Big Data Tools & Techniques
MSc Data Science

ANALYZING CLINICAL TRIALS USING PYSPARK AND SPARK SQL AND PREVENTIVE MAINTENANCE USING VIBRATION SENSOR DATA

NAME: ABDULAZEEZ MOMOH

STUDENT ID: @00686172

DUE DATE: 28th APRIL 2023

Table of Contents

TASK 1: ANALYZING CLINICAL TRIALS USING PYSPARK AND SPARK SQL

1.0 INTRODUCTION

Clinical trials are a critical aspect of medical research, as they help assess the safety and efficacy of new treatments, drugs, and medical devices. This report presents a detailed analysis of clinical trials data using PySpark and Spark SQL, a powerful open-source data processing engine for large-scale data processing. The dataset contains information about various aspects of clinical trials, such as study types, conditions, sponsors, and completion dates. We will answer five specific questions related to the dataset, discussing the assumptions, implementation, and results for each question. We will also implement some extra features to enable us

1.1 DESCRIPTION OF SET UP REQUIRED TO BE ABLE TO COMPLETE THE TASK

The following are a brief description of the set up required to complete the task;

1. The first step is to log in to our already created Databricks community edition account.
2. We will then proceed to download the required csv file ‘FaultDataset.csv’ which has been provided for the task.
3. On the Databricks home screen, we select ‘Browse files’ under the Data Import option on the screen, and then navigate to the directory location where the csv file was saved. We then upload the file to Databricks.
4. Once the data has been uploaded, we then create a cluster by clicking on “Create Compute”. Select the most recent runtime.
5. In creating the cluster for this particular task, I selected the standard cluster
6. The final step will be to create a notebook, by clicking on create and selecting notebook. The notebook is titled as defined in the assessment brief’

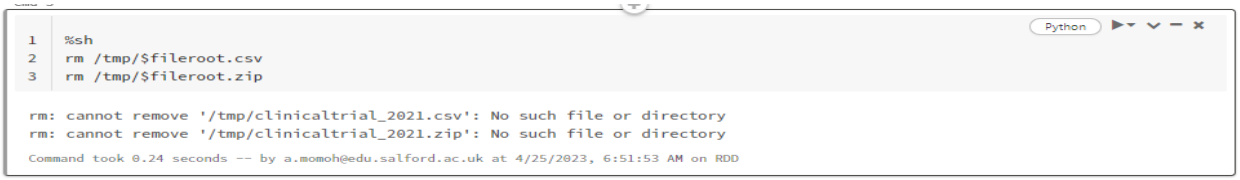
1.2 DATA CLEANING AND PREPARATION (INCLUDING DESCRIPTIONS, JUSTIFICATIONS, AND SCREENSHOTS OF ALL CODE

1. RDD

Below are the steps taken to make the notebook rerunnable alongside the cleaning and preparation? Screenshots of all codes are provided as well.

1. We set the year of the clinical trial to 2021, and create a file root and a file path based on the year. Simply change the year and run all to view details of other year's i.e. 2019 or 2020. The code is set for 2021 as advised in the brief.
2. Import the os library and set the fileroot environment variable.

3. Remove any existing files in the temporary directory, if any.



4. Clean the DBFS (Databricks File System) by removing any existing files related to the data.
5. Copy the required files from the DBFS to the temporary directory.

Cmd 4

```
# Clean the DBFS from the contents that the notebook needs to create again

dbutils.fs.rm("/FileStore/tables/" + fileroot + ".csv", True)
dbutils.fs.rm("/FileStore/tables/pharma.csv", True)
```

```
Out[4]: False
Command took 1.39 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:53 AM on RDD
```

Cmd 5

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

```
Out[5]: True
Command took 5.06 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:53 AM on RDD
```

6. Unzip the files in the temporary directory.

Cmd 6

```
%sh
unzip -d /tmp /tmp/$fileroot.zip
unzip -d /tmp /tmp/pharma.zip
```

```
Archive:  /tmp/clinicaltrial_2021.zip
  inflating: /tmp/clinicaltrial_2021.csv
Archive:  /tmp/pharma.zip
  inflating: /tmp/pharma.csv
Command took 1.14 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:53 AM on RDD
```

7. Move the unzipped files back to the DBFS and list the files in the DBFS directory

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

```
Out[7]: True
Command took 7.82 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:53 AM on RDD
```

Cmd 8

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

```
Out[8]: True
Command took 0.59 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:53 AM on RDD
```

Cmd 9

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

```
Out[9]: [FileInfo(path='dbfs:/FileStore/tables/FaultDataset-1.csv', name='FaultDataset-1.csv', size=1703184, modificationTime
=1678127211000),
 FileInfo(path='dbfs:/FileStore/tables/FaultDataset-2.csv', name='FaultDataset-2.csv', size=1703184, modificationTime=1678127
478000),
 FileInfo(path='dbfs:/FileStore/tables/FaultDataset-3.csv', name='FaultDataset-3.csv', size=1703184, modificationTime=1678128
014000),
 FileInfo(path='dbfs:/FileStore/tables/FaultDataset-4.csv', name='FaultDataset-4.csv', size=1703184, modificationTime=1678128
595000),
 FileInfo(path='dbfs:/FileStore/tables/FaultDataset-5.csv', name='FaultDataset-5.csv', size=1703184, modificationTime=1678132
279000),
 FileInfo(path='dbfs:/FileStore/tables/FaultDataset.csv', name='FaultDataset.csv', size=1703184, modificationTime=16781270970
00),
 FileInfo(path='dbfs:/FileStore/tables/Occupancy_Detection_Data.csv', name='Occupancy_Detection_Data.csv', size=50968, modifi
cationTime=1677962483000),
 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/activations/', name='activations/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/FileStore/tables/activations.zip', name='activations.zip', size=8411369, modificationTime=167580594400
0),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2019.zip', name='clinicaltrial_2019.zip', size=9707871, modificationTime
=1679082724000),
Command took 0.49 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD
```

8. Import required libraries for data processing and visualization.
9. Define a function rddcreator that takes a file name as input and returns an RDD after removing the header row.

10. Create clinicaltrial_rdd by calling the rddcreator function with the clinical trial file path and print the first 5 elements

Cmd 11

```
1  from pyspark.sql import *
2  from pyspark.sql.functions import *
3  import matplotlib.pyplot as plt
```

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

Cmd 12

```
1  # A method for creating rdds
2  def rddcreator(file_name):
3      created_rdd=sc.textFile(file_name)
4      firstrow = created_rdd.first()
5      # removing the header
6      rest_of_rdd = created_rdd.filter(lambda row: row!=firstrow)
7      return rest_of_rdd
```

Command took 0.06 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

Cmd 13

```
1  clinicaltrial_rdd = rddcreator(clinicaltrial_filepath)
```

▸ (1) Spark Jobs

Command took 10.72 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

Cmd 14

```
1  clinicaltrial_rdd.take(5)
```

▸ (1) Spark Jobs

```
Out[14]: ['NCT02758028|The University of Hong Kong|Recruiting|Aug 2005|Nov 2021|Interventional|Apr 2016||',
 'NCT02751957|Duke University|Completed|Jul 2016|Jul 2020|Interventional|Apr 2016|Autistic Disorder,Autism Spectrum Disorder
|',
 'NCT02758483|Universidade Federal do Rio de Janeiro|Completed|Mar 2017|Jan 2018|Interventional|Apr 2016|Diabetes Mellitus|',
 'NCT02759848|Istanbul Medeniyet University|Completed|Jan 2012|Dec 2014|Observational|May 2016|Tuberculosis,Lung Diseases,Pulm
onary Disease|',
 'NCT02758860|University of Roma La Sapienza|Active, not recruiting|Jun 2016|Sep 2020|Observational [Patient Registry]|Apr 201
6|Diverticular Diseases,Diverticulum,Diverticulosis|']
```

Command took 3.04 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

11. Create pharma_rdd by reading the pharma CSV file with a header and inferred schema.
12. Print the first 5 elements of pharma_rdd.

```
pharma_rdd = spark.read.option("header","true").option("inferSchema", "true").csv("/FileStore/tables/pharma.csv").rdd
```

