BIG DATA TOOLS AND TECHNIQUES

TASK 1

DESCRIPTION OF THE SET UP REQUIRED TO COMPLETE THE TASK

OPENING UP THE DATABRICKS WORKSPACE

To start my analysis in Databricks Community Edition, logged into my account and created a cluster. To create cluster, navigate to the "Compute" section on the main page of my Databricks Community Edition account, clicked on the "Create Compute" button and entered a name for the given cluster. The most recent runtime was selected which is "runtime 12.1 (Scala 2.12, Spark 3.3.1)". After creating the cluster, a green circle with a green tick will display next to the cluster name. this means that the cluster has been successfully created.

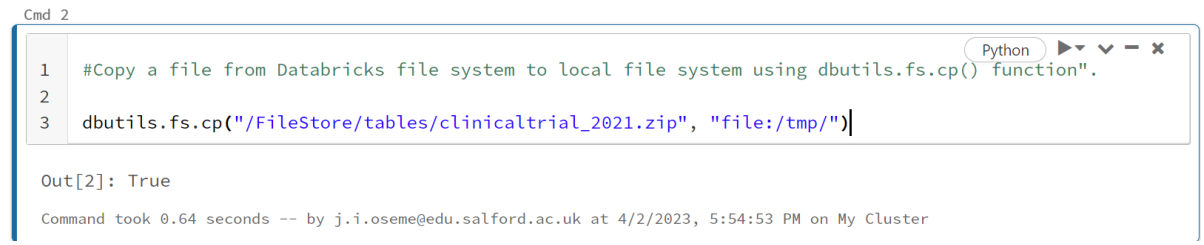
GENERATE A NEW NOTEBOOK IN PYTHON.

Once the cluster has been successfully created, a new notebook in python will be generated. Notebook is a collection of runnable cells which contain your commands. To create a notebook, click the button “Create” and select “Notebook”.

LOAD CLINICALTRIAL_2021 AND PHARMA DATA ON DATABRICKS

Download the clinicaltrial_2021.zip file and pharma.zip file from Blackboard into local machine, logged into the Databricks account and navigated to the home screen labelled as "Data Science & Engineering". This was access on the screen by clicking on the left icon on the navigation bar. Search for “Data Import" column in the middle of the screen and click "Browse files". Select the clinicaltrial_2021 and pharma data.zip file that was downloaded to local system and wait for it to finish uploading. Once the upload is complete, go back to the notebook that was created to check that the file has been uploaded. See screenshot below.

Extract the content of the zip archive using shell command to copy the zip archive file to the local system.



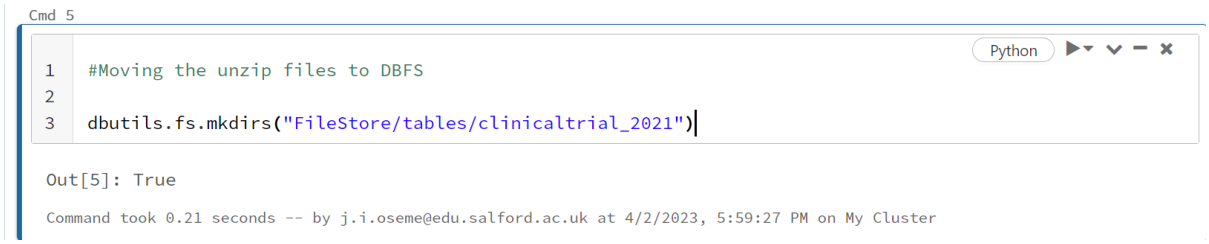
The screenshot above can be used to copy a file from databricks file system to local file system.

After copying the clinicaltrial_2021 file from the Databricks file system, verify if the file has been copied successfully by checking if clinicaltrial_2021. Zip file is present in the /tmp directory.

Confirm the tmp directory to see if clinicaltrial_2021.zip is in it, use the screenshot above.

To extract the contents of the clinicaltrial_2021.zip file and save it to the /tmp directory, use the screenshot below.

The output provided is as a result of running the unzip command on the clinicaltrial_2021.zip file.



After creating the DBFS file, move the content of the local file directory to the newly created DBFS.

Cmd 6

```python
#Move the content of the local file directory into the newly created DBFS
dbutils.fs.mv("file:/tmp/clinicaltrial_2021.csv", "/FileStore/tables/clinicaltrial_2021.csv", True)
```

```
Out[6]: True
```

Command took 2.73 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:00:17 PM on My Cluster

The dbutils.fs.mv method is used to move a file in the Databricks file system (DBFS) from one location to another. the file /tmp/clinicaltrial_2021.csv will move to the file store path at /FileStore/tables/clinicaltrial_2021.csv.

Cmd 7

```python
dbutils.fs.ls('FileStore/tables/clinicaltrial_2021.csv/')
```

```
Out[7]: [FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1680454820000)]
```

Command took 0.15 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:01:32 PM on My Cluster

The screenshot above list the contents of a directory in the Databricks file system (DBFS). The output indicates that the dbutils.fs.ls method successfully listed the contents of the specified path and returned a single FileInfo object corresponding to the file 'clinicaltrial_2021.csv'. The FileInfo object contains information about the file, including its path, name, size, and modification time with their corresponding values as shown in the screenshot above.

introducing pharma.zip file, the dbutils.fs.cp method is used to copy a file from a source DBFS path to a destination local path. In this case, the source path is "/FileStore/tables/pharma.zip" which is a file located in the DBFS file system, and the destination path is "file:/tmp/" which is a local temporary directory.

Cmd 8

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

```
Out[8]: True
```

Command took 0.41 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:02:14 PM on My Cluster

The output shows that dbutils.fs.cp method is “True” indicates the file was copied from the original location to the destination location without any errors. In the screenshot below, this command extracts the contents of a ZIP file located at /tmp/pharma.zip into the /tmp/ directory.

Cmd 9

```
%sh

unzip -d /tmp/ /tmp/pharma.zip
```

```
Archive:  /tmp/pharma.zip
  inflating: /tmp/pharma.csv
```

Command took 0.14 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:02:56 PM on My Cluster

The output indicates that the unzip command successfully extracted the contents of the ZIP file located at /tmp/pharma.zip into the /tmp/ directory.

Use the code below to list the contents of the /tmp/ directory and verify that the pharma.csv file was successfully extracted from the ZIP archive.

Cmd 10

```
%sh

ls /tmp/pharma.csv
```

```
/tmp/pharma.csv
```

Command took 0.09 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:03:38 PM on My Cluster

This output confirms that the pharma.csv file is available in the local file system and extraction operation was successful. A directory in the DBFS file system named "pharma.csv" shall be created.

Cmd 11

```
dbutils.fs.mkdirs("FileStore/tables/pharma.csv")
```

```
Out[11]: True
```

Command took 0.23 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:04:28 PM on My Cluster

The resulting output indicates that dbutils.fs. mkdirs method returned “True” to indicate that the directory was created in the DBFS file system without any errors.

Move the file from a temporary location to a permanent location in the databricks file system. "/FileStore/tables/pharma.csv" is the destination file path.

Cmd 12

```
dbutils.fs.mv("file:/tmp/pharma.csv", "/FileStore/tables/pharma.csv", True)
```

```
Out[12]: True
```

Command took 0.49 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:05:23 PM on My Cluster

The resulting output dbutils.fs.mv function is “True” to indicate that the file move operation was successful.

List the files and directories within a directory named pharma.csv that is in the /FileStore/tables/ directory of the Databricks file system.

Cmd 13

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

```
Out[13]: [FileInfo(path='dbfs:/FileStore/tables/pharma.csv/pharma.csv', name='pharma.csv', size=678999, modificationTime=1680455125000)]
```

Command took 0.15 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:06:19 PM on My Cluster

The output will include information such as the file name, file size, and last modification time with their respective values as seen in the screen shot.

To get the length of the list returned by dbutils.fs.ls (). Use the len () function.

Cmd 14

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

Python

```
Out[14]: 1

Command took 0.16 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:07:23 PM on My Cluster
```

The output 1 indicates that there is one file or directory within the /FileStore/tables/pharma.csv/ directory of the Databricks file system. Therefore, the length of the list is 1, and the output of len(dbutils.fs.ls("FileStore/tables/pharma.csv/")) is also 1.

Install ‘Bokeh’ to provides interactive visualization tools for modern web browsers.

```
1  #installs the Python package "Bokeh" using the pip package installer.
2  %pip install bokeh
```

```
Python interpreter will be restarted.
Collecting bokeh
  Downloading bokeh-3.1.0-py3-none-any.whl (8.3 MB)
Collecting contourpy>=1
  Downloading contourpy-1.0.7-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (299 kB)
Collecting PyYAML>=3.10
  Downloading PyYAML-6.0-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (661 kB)
Requirement already satisfied: Jinja2>=2.9 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (2.11.3)
Collecting xyzservices>=2021.09.1
  Downloading xyzservices-2023.2.0-py3-none-any.whl (55 kB)
Requirement already satisfied: tornado>=5.1 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (6.1)
Requirement already satisfied: numpy>=1.16 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (1.21.5)
Requirement already satisfied: pandas>=1.2 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (1.4.2)
Requirement already satisfied: pillow>=7.1.0 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (9.0.1)
Requirement already satisfied: packaging>=16.8 in /databricks/python3/lib/python3.9/site-packages (from bokeh) (21.3)
Requirement already satisfied: MarkupSafe>=0.23 in /databricks/python3/lib/python3.9/site-packages (from Jinja2>=2.9->bokeh) (2.0.
1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /databricks/python3/lib/python3.9/site-packages (from packaging>=16.8->b
okeh) (3.0.4)
Requirement already satisfied: python-dateutil>=2.8.1 in /databricks/python3/lib/python3.9/site-packages (from pandas>=1.2->bokeh)
(2.8.2)
```

1

```
Command took 9.07 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:12:18 PM on My Cluster
```

Cmd 16

```
1  # Setting clinicaltrialdata = 2021
2
3  clinicaldata = 2021
```

Python

```
Command took 0.07 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:14:07 PM on My Cluster
```

From the screenshot above, assign the variable ‘clinicaldata’ to the integer value 2021.

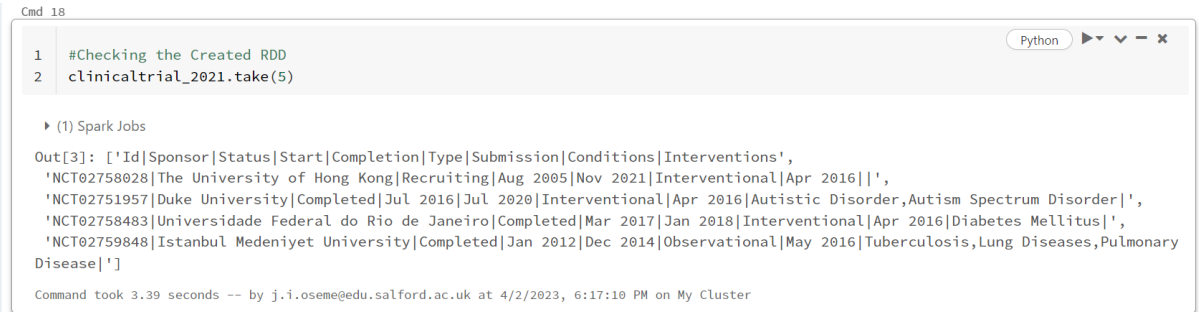
Using the screenshot below, read a file named clinicaltrial_2021.csv, where the year 2021 is obtained from the previously defined clinicaldata variable.

Cmd 17

```
1  # Creating a Resilient Distributed Dataset (RDD) by loading a CSV file and storing it in a newly created variable initialized
   as empty.
2
3  clinicaltrial_2021 = sc.textFile("/FileStore/tables/clinicaltrial_"+str(clinicaldata)+".csv")
4
5  pharma = sc.textFile("/FileStore/tables/pharma.csv")
```

```
Command took 0.20 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:14:48 PM on My Cluster
```

The screenshot below shows clinicaltrial_2021 is being used to retrieve the first 5 elements from the clinicaltrial_2021 RDD.



The resulting first five elements are shown in the output above.

Question 1

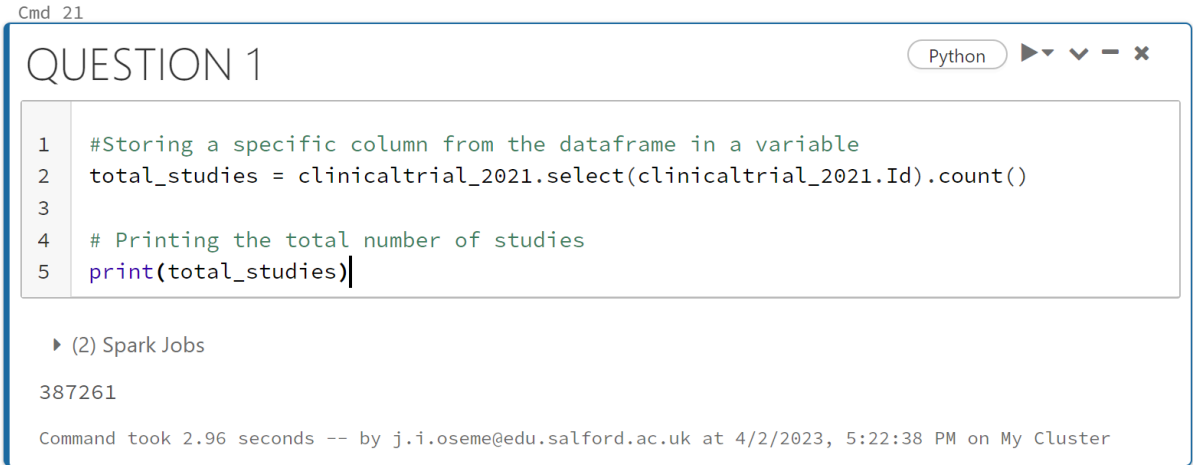
ASSUMPTIONS MADE ON THE DATASET.

The dataset contains a clinicaltrial_2021 dataset. The code removes the first column of the dataset and checks the number of distinct studies available in the remaining columns. It is also assumed that each row represents a unique clinical trial study where they are no duplicates that could affect the accuracy of the result.

PySpark implementation outline in RDD

The first step in the data preprocessing process was to remove the first column of the clinicaltrial_2021 dataset which is assumed to be the ID column, and return a new dataset called Remove_First_Column. Next, the number of distinct rows in the Remove_First_Column dataset was counted using the distinct () and count () methods. This step helps to identify any duplicate rows in the data. The output indicates that the Remove_First_Column dataset contains 387261 distinct rows after removing the ID column.

PySpark implementation outline in DataFrame



Selects the Id column from the clinicaltrial_2021 dataframe. The selected column is stored in a variable named total_studies. The code uses the count () function to count the total number of studies in the Id column and prints the result using the print () function.

SQL implementation outline

Selects the id column from the clinicaltrial_2021 table and applies the keyword “DISTINCT” to count the number of unique studies. A column named Number_of_Studies represents the total number of distinct studies in the table.

Discussion of result

The results obtained for RDD, data frame, and SQL in 2021, 2020 and 2019 are 387261, 356466, 326348. Comparing the different results obtained for RDD, data frame, and SQL in each year, indicates that they are consistent and yield the same number of distinct studies. This result shows that there is an increase in the number

of distinct studies from 2019-2021. For instance, there is a rise of around 30000 studies between 2019 and 2020 and another surge of about 30000 studies between 2020 and 2021. That is to say that there was a 9.24% increase in the number of distinct studies between 2019 and 2020 and 8.64% increase between 2020 and 2021. These percentages indicate a slowing in the rise of the number of separate studies, as the percentage increase is lower from 2020 to 2021 compared to 2019 to 2020. However, the trend suggests that the number of clinical trial studies is increasing over time. In conclusion, the comparison of the results obtained for RDD, data frame, and SQL suggests that they are reliable and produce consistent results.

Question 2

ASSUMPTIONS MADE ON THE DATASET.

