BIG DATA TOOLS AND TECHNIQUES

STUDENT ID: @00704794

MSC DATA SCIENCE
School of Computing, Science & Engineering
University of Salford, Manchester.

KOSEMANI ABIOLA M.
APRIL 11, 2023

TASK ONE

ABSTRACT.

This task is aimed at providing a broad overview of the general area of big data systems through developing specific knowledge in areas demonstrating an interaction and synergy between ongoing research and practical use.

The use of common Big Data tools and techniques, the ability to perform a standard data analysis process which includes loading information, cleansing it, analysing it, and reporting with Python, SQL or Linux terminal commands are essential skills that will be evaluated in this evaluation.

INTRODUCTION

I'm going to use clinical trial data in this work and combine it with the list of pharmaceutical companies. To verify the core solution to these problems, I was provided with answers on two main implementations from historic data sets.

The dataset used for this assessment was downloaded as .csv files, The .csv files have a header describing the file's contents. These csv files include. Clinicaltrial_2021.csv, Clinicaltrial_2019.csv, Clinicaltrial2020.csv. Each row in the Clinicaltrial files represent an individual clinical trial, identified by an ID, listing the sponsor, the status of the study and other necessary details.

The pharma.csv file contains a small number if a publicly available list of pharmaceutical violations. For the purpose of this work, the assessment focus is on the parent company which contains the name of the pharmaceutical company in question

I am to implement three steps three times once in Spark SQL and twice in PySpark for RDD and DataFrame, For the visualisation of the results, I decided to use a tool of my choice which fulfil the assessment requirements.

DATA CLEANING AND PREPARATION

The Databricks compute tool has been created using the Create tab, it provides a powerful and scalable platform for data processing and analysis. While several computes have been created during the process of solving this evaluation due to a timeout caused by inactivity, the following example shows how compute has been created.

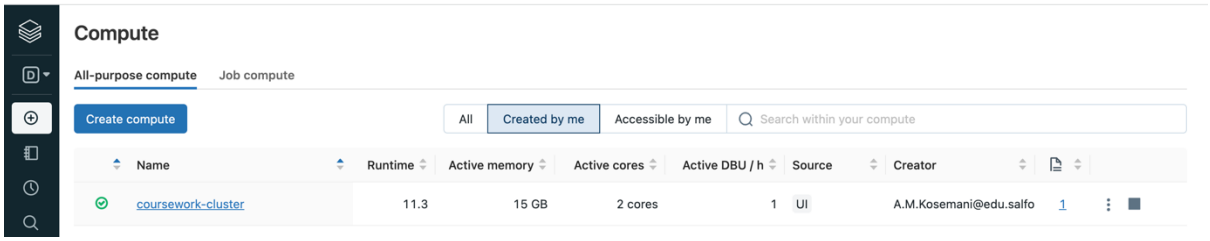


Fig 1.0 Creating a compute

Uploading of datasets was done by using graphical user's graphical interphase(GUI) tab of the Databricks File System(DBFS) and the imported files was saved in the FileStore/tables directory. A screenshot of this process is shown below.

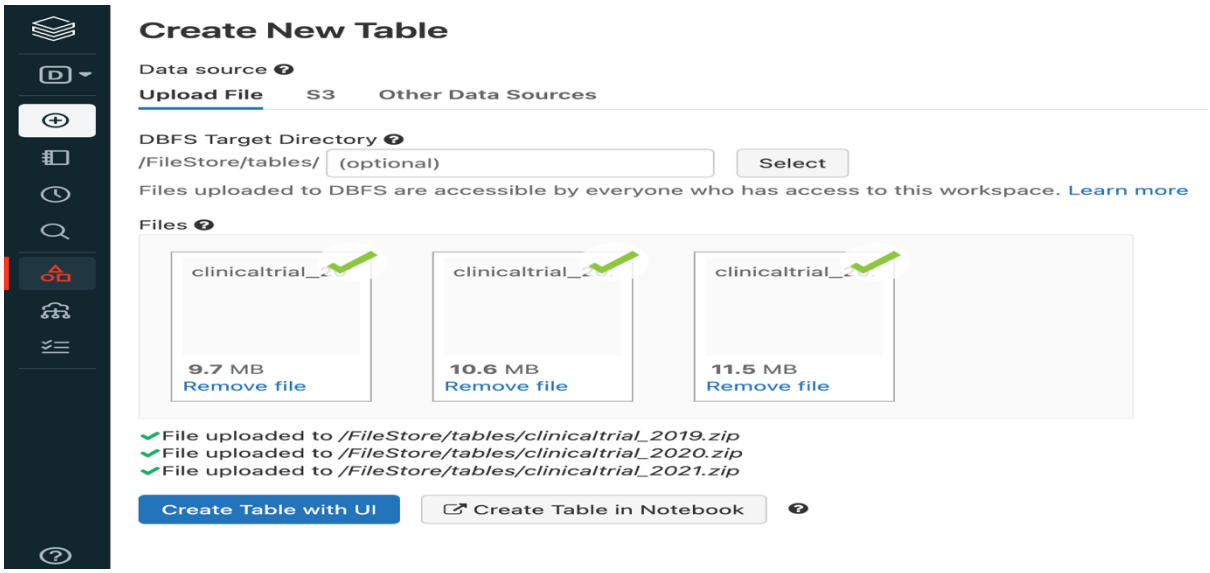


Fig 1.1 uploading clinicaltrial .csv files using the Databricks create tab GUI.

LIBRARIES IMPORTATION AND FILE UNZIPPING

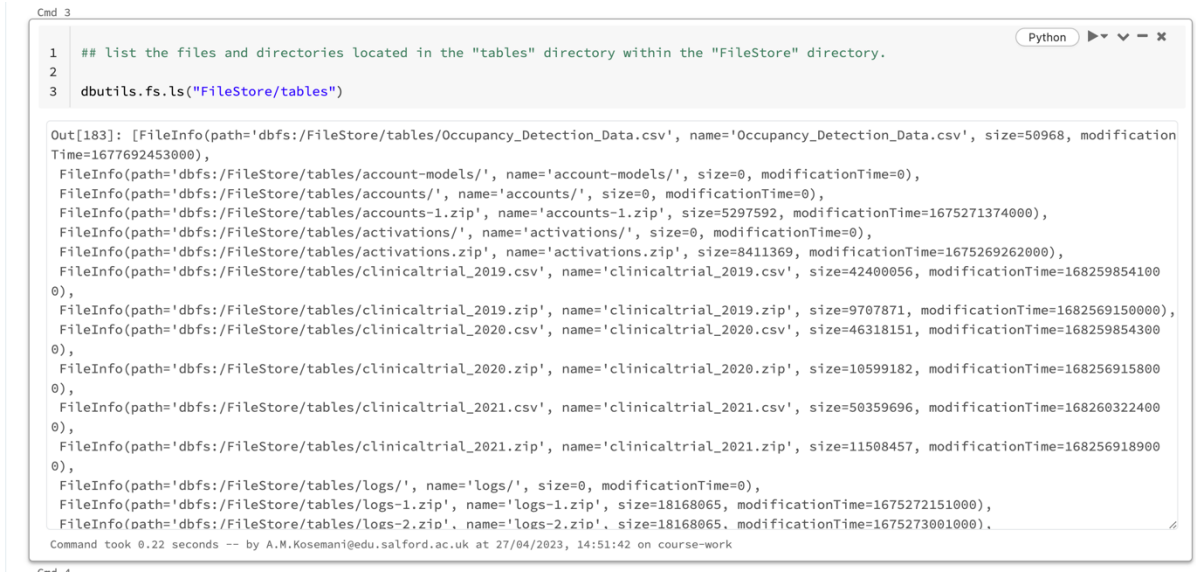
After uploading the 3 clinicaltrial .csv files into the FileStore/tables directory, some series of other operation were required for the process of unzipping the files, but first I was able to import the required packages which are needed for this project which is shown below

Fig 1.2 Libraries setup and importation

The first line import csv module and the following like imports the add operator from the operator module, both modules are responsible for reading and manipulating data.

The third line imports the pyplot module from the matplotib while the fourth line created a SparkSession object, the pyplot is a data visualization library and will be used to create graphs and charts in this assessment. The SparkSession serves as an entry point to spark functionality. Hence, the **"sc"** param is a SparkContext which is purpose is to set up connection to the spark cluster.

After the library's importation, the imported files were then moved from FileStore/tables to what is called the driver node which is the /tmp directory this is because the FileStore/tables doesn't support files unzipping. For this reason, it has no command for file unzipping.



```
Cmd 3

## list the files and directories located in the "tables" directory within the "FileStore" directory.

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

```
Out[183]: [FileInfo(path='dbfs:/FileStore/tables/Occupancy_Detection_Data.csv', name='Occupancy_Detection_Data.csv', size=50968, modificationTime=1677692453000),
 FileInfo(path='dbfs:/FileStore/tables/account-models/', name='account-models/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/FileStore/tables/accounts/', name='accounts/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/FileStore/tables/accounts-1.zip', name='accounts-1.zip', size=5297592, modificationTime=1675271374000),
 FileInfo(path='dbfs:/FileStore/tables/activations/', name='activations/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/FileStore/tables/activations.zip', name='activations.zip', size=8411369, modificationTime=1675269262000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2019.csv', name='clinicaltrial_2019.csv', size=42400056, modificationTime=1682598541000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2019.zip', name='clinicaltrial_2019.zip', size=9707871, modificationTime=1682569150000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2020.csv', name='clinicaltrial_2020.csv', size=46318151, modificationTime=1682598543000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2020.zip', name='clinicaltrial_2020.zip', size=10599182, modificationTime=1682569158000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1682603224000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.zip', name='clinicaltrial_2021.zip', size=11508457, modificationTime=1682569189000),
 FileInfo(path='dbfs:/FileStore/tables/logs/', name='logs/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/FileStore/tables/logs-1.zip', name='logs-1.zip', size=18168065, modificationTime=1675272151000),
 FileInfo(path='dbfs:/FileStore/tables/logs-2.zip', name='logs-2.zip', size=18168065, modificationTime=1675273001000),
```

```
Command took 0.22 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:51:42 on course-work
```

Fig 1.3 Checking the for the imported files with the dbutils.fs.ls() command

```
## copy the clinicaltrail_2019.zip file to file:/tmp/ for unziping

dbutils.fs.cp("dbfs:/FileStore/tables/clinicaltrial_2019.zip", "file:/tmp/")
```

```
Out[184]: True
```

```
Command took 0.55 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:51:45 on course-work
```

```
Cmd 5

## copy the clinicaltrail_2020.zip file to file:/tmp/ for unziping

dbutils.fs.cp("dbfs:/FileStore/tables/clinicaltrial_2020.zip", "file:/tmp/")
```

```
Out[185]: True
```

```
Command took 0.47 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:51:48 on course-work
```