▸ (2) Spark Jobs

Command took 11.47 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

Cmd 16

```
pharma_rdd.take(5)
```

▸ (1) Spark Jobs

```
Out[16]: [Row(Company='Abbott Laboratories', Parent_Company='Abbott Laboratories', Penalty_Amount='$5,475,000', Subtraction_From_Penalty='$0', Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting='$5,475,000', Penalty_Year=2013, Penalty_Date=20131227, Offense_Group='government-contracting-related offenses', Primary_Offense='False Claims Act and related', Secondary_Offense='kickbacks and bribery', Description="Abbott Laboratories agreed to $5.475 million to resolve allegations that it violated the False Claims Act by paying kickbacks to induce doctors to implant the company's carotid, biliary and peripheral vascular products.", Level_of_Government='federal', Action_Type='agency action', Agency='Justice Department Civil Division', Civil/Criminal='civil', Prosecution_Agreement=None, Court=None, Case_ID=None, Private_Litigation_Case_Title=None, Lawsuit_Resolution=None, Facility_State=None, City=None, Address=None, Zip=None, NAICS_Code=None, NAICS_Translation=None, HQ_Country_of_Parent='USA', HQ_State_of_Parent='Illinois', Ownership_Structure='publicly traded', Parent_Company_Stock_Ticker='ABT', Major_Industry_of_Parent='pharmaceuticals', Specific_Industry_of_Parent='pharmaceuticals', Info_Source='https://www.justice.gov/opa/pr/abbott-laboratories-pays-us-5475-million-settle-claims-company-paid-kickbacks-physicians', Notes=None),
 Row(Company='Abbott Laboratories Inc.', Parent_Company='AbbVie', Penalty_Amount='$1,500,000,000', Subtraction_From_Penalty='$0', Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting='$1,500,000,000', Penalty_Year=2012, Penalty_Date=20120507, Offense_Group='healthcare-related offenses', Primary_Offense='off-label or unapproved promotion of medical products', Secondary_Offense=None, Description="Global Health Care Company Abbott Laboratories Inc. has pleaded guilty and agreed to pay $1.5 billion to resolve its criminal and civil liability arising from the company's unlawful promotion of the prescription drug Depakote for uses not approved as safe and effective by the Food and Drug Administration. The resolution - the second largest payment by a drug company - includes a criminal fine and forfeiture totaling $700 million and civil settlements with the federal government and the states totaling $800 million. Abbott also will be subject to court-supervised probation and reporting obligations for Abbotts CEO and Board of Directors.", Level_of_Government='federal', Action_Type='agency action', Agency='Food and Drug Administration referral to the Justice Department', Civil/Criminal='civil and criminal', Prosecution_Agreement=None
```

Command took 1.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

2. DATAFRAME

Cmd 1

```
# Year of clinical trial
ct_year = str(2021)
```

Command took 0.08 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:54:35 AM on RDD

1. We simply set the year of the clinical trial to the year we intend to analyse and run all cells. The year is currently set for 2021 as advised in the brief.
2. Creating the clinicaltrial_df and displaying a few rows.

```
clinicaltrial_df = spark.read.option("header","true").option("inferSchema", "true").option("delimiter", '|').csv("/FileStore/tables/clinicaltrial_" + ct_year + ".csv")
```

(2) Spark Jobs

clinicaltrial_df: pyspark.sql.dataframe.DataFrame

```
Id: string
Sponsor: string
Status: string
Start: string
Completion: string
Type: string
Submission: string
Conditions: string
Interventions: string
```

Command took 6.84 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:54:36 AM on RDD

Cmd 14

```
clinicaltrial_df.show()
```

(1) Spark Jobs

Id	Sponsor	Status	Start	Completion	Type	Submission	Conditions	Interventions
NCT02758028	The University of...	Recruiting	Aug 2005	Nov 2021	Interventional	Apr 2016	null	null
NCT02751957	Duke University	Completed	Jul 2016	Jul 2020	Interventional	Apr 2016	Autistic Disorder...	null
NCT02758483	Universidade Fede...	Completed	Mar 2017	Jan 2018	Interventional	Apr 2016	Diabetes Mellitus	null
NCT02759848	Istanbul Medeniye...	Completed	Jan 2012	Dec 2014	Observational	May 2016	Tuberculosis,Lung...	null
NCT02758860	University of Rom...	Active, not recru...	Jun 2016	Sep 2020	Observational [Pa...	Apr 2016	Diverticular Dise...	null
NCT02757209	Consorzio Futuro ...	Completed	Apr 2016	Jan 2018	Interventional	Apr 2016	Asthma	Fluticasone,Xhanc...
NCT02752438	Ankara University	Unknown status	May 2016	Jul 2017	Observational [Pa...	Apr 2016	Hypoventilation	null
NCT02753543	Ruijin Hospital	Unknown status	Nov 2015	Nov 2019	Interventional	Apr 2016	Lymphoma	null
NCT02757508	Washington Univer...	Completed	Mar 2016	Jul 2017	Interventional	Apr 2016	null	Vitamins
NCT02753530	Orphazyme	Completed	Aug 2017	Jan 2021	Interventional	Apr 2016	Myositis	null
NCT02754817	Novo Nordisk A/S	Completed	Apr 2016	Oct 2016	Observational	Apr 2016	Diabetes Mellitus	Liraglutide,Xultophy
NCT02759276	Daniel Alexandre ...	Completed	May 2015	Dec 2015	Observational	Apr 2016	Hypertension	null
NCT02750956	Bulent Ecevit Uni...	Completed	Jun 2015	Mar 2016	Observational	Apr 2016	Periodontal Diseases	null
NCT02752113	Institut für Phar...	Completed	Apr 2016	May 2019	Interventional	Apr 2016	Diabetes Mellitus	Metformin,Empagli...
NCT02752698	The Third Xiangya...	Active, not recru...	Jan 2015	Dec 2021	Interventional	Jun 2015	Appendicitis,Stom...	null
NCT02755779	Tel Aviv Medical ...	Unknown status	Jun 2016	Jun 2017	Observational	Apr 2016	null	null
NCT02750384	Medicines for Mal...	Terminated	May 2016	Jul 2016	Interventional	Apr 2016	null	null
NCT02754609	James Cook Univer...	Completed	Sep 2016	Oct 2019	Interventional	Apr 2016	Hookworm Infectio...	null

Command took 0.67 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:54:36 AM on RDD

3. Creating the pharma_df

Cmd 15

```
pharma_df = spark.read.option("header","true").option("inferSchema", "true").csv("/FileStore/tables/pharma.csv")
```

(2) Spark Jobs

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

Command took 1.80 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:54:36 AM on RDD

3. SQL

1. We initialize the variable ct_year to store the year of the clinical trial, which is 2021 in this case.

Cmd 1

```python
%python
# Year of clinical trial
ct_year = "2021"
```

Command took 0.08 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

2. We run to code to create the tables same as what we did in RDD.
3. The way Databricks is configured. Variable defined in the cell/notebook of a language is not available in the REPL of another language/another cell. REPLs can share state only through external resources such as files in DBFS or objects in object storage. This means that the year variable defined in RDD file is not available in this SQL notebook. Thus, we need to create the variable again in this notebook. So basically we will need to change the year in two places to be able to run the code in SQL unlike RDD and df

```sql
-- The way Databricks is configured. Variable defined in the cell/notebook of a language is not available in the
REPL of another language/another cell. REPLs can share state only through external resources such as files in DBF
or objects in object storage.
-- This means that the year variable defined in RDD file is not available in this SQL notebook. Thus, we need to
create the variable again in this notebook

SET year=2021;
```

Table +

	key	value
1	year	2021

1 row | 0.13 seconds runtime Refreshed 6 hours ag

Command took 0.13 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 11

```sql
-- Loading the Data
```

Command took 0.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 12

```sql
-- Remove the existing tables and views
```

Command took 0.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 13

```sql
DROP TABLE IF EXISTS clinicaltrial_${hivevar:year};
DROP VIEW IF EXISTS clinicaltrial_view;
DROP VIEW IF EXISTS conditions_view;
DROP VIEW IF EXISTS sponsors_view;
DROP VIEW IF EXISTS completion_view;
```

OK

Command took 2.66 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

4. We then proceed to load the data and remove the existing clinicaltrial tables and views as seen in the code above.

5. We also drop from the pharma if it exists
6. We proceed to create the clinicaltrial table and load the path to overwrite into the table.

