

BIG DATA TOOLS AND TECHNIQUES

TASK ONE

ABSTRACT

The goal of this job is to get a thorough grasp of large data systems through the exploration of particular subjects that highlight the interaction between theory and real-world applications. It entails employing well-liked big data tools and methodologies in addition to having competence with common data analysis tasks including loading, cleaning, analyzing, and producing reports. It is essential to be proficient with Python, SQL, or Linux terminal commands in order to pass this test.

INTRODUCTION

In my capacity as an AI engineer and data scientist given the opportunity to investigate more about clinical trials, I've carefully examined the dataset that is given me. Several important topics are intended to be addressed by this investigation, which skillfully uses visuals to bolster findings.

First, I'll look at the total number of studies in the dataset, making sure to explicitly account for different research. The types of research that are here will next be discussed, with each type listed with its corresponding frequency in order of least to most frequent. I'll also include the top 5 conditions found in the dataset along with their corresponding frequencies.

I'll also look for sponsors, paying special attention to non-pharmaceutical businesses. I'll count the number of clinical studies that the top 10 sponsors outside the pharmaceutical industry have supported. Etc.

SETUP

Fire up the Databricks workspace

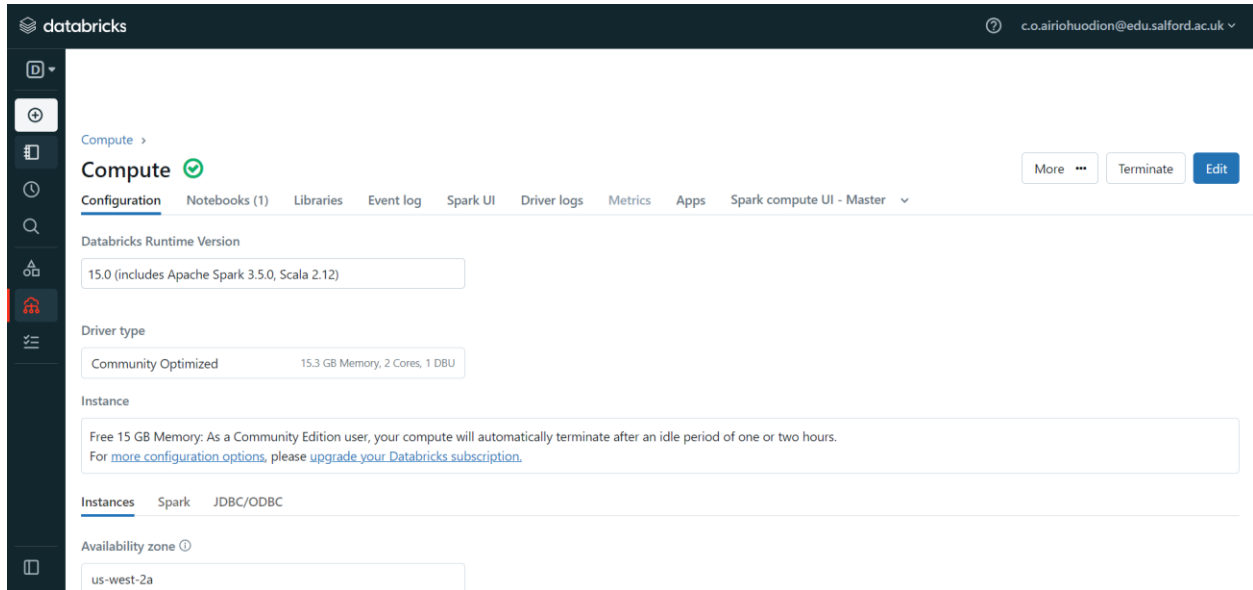


Figure 1.1.0 Setting up the platform required to kick start the task given

After logging into my Databricks Community Edition account, I created a cluster to start my analysis to have access to a machine to use then I click on create Compute and type a new name for the cluster in this case Clement_Airiohuodion_rdd for my RDD solution. I picked the most recent run time 15.0, after the green tick meaning successfully started up.

DATA CLEANING AND PREPARATION

(including descriptions, justifications and screenshots of all code).



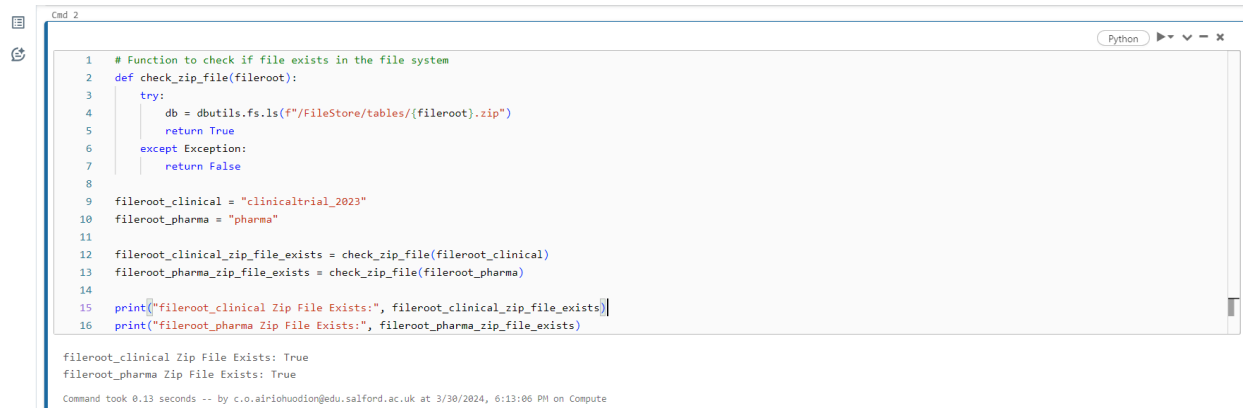
Figure 1.1.1 The screen short above shows uploaded clinical.csv of year 2020, 2021, 2023 files and a pharma file.

FILE UNZIPPING

I created a python notebook to have access to runnable cells under the active cluster. Given clinical_trial 2020, 2021,2023 and pharma file we have been instructed to answer some questions. Further steps were required to unzip the 4 files after they were uploaded to the FileStore/tables directory, as shown in the Figures below.