```
Cmd 6

## copy the clinicaltrail_2021.zip file to file:/tmp/ for unziping

dbutils.fs.cp("dbfs:/FileStore/tables/clinicaltrial_2021.zip", "file:/tmp/")
```

```
Out[186]: True
```

```
Command took 0.47 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:51:52 on course-work
```

Fig 1.4 Coping the .zip files from FileStore/tables to /tmp dir.

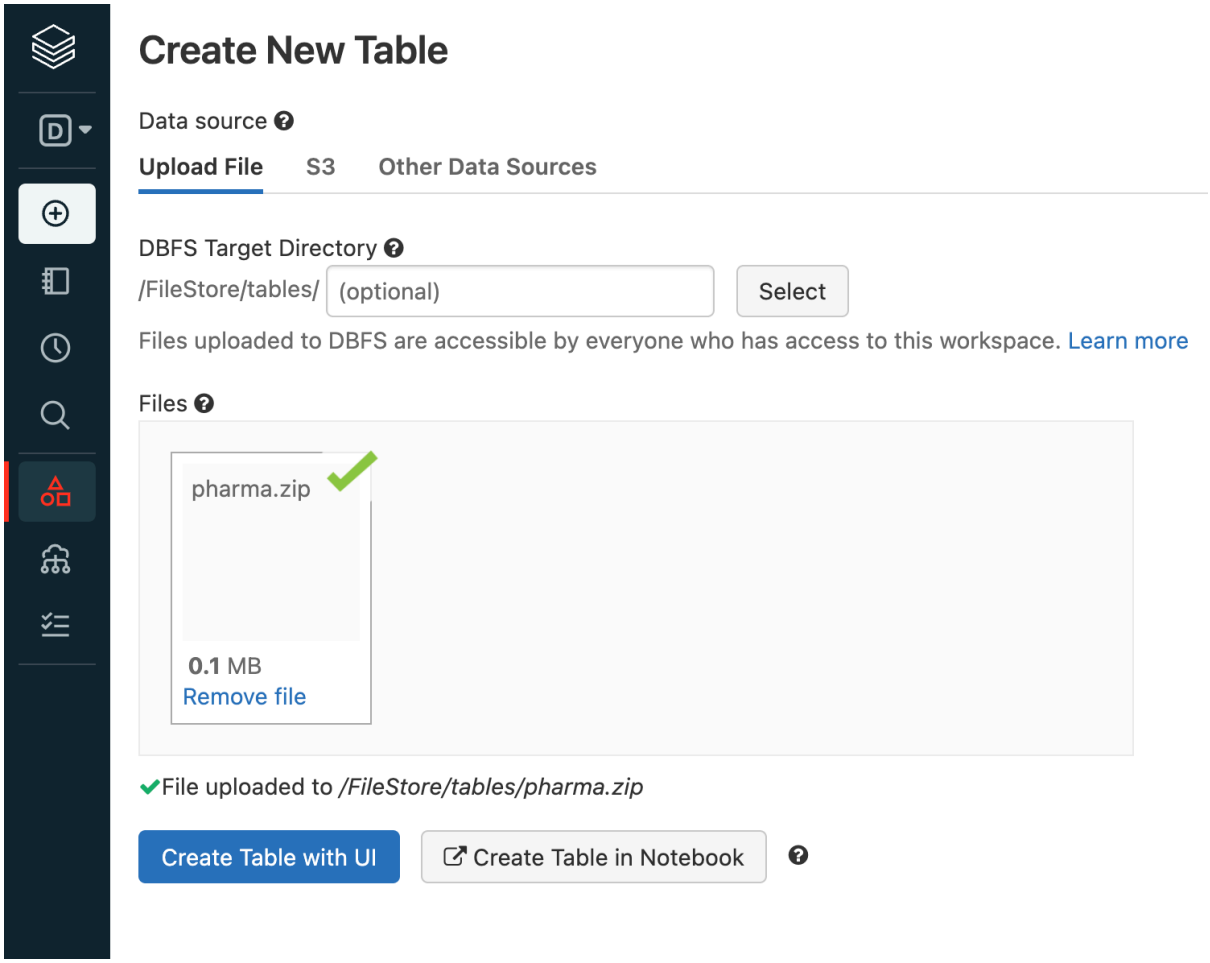


Fig 1.5 uploading pharma.csv file using the Databricks create tab GUI.

Fig 1.6

The ls command checks the **FileStore/tables** dir to confirm that pharma.csv file has been uploaded, and the UNIX **“cp”** command moves this file to \tmp directory for unzipping

Fig 1.7 Checking the /tmp directory for the moved .zip files.

Fig 1.8 unzipping the .zip files

The **“unzip”** command is used to unzip files which are in the zip format, the **“-d”** which is followed by the dir path is used to specify the directory the extracted files should be placed.

The above image in **Fig 1.8** extracts the contents of the .zip files into the **/tmp/** directory.

Fig 1.9 Creating tables directory in the FileStore.

A directory named "tables" in the FileStore folder is created in **Fig 1.9**

Fig 2.0 Moving the unzipped .csv files to the FileStore/tables

In Fig 2.0, the dbutils module is used to move a file from a location file path to another. This dbutils.fs.mv method is used in the Fig above to move a file within the Databricks file system (DBFS). The ‘true’ options which is provide in the argument in the method which means to move the subdirectories and files under the source directory is a recurse statement indicating whether to move files reclusively.

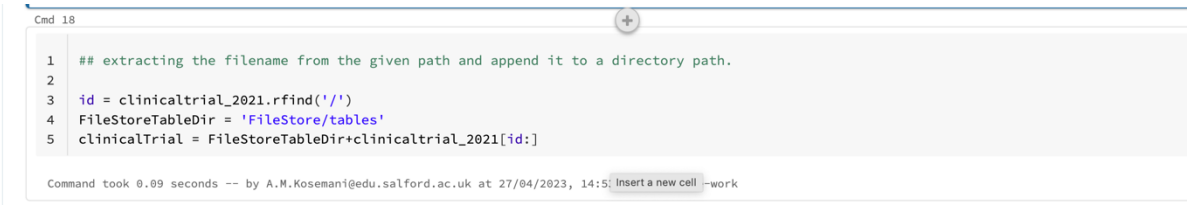


Fig 2.1 extracting the filename from the given path to append it to a directory path.

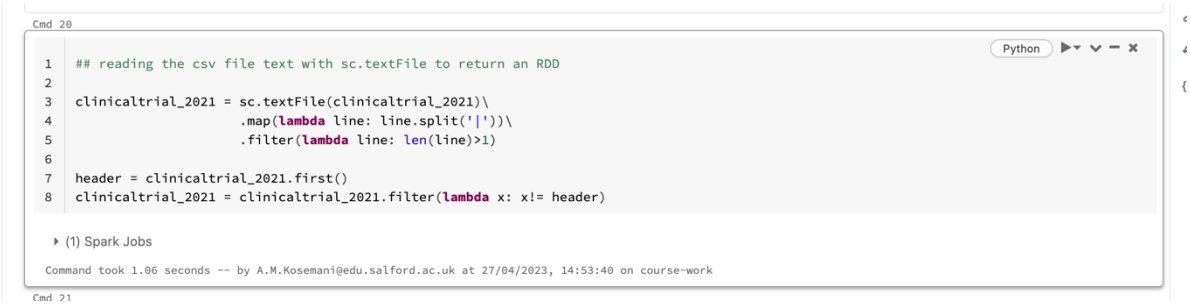


Fig 2.2 reading the csv file text with sc.textFile to return an RDD

It is safe to assume that the text file contains a "|" delimiter and a header as the first row, and that the header has more than has more than one element, if these assumptions are not met for some reasons this code will produce error or an unexpected result.

In **Fig 2.2** the clinicaltrial_2021 file was read and splits each line by a delimiter which filters out the lines with only one element. The header is stored separately in the header variable which is filtered out of the dataset.

The result of this operation is a Resilient Distributed Dataset object which called RDD.

Fig 2.3, using collect method to return all rows in the DataFrame

PROBLEMS ANSWERS IMPLEMENTATION(PySpark RRD).

SOLUTION 1.

Fig 2.4 Numbers of count in a dataset.

The number of studies was obtained by using the count method. In essence, it returns the numbers of rows in an entire dataset that correspond to the number of different IDs within a given Clinicaltrial Dataset. Therefore, the number of studies can be seen in the screenshot above for the clinicaltrial_2021 dataset with a total number of distinct counts is 387261.

SOLUTION 2.

Fig 2.5 Checking the file header's values.

Fig 2.6

As shown in Fig 2.5, the first step is to check for the value of Type in the header's Array, with the value of an array read from zero, the value of Type is equals 5, After which a map function is used to select the Type column in the clinicaltrial-2021 file for all the years rows and observations under the column, any missing observation in the column was filtered out. Aggregation was done which returns an output in a random order. The output was then sorted out in ascending order which a collection method invoked to display the result.

SOLUTION 3

Fig 2.7

Fig 2.8 The five top conditions in the dataset.

The assumption was that the column condition is not empty or null and that there are rows in a column with more than one conditions which require separation to get an output

The value of the conditions was confirmed by checking the header as show in **Fig 2.5**, A map function was then used to select the conditions column inside the clinicaltrial file which returns every observation or rows under conditions. Filtering was done to take care of the missing observations. There are rows with more than one condition and a split was needed to separate the conditions rows.

Finally, to count the number of different conditions in each row under conditions column a map function was used, the resulting count was now aggregated which was sorted sing the **sortByKey** function as seen in **Fig 2.8**

SOLUTION 4

Fig 2.9 reading the .csv file text with sc.textFile

```
1  pharma_header
```

```
Out[211]: ['"Company"',
 '"Parent_Company"',
 '"Penalty_Amount"',
 '"Subtraction_From_Penalty"',
 '"Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting"',
 '"Penalty_Year"',
 '"Penalty_Date"',
 '"Offense_Group"',
 '"Primary_Offense"',
 '"Secondary_Offense"',
 '"Description"',
 '"Level_of_Government"',
 '"Action_Type"',
 '"Agency"',
 '"Civil/Criminal"',
 '"Prosecution_Agreement"',
 '"Court"',
 '"Case_ID"',
 '"Private_Litigation_Case_Title"',
 '"Lawsuit_Resolution"',
 '"Facility_State"',
```

```
Command took 0.13 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:54:43 on course-work
```

Fig 3.0 Checking the pharma's columns.

```
Cmd 37
```

Python

```
1  Parent_Company = 1
2  Sponsor = 1
```

```
Command took 0.04 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:54:47 on course-work
```

```
Cmd 38
```

Fig 3.1 giving a value position to parent company and sponsor column

```
Cmd 38
```

Python

```
1  sponsor = clinicaltrial_2021.map(lambda col: col[Sponsor])
2  pharmaceutical = pharma.map(lambda col: col[Parent_Company])\
3  .map(lambda x: x.strip('\"'))
4
5  pharmaceutical_rdd = sc.broadcast(pharmaceutical.collect())
6
7  sponsor.filter(lambda x: x not in pharmaceutical_rdd.value)\
8  .map(lambda x: (x, 1))\
9  .reduceByKey(add)\
10 .sortBy(lambda x: x[1], ascending = False)\
11 .take(10)
```

▸ (4) Spark Jobs

```
Out[213]: [('National Cancer Institute (NCI)', 3218),
 ('M.D. Anderson Cancer Center', 2414),
 ('Assistance Publique - Hôpitaux de Paris', 2369),
 ('Mayo Clinic', 2300),
 ('Merck Sharp & Dohme Corp.', 2243),
 ('Assiut University', 2154),
 ('Novartis Pharmaceuticals', 2088),
 ('Massachusetts General Hospital', 1971),
 ('Cairo University', 1928),
 ('Hoffmann-La Roche', 1828)]
```

```
Command took 8.81 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:54:54 on course-work
```

Fig 3.2 ten most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

Before carrying out the task, my assumption was that the parent company column should have all the pharmaceuticals, but the not sponsors and the sponsors. Based on **Fig 3.0**, the column number for both parent company and sponsor are 1 according to **Fig 3.1**. In **Fig 3.2**, the RDD created for sponsor was filtered out to get a list of 10 sponsors that are not pharmaceuticals.

SOLUTION 5

QUESTION 5

Cmd 40

```
year='2021'

ans = clinicaltrial_2021.map(lambda col: [col[x] for x in [2,4]])\
.filter(lambda x: x[0] == 'Completed')\
.map(lambda x: x[1])\
.filter(lambda x: x!= '')\
.map(lambda x: tuple(x.split()))\
.filter(lambda x: x[1] == year)\
.groupByKey().mapValues(len)\
.sortBy(lambda x: x[1], ascending=False)

ans.collect()
```

▸ (3) Spark Jobs

```
Out[214]: [('Mar', 1227),
 ('Jan', 1131),
 ('Jun', 1094),
 ('May', 984),
 ('Apr', 967),
 ('Feb', 934),
 ('Jul', 819),
 ('Aug', 700),
 ('Sep', 528),
 ('Oct', 187)]
```

Command took 3.44 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:55:07 on course-work

Cmd 41