Cmd 14

```
DROP TABLE IF EXISTS pharma;
DROP VIEW IF EXISTS pharma_view;
```

OK

Command took 0.68 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 15

```
-- Creating clinical trial table, if it does not exist
```

Command took 0.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 16

```
CREATE TABLE IF NOT EXISTS clinicaltrial_${hivevar:year}(
  Id STRING,
  Sponsor STRING,
  Status STRING,
  Start STRING,
  Completion STRING,
  Type STRING,
  Submission STRING,
  Conditions STRING,
  Interventions STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LOCATION '/FileStore/clinicaltrial_tables';
```

OK

Command took 0.30 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

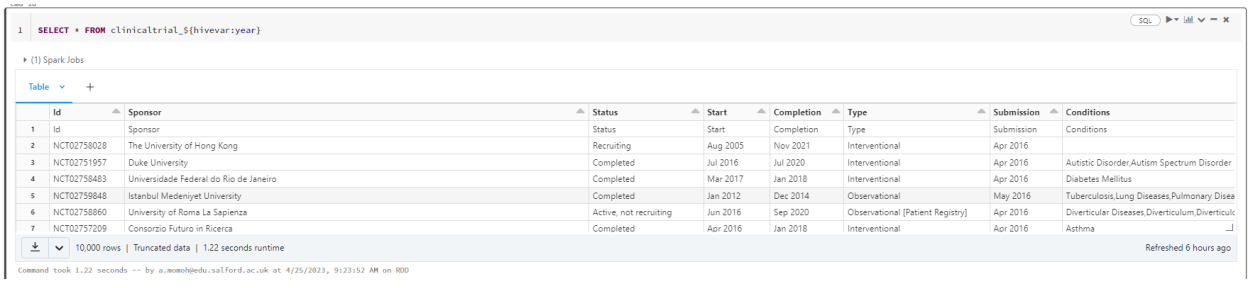
Cmd 17

```
LOAD DATA INPATH '/FileStore/tables/clinicaltrial_${hivevar:year}.csv' OVERWRITE INTO TABLE
clinicaltrial_${hivevar:year};
```

OK

Command took 2.10 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

7. We view the first few rows of the clinicaltrial dataset.



```
SELECT * FROM clinicaltrial_${hivevar:year}
```

(1) Spark Jobs

Table

	Id	Sponsor	Status	Start	Completion	Type	Submission	Conditions
1	Id	Sponsor	Status	Start	Completion	Type	Submission	Conditions
2	NCT02758028	The University of Hong Kong	Recruiting	Aug 2005	Nov 2021	Interventional	Apr 2016	
3	NCT02751957	Duke University	Completed	Jul 2016	Jul 2020	Interventional	Apr 2016	Autistic Disorder,Autism Spectrum Disorder
4	NCT02758483	Universidade Federal do Rio de Janeiro	Completed	Mar 2017	Jan 2018	Interventional	Apr 2016	Diabetes Mellitus
5	NCT02759848	Istanbul Medeniyet University	Completed	Jan 2012	Dec 2014	Observational	May 2016	Tuberculosis,Lung Diseases,Pulmonary Disea
6	NCT02758860	University of Roma La Sapienza	Active, not recruiting	Jun 2016	Sep 2020	Observational [Patient Registry]	Apr 2016	Diverticular Diseases,Diverticulum,Diverticulc
7	NCT02757209	Consorzio Futuro in Ricerca	Completed	Apr 2016	Jan 2018	Interventional	Apr 2016	Asthma

10,000 rows | Truncated data | 1.22 seconds runtime

Refreshed 6 hours ago

Command took 1.22 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

8. Same as what we did for clinicaltrial, we will now create the pharma table and load the path to overwrite into the table.

```
-- Creating pharma table, if it does not exist
```

Command took 0.01 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 20

```
CREATE TABLE IF NOT EXISTS pharma(
  Company STRING, Parent_Company STRING, Penalty_Amount STRING,
  Subtraction_From_Penalty STRING, Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting STRING,
  Penalty_Year STRING, Penalty_Date STRING, Offense_Group STRING, Primary_Offense STRING,
  Secondary_Offense STRING, Description STRING, Level_of_Government STRING, Action_Type STRING,
  Agency STRING, `Civil/Criminal` STRING, Prosecution_Agreement STRING, Court STRING,
  Case_ID STRING, Private_Litigation_Case_Title STRING, Lawsuit_Resolution STRING,
  Facility_State STRING, City STRING, Address STRING, Zip STRING, NAICS_Code STRING,
  NAICS_Translation STRING, HQ_Country_of_Parent STRING, HQ_State_of_Parent STRING,
  Ownership_Structure STRING, Parent_Company_Stock_Ticker STRING, Major_Industry_of_Parent STRING,
  Specific_Industry_of_Parent STRING, Info_Source STRING, Notes STRING
  )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/FileStore/pharma_tables';
```

OK

Command took 0.33 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 21

```
LOAD DATA INPATH '/FileStore/tables/pharma.csv' OVERWRITE INTO TABLE pharma;
```

OK

Command took 1.50 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

9. We explore the first few roles of the pharma data

```
SELECT * FROM pharma
```

(1) Spark Jobs

Table

	Company	Parent_Company	Penalty_Amount	Subtraction_From_Penalty	Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting	Pe
1	"Company"	"Parent_Company"	"Penalty_Amount"	"Subtraction_From_Penalty"	"Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting"	"Pe
2	"Abbott Laboratories"	"Abbott Laboratories"	"$5	475	000"	"$
3	"Abbott Laboratories Inc."	"AbbVie"	"$1	500	000	00
4	"Abbott Laboratories Inc."	"AbbVie"	"$126	500	000"	"$
5	"Abbott Laboratories Puerto Rico	Inc."	"Abbott Laboratories"	"$49	045"	"$
6	"Acclarent Inc."	"Johnson & Johnson"	"$18	000	000"	"$
7	"Advanced Medical Optics"	"Abbott Laboratories"	"$16	800"	"$0"	"$

969 rows | 0.64 seconds runtime

Refreshed 6 hours ago

Command took 0.64 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

10. Now we will create temporary views for the tables and explore a few rows.

```
1  -- Creating temporary views from tables created
```

Command took 0.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 24

```
1  -- Creating the view for clinical trial data
2  CREATE TEMPORARY VIEW clinicaltrial_view
3  AS SELECT * FROM clinicaltrial_${hivevar:year} WHERE Id <> 'Id';
```

OK

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 25

```
1  SELECT * FROM clinicaltrial_view
```

(1) Spark Jobs

Table +

	Id	Sponsor	Status
1	NCT02758028	The University of Hong Kong	Recruiting
2	NCT02751957	Duke University	Complete
3	NCT02758483	Universidade Federal do Rio de Janeiro	Complete
4	NCT02759848	Istanbul Medeniyet University	Complete
5	NCT02758860	University of Roma La Sapienza	Active, no
6	NCT02757209	Consorzio Futuro in Ricerca	Complete
7	NCT02752438	Ankara University	Unknown

10,000 rows | Truncated data | 0.96 seconds runtime Refreshed 6 hours ago

Command took 0.96 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

Cmd 26

```
1  -- Creating the view for pharma data
2  -- Selecting only the two important coloumns
3  CREATE TEMPORARY VIEW pharma_view
4  AS SELECT REGEXP_REPLACE(Company, '["]','') AS Company, REGEXP_REPLACE(Parent_Company, '["]','') AS Parent_Company
5  FROM pharma WHERE Company != '"Company"';
```

OK

Command took 0.10 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

```
1  SELECT * FROM pharma_view
```

SQL

(1) Spark Jobs

Table +

	Company	Parent_Company
1	Abbott Laboratories	Abbott Laboratories
2	Abbott Laboratories Inc.	AbbVie
3	Abbott Laboratories Inc.	AbbVie
4	Abbott Laboratories Puerto Rico	Inc.
5	Acclarent Inc.	Johnson & Johnson
6	Advanced Medical Optics	Abbott Laboratories
7	Advanced Neuromodulation Systems	Inc.

968 rows | 0.68 seconds runtime Refreshed 6 hours ago

Command took 0.68 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:52 AM on RDD

1.3 QUESTION 1: The number of studies in the dataset. You must ensure that you explicitly check distinct studies.

1. Assumptions made on the dataset before answering this question:

It is assumed that the dataset have duplicate values hence why we are required to check for distinct studies.

2. PySpark implementation outline in RDD:

```
1  # Question 1. To find the number of Distinct studies
```

Python

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

Cmd 18

```
1  clinicaltrial_rdd.distinct().count()
```

Python

(1) Spark Jobs

Out[18]: 387261

Command took 6.55 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

The code clinicaltrial_rdd.distinct().count() is an operation performed on an RDD called clinicaltrial_rdd. The purpose of this operation is to find the number of distinct records in the dataset.

