16 runs to an experiment in MLflow. Learn more

2024/05/01 22:03:44 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '79603816e92f4ce99d723030b44fe62a', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
2024/05/01 22:03:46 WARNING mlflow.utils.autologging_utils: MLflow autologging encountered a warning: "/databricks/python/lib/python3.11/site-packages/mlflow/types/utils.py:393: UserWarning: Hint: Inferred schema contains integer column(s). Integer columns in Python cannot represent missing values. If your input data contains missing values at inference time, it will be encoded as floats and will cause a schema enforcement error. The best way to avoid this problem is to infer the model schema based on a realistic data sample (training dataset) that includes missing values. Alternatively, you can declare integer columns as doubles (float64) whenever these columns may have missing values. See `Handling Integers With Missing Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-with-missing-values>`_ for more details."
2024/05/01 22:11:35 WARNING mlflow.pyspark.ml: Model TrainValidationSplitModel_78b0d47cfd29 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 7.87 minutes -- by c.e.emeh@edu.salford.ac.uk at 5/1/2024, 11:03:43 PM on task
```

HYPERPARAMETERS VIEW



Experiments ›

CHARLES _EMEH_ml Provide Feedback

Permissions Share

Experiment ID: 627497866300901 Artifact Location: dbfs:/databricks/mlflow-tracking/627497866300901

› Description Edit

metrics.rmse < 1 and params.model = "tree" Time created State: Active Datasets Sort: Created Columns Expand rows

Table Chart Evaluation Preview

				Metrics					Parameters						
Run Name	Created	Duration	rmse-2_test_set-	rmse-3_test_set-	rmse-4_test_set-	rmse_test_set	best_ALS.regPar.	predictionCol	rank	ratingCol	regParam	seed	userCol
upbeat-sow-696	1 hour ago	7.9min	-	-	-	-	0.1	-	-	-	-	100	-
orderly-hare-700	2 hours ago	1.0min	239.6390851...	-	-	0.100700811...	-	prediction	10	Play Time	0.1	-777745628...	User ID
bittersweet-cow-809	2 days ago	25.7s	233.7518041...	-	-	0.100700811...	-	prediction	10	Play Time	0.1	-777745628...	User ID
awesome-rook-406	3 days ago	1.0min	233.7518041...	-	-	0.100976368...	-	prediction	10	Play Time	0.1	-777745628...	User ID
grandiose-shrew-208	22 days ago	46.9s	233.7518041...	239.6390851...	239.6390851...	0.100976368...	-	prediction	10	Play Time	0.1	-777745628...	User ID
respected-fowl-888	23 days ago	34.3s	-	-	-	150.2835854...	-	prediction	10	Play Time	0.1	-777745628...	User ID
wise-fowl-719	23 days ago	32.4s	-	-	-	0.100543642...	-	prediction	10	Play Time	0.1	-777745628...	User ID
upset-shrike-832	23 days ago	34.3s	-	-	-	0.100635569...	-	prediction	10	Play Time	0.1	-777745628...	User ID
dapper-frog-709	23 days ago	39.6s	-	-	-	385.3957627...	-	prediction	10	Play Time	0.1	-777745628...	User ID
bustling-ray-976	23 days ago	48.2s	-	-	-	475.0339810...	-	prediction	10	Play Time	0.1	-777745628...	User ID
intrigued-hen-911	28 days ago	1.0min	-	-	-	-	-	prediction	10	Play Time	0.1	-777745628...	User ID
magnificent-ox-308	1 month ago	44.8s	-	-	-	-	-	prediction	10	Play Time	0.1	-151715756...	User ID