Figure 1.1.2 Using the `dbutils.fs.ls()` command, generally listing the files and folders in the “tables” directory inside the “FileStore” directory.



```
1 # Function to check if file exists in the file system
2 def check_zip_file(fileroot):
3     try:
4         db = dbutils.fs.ls(f"/FileStore/tables/{fileroot}.zip")
5         return True
6     except Exception:
7         return False
8
9 fileroot_clinical = "clinicaltrial_2023"
10 fileroot_pharma = "pharma"
11
12 fileroot_clinical_zip_file_exists = check_zip_file(fileroot_clinical)
13 fileroot_pharma_zip_file_exists = check_zip_file(fileroot_pharma)
14
15 print(fileroot_clinical Zip File Exists:", fileroot_clinical_zip_file_exists)
16 print(fileroot_pharma Zip File Exists:", fileroot_pharma_zip_file_exists)

fileroot_clinical Zip File Exists: True
fileroot_pharma Zip File Exists: True

Command took 0.13 seconds -- by c.o.airiohuodion@edu.salford.ac.uk at 3/30/2024, 6:13:06 PM on Compute
```

Figure 1.1.3 Creating and Using a function to check specific files (`clinicaltrial_2023` and `pharma` file) respectively.



```
1 def copy_into_localfile(fileroot):
2     try:
3         dbutils.fs.cp(f"/FileStore/tables/{fileroot}.zip", "file:/tmp/")
4         dbutils.fs.ls(f"/FileStore/tables/{fileroot}.zip")
5     except Exception:
6         pass
7
8 if fileroot_clinical_zip_file_exists:
9     copy_into_localfile(fileroot_clinical)
10 if fileroot_pharma_zip_file_exists:
11     copy_into_localfile(fileroot_pharma)

Command took 1.36 seconds -- by c.o.airiohuodion@edu.salford.ac.uk at 3/30/2024, 6:18:11 PM on Compute
```

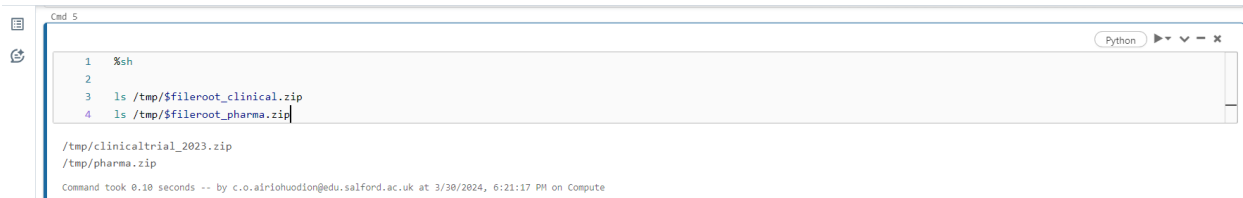
Figure 1.1.4 This defined a function called `copy_into_localfile()` in the supplied code, with `dbutils.fs.cp` it copies the two ZIP files from the **FileStore** directory to the **local file system** directory `/temp/`. It then displays the contents of the transferred ZIP file if the process is successful. Based on the existence of each file which was previously established, the function is called twice: once for the `fileroot_clinical` ZIP file and again for the `fileroot_pharm` ZIP file.



```
1 import os
2
3 os.environ['fileroot_clinical'] = fileroot_clinical
4 os.environ['fileroot_pharma'] = fileroot_pharma

Command took 0.10 seconds -- by c.o.airiohuodion@edu.salford.ac.uk at 3/30/2024, 6:19:55 PM on Compute
```

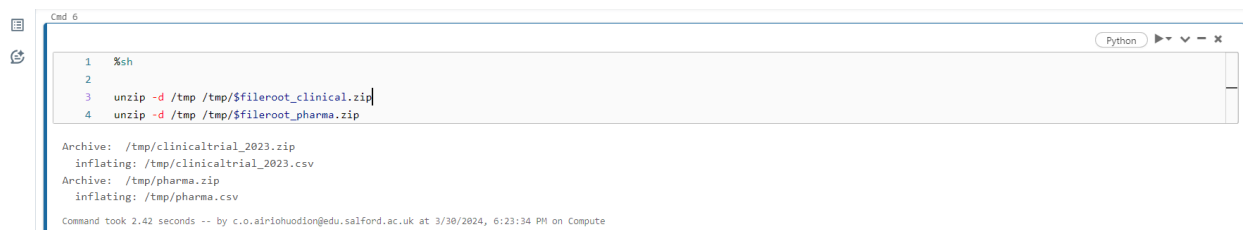
Figure 1.1.5 Knowing that the local filesystem or shell command line does not automatically know about the “**fileroot**” variable, we set the environment variables “**fileroot_clinical**” and “**fileroot_pharma**” to the variables. We set this value as an environment variable so that it may be accessed in shell commands.



```
Cmd 5
1 %sh
2
3 ls /tmp/$fileroot_clinical.zip
4 ls /tmp/$fileroot_pharma.zip

/tmp/clinicaltrial_2023.zip
/tmp/pharma.zip
Command took 0.10 seconds -- by c.o.airlohuodion@edu.salford.ac.uk at 3/30/2024, 6:21:17 PM on Compute
```

Figure 1.1.6 This “ls” shell command above checks the **/tmp/** to confirm that **clinical_trail.zip** and **pharma.zip** was moved successfully.

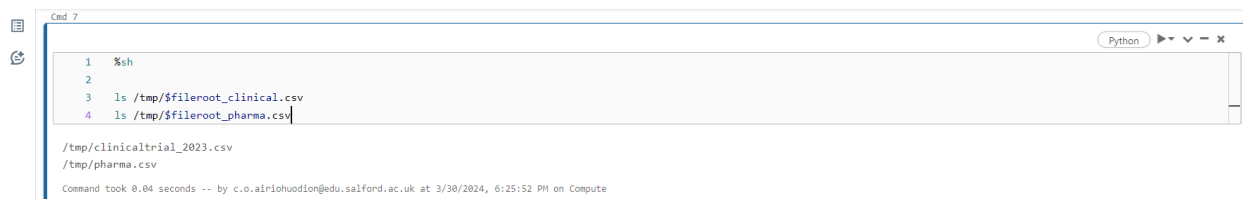


```
Cmd 6
1 %sh
2
3 unzip -d /tmp /tmp/$fileroot_clinical.zip
4 unzip -d /tmp /tmp/$fileroot_pharma.zip

Archive: /tmp/clinicaltrial_2023.zip
  inflating: /tmp/clinicaltrial_2023.csv
Archive: /tmp/pharma.zip
  inflating: /tmp/pharma.csv
Command took 2.42 seconds -- by c.o.airlohuodion@edu.salford.ac.uk at 3/30/2024, 6:23:34 PM on Compute
```

Figure 1.1.7 This code above Unzips the two .zip files

The Files designated by the variables \$fileroot_clinical and \$fileroot_pharma is unzipped into the /tmp/ directory, and the extracted files and their corresponding archives are then shown.



```
Cmd 7
1 %sh
2
3 ls /tmp/$fileroot_clinical.csv
4 ls /tmp/$fileroot_pharma.csv

/tmp/clinicaltrial_2023.csv
/tmp/pharma.csv
Command took 0.04 seconds -- by c.o.airlohuodion@edu.salford.ac.uk at 3/30/2024, 6:25:52 PM on Compute
```

Figure 1.1.8 This “ls” shell command above checks the **/tmp/** directory to confirm that **clinical_trail.csv** and **pharma.csv** now exist to ensure successful unzipping.