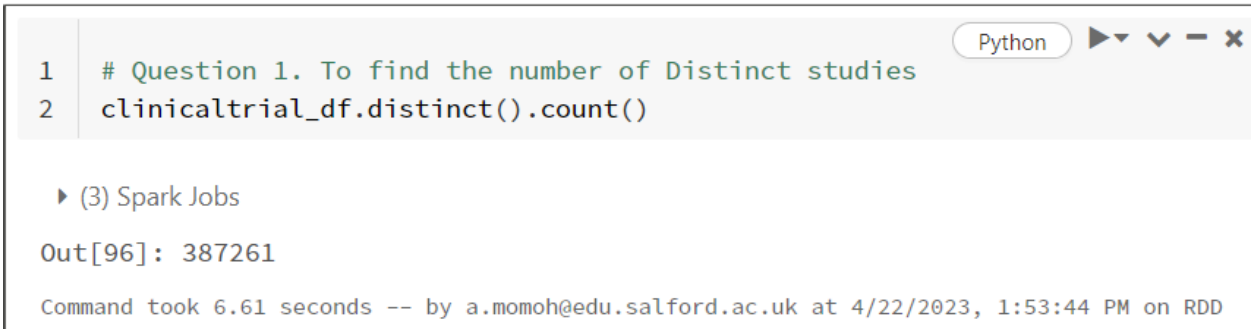
Here is a breakdown of the implementation.

clinicaltrial_rdd: This is the RDD containing the clinical trial data. Each record in this RDD represents a unique clinical trial.

.distinct(): This method is called on the RDD to remove duplicate records. It returns a new RDD containing only the distinct records from the original RDD. If there are any duplicate records in clinicaltrial_rdd, they will be removed in the output RDD.

.count(): This method is called on the distinct RDD to count the number of records in the RDD. Since the RDD now contains only distinct records, the count represents the number of unique clinical trials in the dataset.

3. PySpark implementation outline in Dataframe:



The code clinicaltrial_df.distinct().count() is an operation performed on a DataFrame called clinicaltrial_df. The purpose of this operation is to find the number of distinct records in the dataset.

Here's a breakdown of the implementation.
clinicaltrial_df: This is the DataFrame containing the clinical trial data. Each row in this DataFrame represents a unique clinical trial.

.distinct(): This method is called on the DataFrame to remove duplicate rows. It returns a new DataFrame containing only the distinct rows from the original DataFrame. If there are any duplicate rows in clinicaltrial_df, they will be removed in the output DataFrame.

.count(): This method is called on the distinct DataFrame to count the number of rows in the DataFrame. Since the DataFrame now contains only distinct rows, the count represents the number of unique clinical trials in the dataset.
In summary, the code clinicaltrial_df.distinct().count() calculates the number of distinct clinical trials in the dataset by removing duplicate rows and counting the remaining unique rows.

4. SQL implementation outline

```
1   -- Question 1. To find the number of Distinct studies
```

Command took 0.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

Cmd 29

```
1   -- Result
2   SELECT DISTINCT COUNT(*) AS TotalCount FROM clinicaltrial_view;
```

SQL

▸ (2) Spark Jobs

Table +

	TotalCount
1	387261

1 row | 1.43 seconds runtime Refreshed 5 hours ago

Command took 1.43 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

This query will first find all distinct rows in the clinicaltrial_view table and then count them. The result will be returned as a single-row table with the column name TotalCount, representing the total number of distinct clinical trials in the dataset.

5. Discussion of result:

The result obtained from the PySpark implementation and Spark SQL will provide the total number of distinct clinical trials in the dataset which is 387261. This information is essential for understanding the overall scope and variety of clinical trials conducted in the field.

1.4 QUESTION 2: Types of studies in the dataset alongside the frequencies ordered from most frequent to least frequent

1. Assumptions made on the dataset before answering this question:

We assumed that the dataset contains a column with the type of each study and that the study has only one status value.

2. PySpark implementation outline in RDD:

   First we will split the clinical trial data by delimiter

```
Python
1  # Splitting the clinical trial data by delimeter
2  clinicaltrial_rdd = clinicaltrial_rdd.map(lambda line: line.split('|'))

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
Cmd 20
```

This line of code takes the original 'clinicaltrial_rdd' RDD object and applies a map transformation to split each line of the dataset by the delimiter '|'. The result is an RDD where each element is a list of strings representing the columns of the dataset.

```
Python
1  #Types of studies and their frequencies
2  types_rdd = clinicaltrial_rdd.map(lambda line: (line[5], 1))
3  types_rdd.take(5)

▸ (1) Spark Jobs
Out[21]: [('Interventional', 1),
 ('Interventional', 1),
 ('Interventional', 1),
 ('Observational', 1),
 ('Observational [Patient Registry]', 1)]
Command took 1.60 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

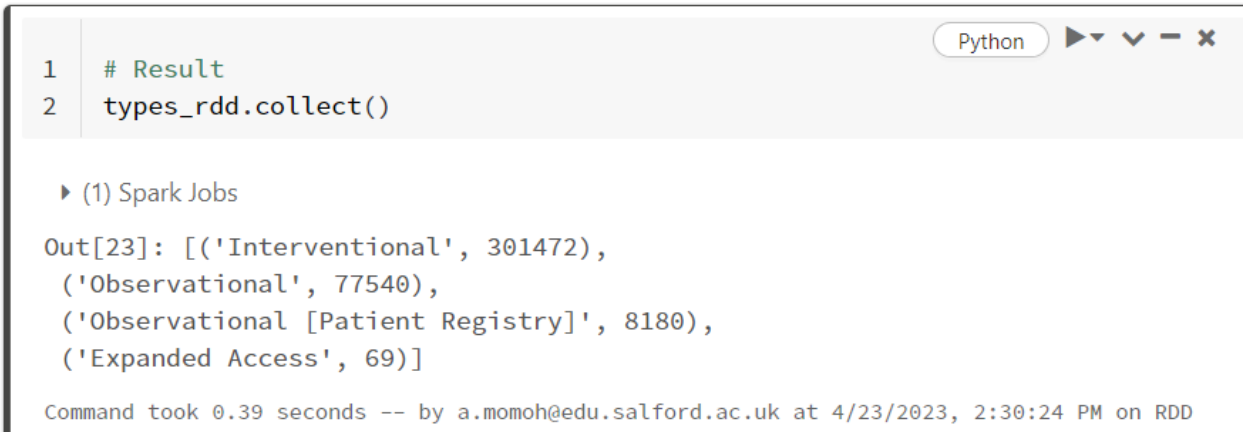
We then proceed to create a new RDD called 'types_rdd by applying another map transformation on the clinicatrial_rdd. This selects the type of study (which is the 6th column or index 5) from each line and creates a tuple with the study type and a value of 1. The purpose of adding 1 is to count the occurrences of each study type in the next steps. We also printed the first 5 types for a quick inspection.

```
Python
1  types_rdd = types_rdd.reduceByKey(lambda accumulation,current:
   accumulation + current).sortBy(lambda row: -row[1], ascending=True)

▸ (2) Spark Jobs
Command took 3.12 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

Now that we have found the types of studies, we need to find their corresponding frequencies. To do this, we apply two transformation to the types_rdd; as seen in the code above.

reduceByKey: This transformation groups the RDD elements by the study type (the first element in each tuple) and applies a reduce function (in this case, a summation) to the second element (the counts) of each tuple. This will effectively compute the frequency of each study type in the dataset.

sortBy: This transformation sorts the RDD elements based on the frequencies (the second element in each tuple) in descending order (by using -row [1]).



Finally, the collect() action is called on the types_rdd to return the resulting list of tuples, containing the study types and their corresponding frequencies, sorted in descending order by frequency.

3. PySpark implementation outline in Dataframe:

In Dataframe, to list the types and their frequencies, we can use the groupBy() and count() functions in PySpark. These functions allow you to group the data by intervention type and count the number of occurrences for each group.

```
# Question 2. To list all the types and frequency

clinicaltrial_df.groupBy("Type").count().orderBy("count",
ascending=False).show(truncate=False)
```

▸ (2) Spark Jobs

```
+-----------------------------------+------+
|Type                               |count |
+-----------------------------------+------+
|Interventional                     |301472|
|Observational                      |77540 |
|Observational [Patient Registry]   |8180  |
|Expanded Access                    |69    |
+-----------------------------------+------+
```

Command took 6.65 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

Here is a breakdown of the code;

clinicaltrial_df.groupBy("Type"): This operation groups the DataFrame clinicaltrial_df by the "Type" column. This will create groups of rows with the same study type.
.count(): This operation counts the number of rows in each group created in the previous step. It will result in a new DataFrame with two columns: "Type" and "count". The "Type" column contains the distinct study types, and the "count" column contains the number of occurrences (or frequencies) of each study type in the dataset.
.orderBy("count"): This operation sorts the resulting DataFrame by the "count" column in ascending order (by default).
.show(truncate=False): This operation displays the sorted DataFrame with study types and their corresponding frequencies. The truncate=False parameter ensures that the full content of each cell is displayed without truncation.

In summary, this line of code groups the dataset by the study type, calculates the frequency of each study type, sorts the results in ascending order by frequency, and displays the output.