```
def changer(month):
  if month[0]=='Jan':
    return (1,month)
  elif month[0]=='Feb':
    return (2,month)
  elif month[0]=='Mar':
    return (3, month)
  elif month[0]=='Apr':
    return (4, month)
  elif month[0]=='May':
    return (5, month)
  elif month[0]=='Jun':
    return (6, month)
  elif month[0]=='Jul':
    return (7, month)
  elif month[0]=='Aug':
    return (8, month)
  elif month[0]=='Sep':
    return (9, month)
  elif month[0]=='Oct':
    return (10, month)
  elif month[0]=='Nov':
    return (11, month)
  else:
    return (12,month)
```

Command took 0.08 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:55:15 on course-work

Fig 3.3

Cmd 42

```
final = ans.map(changer)\
.sortBy(lambda x: x[0], ascending=True)\
.map(lambda x: x[1])
final.collect()
```

▸ (3) Spark Jobs

```
Out[216]: [('Jan', 1131),
 ('Feb', 934),
 ('Mar', 1227),
 ('Apr', 967),
 ('May', 984),
 ('Jun', 1094),
 ('Jul', 819),
 ('Aug', 700),
 ('Sep', 528),
 ('Oct', 187)]
```

Command took 0.47 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:55:19 on course-work

Cmd 43

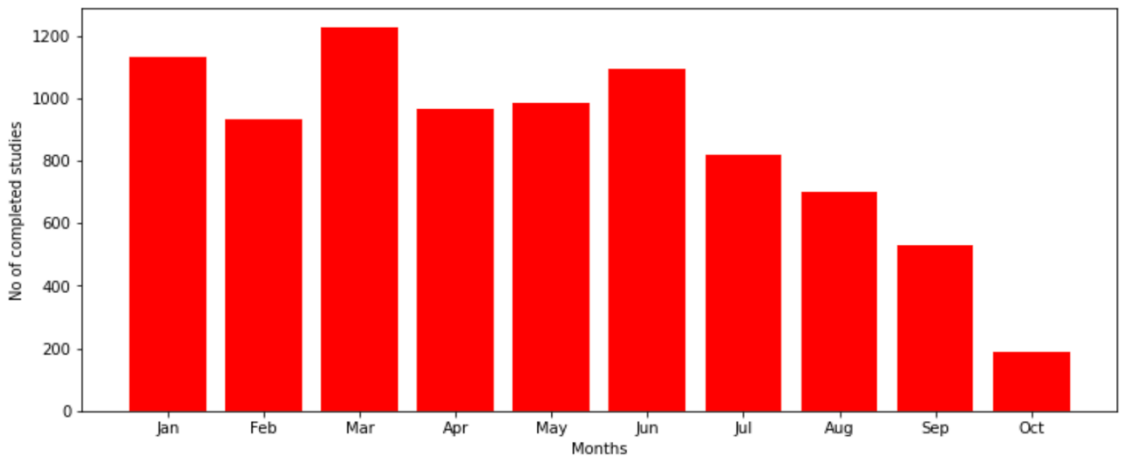
```
months = final.map(lambda x: x[0]).collect()
counts =final.map(lambda x: x[1]).collect()
```

▸ (2) Spark Jobs

Command took 0.31 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:55:24 on course-work

Cmd 44

```
plt.figure(figsize=(12, 5))
plt.bar(months, counts, color='red')
plt.xlabel('Months')
plt.ylabel('No of completed studies')
plt.show()
```



Command took 0.37 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 14:55:28 on course-work

Cmd 45

Fig 3.4 A graph of no of completed studies against the months.

In **Fig 3.3,** A RDD was created for completed studies by writing a for loop to loop through and select all studies under the status column and the corresponding date in the completion column then filtering was done in other to filter out the completed and the missing observations and splitting is then done after this to separate the month from year.

The next operation was the filtering by year and sorting the numbers of completed that was save in a variable name, with the result later viewed.

Finally, to have a visualization of the table, the month was then arranged accordingly using a switch if else statement with the visualization done using the matplotlib as shown in **Fig 3.4.**

EXTRAS.

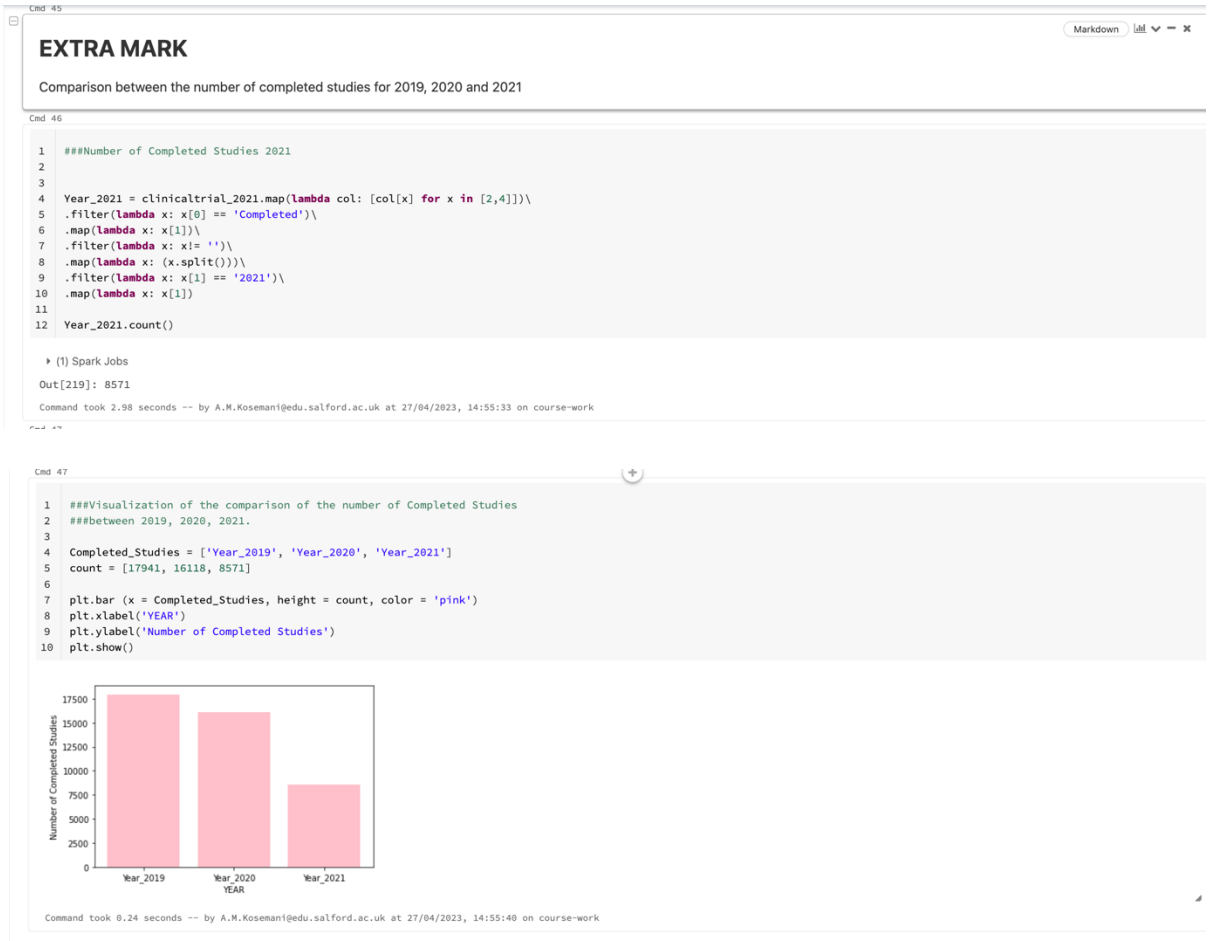


Fig 3.5 The results show a fall in the number of studies that have been finalised during these three years. In the image below, we see a visual representation of this.

PROBLEMS ANSWERS IMPLEMENTATION(PySpark SQL).

Cmd 1

```
drop table if exists clinicaltrial_2021;

create table clinicaltrial_2021
using csv
options(
header='true',
delimiter='|',
inferSchema='true',
mode='FAILFAST',
path='/FileStore/tables/clinicaltrial_2019.csv'
);

cache table clinicaltrial_2021
```

▸ (4) Spark Jobs

OK

Command took 11.96 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:17:05 on course-work

Fig 3.5 Creating clinicaltrial_2021 table.

This SQL query drop clinicaltrial_2021 table if it already exists and create a new table clinicaltrial_2021, the using csv implies csv as the source of data, the options take in arguments such as header, delimiter and whether the schema should be inferred automatically.

The cache table clinicaltrial_2021 SQL statement caches a table in memory which can improve the performance of the query by reducing the need to read from disk every single time.

Cmd 2

```
1
2  drop table if exists pharma;
3
4  create table pharma
5  using csv
6  options(
7  header='true',
8  delimiter=',',
9  inferSchema='true',
10 mode='PERMISSIVE',
11 path='/FileStore/tables/pharma.csv'
12 );
13
14 cache table pharma
```

▸ (4) Spark Jobs

OK

Command took 2.44 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:21:52 on course-work

Fig 3.5 Creating pharma's table.

Cmd 3

```
1  select * from clinicaltrial_2021
```

SQL

▸ (1) Spark Jobs

Table +

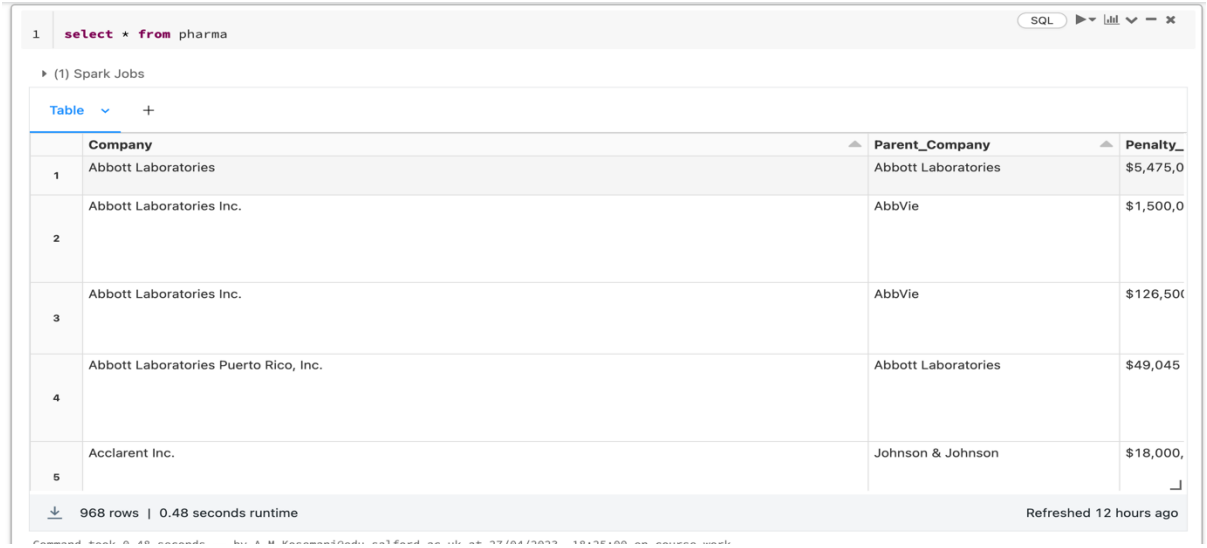
	Id	Sponsor	Status
1	NCT02758028	The University of Hong Kong	Recruiting
2	NCT02751957	Duke University	Completed
3	NCT02758483	Universidade Federal do Rio de Janeiro	Completed
4	NCT02759848	Istanbul Medeniyet University	Completed
5	NCT02758860	University of Roma La Sapienza	Active, not recruiting
6	NCT02757209	Consorzio Futuro in Ricerca	Completed

10,000 rows | Truncated data | 0.61 seconds runtime — Refreshed 12 hours ago

Command took 0.61 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:24:19 on course-work

Cmd 4

Fig 3.6 Retrieving all columns and rows from clinicaltrial_2021 table.