DATA CLEANING AND PREPARATION

RDD IMPLEMENTATION



```
Cmd 11
Python ▶ ▼ - x
1 #creating 1st RDD for fileroot parameter
2 def create_rdd(fileroot):
3     myrddc = sc.textFile("/FileStore/tables/" + fileroot + ".csv")
4     return myrddc
Command took 0.09 seconds -- by c.o.airiohuodion@edu.salford.ac.uk at 3/30/2024, 6:40:04 PM on Compute
```

Figure 1.2.2 At this point the CSV files in use in the DBFS is used to construct an RDD(Resilient distributed Dataset) using the “**create_rdd()**” function. It accepts the given parameter called **fileroot**, which indicates the target CSV files root name. The function uses **sc.textFile()** method with a filename that is obtained from the **fileroot** parameter to read the contents of the CSV file that is located in the file store table directory. The function then returns the final RDD. Using CSV files, this method makes it easier to create RDDs for further data processing tasks.



```
Cmd 12
Python ▶ ▼ - x
1 #split record in RDD based on corresponding delimiter character
2 delimiter_selector = {
3     "clinicaltrial_2023": "\t",
4     "clinicaltrial_2021": "|",
5     "clinicaltrial_2020": "|",
6     "pharma": ",",
7 }
8
9 # creating 2nd RDD
10 def clean_rdd(myrddc, fileroot):
11     Cleanup_myrddc = myrddc.map(lambda x: [i.replace(',', '').replace(' ', '') for i in x.split(delimiter_selector[fileroot])])
12     return Cleanup_myrddc
Command took 0.09 seconds -- by c.o.airiohuodion@edu.salford.ac.uk at 4/12/2024, 8:01:35 AM on sgst
```

Figure 1.2.3 The code defines a dictionary called **delimiter_selector**, linking file origins to appropriate delimiter characters. The function called **clean_rdd()** accept two parameters a file root (**fileroot**) and an RDD(**myrddc**) and converts the RDD to a file root, using the **map()** function to separate records based on the delimiter character, simplifying RDD cleaning.

```

1 # Creating 3rd RDDs
2 ClinicalT_RDD = create_rdd(filerooot_clinical)
3 Pharma_RDD = create_rdd(filerooot_pharma)
4
5 # Clean up RDDs
6 ClinicalT_RDD_clean = clean_rdd(ClinicalT_RDD, filerooot_clinical)
7 Pharma_RDD_clean = clean_rdd(Pharma_RDD, filerooot_pharma)
8
9 # Take first 5 elements from cleaned ClinicalT RDD
10 ClinicalT_RDD_clean.take(5)

```

```

(1) Spark Jobs
[{"id":
  'Study Title',
  'Acronym',
  'Status',
  'Conditions',
  'Interventions',
  'Sponsor',
  'Collaborators',
  'Enrollment',
  'Funder Type',
  'Type',
  'Study Design',
  'Start',
  'Completion'],
['NCT03630471',
  'Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India',
  'PRIDE',
  'COMPLETED',
  'Mental Health Issue (E.G. Depression Psychosis Personality Disorder Substance Abuse)',
  'BEHAVIORAL: PRIDE "Step 1" problem-solving intervention|BEHAVIORAL: Enhanced usual care',
  'Sangath',

```

Figure 1.2.4 The create_rdd() function creates two RDDs, ClinicalT_RDD and Pharma_RDD, representing raw data from pharma files and clinical trials. The clean_rdd() method cleans the raw RDDs, dividing records within each RDD. The take(5) action extracts the first items from the cleaned ClinicalT_RDD, ensuring efficient data management

DATAFRAME IMPLEMENTATION

In this section I have used Spark DataFrames to process the clinical trial and pharma files. Various cleaning process was carried out using unique codes, for example the completion column in the clinical trial 2023 had a double quote ‘ ‘ ’ and lots of commas that needed special code to clean it off. After the cleaning process has been thoroughly done, questions are given are then answered.

```

1 filerooot_clinical = "clinicaltrial_2023"
2 filerooot_pharma = "pharma"

```

Figure 1.2.5 My file roots are file_pharma and filerooot_clinical. The names of the CSV files containing pharmaceutical and clinical trial 2020, 2021 and 2023 data respectively, are held in these variables

```

1 #Creating DataFrame
2 from pyspark.sql.types import * #StructType, StructField
3 from pyspark.sql.functions import *
4
5 delimiter_selector = {
6     "clinicaltrial_2023": "\t",
7     "clinicaltrial_2021": "|",
8     "clinicaltrial_2020": "|",
9     "pharma": ","
10 }
11
12 def create_clinical_dataframe(fileroor):
13     if fileroor == "clinicaltrial_2023":
14         rdd = sc.textFile(f"/FileStore/tables/{fileroor}.csv").map(lambda row: row.split(delimiter_selector[fileroor]))
15         head = rdd.first()
16         rdd = rdd.map(lambda row: row + [" " for i in range(len(head) - len(row))] if len(row) < len(head) else row )
17         df = rdd.toDF()
18         first = df.first()
19         for col in range(0, len(list(first))):
20             df = df.withColumnRenamed(f"_{col + 1}", list(first)[col].replace(",","").replace("'", ""))
21         df = df.withColumn('index', monotonically_increasing_id())
22         df = df.withColumn('completion', regexp_replace("completion", ",", "")).withColumn('Completion', regexp_replace("Completion", "'", ""))
23         return df.filter(~df.index.isin([0])).drop('index')
24     else:
25         return spark.read.csv(f"/FileStore/tables/{fileroor}.csv", sep=delimiter_selector[fileroor], header=True)
26
27 ClinicalT_dataframe = create_clinical_dataframe(fileroor_clinical)
28 pharma_dataframe = create_clinical_dataframe(fileroor_pharma)
29 ClinicalT_dataframe.show(20)
30

```

Figure 1.2 From the top of this code lines I imported Modules, the first contains data types to define schema while the second contains functions for DataFrame operations, then I introduce a `delimiter_selector` dictionary and map each file root name to its delimiter. Next a `fileroor` is required as input for this function, which produces a DataFrame that correspond to the file root.

Going from the “**if**” statement this code reads the `Clinical_trial 2023` file, and divides each row by its **delimiter** in the process generates an RDD, then transforms it into a DataFrame. Rows with fewer columns than the column header are padded with empty strings in this scenario. It gets rid of special characters and changes the names of the columns after which it takes off the DataFrame header row and adds an index column. It then takes the place of double quotes and commas in the completion column, finally the cleaned DataFrame is returned.

From the “**else**” statement the other `clinical_trial CSV` files 2020 and 2021 are read using `spark read.csv` function to read them directly into a DataFrame. To construct DataFrames for the `clinicaltrial` and `parma` data, these lines call the **Create_clinical_dataframe** method and then 20 rows of `clinicaltrial DataFrame` are shown.

▶ (5) Spark Jobs

▶ ClinicalT_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 fields]

Id	Study Title	Acronym	Status	Conditions	Interventions	Sponsor	Collaborators	Enrollment	Funder Type	T	
Study Design	Start	Completion									
"NCT03630471"	Effectiveness of ...	PRIDE	COMPLETED	Mental Health Iss...	BEHAVIORAL: PRIDE...	Sangath	Harvard Medical S...	250.0	OTHER	INTERVENTIO	
NAL	Allocation: RANDO...	2018-08-20	2019-02-28								
"NCT05992571"	Oral Ketone Monoe...										
NAL	Allocation: RANDO...	2023-10-25	2024-08								
"NCT00237471"	Impact of Tight G...										
NAL	Allocation: RANDO...	2005-10	2006-05								
"NCT03820271"	New Prognostic Pr...	SUPERMELD	RECRUITING	Decompensated Cir...	OTHER: SuperMELD	Assistance Publi...		500.0	OTHER	INTERVENTIO	
NAL	Allocation: NA	In...	2020-10-01	2023-10-01							
"NCT06229171"	InTake Care: Deve...	InTakeCare	NOT_YET_RECRUITING	Hypertension	Trea...	OTHER: adherence ...	Istituto Auxologi...	Istituti Clinici ...	206.0	OTHER	INTERVENTIO
NAL	Allocation: RANDO...	2024-10-01	2026-04-01								
"NCT02945371"	Tailored Inhibito...	REV	COMPLETED	Smoking	Alcohol D...	BEHAVIORAL: Perso...	University of Oregon		103.0	OTHER	INTERVENTIO
NAL	Allocation: RANDO...	2014-09	2016-05								
"NCT01855171"	Neuromodulation o...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125371"	Computerized Brie...										
NAL	Allocation: RANDO...	2010-01	2012-08								
"NCT01125											

Figure 2.1 From the top of this code lines I imported Modules, the first contains data types to define schema while the second contains functions for DataFrame operations, then I introduce a delimiter_selector dictionary and map each file root name to its delimiter. Next a fileroot is required as input for this function, which produces a DataFrame that correspond to the file root.