4. SQL implementation outline:

To carry out the implementation using Spark SQL, We will groups the dataset by the "Type" column. Then, we will count the number of rows in each group using the COUNT(*) function and assigns this count to a new column called "Frequency".
Finally, we order the resulting groups by the "Frequency" column in descending order using the ORDER BY Frequency DESC clause.

```
1  -- Result
2  SELECT Type, COUNT(*) AS Frequency FROM clinicaltrial_view GROUP BY Type
   ORDER BY Frequency DESC;
```

SQL

▸ (2) Spark Jobs

Table +

	Type	Frequency
1	Interventional	301472
2	Observational	77540
3	Observational [Patient Registry]	8180
4	Expanded Access	69

4 rows | 2.24 seconds runtime Refreshed yesterday

Command took 2.24 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

5. Discussion of result:

The result displays a list of study types and their corresponding frequencies in the dataset. This information is useful for understanding the most common intervention types used in clinical trials and identifying potential trends in the field.

1.5 QUESTION 3: The top 5 conditions (from Conditions) with their frequencies.

1. Assumptions made on the dataset before answering this question:
   We assumed that each condition is comma separated in the conditions column.
   We assumed it didn't have another delimiter separating the data in the column.

2. PySpark implementation outline in RDD:
   Similar to what we did in question 2, here we will Map the RDD only to contain the conditions and use the reduceByKey to count the frequency of each condition and finally sort the results by frequency and take the top 5 conditions.

```
1  conditions_rdd = clinicaltrial_rdd.flatMap(lambda line:
   line[7].split(",")).map(lambda line: (line,1)).filter(lambda line:
   line[0])
```

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

Cmd 26

```
1  conditions_rdd.take(5)
```

▸ (1) Spark Jobs

```
Out[26]: [('Autistic Disorder', 1),
 ('Autism Spectrum Disorder', 1),
 ('Diabetes Mellitus', 1),
 ('Tuberculosis', 1),
 ('Lung Diseases', 1)]
```

Command took 1.20 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

flatMap: Takes each input record and splits the value in the "Conditions" column (indexed at position 7) by comma. This creates a new RDD containing the individual conditions as separate records. Since flatMap is used, the resulting RDD is flattened and each condition becomes a separate entry.

map: Transforms each condition entry into a key-value pair, where the key is the condition itself and the value is 1. This is done in preparation for counting the occurrences of each condition.

filter: Removes any empty string entries from the RDD by checking if the key (the condition) is not empty using lambda line: line[0].

```
1  # Result
2  conditions_rdd = conditions_rdd.reduceByKey(lambda accumulation,current:
   accumulation + current).sortBy(lambda row: -row[1])
3  conditions_rdd.take(5)
```

▸ (3) Spark Jobs

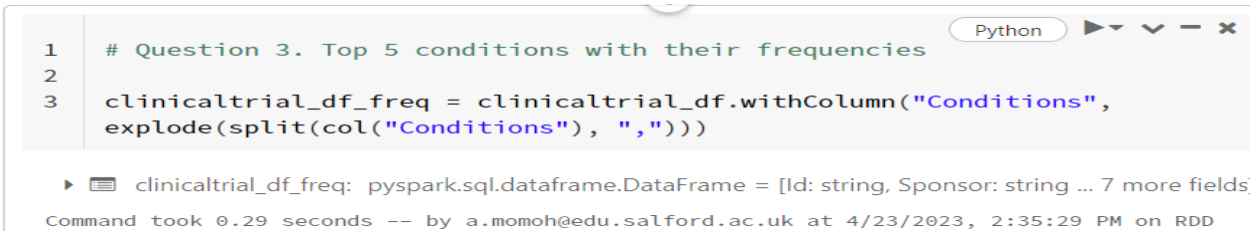
```
Out[27]: [('Carcinoma', 13389),
 ('Diabetes Mellitus', 11080),
 ('Neoplasms', 9371),
 ('Breast Neoplasms', 8640),
 ('Syndrome', 8032)]
```

Command took 4.41 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

reduceByKey: Groups the key-value pairs in conditions_rdd by their keys (the conditions) and sums their values (the count of occurrences). This results in an RDD containing each unique condition and its frequency in the dataset.

sortBy: Sorts the resulting RDD in descending order based on the frequency (count) of each condition. The lambda row: -row[1] function is used to negate the frequency values, which makes the sortBy function sort in descending order and the we take(5): This retrieves the top 5 records from the sorted RDD, which represent the top 5 conditions with the highest frequencies.

3. PySpark implementation outline in Dataframe:
   In DataFrame, to find the top 5 conditions with their frequencies, you can use the groupBy(), count(), and orderBy() functions in PySpark. These functions allow you to group the data by condition, count the number of occurrences for each group, and sort the results in descending order.



   To achieve, we first use withColumn: To add a new column to the clinicaltrial_df DataFrame. The new column is called "Conditions".
   explode(split(col("Conditions"), ",")): The split function is used to split the "Conditions" column of the original DataFrame using the comma (,) as a delimiter. This creates an array of conditions for each row.
   explode: The explode function is then used to create a new row for each condition in the array generated by the split function. In other words, the explode function "flattens" the array of conditions and creates separate rows for each condition in the "Conditions" column of the new DataFrame.

   Upon executing the code we will have now created a new DataFrame called clinicaltrial_df_freq with an additional "Conditions" column containing individual conditions from the original "Conditions" column, separated into different rows. We preview the newly created dataframe as seen below;

```
clinicaltrial_df_freq.show()
```

(1) Spark Jobs

```
+----------+--------------------+--------------------+---------+----------+--------------------+----------+--------------------+--------------------+
|        Id|             Sponsor|              Status|    Start|Completion|                Type|Submission|          Conditions|       Interventions|
+----------+--------------------+--------------------+---------+----------+--------------------+----------+--------------------+--------------------+
|NCT02751957|     Duke University|           Completed|Jul 2016|  Jul 2020|      Interventional|  Apr 2016|   Autistic Disorder|                null|
|NCT02751957|     Duke University|           Completed|Jul 2016|  Jul 2020|      Interventional|  Apr 2016|Autism Spectrum D...|                null|
|NCT02758483|Universidade Fede...|           Completed|Mar 2017|  Jan 2018|      Interventional|  Apr 2016|   Diabetes Mellitus|                null|
|NCT02759848|Istanbul Medeniye...|           Completed|Jan 2012|  Dec 2014|       Observational|  May 2016|        Tuberculosis|                null|
|NCT02759848|Istanbul Medeniye...|           Completed|Jan 2012|  Dec 2014|       Observational|  May 2016|       Lung Diseases|                null|
|NCT02759848|Istanbul Medeniye...|           Completed|Jan 2012|  Dec 2014|       Observational|  May 2016|   Pulmonary Disease|                null|
|NCT02758860|University of Rom...|Active, not recru...|Jun 2016|  Sep 2020|Observational [Pa...|  Apr 2016|Diverticular Dise...|                null|
|NCT02758860|University of Rom...|Active, not recru...|Jun 2016|  Sep 2020|Observational [Pa...|  Apr 2016|        Diverticulum|                null|
|NCT02758860|University of Rom...|Active, not recru...|Jun 2016|  Sep 2020|Observational [Pa...|  Apr 2016|      Diverticulosis|                null|
|NCT02757209|Consorzio Futuro ...|           Completed|Apr 2016|  Jan 2018|      Interventional|  Apr 2016|              Asthma|Fluticasone,Xhanc...|
|NCT02752438|   Ankara University|     Unknown status|May 2016|  Jul 2017|Observational [Pa...|  Apr 2016|     Hypoventilation|                null|
|NCT02753543|     Ruijin Hospital|     Unknown status|Nov 2015|  Nov 2019|      Interventional|  Apr 2016|            Lymphoma|                null|
|NCT02753530|          Orphazyme|           Completed|Aug 2017|  Jan 2021|      Interventional|  Apr 2016|            Myositis|                null|
|NCT02754817|   Novo Nordisk A/S|           Completed|Apr 2016|  Oct 2016|       Observational|  Apr 2016|   Diabetes Mellitus|Liraglutide,Xultophy|
|NCT02759276|Daniel Alexandre ...|           Completed|May 2015|  Dec 2015|       Observational|  Apr 2016|        Hypertension|                null|
|NCT02750956|Bulent Ecevit Uni...|           Completed|Jun 2015|  Mar 2016|       Observational|  Apr 2016|Periodontal Diseases|                null|
|NCT02752113|Institut für Phar...|           Completed|Apr 2016|  May 2019|      Interventional|  Apr 2016|   Diabetes Mellitus|Metformin,Empagli...|
|NCT02752698|The Third Xiangya...|Active, not recru...|Jan 2015|  Dec 2021|      Interventional|  Jun 2015|        Appendicitis|                null|
+----------+--------------------+--------------------+---------+----------+--------------------+----------+--------------------+--------------------+
Command took 0.59 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

Finally we now use the groupBy, count and orderBy functions in PySpark to group the data by condition, count the number of occurrences for each group, and sort the results in descending order.