The assumptions made is that it contains information that has to do with clinical trials that are being researched. It can also be assumed that the dataset has a column that identifies the type of each study. The assumption is that the values in this column are classified based on the study types, and that the categories can be counted to identify the most and least frequent types of studies in the dataset.

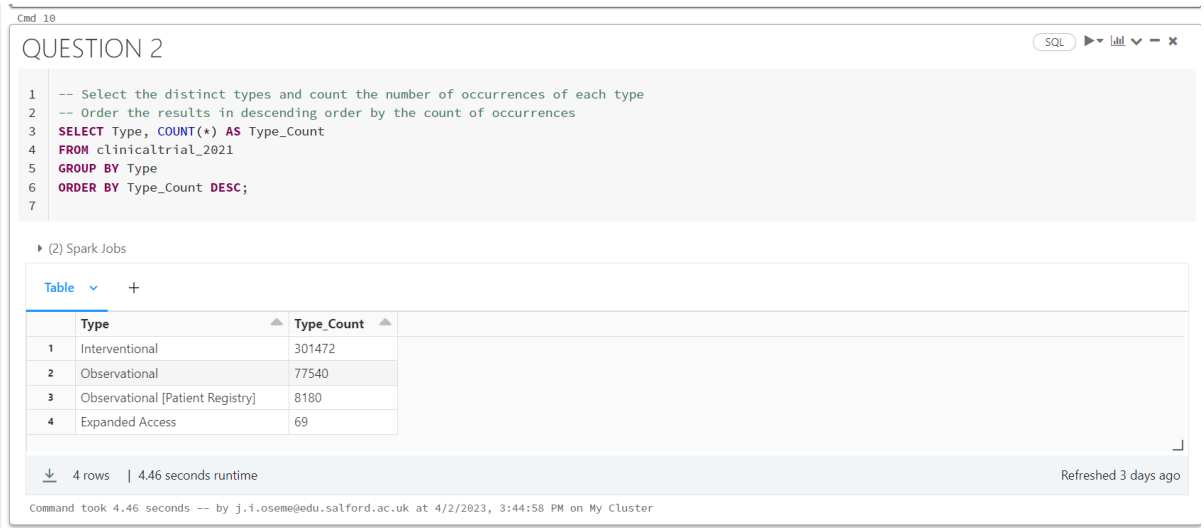
PySpark implementation outline in RDD

The description of the main ideas is to retrieve the different study types from the dataset. The initial column of the dataset will be removed and then retrieves the study types by splitting the data on “|". An RDD of the study types is created. The frequency of each study types is counted by a lambda function. The resulting output shows the study types and their corresponding frequencies, sorted from most frequent to least frequent.

PySpark implementation outline in DataFrame

Using the screenshot above, the code will extract, group, and count the number of instances of each "Type" in the "clinicaltrial_2021" dataframe. The output displays the result in descending order based on their frequency of occurrence.

SQL implementation outline



From the screenshot above, the query categorises the different trials base on their type, count the number of occurrences and then order the results in descending order using the ORDER BY clause. The result will provide a summary of the number of clinical trials for each distinct type in the dataset. The main description of the screenshot above is to extract the different types of clinical trial available in the dataset and count the number of occurrences.

Discussion of result

The results obtained across the various years in 2019, 2020, and 2021 for the different types of clinical trials carried out using RDD, DataFrame, and SQL shows that the number of interventional trials is constantly the highest, followed by observational trials. Now, in 2021, the number of interventional trials conducted using RDD, DataFrame, and SQL was 301,472, while the number of observational trials was 77,540. The number of observational [patient registry] trials was 8,180, and the number of expanded access trials was 69. Comparing the different results obtained across the years, we can observe that the number of clinical trials conducted in 2021 is far higher than those conducted in 2020 and 2019. The number of interventional and observational trials has increased from 277,631 and 64,163, respectively, in 2020 to 301,472 and 77,540, respectively, in 2021. Similarly, the number of “observational [patient registry]” trials has increase from 7,332 in 2020 to 8,180 in 2021. This also indicates that in 2021 77.8% of clinical trials were classified as “interventional”, 20% were categorised as observational, 2.1% as “Observational [Patient Registry]", and 0.02% as "Expanded Access". Overall, the results indicate that the number of clinical trials carried out over the years has constantly been on increase, and interventional trials remain the most common type of trial.

Question 3

ASSUMPTIONS MADE ON THE DATASET.

It is assumed that there are top 5 conditions for which the clinical trials were conducted in the given year. The dataset provides a count of occurrences of each condition, sorted in descending order. The assumption can be made that the clinical trials were conducted based on these conditions as a result of their high prevalence.

PySpark implementation outline in RDD

Cmd 21

QUESTION NUMBER 3

Python

```
#Getting the previous Dropped column and splitting it using lambda function
Splitting = Remove_First_Column.map(lambda line: line.split("|"))

#Getting only the Condition columns
Selecting_Conditions = Splitting.map(lambda line: line[7])

#Splitting it with ,
Splitting_Conditions = Selecting_Conditions.flatMap(lambda line: line.split(","))

#Adding a value 1 to all the values and using reduce by key function getting the count finally sorting it by descending order
Counting_Top_Values = Splitting_Conditions.map(lambda x: (x,1)).reduceByKey(lambda x, y: x+y).sortBy(lambda x: x[1],False)

#Getting the First Row
First_Element = Counting_Top_Values.first()

#Ignoring the First Row
Removing_First_element = Counting_Top_Values.filter(lambda x: x != First_Element)

#Taking only 5 elements
Removing_First_element.take(5)
```

▸ (4) Spark Jobs

```
Out[16]: [('Carcinoma', 13389),
 ('Diabetes Mellitus', 11080),
 ('Neoplasms', 9371),
 ('Breast Neoplasms', 8640),
 ('Syndrome', 8032)]
```

Command took 3.76 seconds -- by j.i.oseme@edu.salford.ac.uk at 4/2/2023, 6:28:11 PM on My Cluster

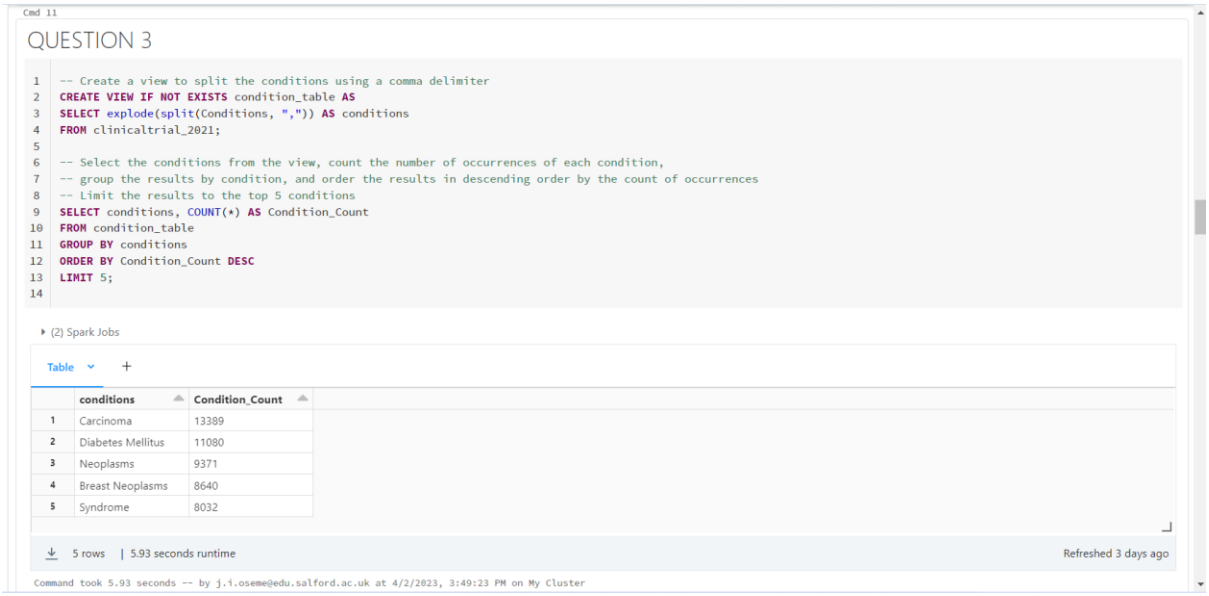
Data analysis is carried out clinicaltrial_2021 dataset. This is done by using a lambda function to extract the column that was initially dropped into distinct columns. After

that, split the condition column using commas by selecting only the condition column from the dataset. A single RDD values are flattened. A value of 1 is added to each value in the RDD and count each value using a reduceByKey () function. Results are sorted in descending order by count. The first row is then removed, and the top 5 values are selected.

PySpark implementation outline in DataFrame

The main description of this code is to analyze the conditions of the clinical trials dataset in order to determine the most shared conditions. The codes import the necessary libraries functions and uses the explode function to separate the “conditions columns by commas. The split condition column are groups and the result of the top 5 are displayed.

SQL implementation outline



The above code uses SQL queries to extract and count the occurrences of the top 5 conditions present in the "Conditions" column of the clinical trial dataset. The SQL script creates a view called “condition_table”, splits the "Conditions" column using a comma delimiter. Select the conditions using the view and count the number of occurrences of each condition. The results are grouped by condition and ordered in descending order by the count of occurrences. The results are limited to the top 5 conditions.

Discussion of Results

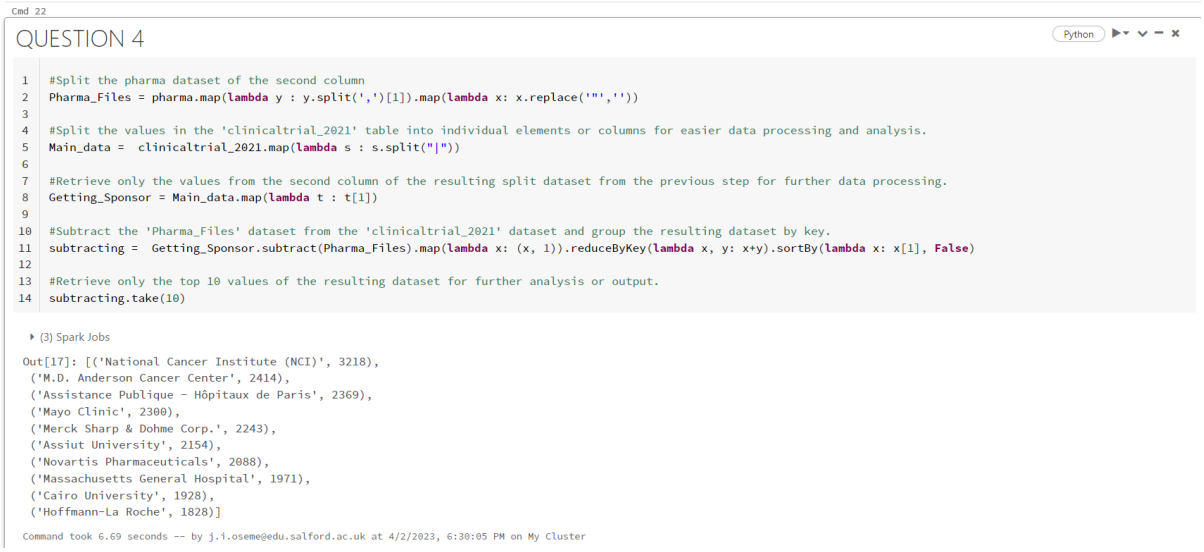
The result output shows that from 2019 to 2021, the top 5 medical conditions using different approaches (Rdd, Dataframe and SQL) have been consistent. Carcinoma and Diabetes Mellitus happens to be the two major prevalent conditions across all three methods. For instance, in 2021, Carcinoma has 12.9%, Diabetes Mellitus with 10.6%, Neoplasms is 9.0%, Breast Neoplasms with 8.3% and Syndrome with 7.7%. In 2020, Carcinoma has 13.4%, Diabetes Mellitus with 11.4%, Neoplasms is 9.4%, Breast Neoplasms has 20.9%, and Syndrome with 8.7%. In 2019, Carcinoma has 14.1%, Diabetes Mellitus with 12.4%, Neoplasms had 9.1%, Breast Neoplasms with 10.6%, Syndrome with 8.3%. It is obvious that the percentage of medical conditions vary by each year. For example, around 48.5% were caused by the top five conditions, with Carcinoma taking the lead. In 2020, there was a percentage increase of about 63.8% of the top conditions carried out, with Breast Neoplasms accounting for the most. In 2019, the top 5 conditions accounted for around 54.5% of all conditions conducted. Carcinoma was the most frequently mentioned. These changes could be due to the increasing number of clinical trials being conducted.

Question 4

ASSUMPTIONS MADE ON THE DATASET

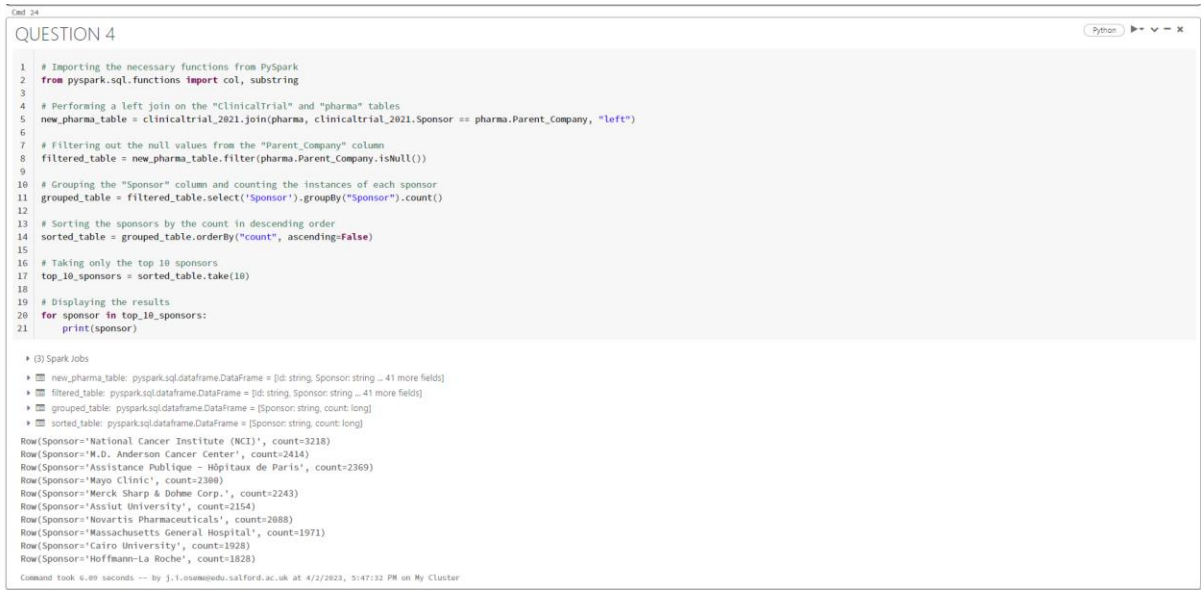
It is assumed that the clinical_trial 2021 table has a column name “sponsor” which consists of the sponsors and clinical trials information. Also, another assumption is that the “pharma table” has a column named parent_company containing information about the parent company of the pharmaceutical company sponsoring the clinical trials. The sponsors are sorted in descending order by their count, assuming that the most frequent sponsors are of interest. Only the top 10 sponsors are selected for display, assuming that displaying more than 10 results would not provide additional value.