Going from the “if” statement this code reads the Clinical_trial 2023 file, and divides each row by its **delimiter** in the process generates an RDD, then transforms it into a DataFrame. Rows with fewer columns than the column header are padded with empty strings in this scenario. It gets rid of special characters and changes the names of the columns after which it takes off the DataFrame header row and adds an index column. It then takes the place of double quotes and commas in the completion column, finally the cleaned DataFrame is returned.

From the “else” statement the other clinical_trial CSV files 2020 and 2021 are read using spark read.csv function to read them directly into a DataFrame. To construct DataFrames for the clinicaltrial and parma data, these lines call the **Create_clinical_dataframe** method and then 20 rows of clinicaltrial DataFrame are shown.

```

1 %python
2
3 pharma_dataframe.createOrReplaceTempView(fileroot_pharma)
4 clinicalT_dataframe.createOrReplaceTempView(fileroot_clinical)

```

Command took 0.22 seconds -- by c.o.alrishuon@edu.salford.ac.uk at 4/12/2024, 8:12:01 AM on sgst

Figure 2.2 This code creates two temporary table (view) Pharma_dataframe and ClinicalT_dataframe. We use createOrReplaceTempView method

Cmd 4

```

1 SELECT * FROM clinicaltrial_2023

```

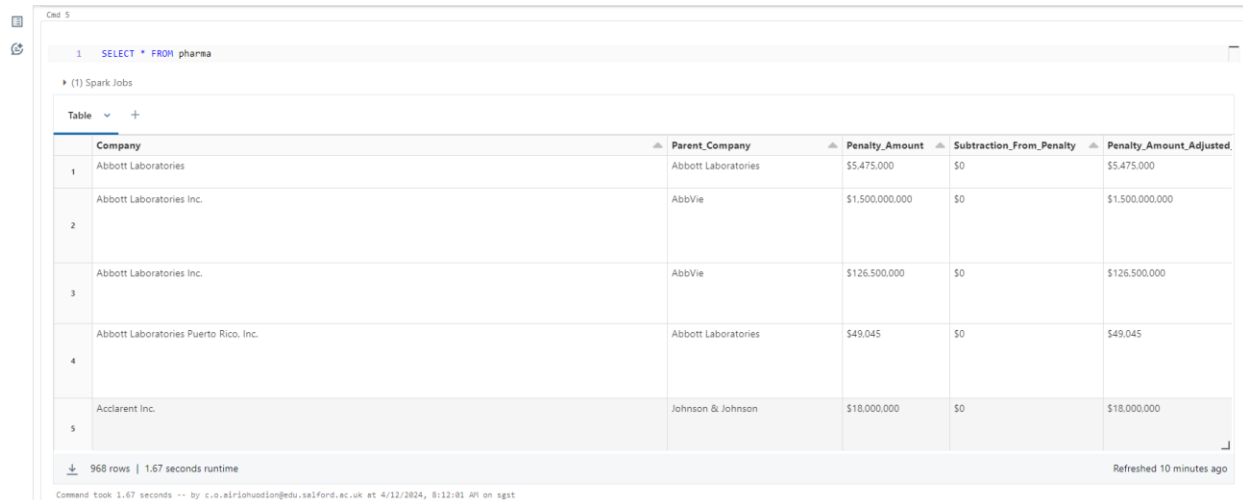
(1) Spark Jobs

Id	Study Title	Acronym	Status	Conditions
1	*NCT03630471 Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India	PRIDE	COMPLETED	Mental Health Issue (E.G.: Depression, Psycho
2	*NCT05992571 Oral Ketone Monoester Supplementation and Resting-state Brain Connectivity		RECRUITING	Cerebrovascular Function Cognition
3	*NCT00237471 Impact of Tight Glycaemic Control in Acute Myocardial Infarction		TERMINATED	Myocardial Infarct Hyperglycemia
4	*NCT03820271 New Prognostic Predictive Models of Mortality of Decompensated Cirrhotic Patients Waiting for Liver Transplantation	SUPERMELD	RECRUITING	Decompensated Cirrhosis Liver Transplantation
5	*NCT06229171 InTake Care: Development and Validation of an Innovative," Personalized Digital Health Solution for Medication Adherence Support in Cardiovascular Prevention	InTakeCare	NOT_YET_RECRUITING	Hypertension Treatment Adherence and Com
6	*NCT02945371 Tailored Inhibitory Control Training to Reverse EA-linked Deficits in Mid-life	REV	COMPLETED	Smoking Alcohol Drinking Prescription Drug A
7	*NCT01055171 Neuromodulation of Trauma Memories in PTSD & Alcohol Dependence		COMPLETED	Alcohol Dependence PTSD

4,839 rows | Truncated data | 3.67 seconds runtime Refreshed 9 minutes ago

Command took 3.67 seconds -- by c.o.alrishuon@edu.salford.ac.uk at 4/12/2024, 8:12:01 AM on sgst

Figure 2.3 The complete dataset is returned by this query, giving users a thorough overview of the clinical trial data for 2023. A single clinical trial is represented by each row in the result, with columns denoting different characteristics such as trial ID, conditions, sponsors.



1 SELECT * FROM pharma

(1) Spark Jobs

	Company	Parent_Company	Penalty_Amount	Subtraction_From_Penalty	Penalty_Amount_Adjusted
1	Abbott Laboratories	Abbott Laboratories	\$5,475,000	\$0	\$5,475,000
2	Abbott Laboratories Inc.	AbbVie	\$1,500,000,000	\$0	\$1,500,000,000
3	Abbott Laboratories Inc.	AbbVie	\$126,500,000	\$0	\$126,500,000
4	Abbott Laboratories Puerto Rico, Inc.	Abbott Laboratories	\$49,045	\$0	\$49,045
5	Acclarent Inc.	Johnson & Johnson	\$18,000,000	\$0	\$18,000,000

968 rows | 1.67 seconds runtime

Command took 1.67 seconds -- by c.o.a.l.r.l.h.u.d.i.o.n@e.d.u.s.a.l.f.o.r.d.a.c.u.k at 4/12/2024, 8:12:01 AM on sgst

Refreshed 10 minutes ago

Figure 2.4 Every field and record in the pharma table will be retrieved by this query