```
1  select * from pharma
```

(1) Spark Jobs

Table +

	Company	Parent_Company	Penalty_
1	Abbott Laboratories	Abbott Laboratories	$5,475,0
2	Abbott Laboratories Inc.	AbbVie	$1,500,0
3	Abbott Laboratories Inc.	AbbVie	$126,50(
4	Abbott Laboratories Puerto Rico, Inc.	Abbott Laboratories	$49,045
5	Acclarent Inc.	Johnson & Johnson	$18,000,

968 rows | 0.48 seconds runtime — Refreshed 12 hours ago

Fig 3.7 Retrieving all columns and rows from pharma's table.

SOLUTION 1.

Cmd 5

QUESTION 1

Cmd 6

```
1  select count(*) from clinicaltrial_2021;
```

(2) Spark Jobs

Table +

	count(1)
1	326348

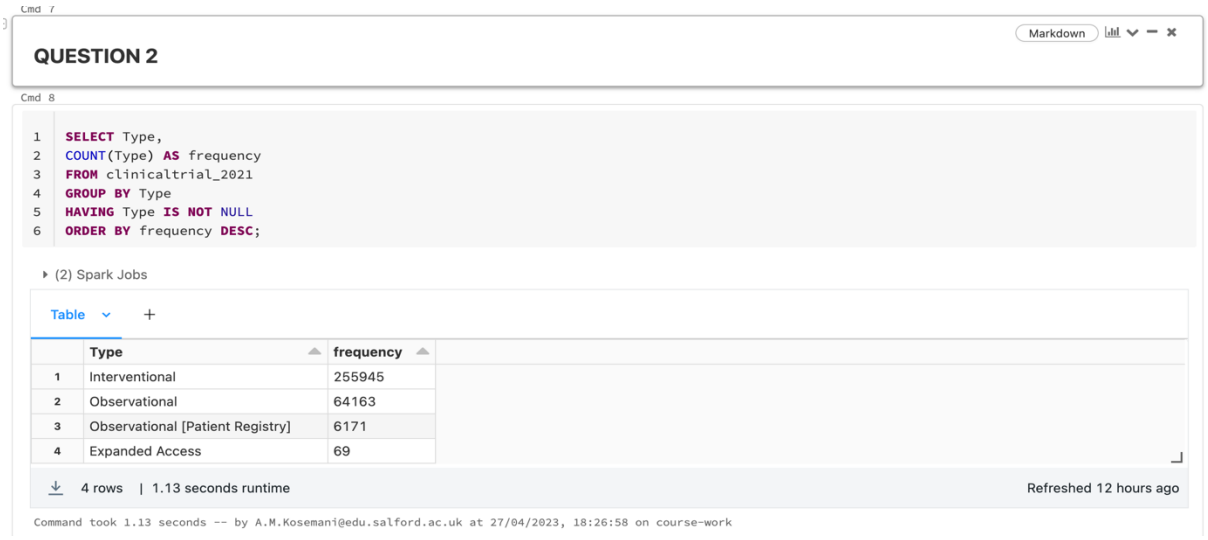
1 row | 0.44 seconds runtime — Refreshed 12 hours ago

Command took 0.44 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:23:24 on course-work

Fig 3.8 Fetching the total number of records in the clinicaltrial_2021 table.

The total number of record(rows) in clinicaltrial_2021 is 326348.

SOLUTION 2



QUESTION 2

```
SELECT Type,
COUNT(Type) AS frequency
FROM clinicaltrial_2021
GROUP BY Type
HAVING Type IS NOT NULL
ORDER BY frequency DESC;
```

(2) Spark Jobs

Table +

	Type	frequency
1	Interventional	255945
2	Observational	64163
3	Observational [Patient Registry]	6171
4	Expanded Access	69

4 rows | 1.13 seconds runtime — Refreshed 12 hours ago

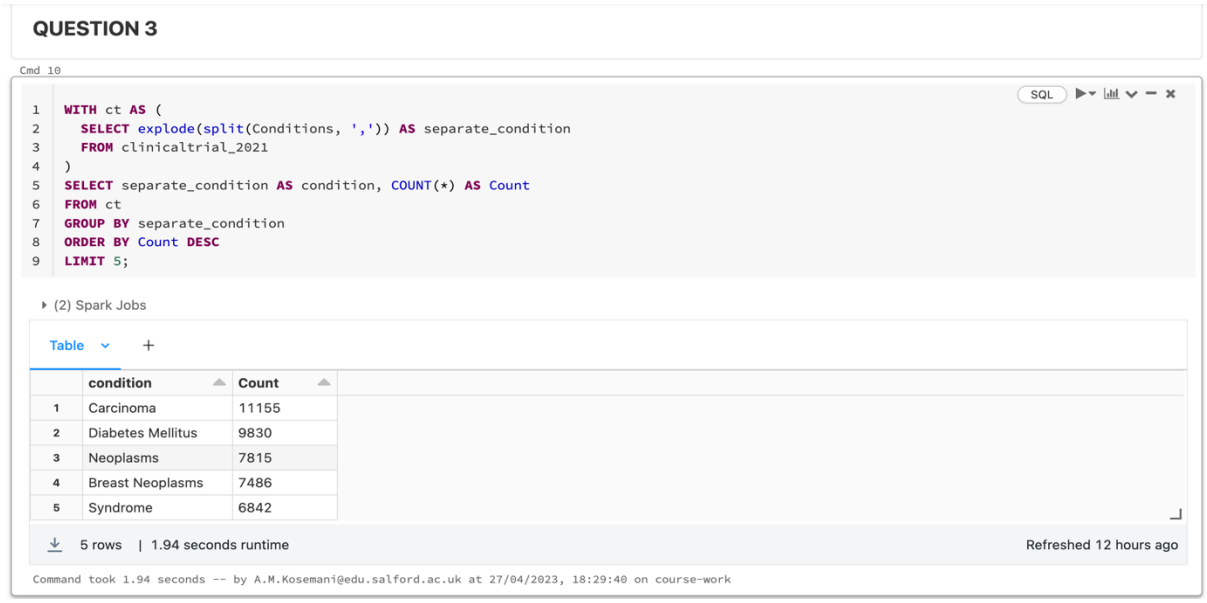
Command took 1.13 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:26:58 on course-work

Fig 3.9

This query retrieves data from the clinicaltrial_2021 table, grouping is done based on the Type column which then counts the number of occurrences of each type. Then ordering was done by the result of frequency occurrence in descending order

The not null value is only included in this result, the null values are filtered out by using where state which creates an exception.

SOLUTION 3



QUESTION 3

```
WITH ct AS (
  SELECT explode(split(Conditions, ',')) AS separate_condition
  FROM clinicaltrial_2021
)
SELECT separate_condition AS condition, COUNT(*) AS Count
FROM ct
GROUP BY separate_condition
ORDER BY Count DESC
LIMIT 5;
```

(2) Spark Jobs

Table +

	condition	Count
1	Carcinoma	11155
2	Diabetes Mellitus	9830
3	Neoplasms	7815
4	Breast Neoplasms	7486
5	Syndrome	6842

5 rows | 1.94 seconds runtime — Refreshed 12 hours ago

Command took 1.94 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:29:40 on course-work

Fig 4.0

In **Fig 4.0**, the query creates a common table expression **CTE** called **ct**, which uses the **WITH** statement. Which uses the explode and split functions to split the values in the conditions column using commas which sperate them into different roles

The other part of this query selects the separate_condition column from the ct. and uses count to count the number of times each condition appears. The result is then grouped by group by and order using ordered by statement.

This query returns the top 5 conditions which appears mostly in the condition column of the table **clinicaltrial_2021.**

SOLUTION 4

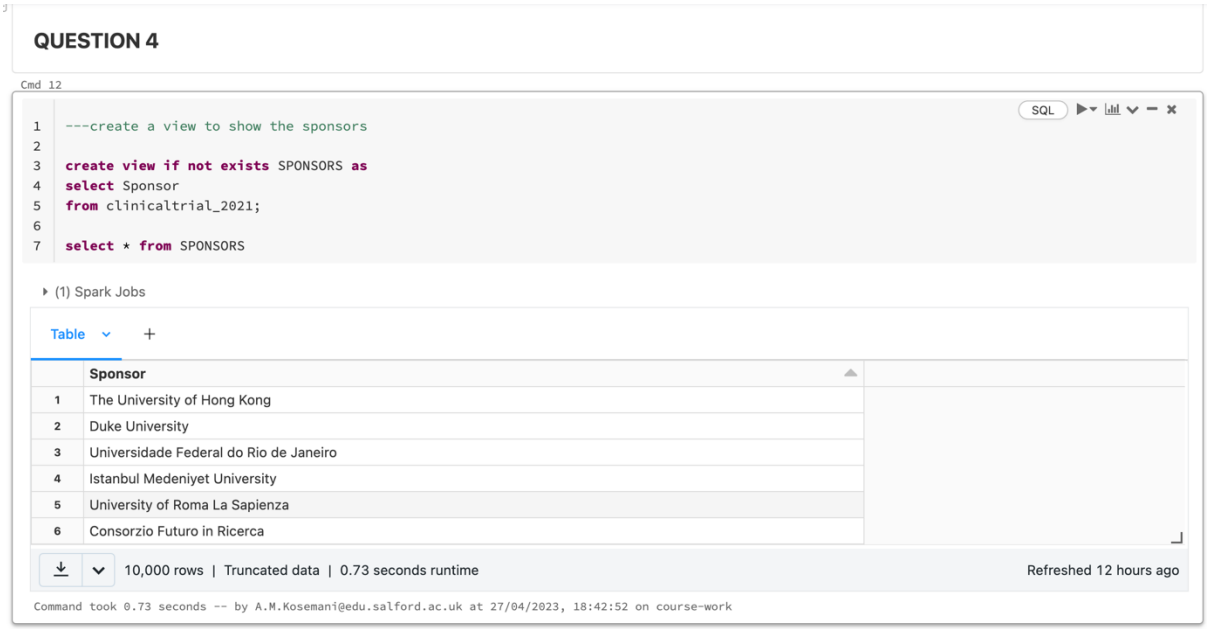


Fig 4.1 Create sponsor's view from the sponsor table.