PySpark implementation outline in RDD



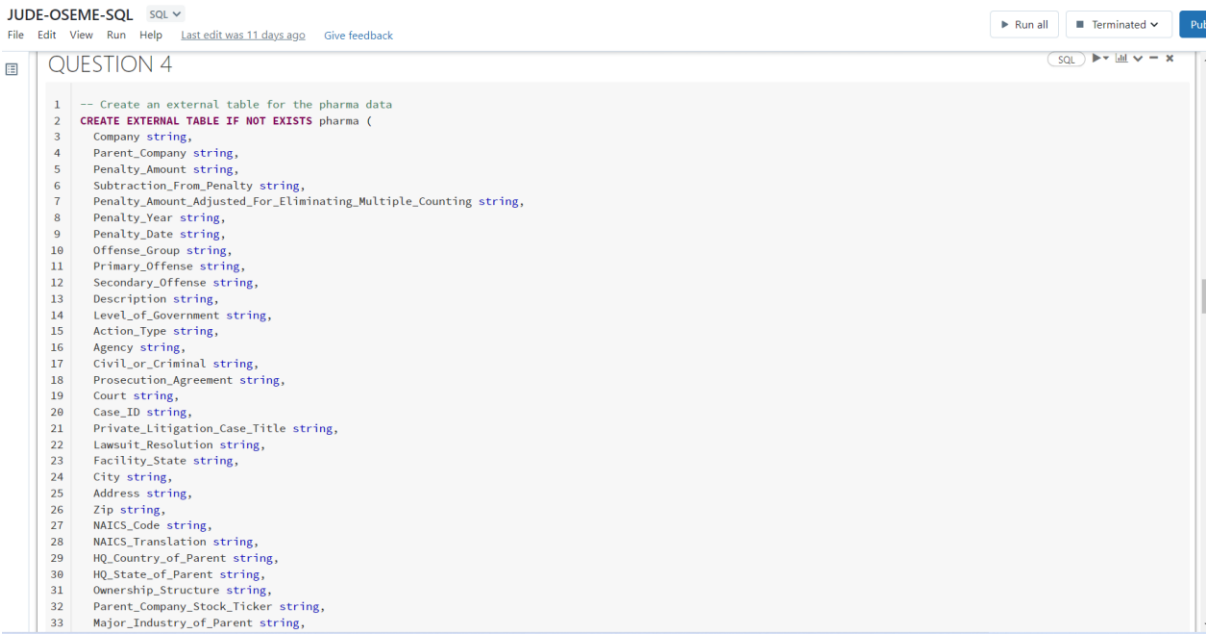
The main description of the screenshot above involves processing two dataset "pharma" and "clinicaltrial_2021" in which key information’s are being retrieved to find the differences between them. Split the first and second column of the pharma dataset as well as the clinicaltrial_2021 dataset into individual columns. Extract the second columns to create the "Getting_Sponsor" dataset. The above code calculates the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored. To achieve this, a list of pharmaceutical companies is created by extracting the names of sponsors from a separate dataset 'pharma'. Next, the list of sponsors from the 'clinicaltrial_2021' dataset is obtained, and the names of pharmaceutical companies are subtracted from this list. The list is sorted in descending order based on the frequency count, and the top 10 entries are displayed using the 'take' method. The resulting output shows the name of the sponsor and the number of clinical trials they have sponsored.

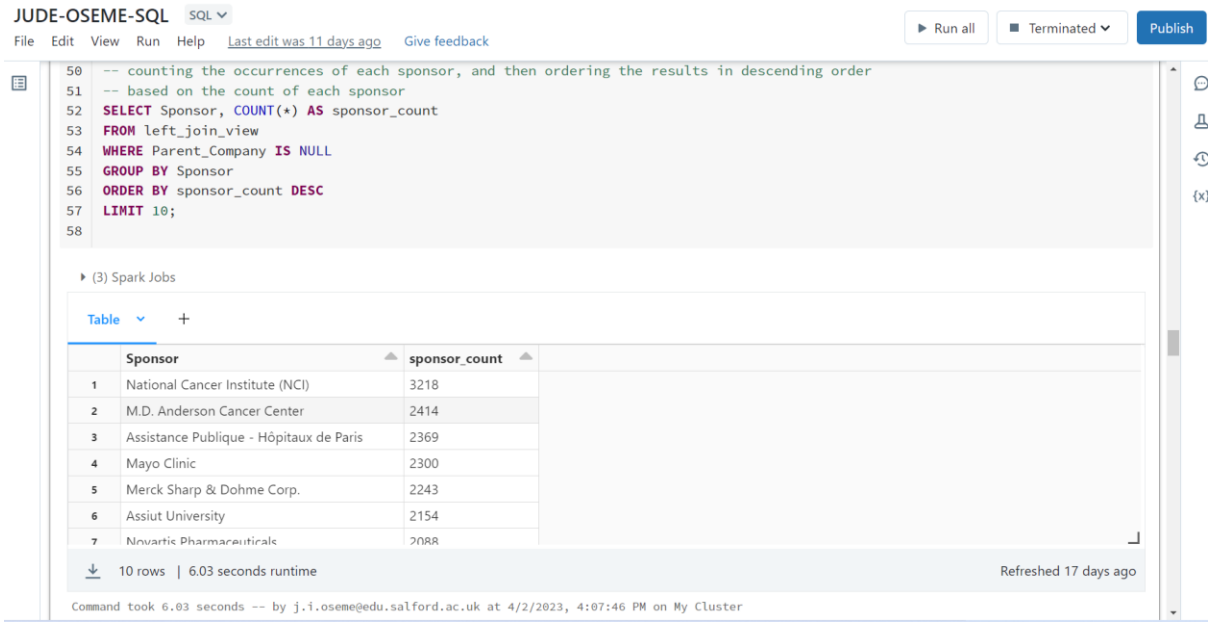
PySpark implementation outline in DataFrame



In the screenshots above, the code imports necessary functions from the PySpark SQL library. A new table is created by joining the clinicaltrial_2021 and pharma tables on the Sponsor and Parent_Company columns respectively, using a left join. The resulting table is sorted in descending order by the count column, and the top 10 sponsors are selected. The resulting output shows the top 10 sponsors who are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

SQL implementation outline





The SQL statements create an external table for the pharma data and a view by left joining the clinical trial table with the pharma table on a condition. It then selects the top 10 sponsors from the left join view by grouping them by sponsor and counting the occurrences of each sponsor, and then ordering the results in descending order based on the count of each sponsor. The resulting query returns the top 10 sponsors who are not part of a parent company in the pharma data.

Discussion of result

The results obtained in different years using different approaches (Rdd, Dataframe, SQL) shows that the top sponsors have been consistent over the years. National Cancer Institute (NCI) and M.D. Anderson Cancer Center have constantly topped the

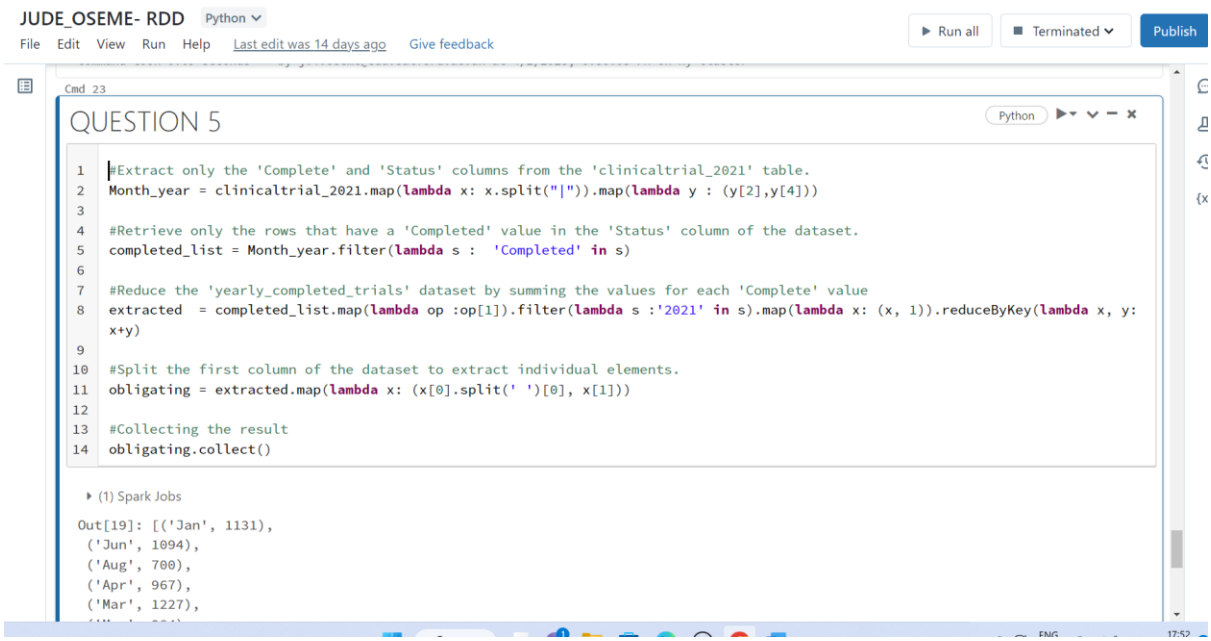
chart. However, some changes were observed in 2021 in the ranking of top sponsors. For example, in 2021, Assistance Publique - Hôpitaux de Paris ranked third, whereas in 2020 and 2019, it ranked fifth and sixth, respectively. This suggests that there might be fluctuations in the amount of funding received by different sponsors each year. The results obtained through the different methods are in line with each other This is because the top sponsors and their funding amounts are similar across all three methods for each year. This is a proof that the different methods of data processing and analysis can yield consistent results. However, there are some changes in the rankings of other sponsors. For example, in 2021, Assistance Publique - Hôpitaux de Paris ranked third, whereas in 2020 and 2019, it ranked fifth and sixth, respectively. This suggests that there might be fluctuations in the amount of funding received by different sponsors each year. By comparing the results obtained through RDD, DataFrame, and SQL, we can see that these methods are reliable and can be used interchangeably to analyze large datasets.

Question 5

ASSUMPTIONS MADE ON THE DATASET

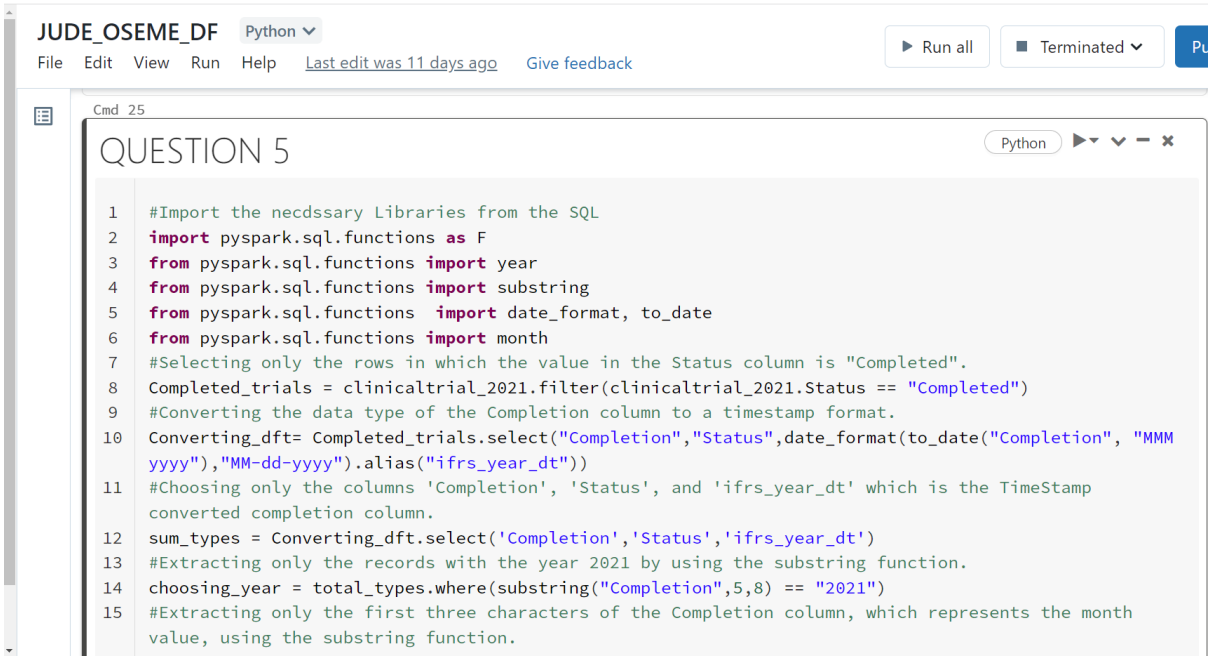
The dataset is assumed to have a particular structure with columns for 'Complete' and 'Status', which filters and extracts important information. It is also assumed that the data in the 'Complete' column represents the number of completed clinical trials and the data in the 'Status' column indicates the status of each trial. The analysis assumes that the data is accurate and complete.

PySpark implementation outline in RDD



The code performs various operations to extract the number of completed clinical trials in 2021 grouped by the month of completion. The code uses PySpark functions such as map, filter, reduceByKey, split, and collect to achieve the desired output. The resulting dataset can be useful for analyzing the distribution of completed clinical trials over time and identifying any trends. to inform decision-making regarding future research and development efforts.

PySpark implementation outline in DataFrame



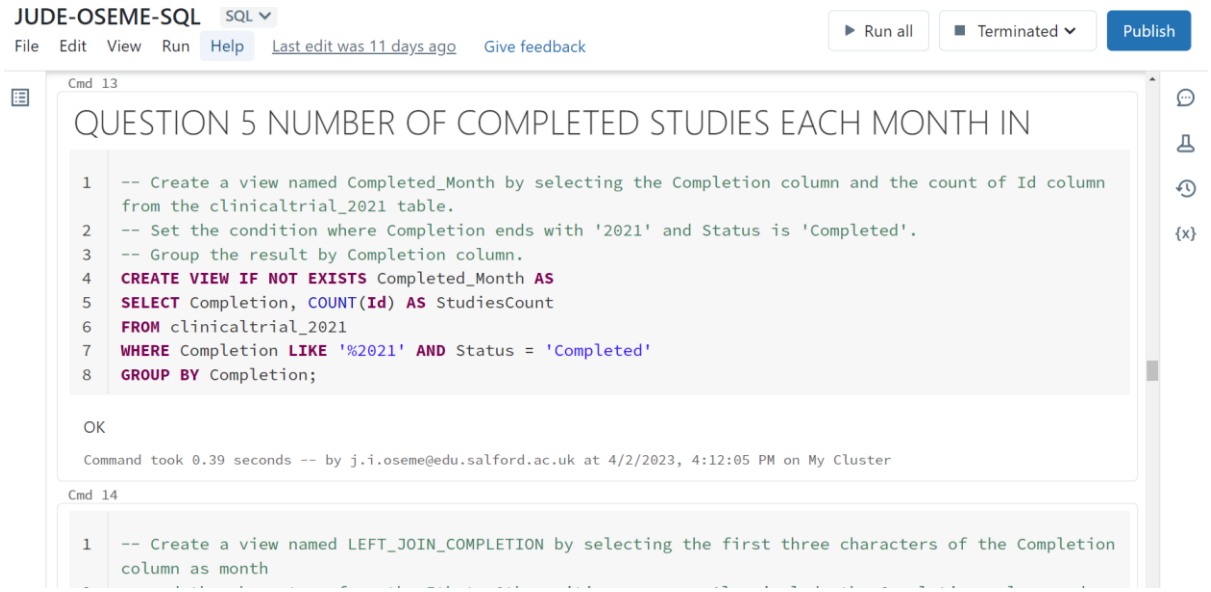

```python
#Import the necdssary Libraries from the SQL
import pyspark.sql.functions as F
from pyspark.sql.functions import year
from pyspark.sql.functions import substring
from pyspark.sql.functions  import date_format, to_date
from pyspark.sql.functions import month
#Selecting only the rows in which the value in the Status column is "Completed".
Completed_trials = clinicaltrial_2021.filter(clinicaltrial_2021.Status == "Completed")
#Converting the data type of the Completion column to a timestamp format.
Converting_dft= Completed_trials.select("Completion","Status",date_format(to_date("Completion", "MMM yyyy"),"MM-dd-yyyy").alias("ifrs_year_dt"))
#Choosing only the columns 'Completion', 'Status', and 'ifrs_year_dt' which is the TimeStamp converted completion column.
sum_types = Converting_dft.select('Completion','Status','ifrs_year_dt')
#Extracting only the records with the year 2021 by using the substring function.
choosing_year = total_types.where(substring("Completion",5,8) == "2021")
#Extracting only the first three characters of the Completion column, which represents the month value, using the substring function.
```