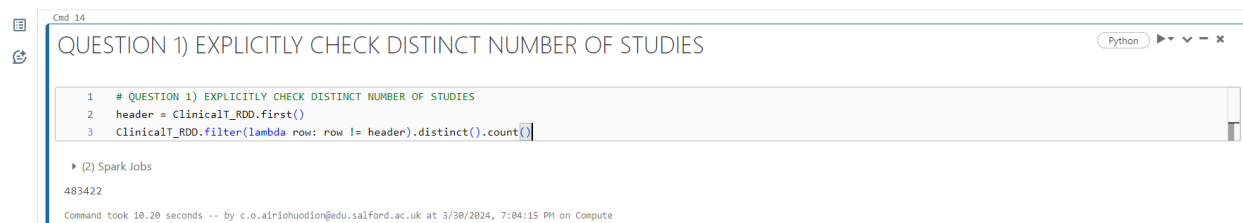
PROBLEM ANSWERS

SOLUTION 1

ASSUMPTIONS:

The aim is to explicitly validate the unique number of studies in the dataset, assuming the csv file is cleaned, consistent and free of error.

RDD Implementation



QUESTION 1) EXPLICITLY CHECK DISTINCT NUMBER OF STUDIES

```

1 # QUESTION 1) EXPLICITLY CHECK DISTINCT NUMBER OF STUDIES
2 header = ClinicalT_RDD.first()
3 ClinicalT_RDD.filter(lambda row: row != header).distinct().count()

```

(2) Spark Jobs

483422

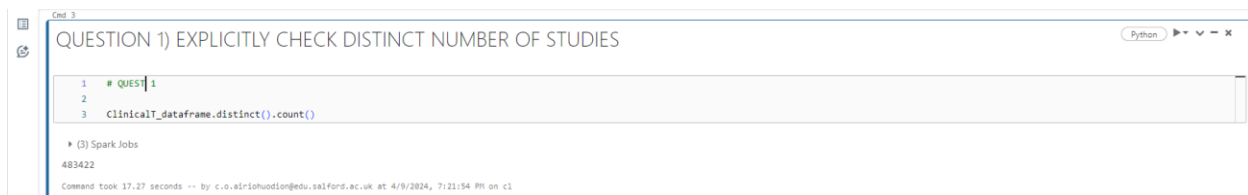
Command took 10.20 seconds -- by c.o.a.l.r.l.h.u.d.i.o.n@e.d.u.s.a.l.f.o.r.d.a.c.u.k at 3/30/2024, 7:04:15 PM on Compute

Figure 2.5 `heading = ClinicalT_RDD.first()`: From **Clinical_RDD**, the first row of data is retrieved and assigned to the variable `header` in this line. The column names or header are presumed to be in the first row.

ClinicalT_RDD.filter(lambda row: row != header): Excluding the header row, this line filters the RDD **ClinicalT_RDD**. Each row is compared to the header row using a lambda function to see if it differs. The RDD's header is basically being removed.

The **.distinct()** function in order to retain just unique rows and eliminate duplicates, this function is applied to the filtered RDD. This code aims to count distinct rows in **ClinicalT_RDD** by removing the header row and deleting duplicates, then counting the remaining rows.

DATAFRAME Implementation



The screenshot shows a Jupyter Notebook interface with a code cell titled "QUESTION 1) EXPLICITLY CHECK DISTINCT NUMBER OF STUDIES". The code cell contains the following Python code:

```
1 # QUEST 1
2
3 ClinicalT_dataframe.distinct().count()
```

Below the code cell, the output shows "(3) Spark Jobs" and the number "483422". At the bottom, a status bar indicates "Command took 37.27 seconds -- by c.o.eiriohuodlon@edu.salford.ac.uk at 4/9/2024, 7:23:54 PM on cl".

Figure 1.4 From the question given I counted the unique entries in the **ClinicalT_dataframe** using pyspark. By taking into account all columns, this count indicates the total number of unique rows in the DataFrame. Using '**Distinct**' to get unique entries and then '**count**' them.

SQL Implementation



The screenshot shows a Jupyter Notebook interface with a code cell titled "1. The nos. of Distinct studies". The code cell contains the following SQL query:

```
1 SELECT DISTINCT count(*) FROM clinicaltrial_2023
```

Below the code cell, the output shows "(2) Spark Jobs" and a table with the following data:

count(1)
483422

At the bottom, a status bar indicates "1 row | 12.38 seconds runtime" and "Refreshed 10 minutes ago".

Figure 2.5 The number of unique records in the **clinicaltrial_2023** table will be counted by this query, which will return the result in a **distinct_count** column.

RESULT DISCUSSION:


The collection contains 483,422 different studies, according to the analysis, each study is counted once.

SOLUTION 2

ASSUMPTIONS:

The “Type” column in the dataset provide trustworthy labels for each study type, and the analysis is predicated on the dataset having accurate and consistent information on clinical trials.

RDD Implementation



The screenshot shows a Jupyter Notebook interface with a code cell titled "2. QUESTION 2) filter, map, reduce, and sort a cleaned RDD then collect". The code cell contains the following PySpark code:

```
1 # QUESTION 2)
2 typ_idx = ClinicalT_RDD_clean.first().index('Type')
3 ClinicalT_RDD_clean.filter(lambda x: len(x) > typ_idx + 1) \
4     .map(lambda x: (x[typ_idx], 1)) \
5     .filter(lambda row: row[0] != 'Type') \
6     .reduceByKey(lambda a, b: a + b) \
7     .filter(lambda x: x[0] != '') \
8     .sortBy(lambda x: x[1], ascending=False) \
9     .collect()
```

Below the code cell, the "Spark Jobs" section shows the output of the collect action:

```
[('INTERVENTIONAL', 371382),
 ('OBSERVATIONAL', 110221),
 ('EXPANDED_ACCESS', 928)]
```

At the bottom, a status bar indicates: "Command took 10.37 seconds -- by c.o.eiriohuodion@edu.salford.ac.uk at 4/1/2024, 12:54:35 AM on clust"

Figure 2.6 Using a lambda function to check for inequality with the header row, it filters out the header row from the RDD ClinicalT_RDD, which is represented by the variable header. After deleting duplicates, it uses the distinct function to procedure unique rows. Lastly, it counts the number of distinct rows in the dataset, which is a direct correlation to the number of distinct studies.

DATAFRAME Implementation


```

2. list all the types, freq., desc

1 #QUESTION 2
2
3 ClinicalT_dataframe.groupBy('Type').count().orderBy('count', ascending=False).show(3)

```

(2) Spark Jobs
 -----+-----
 | Type | count |
 -----+-----
INTERVENTIONAL	371382
OBSERVATIONAL	110221
EXPANDED_ACCESS	928
 -----+-----
 only showing top 3 rows

Command took 11.36 seconds -- by c.o.siriohuodion@edu.salford.ac.uk at 4/9/2024, 7:21:54 PM on cl

Figure 1.5 The pyspark above groups the entire row in the DataFrame ClinicalT_dataframe by the “Type” column using **.groupBy** then after it uses **“count”** function to count the number of rows in each group, to further achieve the result I used the **.orderBy** to place the data in descending order by count, then finally show three (3), results.