```sql
---create a view for the pharmaceutical companies

create view if not exists pharmaceutical as
select Parent_Company
from pharma;

select * from pharmaceutical

```

(1) Spark Jobs

Table +

	Parent_Company
1	Abbott Laboratories
2	AbbVie
3	AbbVie
4	Abbott Laboratories
5	Johnson & Johnson
6	Abbott Laboratories

968 rows | 0.45 seconds runtime — Refreshed 12 hours ago

Command took 0.45 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:43:25 on course-work

SQL

```sql
---10 most common sponsors that are not pharmaceuticals

select Sponsor, count(*) as Count
from Sponsors_not_pharmaceuticals
where Sponsor is not null
group by Sponsor
order by Count desc
limit 10;
```

(3) Spark Jobs

Table +

	Sponsor	Count
1	National Cancer Institute (NCI)	3003
2	Merck Sharp & Dohme Corp.	2124
3	M.D. Anderson Cancer Center	2097
4	Mayo Clinic	1930
5	Novartis Pharmaceuticals	1881
6	Assistance Publique - Hôpitaux de Paris	1764

10 rows | 2.48 seconds runtime — Refreshed 12 hours ago

Command took 2.48 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:44:31 on course-work

Cmd 16

SQL

```sql
---create a view for sponsors that are not pharmaceutical companies

create view if not exists Sponsors_not_pharmaceuticals as
select * from SPONSORS
where Sponsor not in (select Parent_Company from pharmaceutical);


select * from Sponsors_not_pharmaceuticals
```

(2) Spark Jobs

Table +

	Sponsor
1	The University of Hong Kong
2	Duke University
3	Universidade Federal do Rio de Janeiro
4	Istanbul Medeniyet University
5	University of Roma La Sapienza
6	Consorzio Futuro in Ricerca

10,000 rows | Truncated data | 1.23 seconds runtime — Refreshed 12 hours ago

Command took 1.23 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 18:43:50 on course-work

Fig 4.2

From the above result, sponsor was selected from sponsor not pharmaceutical then a count was taken to get an aggregation aliased count, which was then ordered in descending order

Fig 4.1, Create a table that alias sponsors consisting of sponsor from the sponsor table **clinicaltrial_2021**, the same way is used to create pharmaceutical view which creates a view aliased pharmaceutical consisting of all companies from pharma.

A select statement was then used to select sponsors which were not same with parent companies

SOLUTION 6

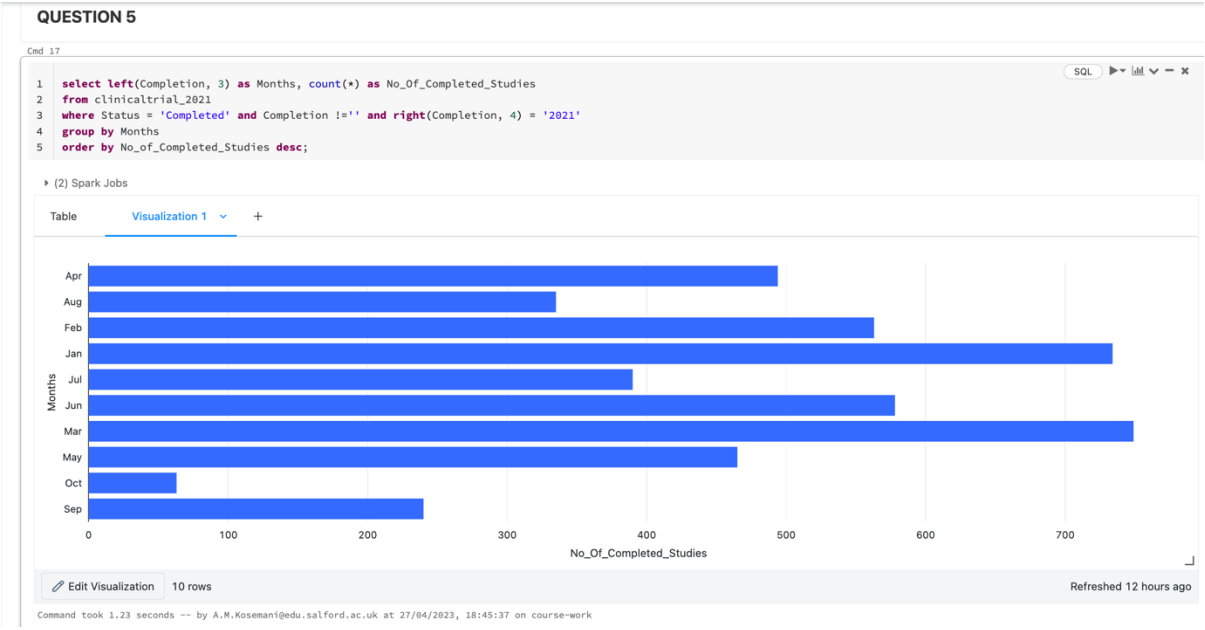


Fig 4.3

The substring and count functions are used to list the first three elements of a completion date column, while each month's occurrence is counted. Where filter is used to filter to where the status of the study is completed, and the year is 2021. The group by function is also employed due to the use of the count aggregate function.

Using standard Databricks visualisation tools, a graph of the results shown above can be seen below.

PROBLEMS ANSWERS IMPLEMENTATION(PySpark DataFrame).

DATA CLEANING AND PREPARATION

Data cleaning is the process of identifying errors or inconsistencies in a dataset and resolving them, if necessary, in order to ensure that the data being worked on is correct, consistent and usable. Although we used the correct format for columns as variables and observations in rows, data was processed dynamically so that it made things a lot easier.

The missing values were also investigated by using the pyspark data frame and it was found that there are 3 columns, namely "Completion", "Outcomes" and "Interventions," which do not have any observations.

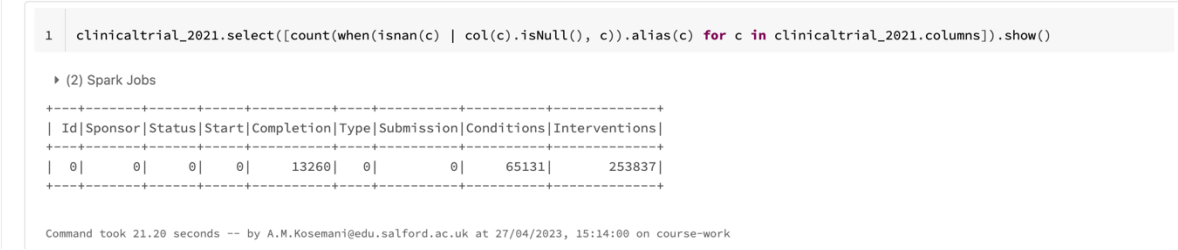


Fig 4.4

Fig 4.4 checks for the number of missing observations for all the column by using the select function to loop over each column in the dataset by using the count function to count when either isnan function return as true or when the column is null

The ‘alias’ basically is a way of temporary renaming the column. As we can see above, this result is the same. However, these missed observations have not been deleted because most of the tasks dealt with are specifically for one variable and if this is to be eliminated, it would lead to significant loss of data in the dataset.

Although no general cleaning of the data base has been carried out, missing observations have been removed from certain tasks where necessary. Under the relevant task, such cleaning shall be further explained.

SOLUTION 1

Fig 4.5

The numbers of studies were gotten by using count function, this function, in essence, returns a number of rows to the whole dataset as it will correspond with the total number of distinct IDs for each clinical trial data set. As a result, the following table shows the number of trials for clinicaltrial_2021 dataset.

SOLUTION 2

Fig 4.6

All rows in the **'Type'** column of the clinicaltrial data was collected using the **'groupBy'** function and result aggregated by 'count'. The use of the truncate false clause with the frequency presented in descending order by using the **'orderBy'** function on the count was used to avoid truncation in the results that usually occur in pyspark, and the result is shown below for the clinical trial_2021 dataset.

SOLUTION 3

QUESTION 3

The top 5 conditions (from Conditions) with their frequencies

Cmd 19

```
from pyspark.sql.window import Window
```

Command took 0.06 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 16:08:51 on course-work

Cmd 20

```
clinicaltrial_2021.select('Conditions').na.drop()\
.select(split('Conditions', ',').alias('Conditions'))\
.withColumn('Individual_condition', explode('Conditions'))\
.drop('Conditions')\
.groupBy('Individual_condition').count()\
.orderBy('count', ascending=False)\
.limit(5)\
.withColumn('S/N', row_number().over(Window.orderBy(monotonically_increasing_id())))\
.select(col('S/N'), col('Individual_condition') , col('count'))\
.show(truncate=False)
```

▸ (2) Spark Jobs

```
+---+--------------------+-----+
|S/N|Individual_condition|count|
+---+--------------------+-----+
|1  |Carcinoma           |13389|
|2  |Diabetes Mellitus   |11080|
|3  |Neoplasms           |9371 |
|4  |Breast Neoplasms    |8640 |
|5  |Syndrome            |8032 |
+---+--------------------+-----+
```

Command took 7.02 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 16:08:54 on course-work

Fig 4.7

Fig 4.7 started by selecting "Conditions" from the DataFrame and deleting any rows that have an empty value in them. Then it applies "split" to the condition columns, splitting each string with commas and creating a new column 'Conditions' which contains an array of individual conditions.

The next step is to create a new row for each condition on the "Conditions" array by using an explode function. At this point, the current "Conditions"

column is deleted. The resulting DataFrame shall then be divided by "Individual_Condition", i.e., the individual conditions, and counts of each condition shall be calculated. The resulting DataFrame is sorted by the total count in a descending order, and top 5 criteria are selected using "limit" option.

Finally, using "row_number" and window function, we can add a new column to the DataFrame named 'SDN'. For each entry in the DataFrame, this column only contains a row number. For the top five conditions, the final DataFrame is selected to display the "SN", "Individual_Condition", and "Count" columns.

SOLUTION 4

QUESTION 4

10 most common sponsors that are not pharmaceutical companies + the number of clinical trials sponsored.

Cmd 22

```
1  parent = list(pharma.select('Parent_Company').toPandas()['Parent_Company'])
2
3  clinicaltrial_2021.select('Sponsor')\
4  .filter(~col('Sponsor').isin(parent))\
5  .groupBy('Sponsor').count()\
6  .orderBy('count', ascending=False)\
7  .show(10, truncate=False)
```

▸ (3) Spark Jobs

```
+---------------------------------------+-----+
|Sponsor                                |count|
+---------------------------------------+-----+
|National Cancer Institute (NCI)        |3218 |
|M.D. Anderson Cancer Center            |2414 |
|Assistance Publique - Hôpitaux de Paris|2369 |
|Mayo Clinic                            |2300 |
|Merck Sharp & Dohme Corp.              |2243 |
|Assiut University                      |2154 |
|Novartis Pharmaceuticals               |2088 |
|Massachusetts General Hospital         |1971 |
|Cairo University                       |1928 |
|Hoffmann-La Roche                      |1828 |
+---------------------------------------+-----+
only showing top 10 rows
```

Command took 14.26 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 16:12:12 on course-work

Fig 4.8

In Fig 4.8 A list is formed of the parent company column from the pharma's DataFrame which contains values, then the sponsor column of the clinicaltrial_2021 was filtered to only include values that are not in the parent list. The resulting DataFrame is ordered by counting and displays the top ten results without truncation.

QUESTION 5

Cmd 24

```
###Table and Visualization for Completed Studies

dataFrame = clinicaltrial_2021.select('Status','Completion').na.drop()\
.filter(col('Status') == 'Completed')\
.withColumn('Month', split(col('Completion'), ' ').getItem(0))\
.withColumn('Year', split(col('Completion'), ' ').getItem(1))\
.filter(col('Year') == '2021')\
.drop('Status','Completion','Year')\
.groupBy('Month')\
.count()

display(Completed_Studies)
```

▸ (2) Spark Jobs

▸ dataFrame: pyspark.sql.dataframe.DataFrame = [Month: string, count: long]

Table +

	Month	count
1	Oct	187
2	Sep	528
3	Aug	700
4	May	984
5	Jun	1094
6	Feb	934

10 rows | 3.34 seconds runtime

1

6	Feb	934

10 rows | 3.34 seconds runtime

1

Command took 3.34 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 16:47:26 on course-work

Cmd 25