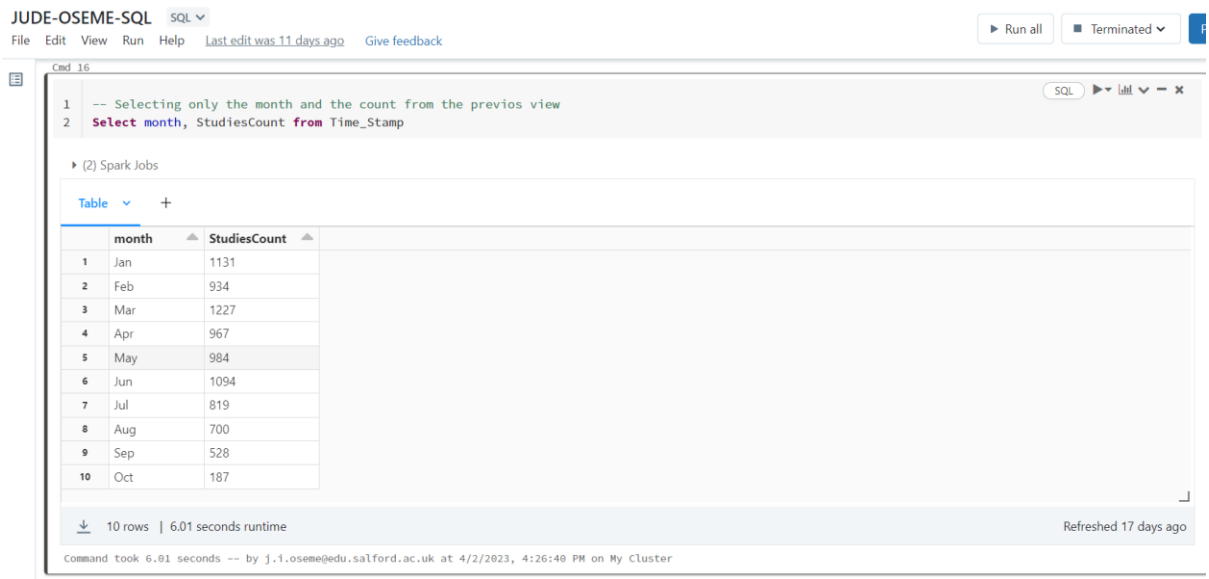

```python
value, using the substring function.
choosing_month_date =
choosing_year.select(substring("Completion",1,3).alias("month"),"Status","ifrs_year_dt")
#Retrieving the month and the Completetion column converted to TimeStamp.
Grouping = choosing_month_date.select("month","ifrs_year_dt")
#Counting the occurrences and sorting them in ascending order based on the Timestamp Converted Completion Column with the corresponding count.
Grouping_1 = Grouping.groupBy("month","ifrs_year_dt").count().orderBy("ifrs_year_dt",ascending = True)
#Next, we extract and keep only the month column and the newly created count column.
Listing = Grouping_1.select("month","count")
#Finally, the data is presented in a monthly format for better understanding and visualization.
Listing.show()
```

```
(2) Spark Jobs
Completed_trials:  pyspark.sql.dataframe.DataFrame = [Id: string, Sponsor: string ... 7 more fields]
Converting_dft:  pyspark.sql.dataframe.DataFrame = [Completion: string, Status: string ... 1 more field]
sum_types:  pyspark.sql.dataframe.DataFrame = [Completion: string, Status: string ... 1 more field]
choosing_year:  pyspark.sql.dataframe.DataFrame = [Completion: string, Status: string ... 1 more field]
choosing_month_date:  pyspark.sql.dataframe.DataFrame = [month: string, Status: string ... 1 more field]
Grouping:  pyspark.sql.dataframe.DataFrame = [month: string, ifrs_year_dt: string]
```

The above screenshots codes extract and analyze data from the 'clinicaltrial_2021' table. The main idea is to extract only the completed trials from the table and convert the 'Completion' column to a timestamp format. Then, filter the data to retrieve only the completed trials in 2021 and extract the month value from the 'Completion' column. After that, the data is grouped by month and year and counted based on the timestamp converted completion column. The result is presented in a monthly format for better understanding and visualization.

SQL implementation outline





In the above screenshots, Time_Stamp that contains two columns, month and StudiesCount. The completion column is selected by the completed_month view and also counts the number of id_columns for the completed trials that ended with the year 2021. The result is grouped by the Completion column. The LEFT_JOIN_COMPLETION view selects the first three characters of the Completion column as month and the characters from the 5th to 8th position as year. It also includes the Completion column and the StudiesCount column from the Completed_Month view. The Time_Stamp view is created to convert the month and year columns to a timestamp format and order the data based on the Date_col column. Finally, the data is extracted from the Time_Stamp view to include only the month and StudiesCount columns.

Discussion of result

The result obtained using RDD, Dataframe and SQL varies for each month. For example, the lowest number of studies counts occurred in October in 2021. While the highest number took place in December 2020. This is an indication that the counts for various month vary for each year. However, the SQL method provides the results in a tabular format, which may be more convenient for certain data analysis tasks. The RDD method provides results in a list format, while the Dataframe method presents results in a more structured and organized manner. Comparing the results obtained using RDD, Dataframe, and SQL, we can see that the counts for each month are constantly the same across all three methods, indicating that all three methods are reliable and consistent in producing accurate results.

FURTHER ANALYSIS

ASSUMPTIONS MADE ON THE DATASET

The assumption made on the dataset is that it contains information on clinical trials conducted in the year 2021, and the data is complete and accurate. The query filters the dataset to only include trials that have a "Suspended" status, and then groups them by the sponsor's name. Finally, it counts the number of trials for each sponsor and lists the top 10 sponsors with the highest count of suspended trials. This information could be useful in understanding which sponsors have the highest number of suspended trials, which could be due to various reasons such as safety concerns, lack of funding, or regulatory issues.

SQL implementation outline

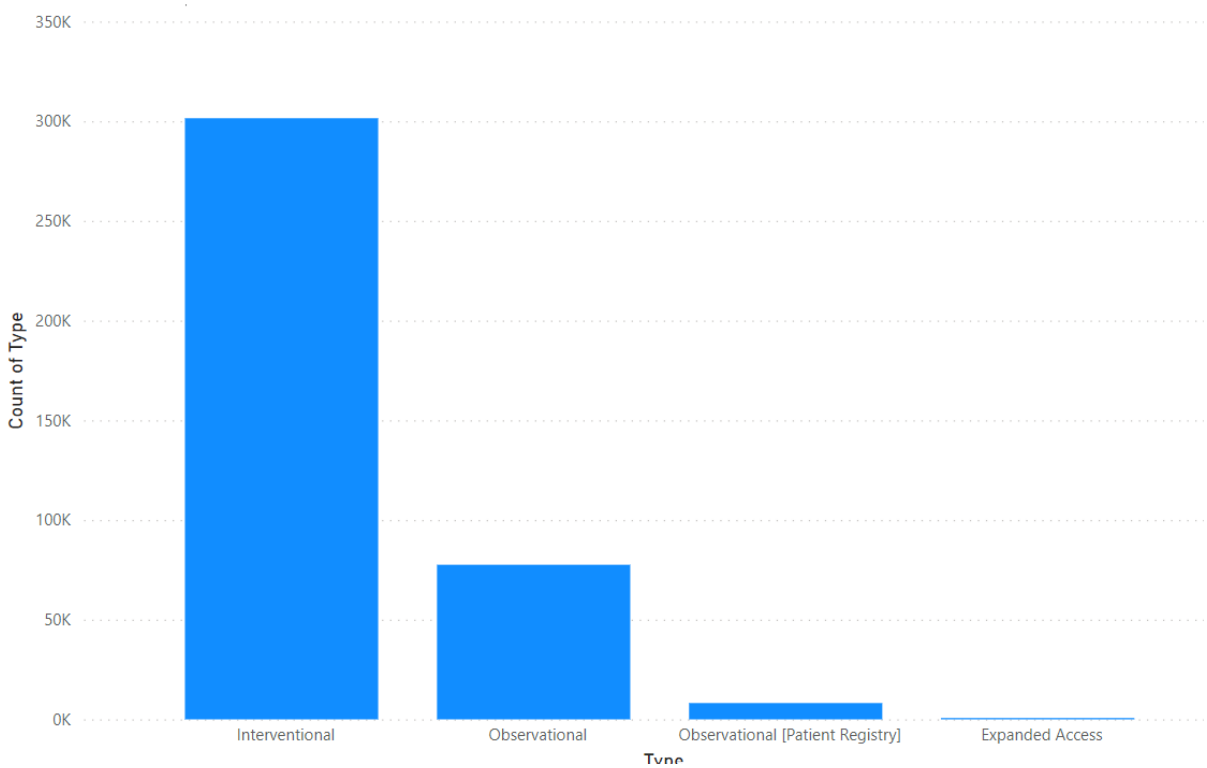
This SQL query retrieves the top 10 sponsors who have the highest count of clinical trials that are currently in "Suspended" status. It filters the data from the "clinicaltrial_2021" table and selects only those records where the "Status" column is "Suspended". Then, it groups the data by "Sponsor" and counts the number of occurrences of "Suspended" status for each sponsor using the "count ()" function. Finally, it orders the results in descending order based on the count and limits the output to the top 10 sponsors with the highest count using the "LIMIT" clause.

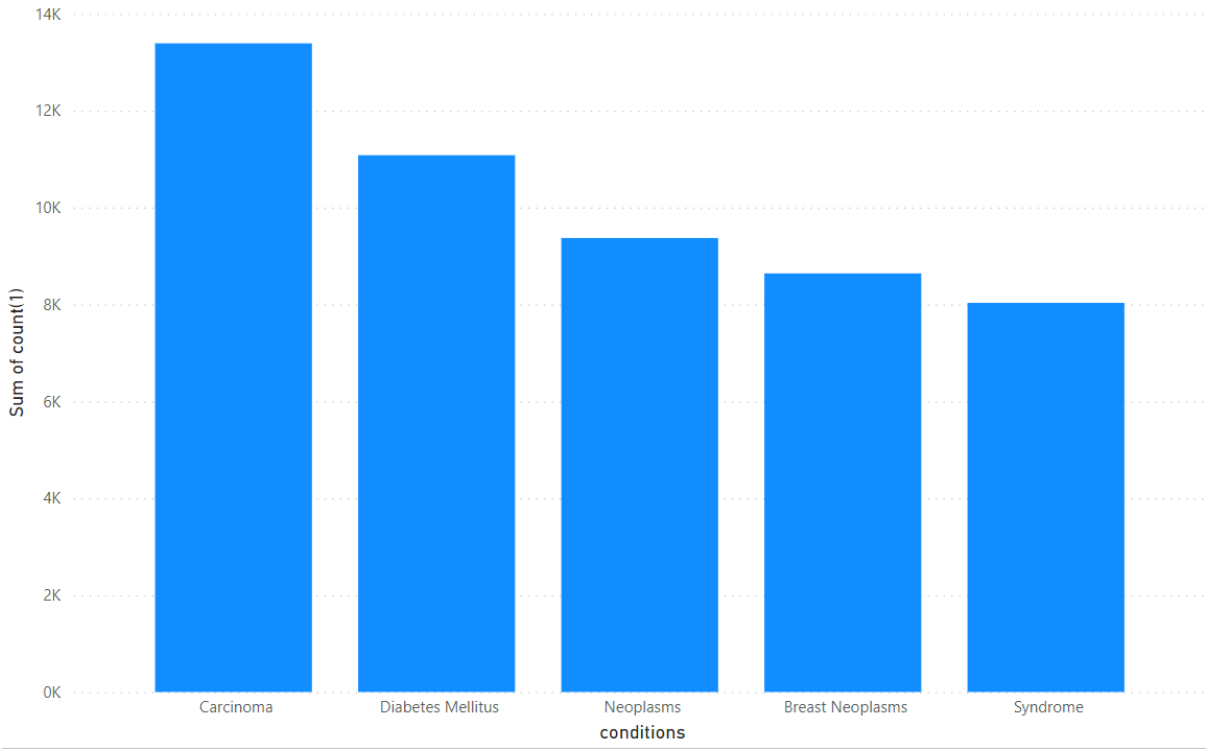
Discussion of result

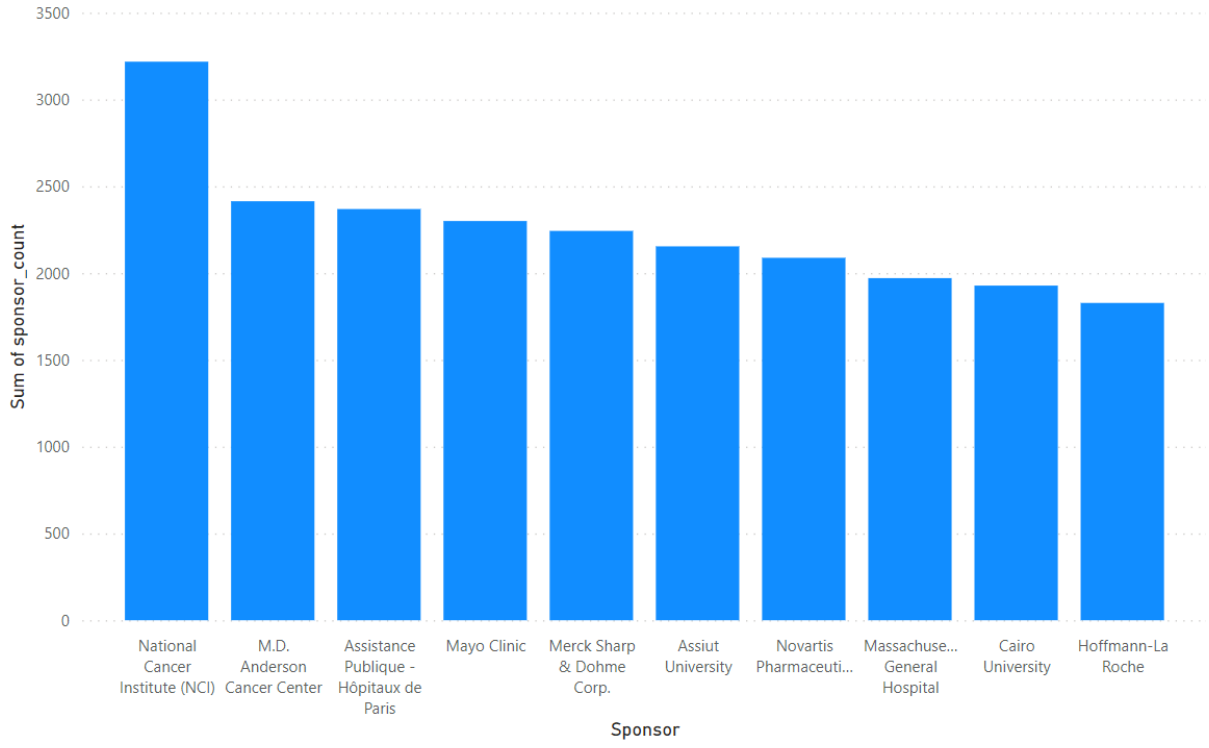
Based on the results, we can see that there are several sponsors who have a high count of suspended trials. This information could be useful for stakeholders in the