SQL Implementation

```

2. list all the types, freq., desc

1 SELECT clinicaltrial_2023.Type, count(*) as count FROM clinicaltrial_2023
2 GROUP BY clinicaltrial_2023.Type
3 ORDER BY count DESC
4 LIMIT 3

```

(2) Spark Jobs
 Table +

	Type	count
1	INTERVENTIONAL	371382
2	OBSERVATIONAL	110221
3	EXPANDED_ACCESS	928

3 rows | 16.10 seconds runtime
 Refreshed 11 minutes ago
 Command took 16.10 seconds -- by c.o.siriohuodion@edu.salford.ac.uk at 4/12/2024, 8:12:01 AM on sgst

Figure 2.6 The clinicaltrial 2023 tables Type column is selected by the SQL query, which then groups the records according to the Type columns values. Using the COUNT(*) function, it counts the number of records within each group and aliases the result as count. Following that, the findings are sorted by count in descending order, and only the top three categories of clinical trials with the highest counts are included.

Discussion of Result:

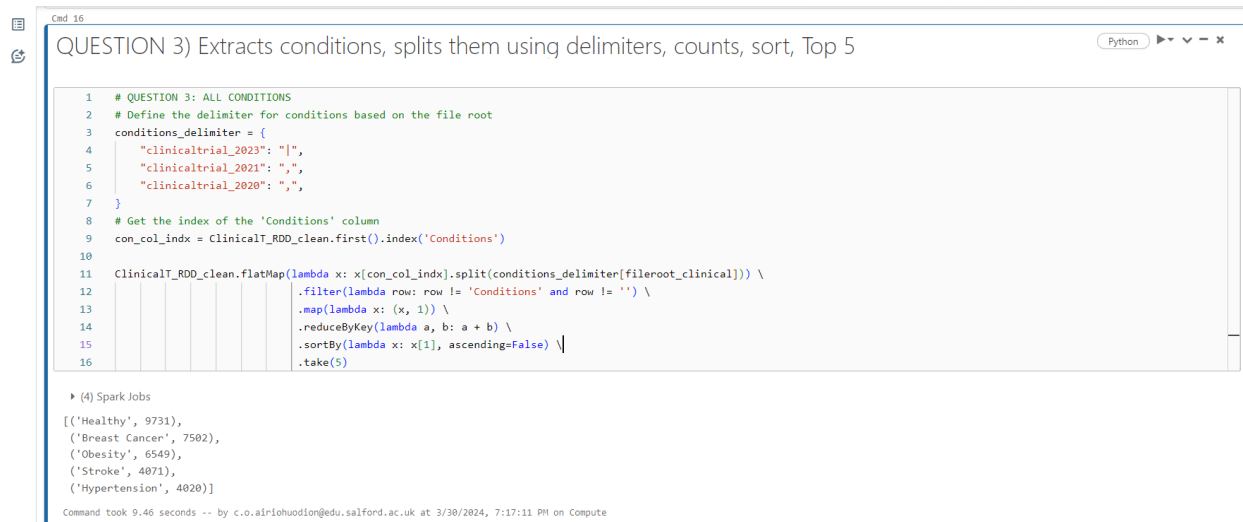
The dataset consists of observational, expanded access, and interventional studies, providing crucial information on the composition and areas of interest in clinical research

SOLUTION 3

ASSUMPTION:

The “Conditions” column contains a list of case-sensitive conditions, divided by a delimiter depending on the particular clinical csv dataset, and is assumed to be accurately reported without any formatting issues.

RDD Implementation



```
1 # QUESTION 3: ALL CONDITIONS
2 # Define the delimiter for conditions based on the file root
3 conditions_delimiter = {
4     "clinicaltrial_2023": "|",
5     "clinicaltrial_2021": ",",
6     "clinicaltrial_2020": ";",
7 }
8 # Get the index of the 'Conditions' column
9 con_col_indx = ClinicalT_RDD_clean.first().index('Conditions')
10
11 ClinicalT_RDD_clean.flatMap(lambda x: x[con_col_indx].split(conditions_delimiter[fileroot_clinical])) \
12     .filter(lambda row: row != 'Conditions' and row != '') \
13     .map(lambda x: (x, 1)) \
14     .reduceByKey(lambda a, b: a + b) \
15     .sortBy(lambda x: x[1], ascending=False) \
16     .take(5)
```

▶ (4) Spark Jobs

```
[('Healthy', 9731),
 ('Breast Cancer', 7502),
 ('Obesity', 6549),
 ('Stroke', 4071),
 ('Hypertension', 4020)]
```

Command took 9.46 seconds -- by c.o.ainohuodion@edu.salford.ac.uk at 3/30/2024, 7:17:11 PM on Compute

Figure 2.7 Define the conditions Delimiter: To assign distinct delimiters to every file root, a dictionary called `conditions_delimiter` is made. This indicates the file root-based division that should be made for the “Conditions” column.

To obtain the ‘conditions’ Column index: the cleaned RDD `ClinicalT_RDD_clean` is where the index for the ‘Conditions’ column is found. We’ll use this index to go to the ‘Conditions’ column in every RDD row.

RDD Processing for Counting Conditions;

- **flatMap():** The delimiter assigned to the file root is used to split the ‘Conditions’ column for each row. A new RDD is produced as a consequence, with each element representing a condition from a single trial.
- **Filter():** It removed rows that contain empty strings or the value ‘Conditions’.
- **Map():** It mapped a tuple (condition, 1) to each condition.

- **reduceByKey:** The total number of occurrences is used to aggregate the counts for each condition.
- **sortBy():** According to their frequency, the resulting counts are arranged in descending order.
- **Take(5):** The five most prevalent conditions are found.

DATAFRAME Implementation



```

3. Top 5 cond. with their freq.

1 # QUESTION 3
2
3 conditions_delimiter = [
4     "clinicaltrial_2023": "\|",
5     "clinicaltrial_2021": ",",
6     "clinicaltrial_2020": ",",
7 ]
8
9 ClinicalT_dataframe.withColumn('Conditions', explode(split(col('Conditions'), conditions_delimiter[filterroot_clinical])))
    .groupBy('Conditions').count().orderBy('count', ascending=False).filter("Conditions != ''").show(5, truncate=False)

```

▶ (2) Spark Jobs

Conditions	count
Healthy	9731
Breast Cancer	7502
Obesity	6549
Stroke	4871
Hypertension	4020

only showing top 5 rows

Command took 12.84 seconds -- by c.o.aiohudson@edu.salford.ac.uk at 4/9/2024, 7:21:54 PM on cl

Figure 1.6 The ‘**Conditions**’ column is divided according to the delimiter that is defined for the specified file root (which is kept in **conditions_delimiter**). It then splits the resultant array into several rows, each of which has a single condition. This guarantees that any condition in a row that includes multiple conditions separated by the designed delimiter will have its own row.

Then I counted the instances of each condition and organized the DataFrame based on the ‘conditions’ column, after which I ordered the result in descending using ‘**.orderBy**’ in other to achieve desired result I **filtered** out unfilled conditions, and showed the **top 5** rows without truncating the data.