```
completed_studies = dataFrame.withColumn('S/N_Month',
  when(col('Month')=='Jan', 1)\
  .when(col('Month')=='Feb', 2)\
  .when(col('Month')=='Mar', 3)\
  .when(col('Month')=='Apr', 4)\
  .when(col('Month')=='May', 5)\
  .when(col('Month')=='Jun', 6)\
  .when(col('Month')=='Jul', 7)\
  .when(col('Month')=='Aug', 8)\
  .when(col('Month')=='Sep', 9)\
  .when(col('Month')=='Oct', 10)\
  .when(col('Month')=='Nov', 11)\
  .when(col('Month')=='Dec', 12)\
 )


completed_studies = completed_studies.orderBy('S/N_Month', ascending = True)\
.drop('S/N_Month')

display(completed_studies)
```

▸ (2) Spark Jobs

▸ completed_studies: pyspark.sql.dataframe.DataFrame = [Month: string, count: long]

Fig 4.9

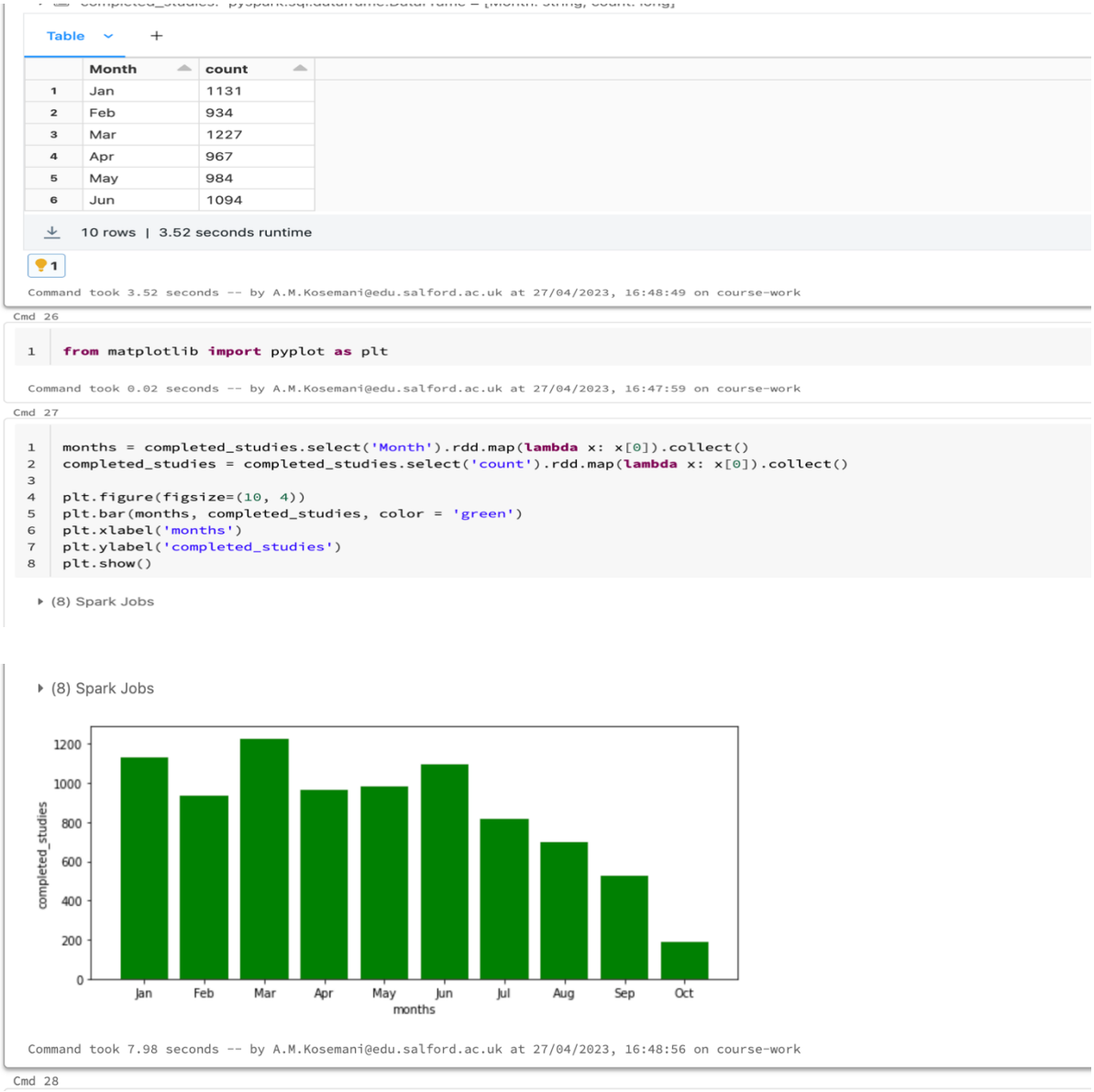


Fig 5.0

In **Fig 4.9**, the completion and status columns were selected and then drops all the rows with null values, after which filtering was done for the completed studies, followed by splitting the completion column into both **Years** and **Months** columns, then after this I was able to filter through for the studies that were completed in 2021. then the data was grouped by month, which allows the counting of the number of studies which was completed for each of the month. With the result of the DataFrame operation stored in variable name **completed_studies** .

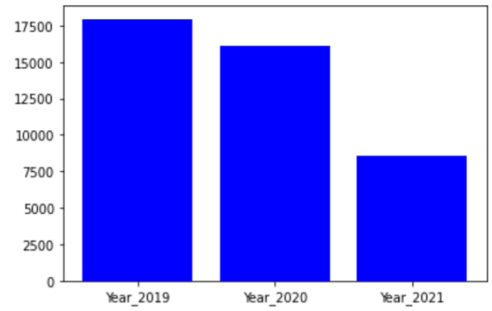
In **Fig 4.9,** in the completed_studies DataFrame, a new column **S/N_Month** was created, which assigns a numerical value to each month. The result of the values is the stored in **the_completed_studies**, A new DataFrame was created by ordering the completed_studies DataFrame by the **S/N_Month** column in ascending order and dropping the **S/N_Month** column.

In **Fig 5.0**, using Matplotlib a bar graph of completed studies by month is shown.

Cmd 30

```
###Visualization of the comparison of the number of Completed Studies
###between 2019, 2020, 2021.
Completed_Studies = ['Year_2019', 'Year_2020', 'Year_2021']
count = [17941, 16118, 8571]