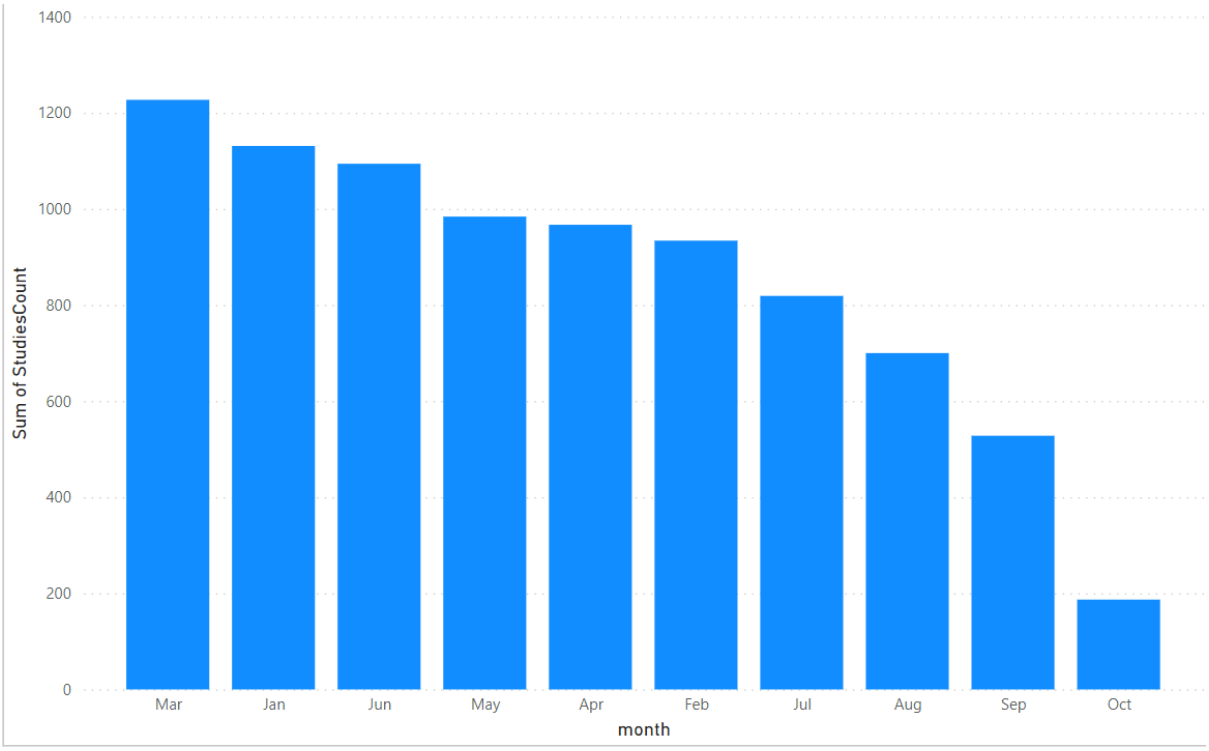
pharmaceutical and healthcare industries, as it could help them make more informed decisions about partnerships or investments in clinical trials. Further analysis of the reasons for suspension and the characteristics of the trials could provide additional insights into the challenges and opportunities in the clinical trials space.

POWER BI VISUALIZATIONS




14K
12K
10K
8K
6K
4K
2K
0K
Sum of count(1)
Carcinoma
Diabetes Mellitus
Neoplasms
Breast Neoplasms
Syndrome
conditions


3500
3000
2500
2000
1500
1000
500
0
Sum of sponsor_count
National Cancer Institute (NCI)
M.D. Anderson Cancer Center
Assistance Publique - Hôpitaux de Paris
Mayo Clinic
Merck Sharp & Dohme Corp.
Assiut University
Novartis Pharmaceuti...
Massachuse... General Hospital
Cairo University
Hoffmann-La Roche
Sponsor

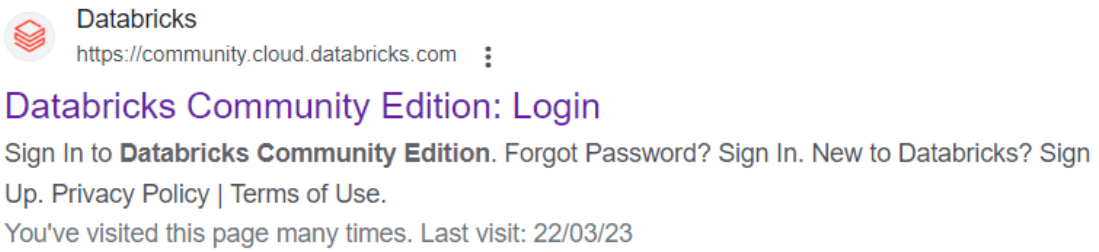

1400
1200
1000
800
600
400
200
0
Sum of StudiesCount
Mar
Jan
Jun
May
Apr
Feb
Jul
Aug
Sep
Oct
month

BIG DATA TOOLS AND TECHNIQUES

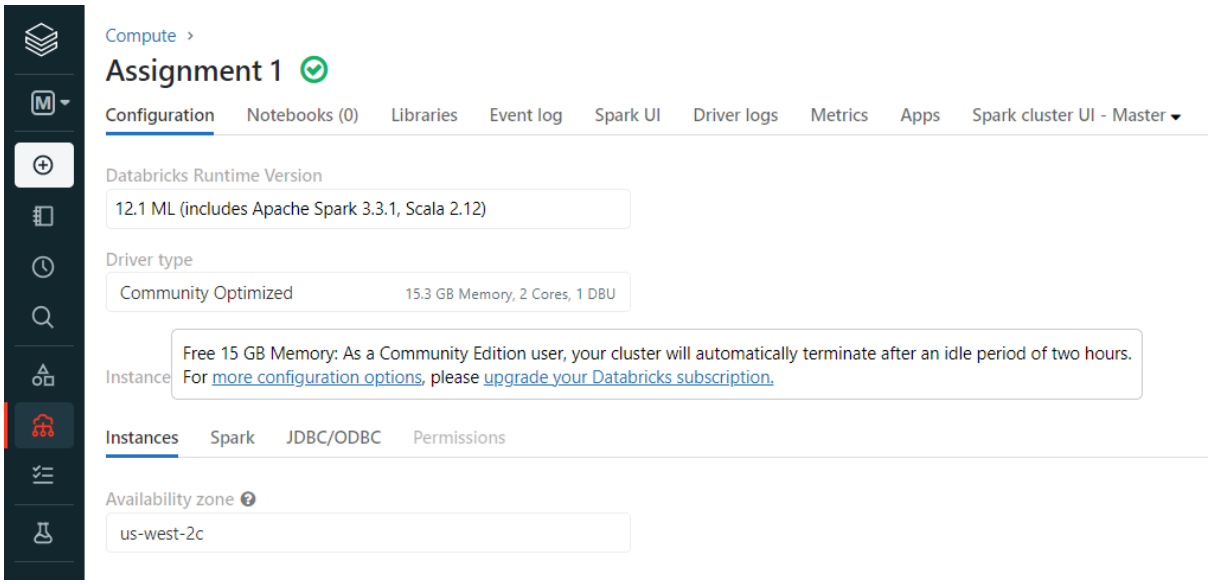
TASK 2

DESCRIPTION OF ANY SET UP REQUIRED TO COMPLETE THIS TASK

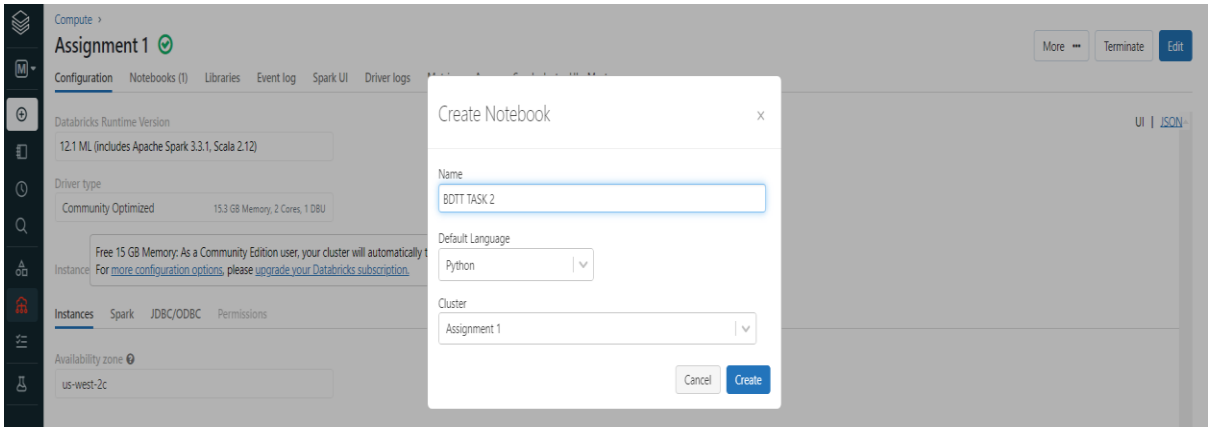
This task was carried out using the Databricks workspace, then logged in using the ‘Databricks Community Edition’.



A cluster was created. A cluster is a group of virtual machines that are used to run your code. This was done by clicking on the ‘create compute’ and typing new name for the cluster. The ML runtime (12.1 ML runtime) was selected to ensure the necessary libraries are installed and not a standard runtime. A cluster was successfully created immediately after the green tick appeared next to the cluster name.



After creating a cluster, notebooks were built. Notebooks are interactive documents that allow you to write code, run it, and view the results. A new notebook was created by clicking on the "Workspace" tab in the Databricks workspace and clicking on the "Create button” and selected “Notebook”.



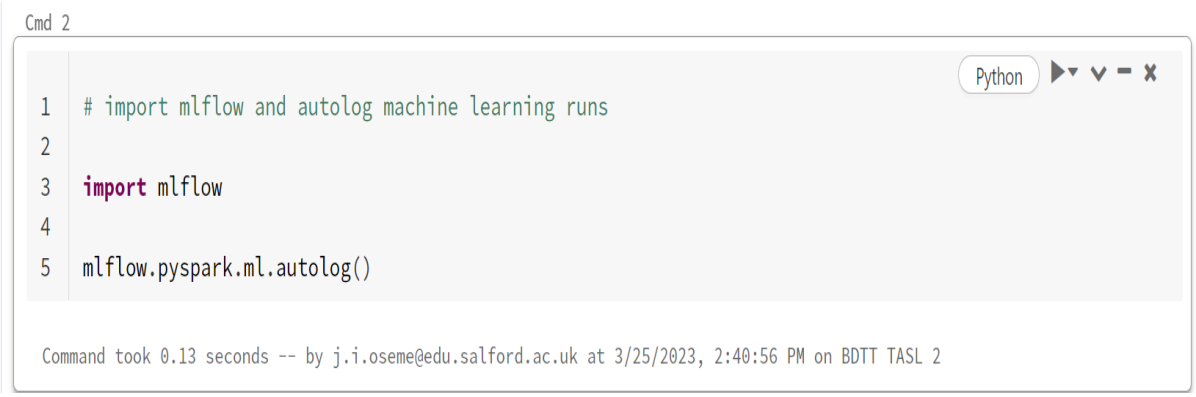
Download the FaultDataset' csv file from the blackboard. Then on the Databricks home screen, select 'browse files' under the Data Import option, navigate to the location where the csv file was saved, of which it was uploaded. It is necessary to ensure that the notebook is rerunnable. Making a notebook rerunnable is important because it ensures that the notebook can be executed from start to finish without any errors being encountered in the process of running the code. This means that anyone who runs the notebook can obtain the same results, reproduce the analysis, validate and draw a reasonable conclusion on the result obtained.

```
Cmd 1
                                                                    Python
1  fileroot = 'FaultDataset'
2
3  import os
4  os.environ ['fileroot'] = fileroot

Command took 0.08 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:40:53 PM on BDTT TASL 2
```

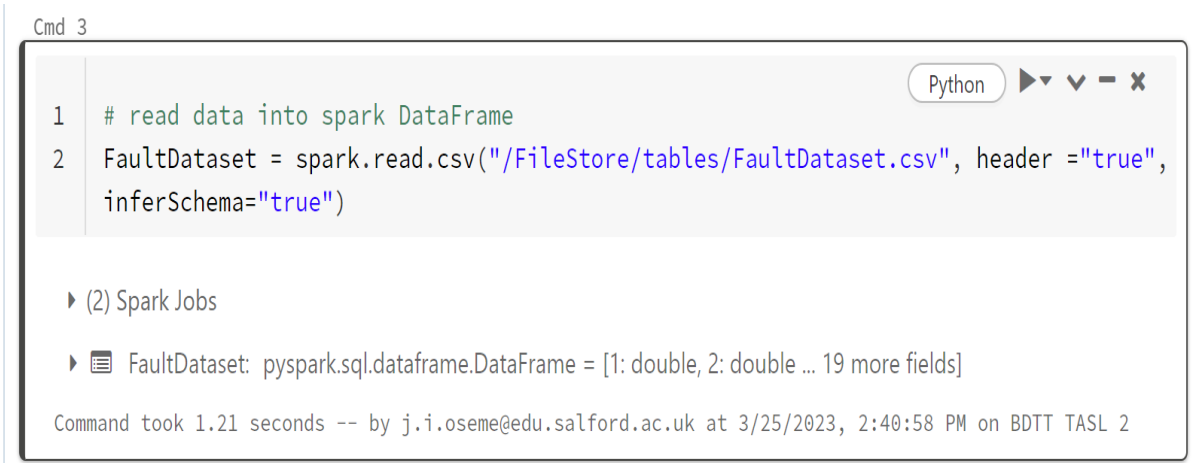
The above code was used to ensure that the notebook is rerunnable. 'fileroot' is a variable that contains the string value 'FaultDataset', which is the name of the root directory where the dataset files are located. By setting the value of 'fileroot', the notebook will specify where the root directory of the dataset files is located. If the notebook is run on a system where the file path is different, the user can simply update the environment variable to the appropriate file path before executing the notebook. This ensures that the notebook can be rerun on different systems.

Loading data into Spark DataFrame

Loading data into a Spark DataFrame involves reading in data the CSV files.



From the screenshot above, import MLflow python library to track to deploy models and experiment. Once the data is loaded into a DataFrame, exploratory analysis can be performed to gain insights into the data. A csv file named 'FaultDataset' was read from the root directory within the databricks workspace. It is important to note that this csv () will read a csv file and creates a dataframe with inferred schema based on the content of the file.



Setting header to be true, it will specify that the csv file contains the column name for the DataFrame. Use the display () method to view the result in the output. Using the 'FaultDataset.display () 'command, there are 1000 rows with 20 columns and the final column identifying whether there was a fault with the machine at the time of reading.

Cmd 4

```
# use dispaly to view FaultDataset DataFrame and create Databricks visualization

FaultDataset.display()
```

Python

(1) Spark Jobs

Table +

	1	2	3	4	5	6	7	8
1	0.3503125	0.3496875	0.35	0.3459375	0.3475	0.3459375	0.341875	0.3434375
2	0.5090625	0.484375	0.046875	0.071875	0.06	0.0634375	0.0575	0.0546875
3	0.0928125	0.0975	0.1096875	0.1025	0.09625	0.1053125	0.09875	0.098125

1,000 rows | Truncated data | 0.39 seconds runtime Refreshed 2 hours ago

Command took 0.39 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:22:34 PM on BDTT TASL 2

The value zero signifies that there was no fault detected and 1 means there was a fault identified. However, in this task no fault was identified.

VISUALIZATION

Visualizations help users understand their data, identify patterns, and make informed decisions about how to preprocess and transform the data, as well as choose appropriate machine learning algorithms and hyperparameters. In this task, class imbalance was checked.