SQL Implementation

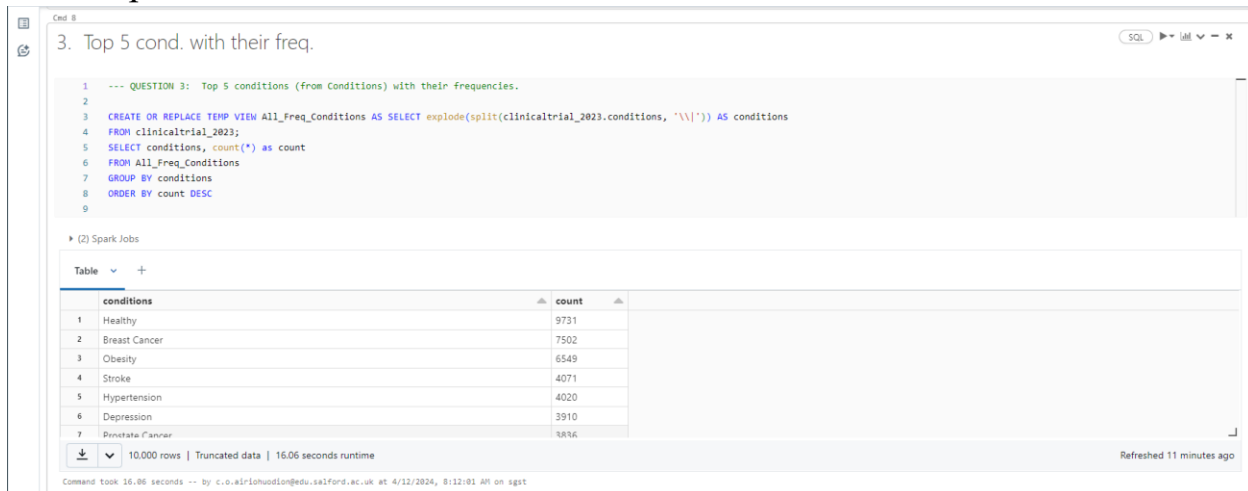


Figure 2.7 By exploding the conditions column, this SQL query produces a temporary view called `all_conditions`. It then counts the frequency of each conditions and arranges the result in decreasing order of count. Lastly, it restricts the outcome to the top five circumstances.

Discussion of Result:

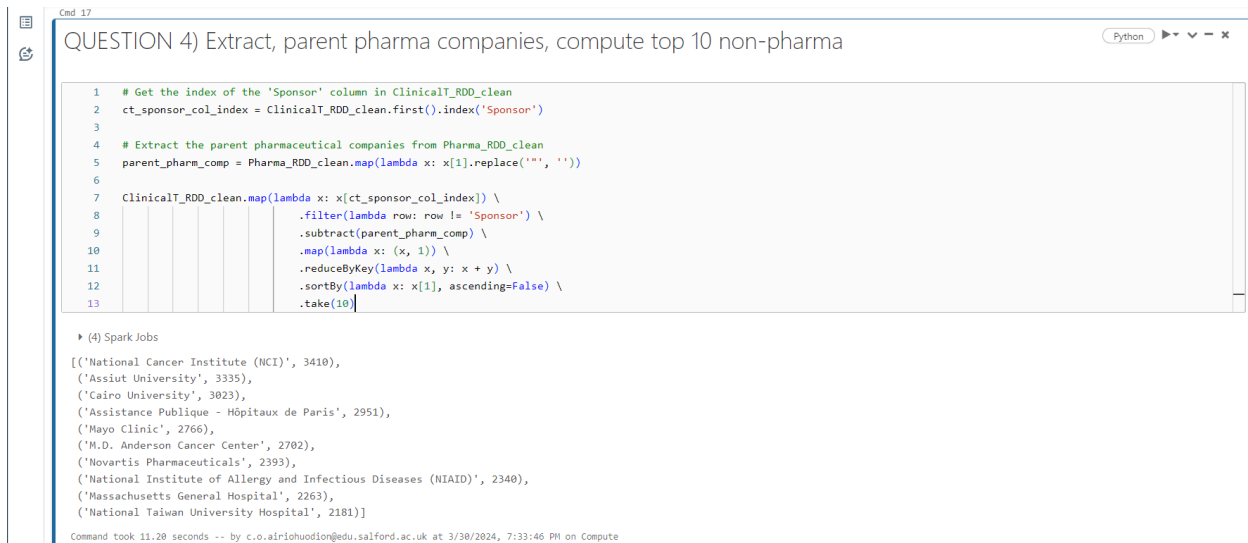
The clinical trial dataset reveals the frequency of common conditions like obesity, stroke, breast cancer, and hypertension, providing valuable insights for further analysis and interpretation.

SOLUTION 4

ASSUMPTION:

The 'Parent Company' column in the `Pharma_RDD_clean` dataset includes all potential pharmaceutical businesses, while the "Sponsor" column in the `ClinicalT_RDD_clean` dataset represents clinical trial sponsors.

RDD Implementation



```
1 # Get the index of the 'Sponsor' column in ClinicalT_RDD_clean
2 ct_sponsor_col_index = ClinicalT_RDD_clean.first().index('Sponsor')
3
4 # Extract the parent pharmaceutical companies from Pharma_RDD_clean
5 parent_pharm_comp = Pharma_RDD_clean.map(lambda x: x[1].replace(' ', ''))
6
7 ClinicalT_RDD_clean.map(lambda x: x[ct_sponsor_col_index]) \
8     .filter(lambda row: row != 'Sponsor') \
9     .subtract(parent_pharm_comp) \
10    .map(lambda x: (x, 1)) \
11    .reduceByKey(lambda x, y: x + y) \
12    .sortBy(lambda x: x[1], ascending=False) \
13    .take(10)
```

► (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 11.20 seconds -- by c.o.airlohuodion@edu.salford.ac.uk at 3/30/2024, 7:33:46 PM on Compute

Figure 2.8 The RDD’s ‘sponsor’ column index is obtained using the **index()** function. The algorithm extracts parent pharmaceutical companies from the **Pharma_RDD_clean** RDD, filters out rows with equal values, subtracts them, and maps each surviving “Sponsor” to a 1-valued tuple, then it counts occurrences by key, and finally sorts results in **descending** order by count. The ten (10) most prevalent conditions are found by **.take(10)**.

DATAFRAME Implementation



```
1 # QUESTION 4
2 pharma_list = create_clinical_dataframe(fileroot_pharma).select("Parent_Company").rdd.flatMap(lambda x: x).collect()
3 ct_sponsor_dataframe = ClinicalT_dataframe.select("Sponsor")
4
5 non_pharma_sponsors = ct_sponsor_dataframe.groupBy("Sponsor").count().orderBy("count", ascending=False).filter(~ct_sponsor_dataframe.Sponsor.isin(pharma_list)).show(10, truncate=False)
```

► (4) Spark Jobs

ct_sponsor_dataframe: pyspark.sql.dataframe.DataFrame = [Sponsor:string]

Sponsor	count
National Cancer Institute (NCI)	3410
Assiut University	3335
Cairo University	3023
Assistance Publique - Hôpitaux de Paris	2951
Mayo Clinic	2766
M.D. Anderson Cancer Center	2702
Novartis Pharmaceuticals	2393
National Institute of Allergy and Infectious Diseases (NIAID)	2340
Massachusetts General Hospital	2263
National Taiwan University Hospital	2181

only showing top 10 rows

Command took 25.45 seconds -- by c.o.airlohuodion@edu.salford.ac.uk at 4/9/2024, 7:21:54 PM on c1

Figure 1.7 The first line of the pyspark code in this solution takes the ‘Parent_Company’ column of the pharmaceutical DataFrame and generates a list of

distinct parent firms. The second line of the code picks the ‘Sponsor’ column from the clinical trial DataFrame (ClinicalT_dataframe).