```
# Result
clinicaltrial_df_freq.groupBy("Conditions").count().orderBy("count",ascending=False).show(5,truncate=False)
```

(2) Spark Jobs

```
+------------------+-----+
|Conditions        |count|
+------------------+-----+
|Carcinoma         |13389|
|Diabetes Mellitus|11080|
|Neoplasms         |9371 |
|Breast Neoplasms |8640 |
|Syndrome          |8032 |
+------------------+-----+
only showing top 5 rows

Command took 5.42 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

4. SQL Implementation

   First we create a temporary view by exploding the Conditions column into separate rows, then use a GROUP BY query to count the frequency of each condition and select the top 5;

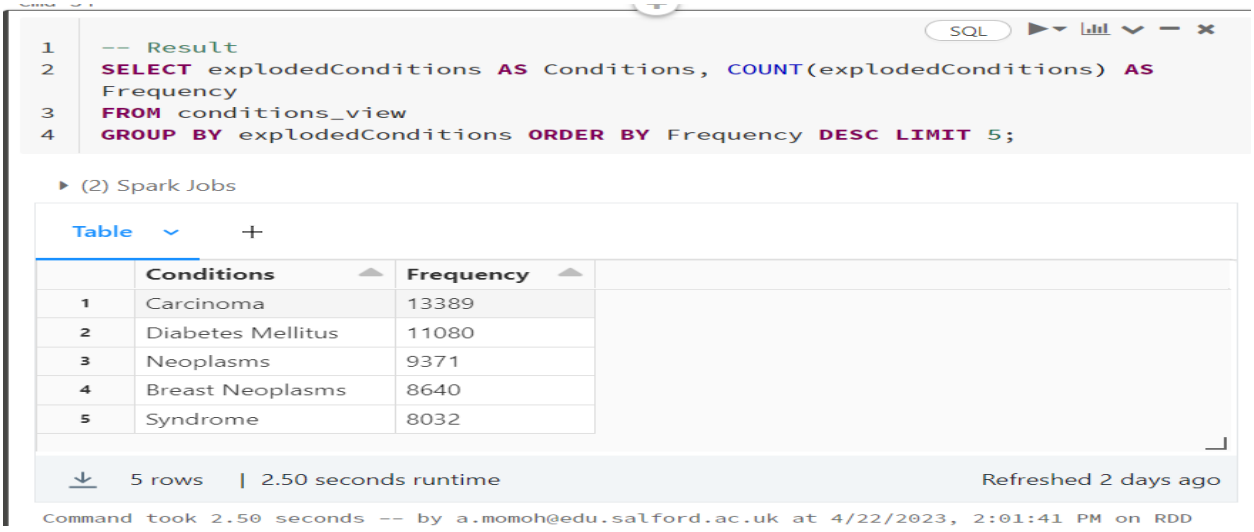
```
-- Exploding Conditions from clinical trial data to separate rows
CREATE TEMPORARY VIEW conditions_view
AS
SELECT *, explode(split(Conditions, ',')) AS explodedConditions
FROM clinicaltrial_view WHERE Conditions <> '';
```

OK

```
Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD
```

CREATE TEMPORARY VIEW conditions_view: Creates a temporary view with the name conditions_view. A temporary view is a named query result that can be used for further SQL queries as if it were a table.
AS SELECT *: This part selects all the columns from the original table clinicaltrial_view. explode(split(Conditions, ',')) AS explodedConditions: The split function is used to split the "Conditions" column of the clinicaltrial_view table using the comma (,) as a delimiter, creating an array of conditions. The explode function is then used to create a new row for each condition in the array, "flattening" the array of conditions into separate rows. The new column with these individual conditions is given the name explodedConditions.
FROM clinicaltrial_view WHERE Conditions <> '': This specifies the source table clinicaltrial_view and filters out any rows where the "Conditions" column is empty.

Finally, we then use a GROUP BY query to count the frequency of each condition and select the top 5.



5. Discussion of result

The result obtained from the PySpark implementation and Spark SQL will provide a list of the top 5 conditions and their frequencies. This information is valuable for identifying the most prevalent conditions studied in clinical trials, which may highlight areas of significant research interest and potential unmet medical needs.

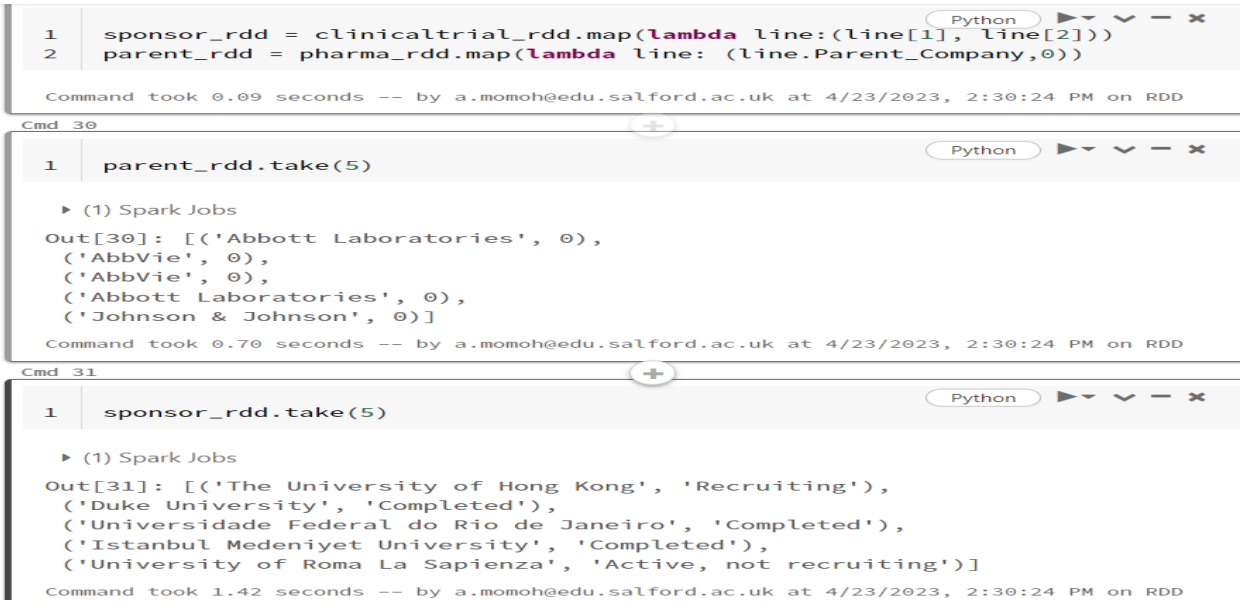
1.6 QUESTION 4: 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

1. Assumptions made on the dataset before answering this question:

Considering the pharma data contains only pharmaceutical companies or sponsors, We assumed that non-pharmaceutical sponsors are those that are not in the pharma data. Since the question said "--with the number of clinical trials they have sponsored", We then assumed that it means all non-active trials, not just the completed ones, all the trials that are not active or ongoing.

2. PySpark implementation outline in RDD:

To carry out this implementation, we map the RDD to only contain the sponsors
Filter out pharmaceutical companies by performing a left outer join with the pharmaceutical company's dataset.
Use reduceByKey() to count the frequency of each sponsor and sort the results by frequency and take the top 10 sponsors.



```
Python
1  sponsor_rdd = clinicaltrial_rdd.map(lambda line:(line[1], line[2]))
2  parent_rdd = pharma_rdd.map(lambda line: (line.Parent_Company,0))

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

Cmd 30

Python
1  parent_rdd.take(5)

▸ (1) Spark Jobs
Out[30]: [('Abbott Laboratories', 0),
 ('AbbVie', 0),
 ('AbbVie', 0),
 ('Abbott Laboratories', 0),
 ('Johnson & Johnson', 0)]
Command took 0.70 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

Cmd 31

Python
1  sponsor_rdd.take(5)

▸ (1) Spark Jobs
Out[31]: [('The University of Hong Kong', 'Recruiting'),
 ('Duke University', 'Completed'),
 ('Universidade Federal do Rio de Janeiro', 'Completed'),
 ('Istanbul Medeniyet University', 'Completed'),
 ('University of Roma La Sapienza', 'Active, not recruiting')]
Command took 1.42 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

First we create a sponsor_rdd = clinicaltrial_rdd. Map(lambda line:(line[1], line[2])): Creates an RDD named sponsor_rdd by extracting the Sponsor and Status columns from the clinicaltrial_rdd.

parent_rdd = pharma_rdd.map(lambda line: (line.Parent_Company,0)): Creates an RDD named parent_rdd by extracting the Parent_Company column from the pharma_rdd.