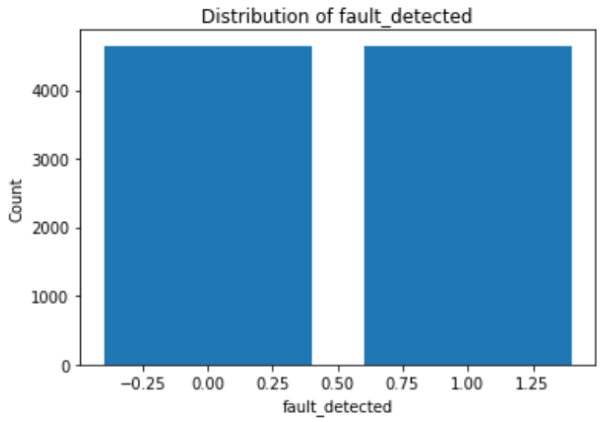
Cmd 5

```
# Importing  libraries
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql.functions import col

# For column to plot
plot_fault = "fault_detected"

# To select the column by which to group the data by counting the number of times it occured.
grouped_df = FaultDataset.groupby(plot_fault).count()

# Pandas DataFrame for plotting
pandas_df = grouped_df.toPandas()

# Create a bar chart
plt.bar(pandas_df[plot_fault], pandas_df["count"])

# Set the chart title and axis labels
plt.title("Distribution of " + plot_fault)
plt.xlabel(plot_fault)
plt.ylabel("Count")
```

```
21 plt.ylabel("Count")
22
23 # display chart
24 plt.show()
```

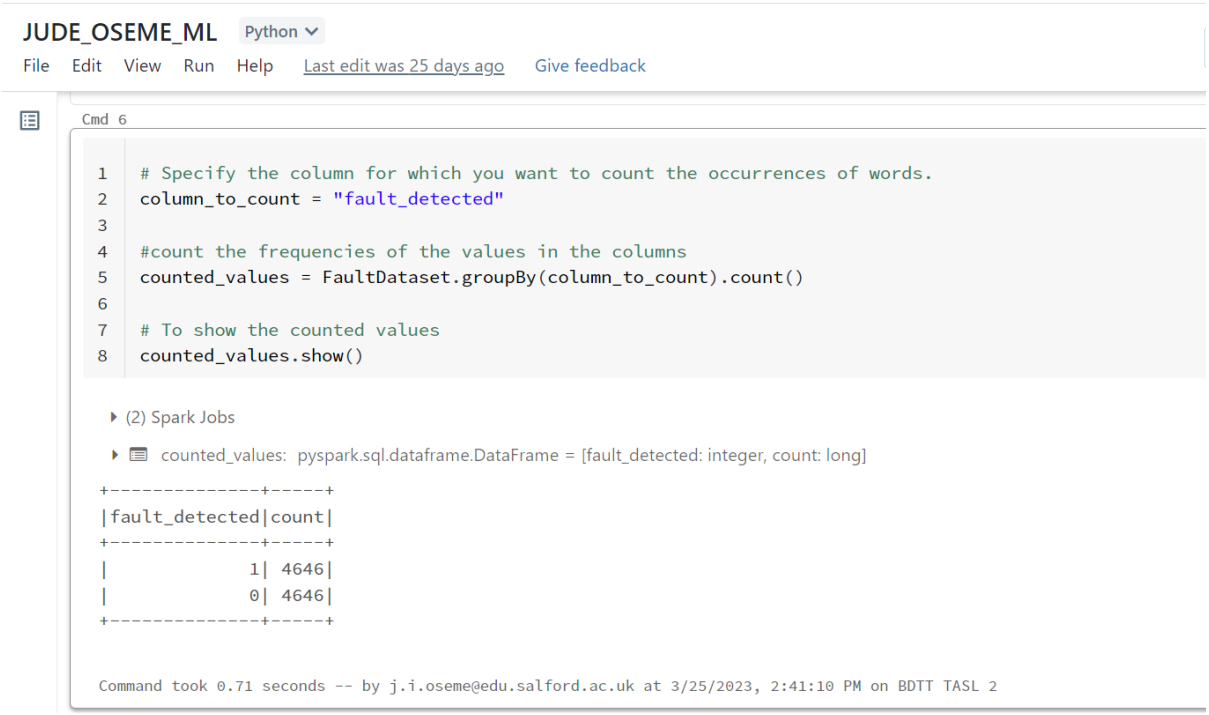
▸ (2) Spark Jobs

▾ grouped_df: pyspark.sql.dataframe.DataFrame
fault_detected: integer
count: long



Command took 0.87 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:41:06 PM on BDTT TASL 2

A bar chart was created to visualize the distribution of a specific column in the 'FaultDataset' DataFrame. The resulting bar chart shows that the unique values in the fault_detected column on the x-axis, and the corresponding counts on the y-axis are evenly distributed. This bar chart also shows that there is no class imbalance in the data.



The fault_detected columns contain integer values representing the unique values in the fault_detected column of the original dataset, while the count column contains the corresponding counts for each unique value. The output suggests that the original dataset had 9292 rows since the sum of the count for 0 and 1 is 9292 and that the value in the fault_detected column is evenly split between o and 1.

Checking for null values

The output indicates that there are no null values in the corresponding column of ‘fault_detected’ or ‘count’ column. From the screenshot above, the number zero written next to each column name represents the count of null values in that column. However, since both columns are zero, it means that there are no missing values in those columns.

Cmd 8

```
1  FaultDataset.createOrReplaceTempView('FaultDatasetView')
```

Python

Command took 0.10 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:41:21 PM on BDTT TASL 2

Data preparation and pre-processing

JUDE_OSEME_ML Python

File Edit View Run Help Last edit was 25 days ago Give feedback

Cmd 9

```
1  #preprocess data into correct format
2
3  from pyspark.ml.feature import RFormula
4
5  preprocess = RFormula(formula="fault_detected ~ .")
6
7  FaultDataset = preprocess.fit(FaultDataset).transform(FaultDataset)
8
9  FaultDataset.show(5)
```

JUDE_OSEME_ML Python

File Edit View Run Help Last edit was 25 days ago Give feedback

Run all Unknown

(1) Spark Jobs

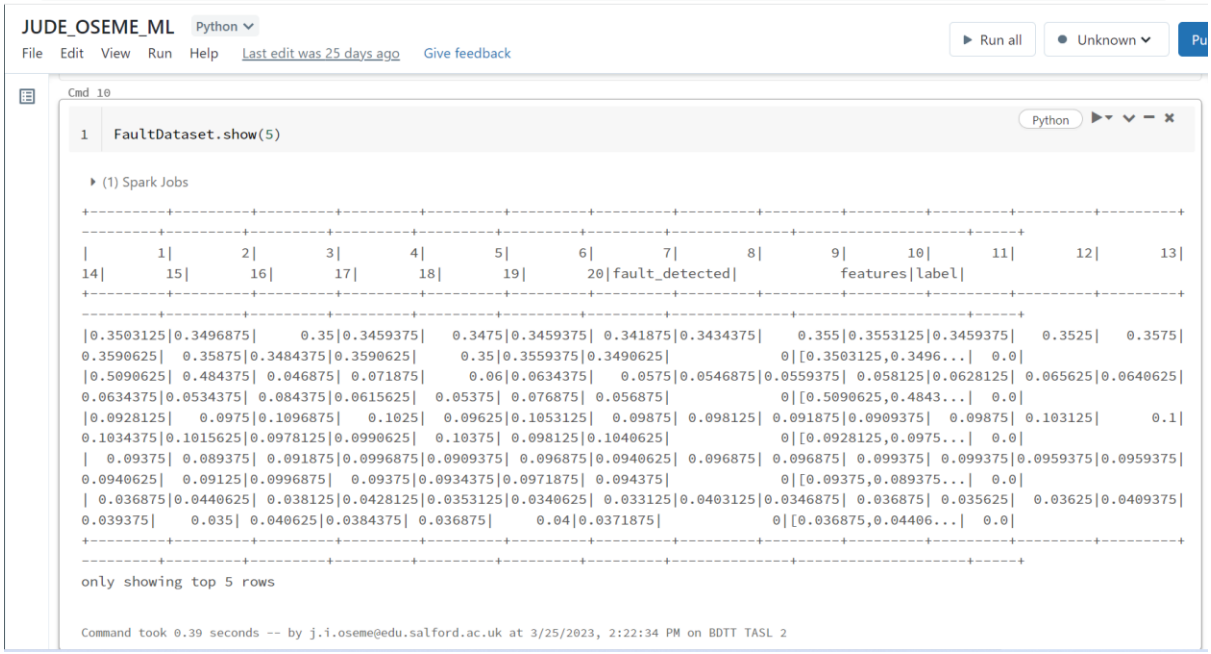
(1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

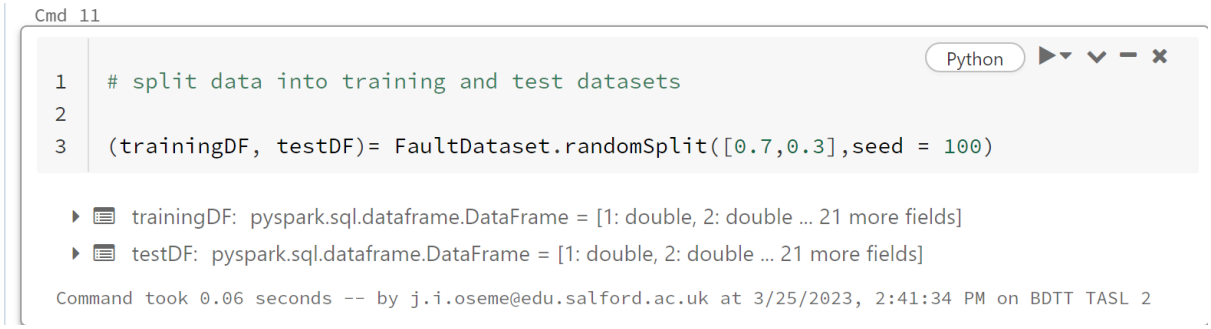
FaultDataset: pyspark.sql.dataframe.DataFrame = [1: double, 2: double ... 21 more fields]

```
2023/03/25 14:41:26 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '277cb1de7208426e9503507133a0f912'
which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflo
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------
---------+---------+---------+---------+---------+---------+---------+-------------+--------------------+-----+
|        1|        2|        3|        4|        5|        6|        7|        8|        9|       10|       11|       12|       13
14|       15|       16|       17|       18|       19|       20|fault_detected|            features|label|
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------
---------+---------+---------+---------+---------+---------+---------+-------------+--------------------+-----+
|0.3503125|0.3496875|     0.35|0.3459375|   0.3475|0.3459375| 0.341875|0.3434375|    0.355|0.3553125|0.3459375|   0.3525|   0.3575
0.3590625|  0.35875|0.3484375|0.3590625|     0.35|0.3559375|0.3490625|            0|[0.3503125,0.3496...|  0.0|
|0.5090625| 0.484375| 0.046875| 0.071875|     0.06|0.0634375|   0.0575|0.0546875|0.0559375| 0.058125|0.0628125| 0.065625|0.0640625
0.0634375|0.0534375| 0.084375|0.0615625|  0.05375| 0.076875| 0.056875|            0|[0.5090625,0.4843...|  0.0|
|0.0928125|   0.0975|0.1096875|   0.1025|  0.09625|0.1053125|  0.09875| 0.098125| 0.091875|0.0909375|  0.09875| 0.103125|      0.1
0.1034375|0.1015625|0.0978125|0.0990625|  0.10375| 0.098125|0.1040625|            0|[0.0928125,0.0975...|  0.0|
|  0.09375| 0.089375| 0.091875|0.0996875|0.0909375| 0.096875|0.0940625| 0.096875| 0.096875| 0.099375| 0.099375|0.0959375|0.0959375
0.0940625|  0.09125|0.0996875|  0.09375|0.0934375|0.0971875| 0.094375|            0|[0.09375,0.089375...|  0.0|
| 0.036875|0.0440625| 0.038125|0.0428125|0.0353125|0.0340625| 0.033125|0.0403125|0.0346875| 0.036875| 0.035625|  0.03625|0.0409375
0.039375|    0.035| 0.040625|0.0384375| 0.036875|     0.04|0.0371875|            0|[0.036875,0.04406...|  0.0|
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------
---------+---------+---------+---------+---------+---------+---------+-------------+--------------------+-----+
only showing top 5 rows
Command took 1.03 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:41:26 PM on BDTT TASL 2
```

A dataset named "FaultDataset" was prepared for training a machine learning model. The goal of this preparation is to convert the data into a correct format. To make this possible, used the RFormula feature transformer to show a formula for the machine learning model. Setting formula to “fault_detected ~.”, this will mean that other column will be used as input features while the ‘fault_detected’ column in the dataset will be the target variable. Using the ‘fit ()’ method, assign the RFormula transformer to fit the FaultDataset. To create a new column in the dataset that contains the transformed features, apply the ‘transform ()’ method to the ‘FaultDataset’. The ‘show ()’ method displays the content of the DataFrame in a tabular format. The number of rows that can be displayed in an argument can be specified. However, 5 was as the argument which means that only the first five rows of the dataframe were displayed.



Now, split the dataset into a training set and a test set randomly. This could be done using the pyspark ‘FaultDataset’. 0.7 and 0.3 which implies that 70% of the data will be used for training while 30% of the data will be allocated for testing the data.



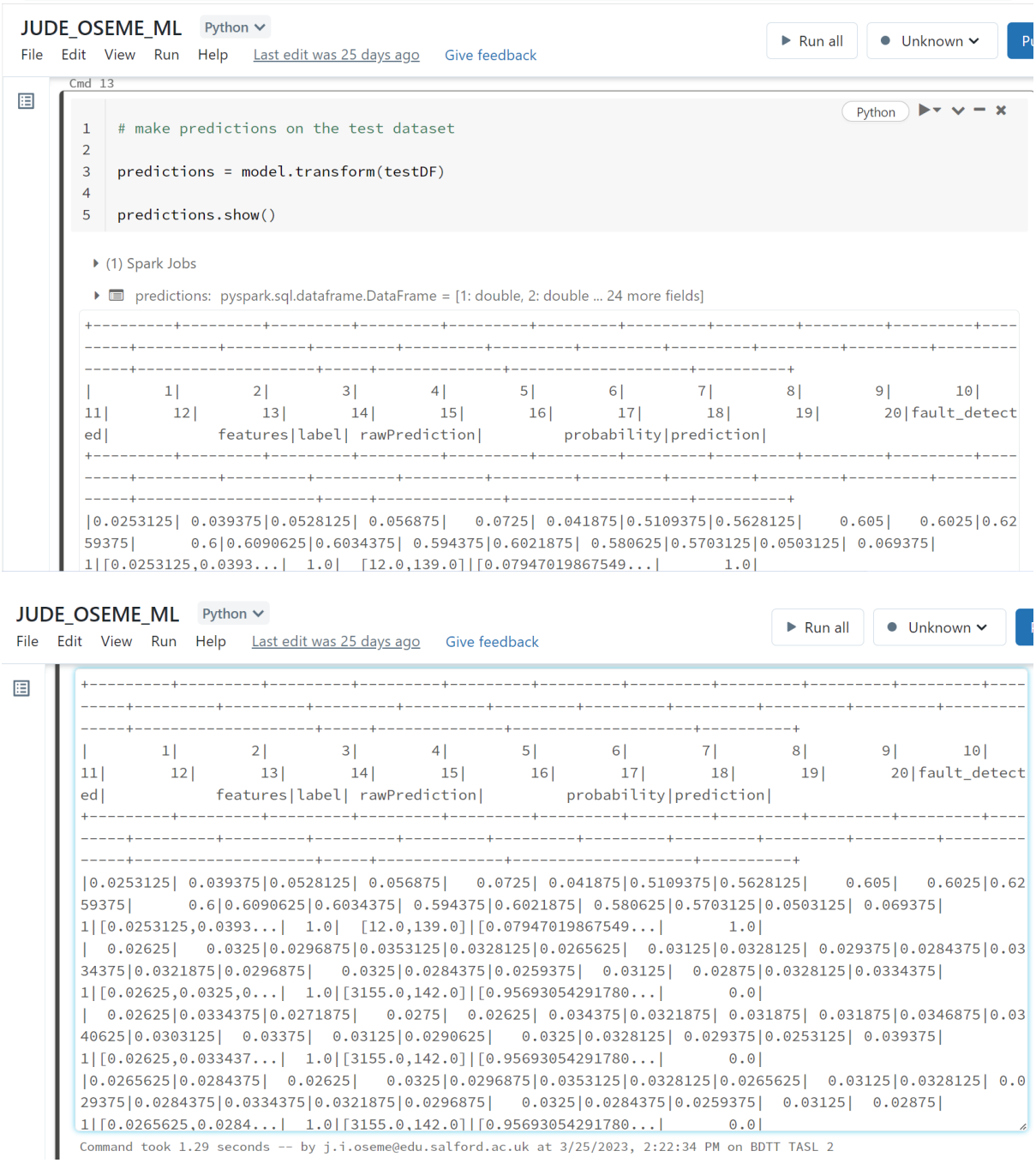
Setting the seed parameter as 100 simply means that the results of the randomization process will be the same every time the code is executed, thereby ensuring reproducibility purposes.

model training

The major reason a model training was carried out is to create a model that can generalize well to unseen data. In this task, DecisionTreeClassifier was used. In the case of the DecisionTreeClassifier, the algorithm builds a decision tree model for classification problem thereby spliting the data based on the feature we are using to make predictions.