The last line classifies the clinical trial DataFrame according to ‘Sponsor’ using ‘.groupby’, counts the instances of each sponsor using .count(), sorts the findings in descending order based on count, removes sponsors that are already on the pharma list, and shows the top 10 non-pharma sponsors without cropping the output.

SQL Implementation



Figure 2.8 In order to filter sponsors from **clinicaltrial_2023** based in their absence in the pharma database, it constructs a temporary view called **Sponsor_Notpharma_Comp**. The output is then **limited** to the top 10 sponsors after **counting** the number of times each sponsor appears in the temporary display and sorting the results by count in descending order.

Discussion of Result:

The outcome reveals the top ten non-pharmaceutical sponsors and their financing of clinical trials, showcasing a diverse range of institutions and groups supporting clinical studies. The National Cancer institute is the most frequent sponsor, alongside academic institutions and medical centers.

SOLUTION 5

ASSUMPTION:

The dataset includes “Status” columns for current study status and 2“Completion” columns for completion date, with completion date in ‘YYYY-M-DD’ format

RDD Implementation



```
1 #number of completed studies for each month in 2023
2 year = {
3     "clinicaltrial_2020": "2020",
4     "clinicaltrial_2021": "2021",
5     "clinicaltrial_2023": "2023",
6 }
7 ClinicalT_RDD_clean = ClinicalT_RDD_clean.map(lambda x: [i.replace(',', '').replace(' ', '') for i in x])
8
9 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"}
10
11 ct_completion_col_index = ClinicalT_RDD_clean.first().index('Completion')
12 ct_status_col_index = ClinicalT_RDD_clean.first().index('Status')
13
14 ClinicalT_RDD_clean.filter(lambda x: len(x) > typ_inx + 1).map(lambda x: (x[ct_completion_col_index], x[ct_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

```
[('Jun', 1619),
('Jan', 1494),
('Aug', 1230),
('Dec', 1082),
('Oct', 1058),
('Mar', 1552),
('Feb', 1272),
('May', 1415),
('Apr', 1324),
('Jul', 1368),
('Nov', 909),
('Sept', 1152)]
```

Command took 10.91 seconds -- by c.o.airohudson@edu.salford.ac.uk at 3/30/2024, 7:58:18 PM on Compute

Figure 2.9 The following procedures are carried out by this code to ascertain the total number of studies that are finished each month in 2023:

The file names are mapped to the corresponding years using a dictionary called year. Commas and double quotes are eliminated from each element of the ClinicalT_RDD_clean RDD. The month_selector dictionary is defined in order to translate the names of the months into their numerical representation.

The ‘Completion’ and ‘Status’ column indexes in the RDD are found. Rows with missing data or those without the status “COMPLETED” OR “Completed” are filtered out. From the completion date, the month and year are extracted.

It applies a 2023 filter. Every month is mapped to a tuple with a value of 1. To count the occurrences of each month, it decreases by key. The month_selector is used to map each month to its associated name. Ultimately, it gathers the outcomes.

DATAFRAME Implementation

```

1 # QUESTION 5
2 from pyspark.sql.functions import split, regexp_replace
3
4 for col in ClinicalT_dataframe.columns:
5     ClinicalT_dataframe = ClinicalT_dataframe.withColumnRenamed(col, col.strip(",").strip(""))
6
7 completed_cd = ClinicalT_dataframe.withColumn('Year', split('Completion', "-")[0]) \
8     .withColumn('Month', split('Completion', "-")[1]) \
9     .withColumn('Month', regexp_replace("Month", ",", "")) \
10    .withColumn('Month', regexp_replace("Month", "'", "")) \
11    .filter(ClinicalT_dataframe.Status.isin(["COMPLETED"])) \
12    .select("Month", "Year", "Status")
13
14 month_counts = completed_cd.filter(completed_cd.Year.isin(["2023"])) \
15    .groupBy("Month").count().orderBy("Month", ascending=True)
16
17 month_counts.show()
18

```

Figure 1.8 After importing `split` and `regexp_replace` the first line of code the `Clinical_dataframe` column names are renamed after all leading and following commas and double quotes are removed. Then Completion column is split into the Year and Month columns, any commas and double quotes are again removed from the Month column, then the DataFrame is filtered to include only records with Status equal to COMPLETED and the **Month**, **Year**, and **Status** columns are selected. This creates a new DataFrame called `completed_cd`. The `completed_cd` DataFrame, filtered to include 2023 entries, it is then grouped by Month and aggregated for each month of 2023, sorting results in ascending order

```

(2) Spark Jobs
ClinicalT_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_counts: pyspark.sql.dataframe.DataFrame = [Month: string, count: long]

+-----+
|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 11.75 seconds -- by c.o.xiriohuo@edu.salford.ac.uk at 4/9/2024, 7:21:54 PM on c1

```

Figure 1.9 The result above is the outcome of the DataFrame code using `show()` method to display the number of clinical trials completed each month in 2023 .

SQL Implementation


```
Cmd 10
QUESTION 5) Extract completion dates and statuses, filter comp stud for yr 2023, counts, show

1 --QUESTION 5: Number of completed studies for each month in 2023
2
3 CREATE OR REPLACE TEMP VIEW Months_Table as SELECT split(clinicaltrial_2023.Completion, '-')[1] as month, count(*) as count
4 FROM clinicaltrial_2023
5 WHERE clinicaltrial_2023.Status = 'COMPLETED'
6 AND split(clinicaltrial_2023.Completion, '-')[0] = '2023'
7 GROUP BY month
8 ORDER BY month

OK
Command took 0.81 seconds -- by c.o.ai@ionuodion@edu.saiford.ac.uk at 4/12/2024, 8:12:01 AM on sgst
```

Figure 2.9.1 I created a temporary view named **Months_Table** that shows the number of studies finished in each month of 2023. This is accomplished by **filtering** for clinical trials that were completed in 2023 and **dividing** the completion date of trials into **month-by-year** components. The outcome is arranged and **classified by month**.

```
Cmd 11
1 select * from months_table
2
```

(2) Spark Jobs

	month	count
1	01	1494
2	02	1272
3	03	1552
4	04	1324
5	05	1415
6	06	1619
7	07	1560

12 rows | 14.99 seconds runtime
Refreshed 14 minutes ago
Command took 14.99 seconds -- by c.o.ai@ionuodion@edu.saiford.ac.uk at 4/12/2024, 8:12:01 AM on sgst

Figure 2.9.2

The data is then retrieved from the Months_Table view using the **SELECT** query to access the total counts of studies for each month in 2023.

Discussion of Result:

In 2023, research completion rates vary by month, with June having the highest number (1619) and January and August following closely behind (1230), indicating potential trends affecting year-round completion rates.