plt.bar(x = Completed_Studies, height = count, color = 'blue')
plt.show()
```



Command took 0.30 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 27/04/2023, 16:59:17 on course-work

Shift+Enter to run
Shift+Ctrl+Enter to run selected text

The steps of task 5 shall be repeated, except for filtering and grouping the month column by year. An analysis for 2019, 2020 and 2021 has been provided in the images above. As shown below, the results for the three years show a decrease in the number of completed studies each year with a visual representation.

TASK TWO

INTRODUCTION

The purpose of this task is to undertake a forecast analysis of FaultData that uses the readings obtained by using them, in order to determine whether machines are operating correctly. For every row containing 20 vibration sensor readings, and for each of the last columns indicating if there has been a failure when these measurements have been made, this dataset is entered in CSV format. 0 in this column indicates that there was no mechanical fault was found, while 1 indicates that a fault was found.

Data was analysed using the Pyspark data frame application, which has been performed in an environment called Databricks. As outlined in the presentation, some preliminary analysis of these figures with visualisation was performed on bokeh and matplotlib.

Pre-processing has been carried out by splitting the training and testing data set prior to the modelling of the data to make it suitable for machine learning. The training of the dataset for classification model has been performed with Spark, MLlib machine learning library and this is followed up by an experiment measuring its performance using MLflow.

COMPUTE SETUP AND FILE IMPORT

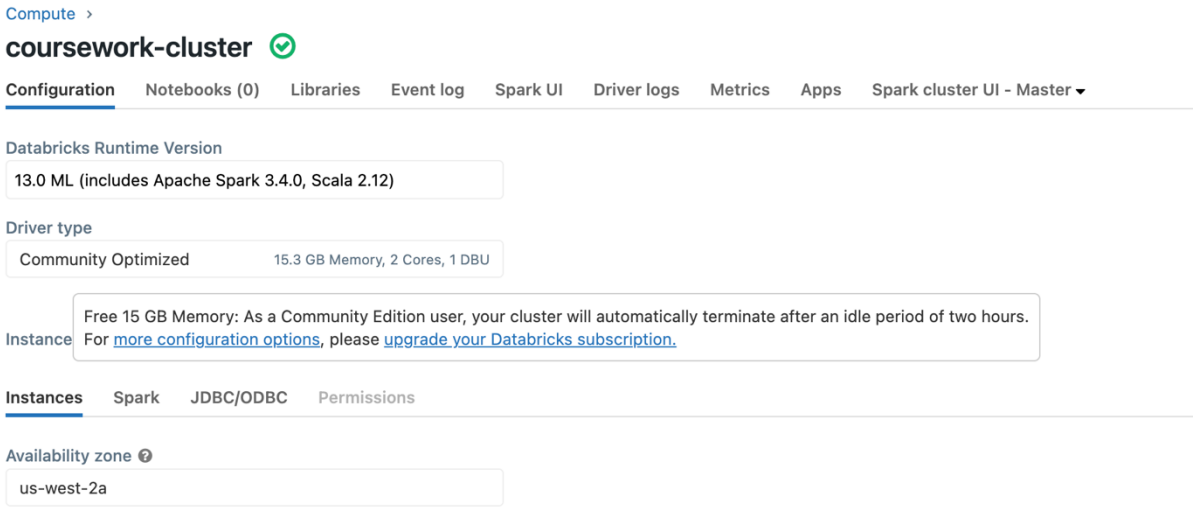


Fig 1.0 Compute setup.

A 15GB cluster that included Apache Spark 3.4.0 and Scala 2.12 has been created which is used for the implementation of this task using a Databricks Community Cloud Platform to create clusters containing two attached cores. To implement the data frame, resources and settings that have been created from cluster creation were used to generate notebooks in a standard Python language.

UPLOADING DATASET

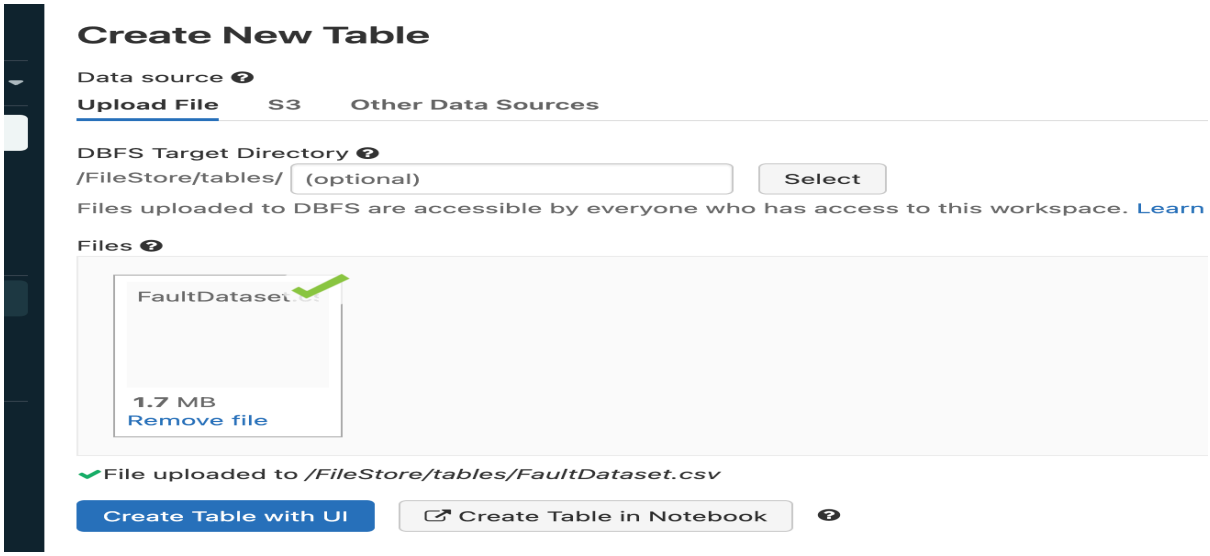


Fig 1.1 Uploading the dataset using the graphical user's interphase(GUI)

IMPORTING LIBRARIES

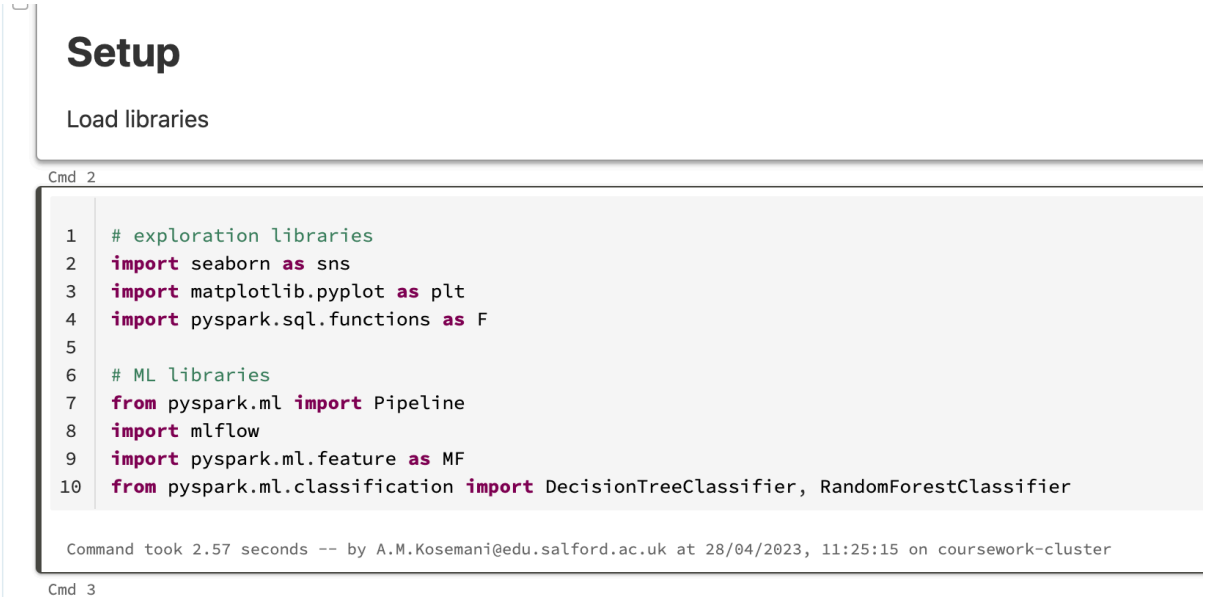


Fig 1.2 Data exploration and machine learning libraries

Fig 1.3 reading the file directory

Calling Show() function in order to show all five rows of DataFrame once I have read the file. This is a good way to look at the data and make sure that they are properly interpreted.

EXPLANATORY DATA ANALYSIS

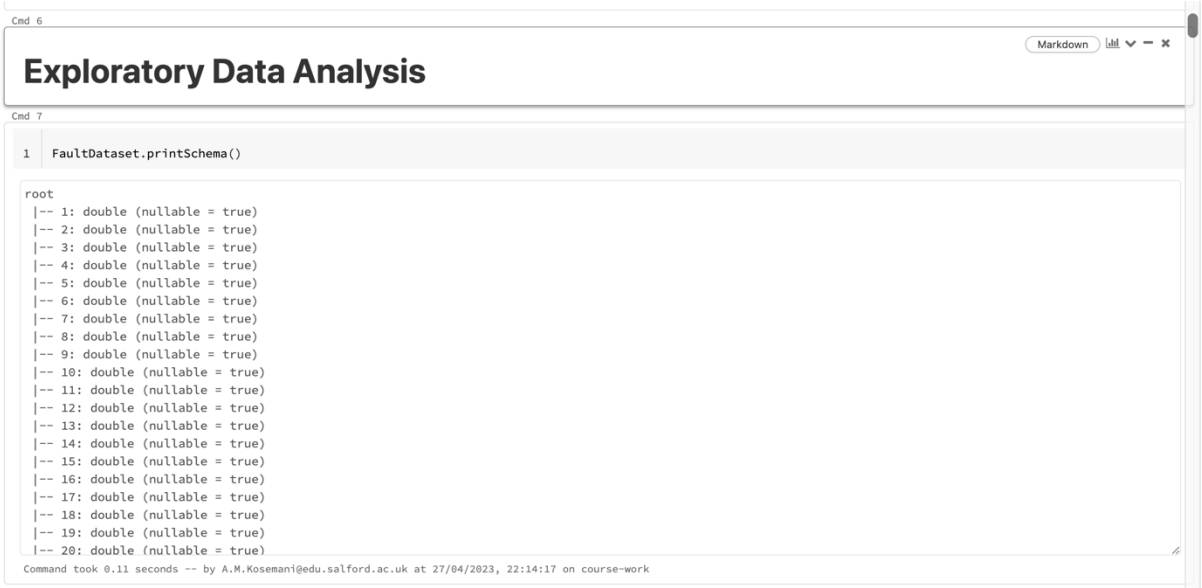


Fig 1.4 Exploratory data analysis

Prior to the creation of the model in machine learning, the EDA was usually used to examine the data for patterns, identify trends where they exist, check for possible assumptions using statistical descriptions and graphical representations.

The first EDA to be carried out for this task has been the checking of the type of data in respect of each variable included in the dataset. This was done using the **printSchema** syntax as seen in **Fig 1.4** below with all the 20 explanatory variables being the same data type while the response variable; **fault_detected** column is an integer consisting of "0" and "1" with 0 meaning no fault was detected with the machine and 1 means a fault was identified.

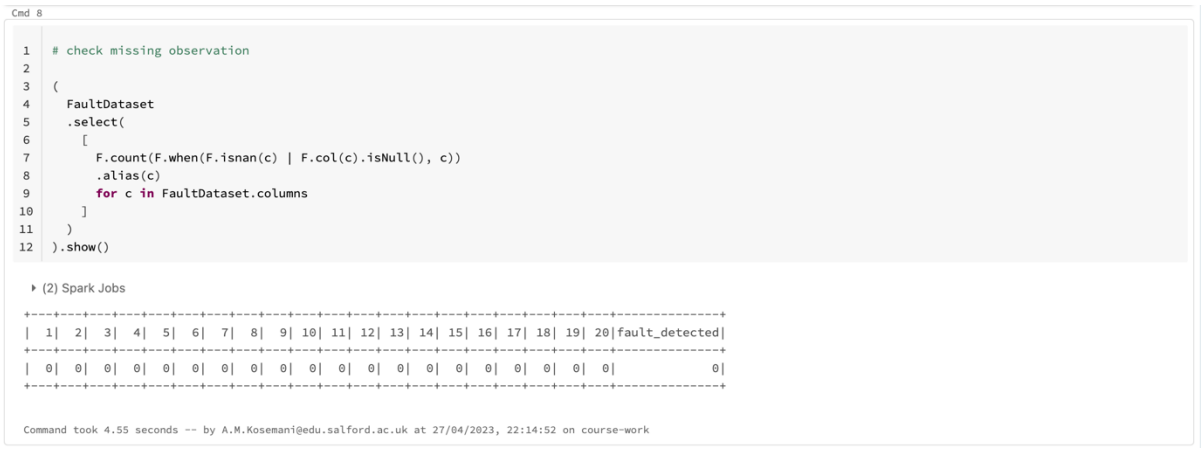


Fig 1.5 Fault dataset count and dimension.

The shape of the Fault Dataset was determined in a further examination carried out under EDA. The length of rows and columns in a two-dimensional data frame structure is computed by means of the number or Len for each row and column, as well as determining the shape dimension of the data. The dimensions of the data are very common to give you an idea of how big it is. The screen shot below shows the size of a FaultDataset with an array of 21 columns.

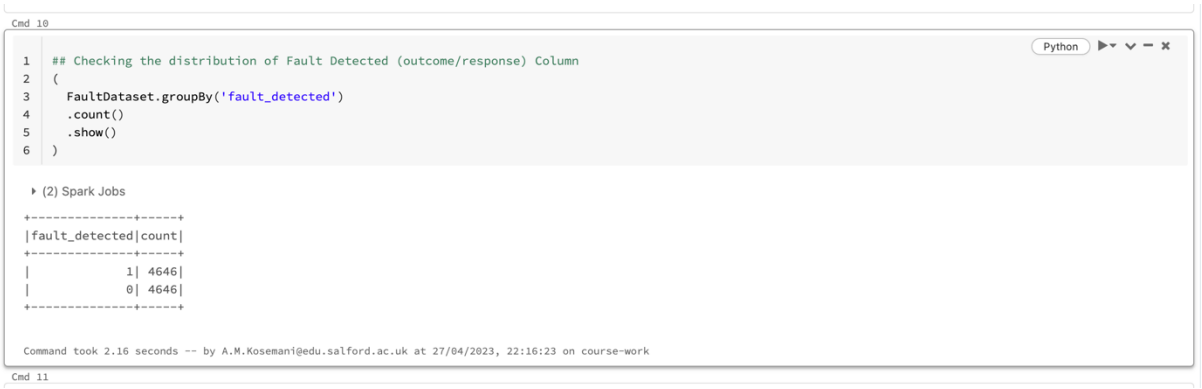


Fig 1.6 Distribution of response variable

In order to determine a pattern for distribution of response variables, a univariate analysis was carried out with the **fault_detection** column as this factor is very important in determining suitable modelling techniques that can give accurate predictions. The response variable for this task has two classes in which observations are categorized, which are “0” and “1”. The data frame was grouped by all columns using the groupBy method, and a pivot table is made with the fault_detected column. The resulting data file **FaultData** shows each unique value separately, including a count of the number of times it has occurred in an **error_detected** column.

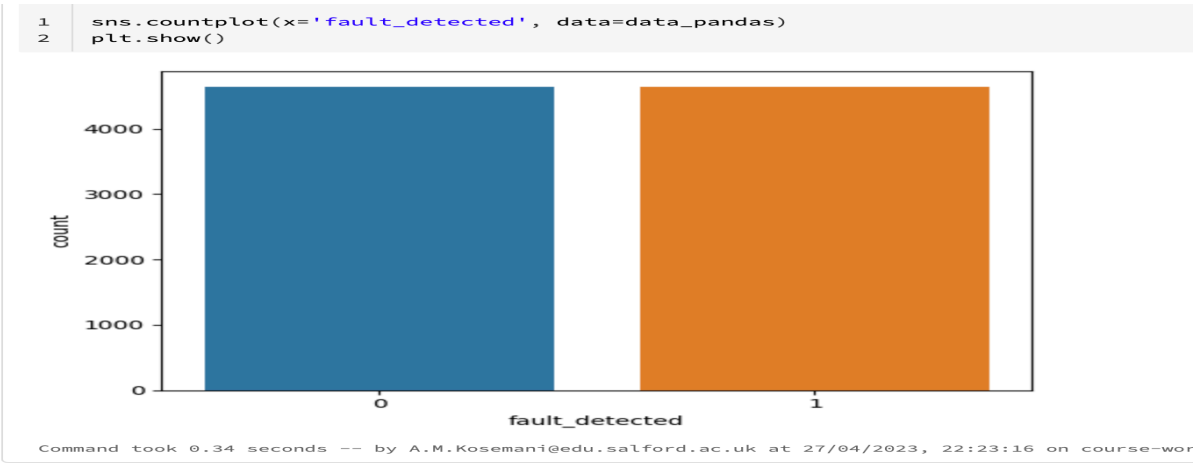


Fig 1.7 To visualize the above result, a library called Seaborn was used.

Cmd 12

Descriptive Statistics

Markdown

Cmd 13