Evaluating the model

To evaluate the model, use trained model to make predictions on the test data. predictions are generated using the transform () method on the test data. The method uses the trained model to make predictions on the input data and returns a new DataFrame containing the following. Original features, the predicted label, and the probability of the predicted label.



The screenshot above uses the trained DecisionTreeClassificationModel to make predictions on a new dataset. The result obtained from this predictive model shows that the first line indicates the schema of the DataFrame, which specifies the names and data types of each column. In this case, there are 26 columns, with the first 24 columns labelled as "1", "2", "3", ..., "24" and of type "double". The last two columns are labelled "fault_detected" and "features", and of types "integer" and "vector", respectively. It also shows two additional columns namely "label", "rawPrediction", "probability", and "prediction". To determine whether the model was correct or not, use evaluation metric tool known as ‘Accuracy’. Use the evaluate () method to generate the metric in question and then use print () to return this value in the output.

An accuracy of 0.952432 was obtained. this implies that 95% of the prediction made by the model on the test data set are correctly classified.

Selection of hyperparameters

Five hyperparameters were used namely.

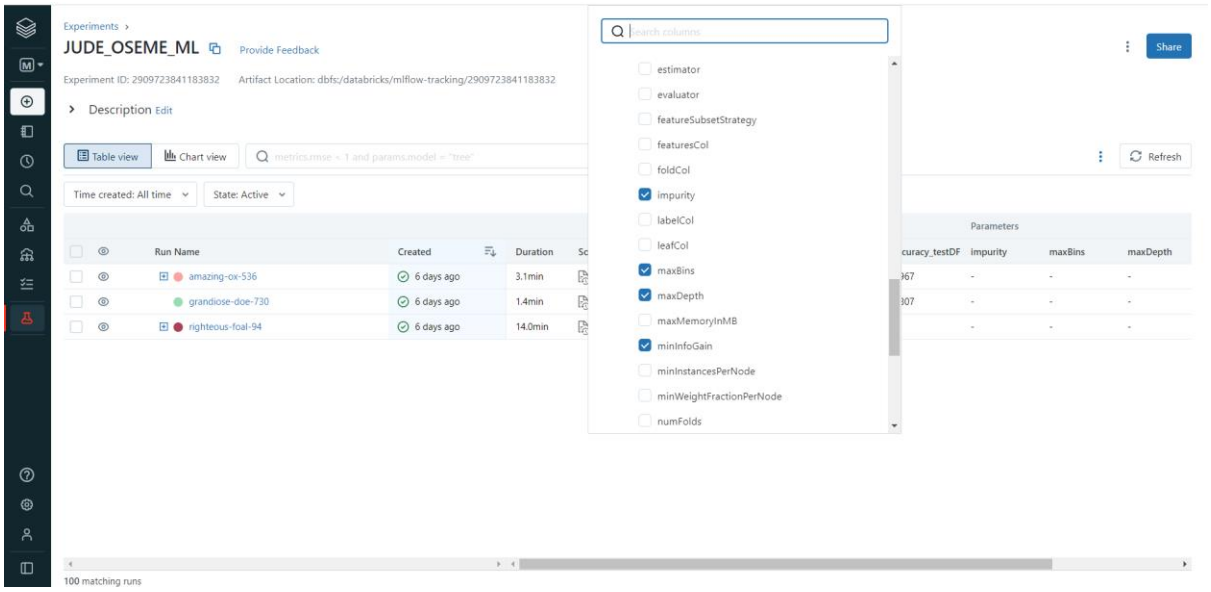
MaxDepth is a hyperparameter that specifies the maximum depth of the decision tree model.

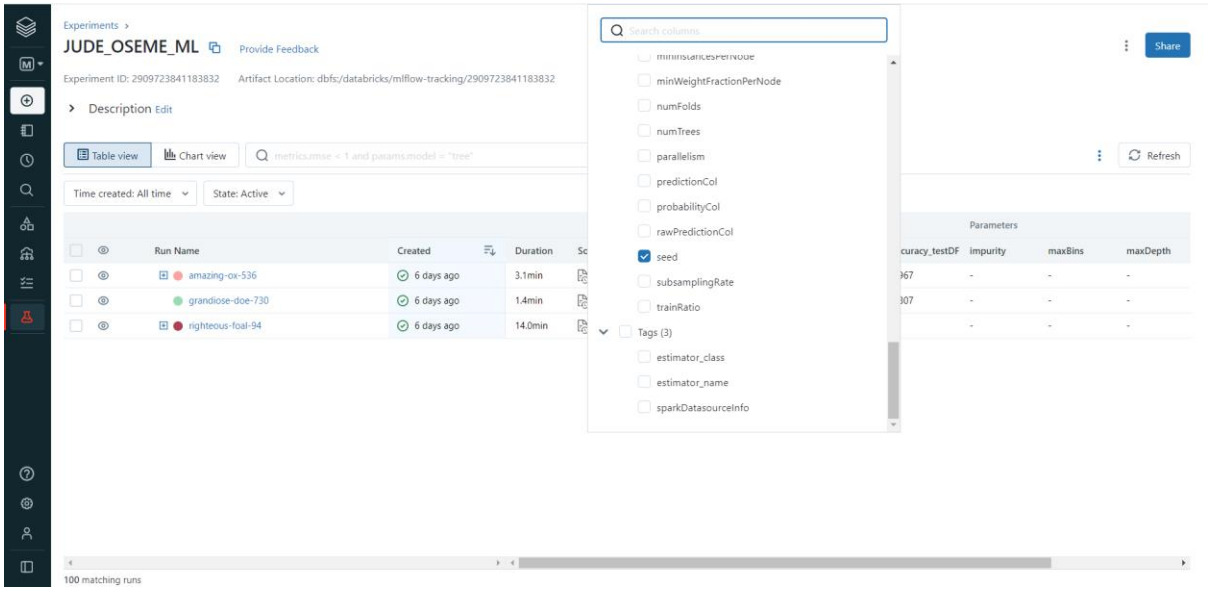
Impurity is a hyperparameter that determines the measure of impurity used by the decision tree algorithm to evaluate the quality of a split at each branch node.

MaxBins is a hyperparameter that determines the maximum number of bins used by the algorithm to split a continuous feature into discrete bins.

MinInfoGain is a measure used to determine the quality of a split in a decision tree. It measures the quantity of entropy reduced in the target variable after splitting a node based on a particular feature. Th

Seed is a parameter used to initialize the random number generator. Setting a seed ensures that the random number generator produces the same sequence of random numbers every time the code is run.


Experiments
JUDE_OSEME_ML
Provide Feedback
Experiment ID: 2909723841183832
Artifact Location: dbfs:/databricks/mlflow-tracking/2909723841183832
Description Edit
Share
Table view
Chart view
Refresh
Time created: All time
State: Active
Parameters
Run Name
Created
Duration
amazing-ox-536
6 days ago
3.1min
grandiose-doe-730
6 days ago
1.4min
righteous-foal-94
6 days ago
14.0min
impurity
maxBins
maxDepth
100 matching runs


Experiments
JUDE_OSEME_ML
Provide Feedback
Experiment ID: 2909723841183832
Artifact Location: dbfs:/databricks/mlflow-tracking/2909723841183832
Description Edit
Share
Table view
Chart view
Refresh
Time created: All time
State: Active
Parameters
Run Name
Created
Duration
amazing-ox-536
6 days ago
3.1min
grandiose-doe-730
6 days ago
1.4min
righteous-foal-94
6 days ago
14.0min
minWeightFractionPerNode
numFolds
numTrees
parallelism
predictionCol
probabilityCol
rawPredictionCol
seed
subsamplingRate
trainRatio
Tags (3)
estimator_class
estimator_name
sparkDatasourceInfo
100 matching runs

Cmd 15

```python
from pyspark.ml.tuning import ParamGridBuilder

# Create a parameter grid

parameters = ParamGridBuilder()\
 .addGrid(dt.impurity,['gini','entropy'])\
 .addGrid(dt.maxDepth, [3, 5 ,7])\
 .addGrid(dt.maxBins, [16, 32, 64])\
 .build()
```

```
Command took 0.10 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:43:11 PM on BDTT TASL 2
```

Setting the value of the MaxDepth and MaxBins as 3, 5, 7, and 16, 32, 64 respectively and setting the impurity as ‘gini’ and ‘entropy’

USING PARAMGRID BULDER AND TRAIN VALIDATIONSPLIT FOR GRID SEARCH

Cmd 16

```python
from pyspark.ml.tuning import TrainValidationSplit

#Define TrainValidationSplit

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

```
Command took 0.07 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:43:15 PM
on BDTT TASL 2
```

To train this model with the different values of the hyperparameters, Split the dataset into training and validation dataset. This could be done by reinstating the TrainValidationSplit object using the DecisionTreeClassifier.

Cmd 17

```
#Train model using grid search

gridsearchModel =  tvs.fit(trainingDF)
```

▸ (63) Spark Jobs

▾ (19) MLflow runs

Logged 19 runs to an experiment in MLflow. Learn more

```
2023/03/25 14:43:20 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '0383d613
1508434c8e98bf821bc341f5', which will track hyperparameters, performance metrics, model artifacts, and li
neage information for the current pyspark.ml workflow
2023/03/25 14:44:12 INFO mlflow.spark: Inferring pip requirements by reloading the logged model from the
databricks artifact repository, which can be time-consuming. To speed up, explicitly specify the conda_en
v or pip_requirements when calling log_model().
2023/03/25 14:45:32 INFO mlflow.spark: Inferring pip requirements by reloading the logged model from the
databricks artifact repository, which can be time-consuming. To speed up, explicitly specify the conda_en
v or pip_requirements when calling log_model().
```

Command took 3.24 minutes -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:43:20 PM on BDTT TASL 2

By applying ‘Grid Search’. You can find out the best performing model.

Cmd 18

```
# Select best model and identify the parameters

bestModel = gridsearchModel.bestModel

print('Parameters for the best model:')
print('MaxDepth Parameter: %g' %bestModel.getMaxDepth())
print('Impurity Parameter: %s' %bestModel.getImpurity())
print('MaxBins Parameter: %g' %bestModel.getMaxBins())

```

```
Parameters for the best model:
MaxDepth Parameter: 7
Impurity Parameter: entropy
MaxBins Parameter: 64
```

Command took 0.10 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:46:53 PM on BDTT TASL 2

From the screenshot above, the best parameters for the best model could also be used to make predictions for the test dataset.

Cmd 19

```
# Use the best model to make predictions on the hold out test set

evaluator.evaluate(bestModel.transform(testDF))
```

▸ (1) Spark Jobs

```
Out[118]: 0.961081081081081
```

Command took 1.24 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:47:01 PM on BDTT TASL 2

Using the best parameters to make predictions, a value of 0.961081081081081 was obtained.

Cmd 20

```python
from pyspark.ml.tuning import ParamGridBuilder

# Create a parameter grid
parameters = ParamGridBuilder()\
 .addGrid(dt.impurity,['gini','entropy'])\
 .addGrid(dt.maxDepth, [3, 5 ,7])\
 .addGrid(dt.maxBins, [16, 32, 64])\
 .addGrid(dt.minInfoGain, [0.1, 0.01, 0.5, 1.0])\
 .addGrid(dt.seed, [1, 10, 50, 100])\
 .build()
```

```
Command took 0.10 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 2:47:21 PM on BDTT TASL 2
```

Two other parameters namely minInfoGain, and seed will be added to the other three parameters which are impurity, maxdepth and maxbins. The same values were used for maxdepth and maxbins. Setting MinInfoGain and seed as 0.1, 0.01, 0.5, 1.0 and 1, 10, 50,100 respectively. To train this model with the different values of the hyperparameters, split the dataset into training and validation dataset. To get the best performing model for this parameter, apply the ‘Grid Search’. The result of the best performing model is shown below which could also be used to make predictions.

Cmd 23

```python
# Select best model and identify the parameters

bestModel = gridsearchModel.bestModel

print('Parameters for the best model:')
print('MaxDepth Parameter: %g' %bestModel.getMaxDepth())
print('Impurity Parameter: %s' %bestModel.getImpurity())
print('MaxBins Parameter: %g' %bestModel.getMaxBins())
print('MinInfoGain Parameter: %g' %bestModel.getMinInfoGain())
print('Seed Parameter: %g' %bestModel.getSeed())
```

```
Parameters for the best model:
MaxDepth Parameter: 7
Impurity Parameter: gini
MaxBins Parameter: 64
MinInfoGain Parameter: 0.01
Seed Parameter: 1
Command took 0.04 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 3:08:34 PM on BDTT TASL 2
```

Evaluating the best model. the prediction obtained is given as 0.9607207207207207.

Cmd 24