```
1  pharma_sponsor_rdd = sponsor_rdd.leftOuterJoin(parent_rdd).filter(lambda
   line: line[1][1]==None and line[1][0]!="Active")
2  pharma_sponsor_rdd = pharma_sponsor_rdd.map(lambda line:
   (line[0],1)).reduceByKey(lambda accumulation,current: accumulation +
   current).sortBy(lambda row: -row[1])
```

Python

▸ (2) Spark Jobs

Command took 7.11 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

sponsor_rdd.leftOuterJoin(parent_rdd): Performs a left outer join between the sponsor_rdd and parent_rdd RDDs based on the Sponsor and Parent_Company columns, respectively.

.filter(lambda line: line[1][1]==None and line[1][0]!="Active"): Filters the resulting RDD to only include rows where the Parent_Company is not a pharmaceutical company (i.e., has a None value) and the Status is not "Active".

.map(lambda line:(line[0],1)): Transforms the filtered RDD by keeping the Sponsor as the key and setting the value to 1 for each row.

.reduceByKey(lambda accumulation,current: accumulation + current): Aggregates the RDD by the Sponsor key, summing up the values (which is counting the number of trials) for each unique Sponsor.

.sortBy(lambda row: -row[1]): Sorts the RDD by the count in descending order.

Finally, we .take(10) Returns the top 10 sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

```
1  # Result
2  pharma_sponsor_rdd.take(10)
```

Python

▸ (1) Spark Jobs

```
Out[33]: [('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 0.49 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD

3. PySpark implementation outline in DataFrame:

To find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored, you can use the groupBy(),

count(), orderBy(), and filter() functions in PySpark. These functions allow you to group the data by sponsor name, count the number of occurrences for each group, sort the results in descending order, and filter out pharmaceutical companies.

We will first perform a left anti join between ‘clinicaltrial_df’ and ‘pharma_df’ DataFrame using SparkSQL.

```
Python
1  # Question 4. 10 most common sponsors that are not pharmaceutical
   companies, along with the number of clinical trials they have sponsored.
2
3  nonpharma_sponsor = clinicaltrial_df.join(pharma_df,
   pharma_df.Parent_Company == clinicaltrial_df.Sponsor, "leftanti")
```

```
▸ nonpharma_sponsor: pyspark.sql.dataframe.DataFrame = [Id: string, Sponsor: string ... 7 more fields]
Command took 0.19 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

The code above creates a new DataFrame called nonpharma_sponsor. It joins the clinicaltrial_df and pharma_df DataFrames based on the condition that the Parent_Company column from pharma_df matches the Sponsor column from clinicaltrial_df. The "leftanti" join type is used, which means that the resulting DataFrame will only include rows from clinicaltrial_df that do not have a match in pharma_df (i.e., the sponsors that are not pharmaceutical companies).

```
Python
1  nonpharma_sponsor.show()
```

▸ (2) Spark Jobs

Id	Sponsor	Status	Start	Completion	Type	Submission	Conditions	Interventions
NCT02758028	The University of...	Recruiting	Aug 2005	Nov 2021	Interventional	Apr 2016	null	null
NCT02751957	Duke University	Completed	Jul 2016	Jul 2020	Interventional	Apr 2016	Autistic Disorder...	null
NCT02758483	Universidade Fede...	Completed	Mar 2017	Jan 2018	Interventional	Apr 2016	Diabetes Mellitus	null
NCT02759848	Istanbul Medeniye...	Completed	Jan 2012	Dec 2014	Observational	May 2016	Tuberculosis,Lung...	null
NCT02758860	University of Rom...	Active, not recru...	Jun 2016	Sep 2020	Observational [Pa...	Apr 2016	Diverticular Dise...	null
NCT02757209	Consorzio Futuro ...	Completed	Apr 2016	Jan 2018	Interventional	Apr 2016	Asthma	Fluticasone,Xhanc...
NCT02752438	Ankara University	Unknown status	May 2016	Jul 2017	Observational [Pa...	Apr 2016	Hypoventilation	null
NCT02753543	Ruijin Hospital	Unknown status	Nov 2015	Nov 2019	Interventional	Apr 2016	Lymphoma	null
NCT02757508	Washington Univer...	Completed	Mar 2016	Jul 2017	Interventional	Apr 2016	null	Vitamins
NCT02753530	Orphazyme	Completed	Aug 2017	Jan 2021	Interventional	Apr 2016	Myositis	null
NCT02754817	Novo Nordisk A/S	Completed	Apr 2016	Oct 2016	Observational	Apr 2016	Diabetes Mellitus	Liraglutide,Xultophy
NCT02759276	Daniel Alexandre ...	Completed	May 2015	Dec 2015	Observational	Apr 2016	Hypertension	null
NCT02750956	Bulent Ecevit Uni...	Completed	Jun 2015	Mar 2016	Observational	Apr 2016	Periodontal Diseases	null
NCT02752113	Institut für Phar...	Completed	Apr 2016	May 2019	Interventional	Apr 2016	Diabetes Mellitus	Metformin,Empagli...
NCT02752698	The Third Xiangya...	Active, not recru...	Jan 2015	Dec 2021	Interventional	Jun 2015	Appendicitis,Stom...	null
NCT02755779	Tel Aviv Medical ...	Unknown status	Jun 2016	Jun 2017	Observational	Apr 2016	null	null
NCT02750384	Medicines for Mal...	Terminated	May 2016	Jul 2016	Interventional	Apr 2016	null	null
NCT02754609	James Cook Univer...	Completed	Sep 2016	Oct 2019	Interventional	Apr 2016	Hookworm Infectio...	null

Command took 1.60 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

As seen above, the resulting nonpharma_sponsor DataFrame contains all the rows from clinicaltrial_df where the Sponsor is not a pharmaceutical company.

Finally, The code below filters the nonpharma_sponsor DataFrame to exclude rows with "Active" status and then groups the rows by the "Sponsor" column, counting the number

of rows per sponsor. It then sorts the results in descending order by the count and displays the top 10 results.

```python
# Result
nonpharma_sponsor.filter(nonpharma_sponsor.Status
!="Active").groupBy("Sponsor").count().orderBy("count",ascending=False).sh
ow(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 6.12 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

4. SQL Implementation:

We create a temporary view that excludes studies sponsored by pharmaceutical companies.

```sql
CREATE TEMPORARY VIEW sponsors_view
AS
SELECT * FROM clinicaltrial_view
LEFT OUTER JOIN pharma_view
ON pharma_view.Parent_Company = clinicaltrial_view.Sponsor
WHERE clinicaltrial_view.Status <> "Active"
AND pharma_view.Parent_Company IS NULL;
```

OK

```
Command took 0.46 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD
```

Finally, we use a GROUP BY query to count the frequency of each sponsor and select the top 10 as seen below. Please note that the screenshot only shows the first 7.

```sql
-- Result
SELECT Sponsor, COUNT(Sponsor) AS Frequency
FROM sponsors_view
GROUP BY Sponsor ORDER BY Frequency DESC LIMIT 10;
```

(4) Spark Jobs

Table +

	Sponsor	Frequency
1	National Cancer Institute (NCI)	3218
2	M.D. Anderson Cancer Center	2414
3	Assistance Publique - Hôpitaux de Paris	2369
4	Mayo Clinic	2300
5	Merck Sharp & Dohme Corp.	2243
6	Assiut University	2154
7	Novartis Pharmaceuticals	2088

10 rows | 2.69 seconds runtime Refreshed 2 days ago

```
Command took 2.69 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD
```

5. Discussion of result:

The result obtained from the PySpark implementation and Spark SQL will provide a list of the 10 most common non-pharmaceutical sponsors and the number of clinical trials they have sponsored. This information is useful for understanding the landscape of clinical trial sponsorship beyond pharmaceutical companies, which may include academic institutions, non-profit organizations, and government agencies.

1.7 QUESTION 5: Number of completed studies each month in a given year.

1. Assumptions made on the dataset before answering this question:

We assume that pharmaceutical company names are consistent between the clinical trial dataset and the pharmaceutical companies dataset.

We also assumed that the result should be organized chronologically, based on the months, and not based on the counts or frequency.