```
1  display(FaultDataset.drop('fault_detected').describe())
```

▸ (2) Spark Jobs

Table +

	summary	1	2	3	4	5	6
1	count	9292	9292	9292	9292	9292	9292
2	mean	0.34162330499354226	0.34263116121394677	0.3421213812957383	0.34213907124407966	0.3428434405940584	0.342827936665949
3	stddev	0.28919489486260785	0.2890875372793958	0.28916422490616933	0.28916356333107296	0.2889646554403878	0.289088989972954
4	min	0.024375	0.024375	0.024375	0.024375	0.024375	0.024375
5	max	1.0809375	1.2134375	1.0809375	1.0809375	1.0809375	1.0809375

5 rows | 5.20 seconds runtime — Refreshed 11 minutes ago

Command took 5.20 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 28/04/2023, 12:40:08 on coursework-cluster

Cmd 14

Fig 1.8 Descriptive statistics

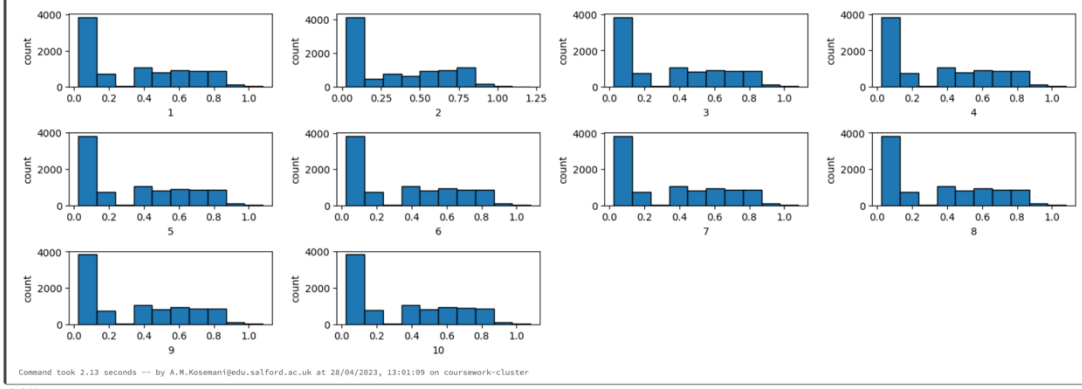
Another important analysis performed on EDA in the task is the descriptive statistics to shed light on the distribution, central tendency, and variability of the Fault Dataset. The mean and the median which reveals the data's basic or essential value were generated using a summary function.

Details regarding the distribution or dispersion of the data from a standard deviation have been revealed, which provide insight into an instrument for measuring variability. The distribution of 25th to 50th, and 75th percentiles is also shown in the data summary.

Cmd 19

```
1  plt.figure(figsize=(15,8))
2
3  for j in range(10):
4    plt.subplot(5, 4, j+1)
5    plt.hist(data_pandas.iloc[:,j], edgecolor='black')
6    plt.xlabel(data_pandas.columns[j], fontsize=10)
7    plt.ylabel('count')
8  plt.tight_layout()
9  plt.show()
```

Python



Command took 2.13 seconds -- by A.M.Kosemani@edu.salford.ac.uk at 28/04/2023, 13:01:09 on coursework-cluster

Cmd 20

Fig 1.9

In **Fig 1.9** A visualization of this summary is presented with histogram and box plot in the screenshot below. histogram gives a frequency distribution for the readings from vibration sensors, showing that there are 10 explanatory variables

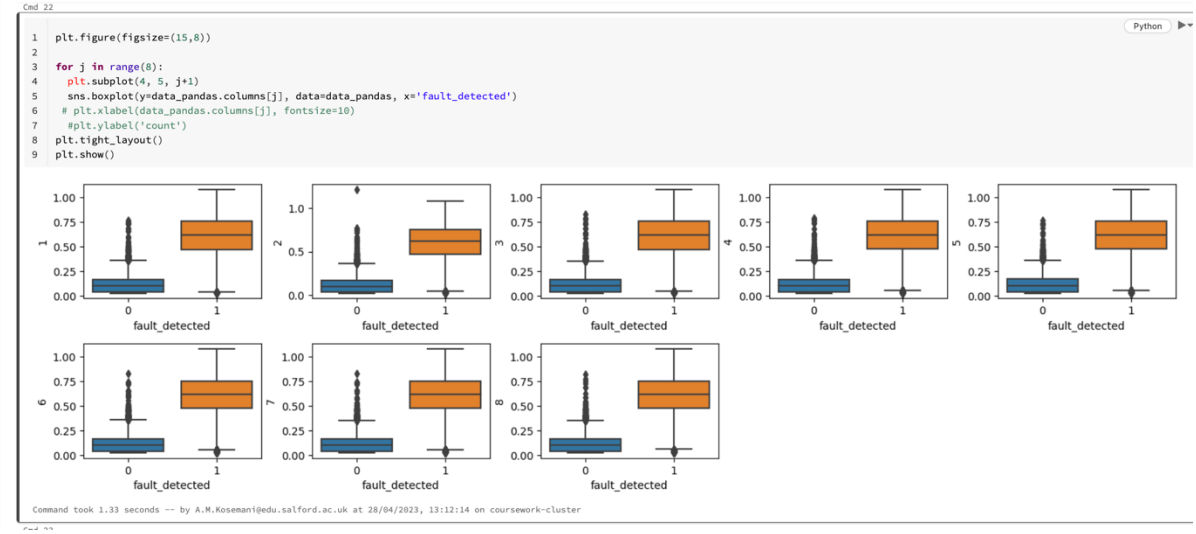


Fig 2.0

A five number summary of FaultDataset, which contains minimum, lower, median, upper quartile and maximum scores, is shown in the box plot. From the diagram above, the above summary was done based on the class of the response variable of either “0” or “1”.

The middle area of the box plot is a median and seems to be positioned in the centre for each set of boxes suggesting normal distribution. Some readings below the lower quartile, which indicate that certain outliers may exist in the data set, can also be seen.

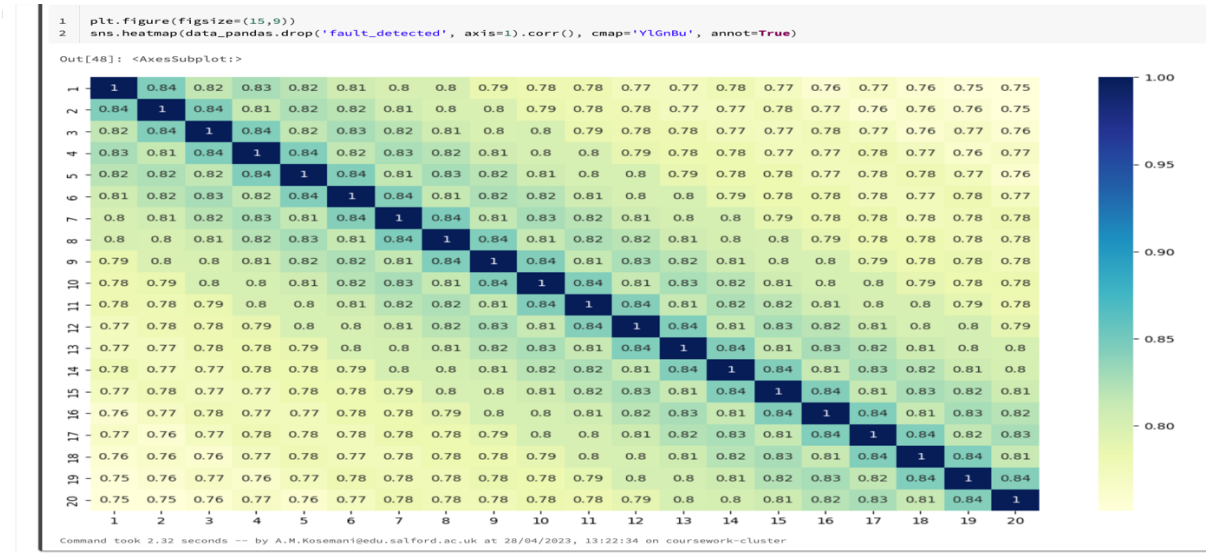


Fig 2.1

This measurement provides details on the relationship of two variables in terms of direction and strength. Correlation coefficients for both positive and negative correlation are found, with 0 of them indicating no correlation.

A correlation matrix was created, in which all explanatory variables are included, and a graph is displayed as a colour coded heatmap showing the very close relation of these data points. Different explanatory variables in the data set also have dark and lighter colours that indicate higher correlation coefficients. The following images show the red to be positive and the blue to be negative correlated.

Fig 2.2 Train-test split

The training data and the testing data for the task has a corresponding number of rows to be 5835 and 3457.

Splitting the dataset into two parts is a first step to pre-processing data after it has been cleaned, namely training data used for machine learning model and testing data that enable models to be tested post training.

As you can see below, a training experiment using the Decision Tree Model was monitored through an ML flow.

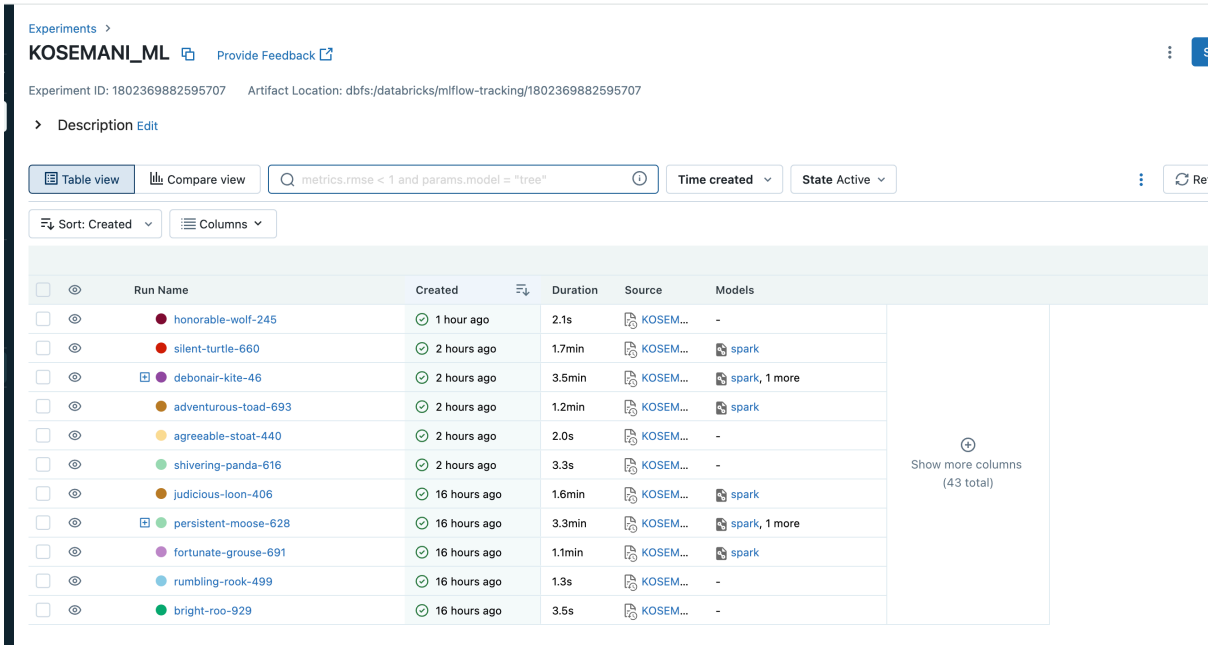


Fig 2.3

Fig 2.3 Evaluating Model on Testing set

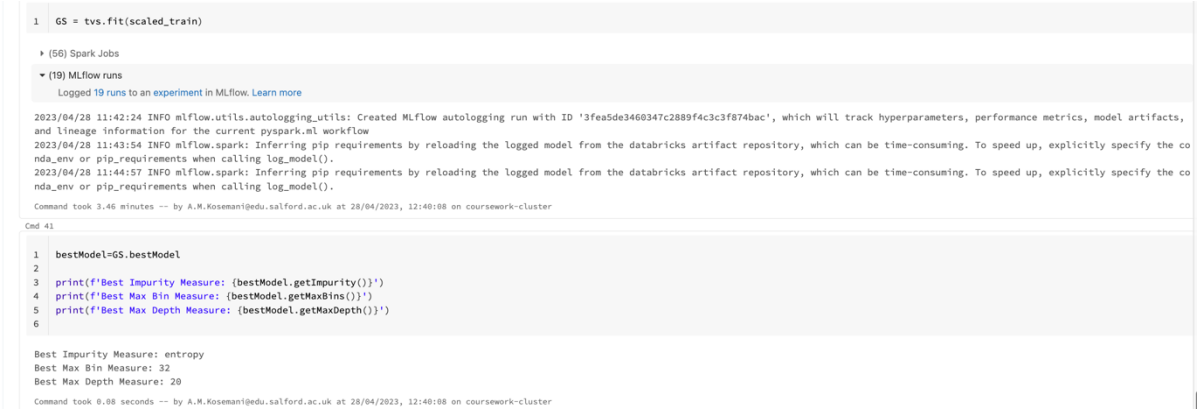


Fig 2.4 Hyperparameter optimization