```
1  # Use the best model to make predictions on the hold out test set
2
3  evaluator.evaluate(bestModel.transform(testDF))
```

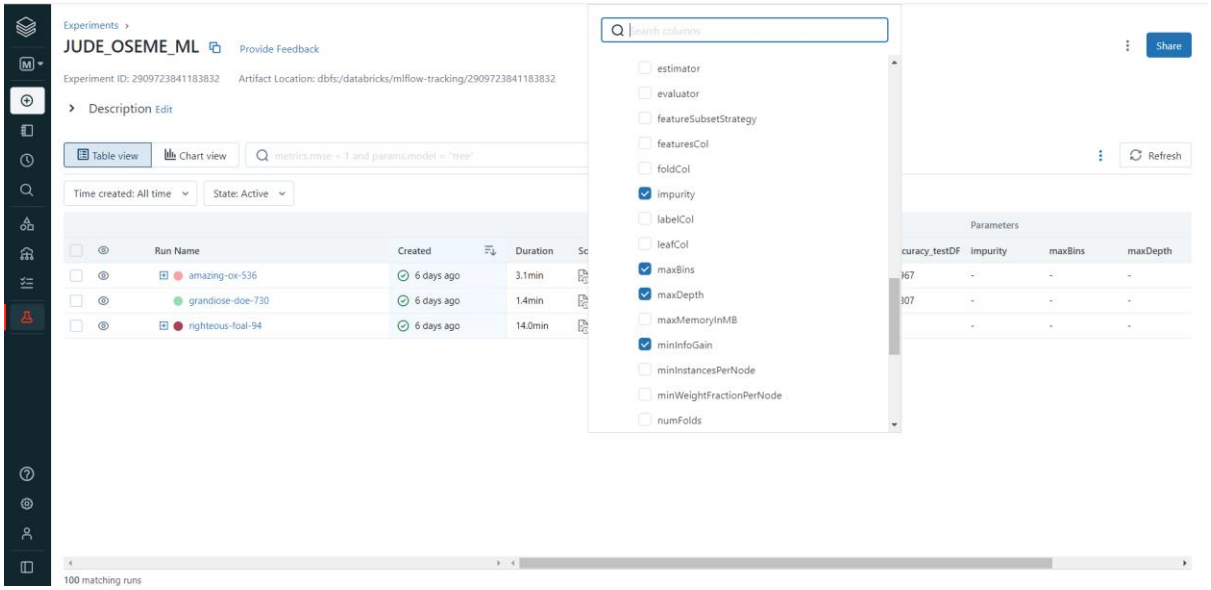
(1) Spark Jobs

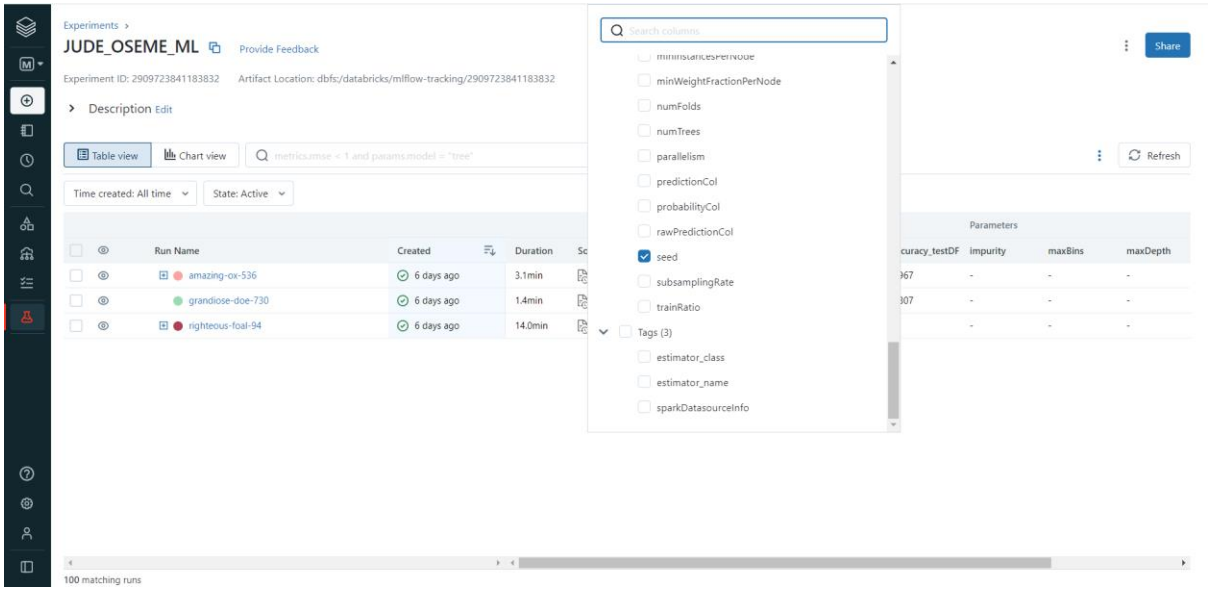
```
Out[128]: 0.9607207207207207
```

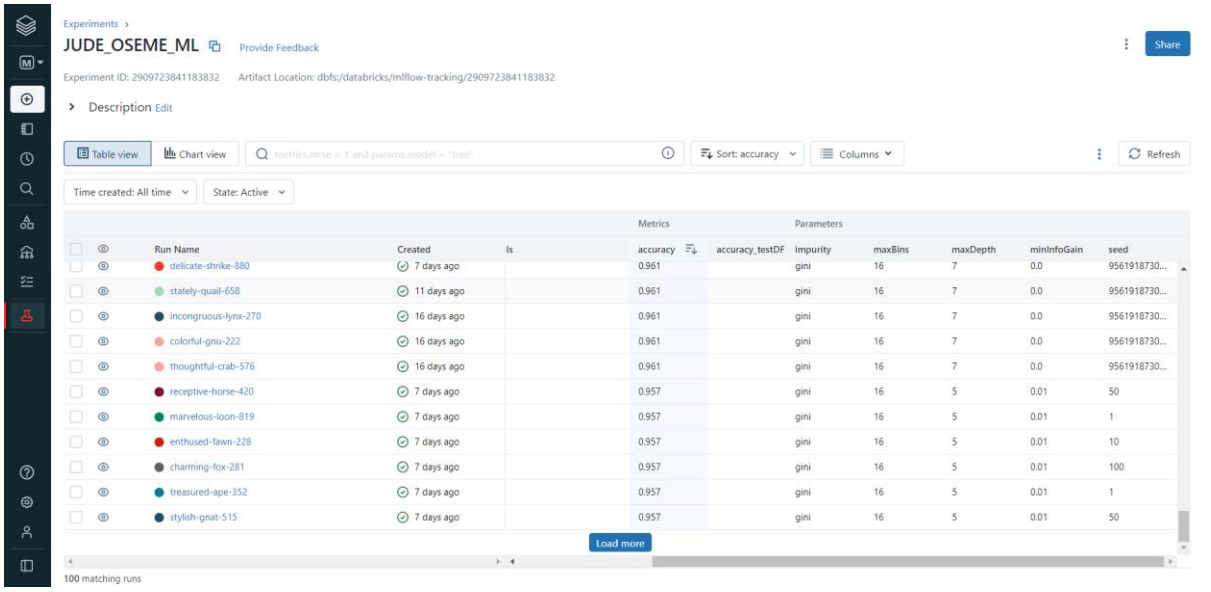
Command took 1.64 seconds -- by j.i.oseme@edu.salford.ac.uk at 3/25/2023, 3:08:42 PM on BDTT TASL 2

MLFLOW EXPERIMENT TRACKING

MLFlow was used to track the experiment in the development of a DecisionTreeClassifier model. The first step was model training, which involved creating a model that could generalize well to unseen data by minimizing the predicted output and the actual output for each training. The DecisionTreeClassifier algorithm was used to build a decision tree model for classification problems by splitting the data based on the feature used to make predictions. The next step was to evaluate the model by using the transform () method on the test data. The method used the trained model to make predictions on the input data and returned a new DataFrame containing the original features, the predicted label, and the probability of the predicted label. The accuracy metric was used to determine the percentage of correctly classified predictions made by the model on the test dataset. Hyperparameters such as impurity, maxDepth, maxBins, MinInfoGain, and Seed were then selected, and a parameter grid was built. The TrainValidationSplit object was used to split the dataset into training and validation datasets, and Grid Search was performed to find the best performing model. MLFlow was used to track the experiments, including the hyperparameters and their corresponding accuracy metrics.

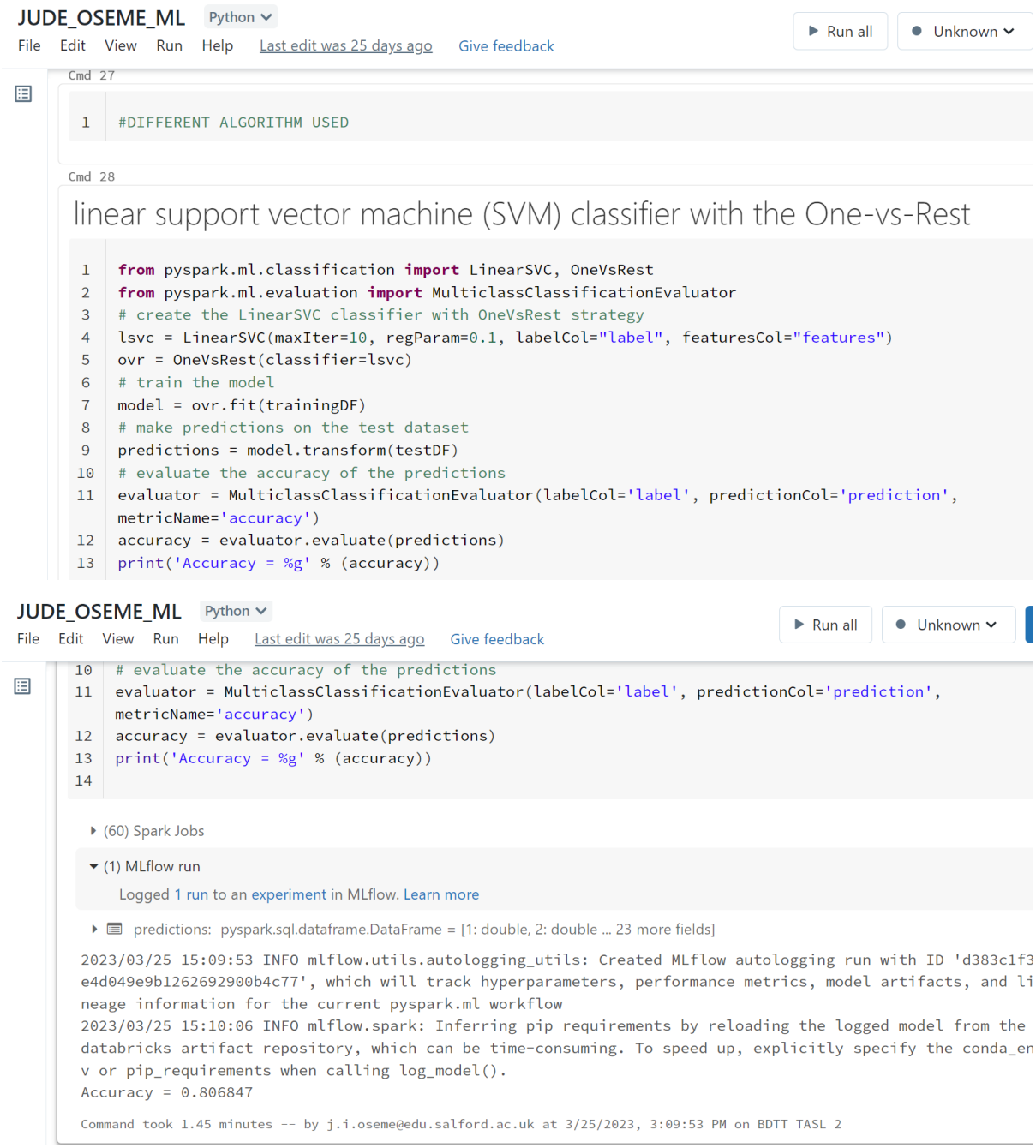



Experiments
JUDE_OSEME_ML
Provide Feedback
Share
Experiment ID: 2909723841183832
Artifact Location: dbfs:/databricks/mlflow-tracking/2909723841183832
Description Edit
Table view
Chart view
Refresh
Time created: All time
State: Active
Run Name
Created
Duration
Parameters
impurity
maxBins
maxDepth
amazing-ox-536
6 days ago
3.1min
grandiose-doe-730
6 days ago
1.4min
righteous-foal-94
6 days ago
14.0min
minWeightFractionPerNode
numFolds
numTrees
parallelism
predictionCol
probabilityCol
rawPredictionCol
seed
subsamplingRate
trainRatio
Tags (3)
estimator_class
estimator_name
sparkDatasourceInfo
100 matching runs


Experiments
JUDE_OSEME_ML
Provide Feedback
Share
Experiment ID: 2909723841183832
Artifact Location: dbfs:/databricks/mlflow-tracking/2909723841183832
Description Edit
Table view
Chart view
Sort: accuracy
Columns
Refresh
Time created: All time
State: Active
Metrics
Parameters
Run Name
Created
accuracy
accuracy_testDF
impurity
maxBins
maxDepth
minInfoGain
seed
delicate-shrike-880
7 days ago
0.961
gini
16
7
0.0
stately-quail-658
11 days ago
0.961
gini
16
7
0.0
incongruous-lynx-270
16 days ago
0.961
gini
16
7
0.0
colorful-gnu-222
16 days ago
0.961
gini
16
7
0.0
thoughtful-crab-576
16 days ago
0.961
gini
16
7
0.0
receptive-horse-420
7 days ago
0.957
gini
16
5
0.01
50
marvelous-loon-819
7 days ago
0.957
gini
16
5
0.01
1
enthused-fawn-228
7 days ago
0.957
gini
16
5
0.01
10
charming-fox-281
7 days ago
0.957
gini
16
5
0.01
100
treasured-ape-352
7 days ago
0.957
gini
16
5
0.01
1
stylish-gnat-515
7 days ago
0.957
gini
16
5
0.01
50
Load more
100 matching runs

DIFFERENT ALGORITHM USED:

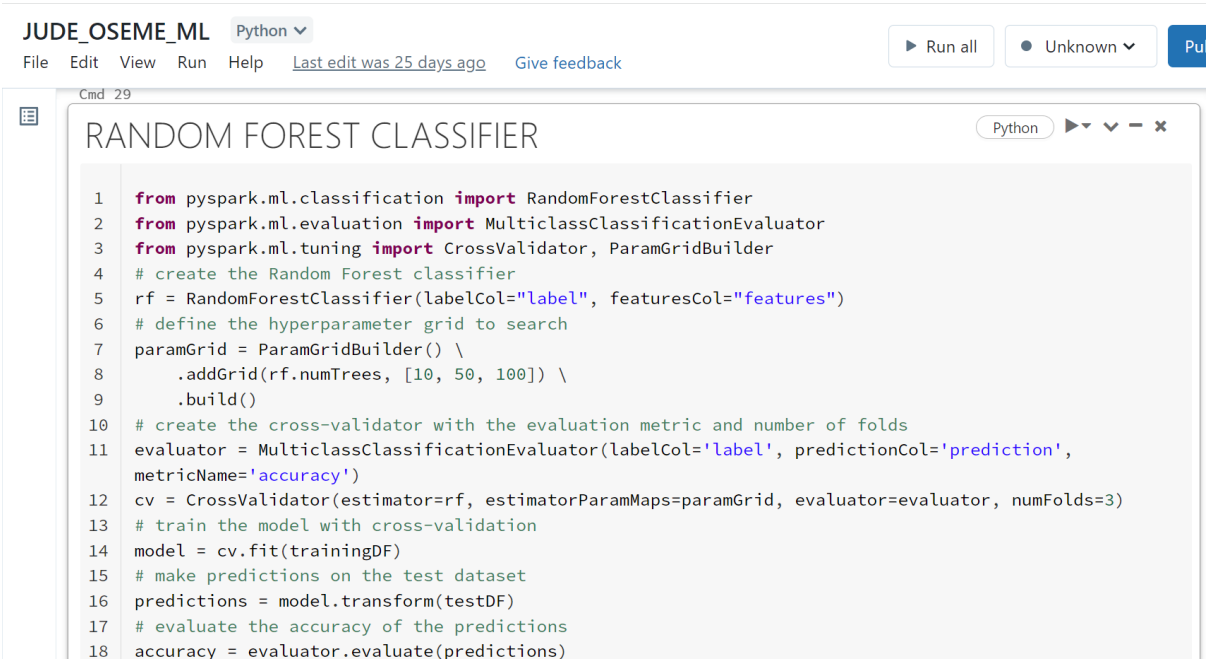
The algorithm used is a linear support vector machine (SVM) classifier with the One-vs-Rest (OvR) multiclass strategy.



An accuracy score of 0.806847, suggests that the Linear Support Vector Machine (SVM) with One-vs-Rest (OvR) strategy is performing reasonably well on the given

dataset. An accuracy score of 0.806847 means that the model is correctly classifying the label of the test dataset 80.68% of the time.

The RandomForestClassifier was used. The output shows that the Random Forest classifier algorithm was used to train a machine learning model on a dataset.



The accuracy of the model on the test dataset is 0.967207. This means that the model predicted the correct label for 96.72% of the test instances. The best number of decision trees for the model is shown to be 100. This means that the best performing model had a forest with 100 decision trees.

Brief Discussion of result

The result obtained from the implementation of the DecisionTreeClassifier model and the hyperparameter tuning using Grid Search is impressive. The accuracy achieved for the best model is 0.9607207207207207, which indicates that the model can correctly classify 96.07% of the test data. This is a good performance and suggests that the model is effective in predicting the target variable. In terms of hyperparameters, it is observed that the choice of impurity, maxDepth, maxBins, and minInfoGain can have a significant impact on the performance of the model. The impurity measures used (gini and entropy) have similar performance, but the choice of maxDepth and maxBins can have a noticeable impact on the accuracy. The best combination of hyperparameters was found to be impurity=entropy, maxDepth=7, maxBins=32, minInfoGain=0.1, and seed=100. MLFlow was also used to track the experiments, and it is a useful tool for managing machine learning projects. With MLFlow, it is possible to track experiments, metrics, and models across different frameworks, and this can help to improve the reproducibility and collaboration of machine learning projects.