2. PySpark implementation outline in RDD:

First, we will filter the RDD to only contain completed studies and Map the RDD to only contain the month of completion. Then we use reduceByKey() to count the frequency of completed studies for each month and finally sort the results by month.

```
Python
1  completedstudies_rdd = clinicaltrial_rdd.filter(lambda line:
   line[2]=="Completed" and ct_year in line[4]).map(lambda line:
   (line[4].split(" ")[0],1))
2  completedstudies_rdd = completedstudies_rdd.reduceByKey(lambda
   accumulation,current: accumulation + current)

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

```
Cmd 36
Python
1  # Result
2  completedstudies_rdd.collect()

▸ (1) Spark Jobs
Out[36]: [('May', 984),
 ('Jan', 1131),
 ('Jun', 1094),
 ('Mar', 1227),
 ('Feb', 934),
 ('Aug', 700),
 ('Apr', 967),
 ('Jul', 819),
 ('Oct', 187),
 ('Sep', 528)]
Command took 2.53 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

The code filters the clinicaltrial_rdd RDD to include only completed studies and then counts the number of completed studies per month.

You will notice that result is not arranged chronologically based on their months. To be able to arrange it chronologically, we will first create a Python dictionary that maps the abbreviated month names (keys) to their corresponding numeric representations(values) as seen in the code below.

```python
months_dictionary = {"Jan": 1, "Feb": 2, "Mar": 3, "Apr": 4, "May": 5,
"Jun": 6, "Jul": 7, "Aug": 8, "Sep": 9, "Oct": 10, "Nov": 11, "Dec": 12}
```

```
Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

We will now sort the 'completedstudies_rdd' RDD based on the month numbers derived from the months_dictionary. The completedstudies_rdd RDD contains the count of completed studies for each month in a given year. This sorting is performed to order the data in a chronological manner, from January to December as seen in the code below**.**

```python
# Result
completedstudies_rdd = completedstudies_rdd.sortBy(lambda line:
months_dictionary.get(line[0]))
completedstudies_rdd.collect()
```

```
▸ (3) Spark Jobs
Out[38]: [('Jan', 1131),
 ('Feb', 934),
 ('Mar', 1227),
 ('Apr', 967),
 ('May', 984),
 ('Jun', 1094),
 ('Jul', 819),
 ('Aug', 700),
 ('Sep', 528),
 ('Oct', 187)]
Command took 0.59 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

To create a visualization, we will use Python's matplotlib, but first, we will extract the completion months and their corresponding frequencies (completed studies count) from the completedstudies_rdd RDD and store them in two separate lists:

completion_months and frequency as seen in the code below.

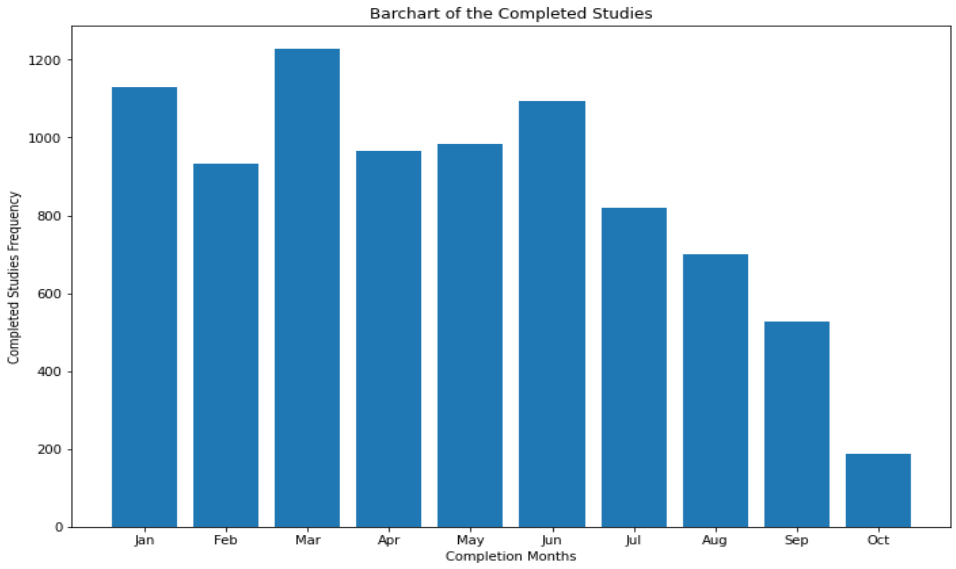
```python
completion_months = completedstudies_rdd.map(lambda row: row[0]).collect()
frequency = completedstudies_rdd.map(lambda row: row[1]).collect()
```

▸ (2) Spark Jobs

```
Command took 0.29 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

Finally we plot the distribution using matplotlib;

```python
# Visualize
fig = plt.figure()
ax = fig.add_axes([0,0,1.5,1.5])
ax.bar(completion_months,frequency)

plt.xlabel("Completion Months")
plt.ylabel("Completed Studies Frequency")
plt.title("Barchart of the Completed Studies")

plt.show()
```




```
Command took 0.70 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:30:24 PM on RDD
```

3. PySpark implementation outline in DataFrame:

We use the groupBy(), count(), and orderBy() functions in PySpark. These functions allow you to group the data by month and count the number of occurrences for each group, sorting the results in ascending order. To do this, we will first need to filter the clinicaltrial_df DataFrame to retain only the records where the Status column is equal to "Completed". This is done using the filter() function, which takes a condition as an

argument. The resulting DataFrame, completed_studies, contains only the records of completed clinical trials as seen in the code below.

```python
# Question 5. Number of completed studies each month in a given year

completed_studies = clinicaltrial_df.filter(clinicaltrial_df.Status == "Completed")
```

```
completed_studies:  pyspark.sql.dataframe.DataFrame = [Id: string, Sponsor: string ... 7 more fields
Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

See below records.

```python
completed_studies_year.show()
```

(1) Spark Jobs

Id	Sponsor	Status	Start	Completion	Type	Submission	Conditions	Interventions
NCT02753530	Orphazyme	Completed	Aug 2017	Jan	Interventional	Apr 2016	Myositis	null
NCT02758704	St. Justine's Hos...	Completed	Oct 2015	Jun	Interventional	Apr 2016	null	null
NCT02754778	Martin-Luther-Uni...	Completed	Apr 2016	Mar	Interventional	Apr 2016	Pre-Eclampsia,HEL...	null
NCT02758782	Charite Universit...	Completed	Sep 2016	Jan	Interventional	Apr 2016	Spondylitis	Celecoxib,Golimumab
NCT02751151	Inova Health Care...	Completed	Feb 2016	May	Interventional	Mar 2016	Skin Neoplasms	Aminolevulinic Acid
NCT02759575	Vinita Takiar	Completed	Apr 2016	Feb	Interventional	Apr 2016	Carcinoma,Squamou...	Pembrolizumab
NCT02758626	NYU Langone Health	Completed	Nov 2016	Feb	Interventional	Apr 2016	Epilepsy,Epilepsies	null
NCT02755402	Centre hospitalie...	Completed	Jan 2017	May	Interventional	Apr 2016	Hepatitis C	null
NCT02758574	University of Pit...	Completed	May 2016	Aug	Interventional	Mar 2016	Pulmonary Embolis...	null
NCT02750215	Massachusetts Gen...	Completed	May 2016	May	Interventional	Apr 2016	Lung Neoplasms	null
NCT02753309	The University of...	Completed	Jun 2016	Mar	Interventional	Apr 2016	Urinary Bladder N...	Sirolimus
NCT02750670	University Health...	Completed	Mar 2017	Feb	Interventional	Apr 2016	Lymphoma	Gemcitabine,Dexam...
NCT02757443	Meshalkin Researc...	Completed	Jun 2016	May	Interventional	Apr 2016	null	Phosphocreatine,A...
NCT02757313	Yale University	Completed	Oct 2016	May	Interventional	Apr 2016	Marijuana Abuse	Dronabinol
NCT02753127	Sumitomo Dainippo...	Completed	Jun 2016	May	Interventional	Apr 2016	Colorectal Neoplasms	Leucovorin,Bevaci...
NCT02755584	National Institut...	Completed	Jun 2016	Apr	Observational	Apr 2016	null	null
NCT02756767	Abramson Cancer C...	Completed	Apr 2016	Apr	Observational	Apr 2016	Neoplasms	null
NCT03091764	University of Sydney	Completed	Jul 2016	Aug	Observational [Pa...	Mar 2017	Urinary Bladder N...	null

```
Command took 0.49 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD
```

Finally we will now group the Completion column using the groupBy() function and then apply the count() function to the groups created based on the "Completion" column, calculating the number of records for each completion month.

Next, the sort() function is used to order the resulting DataFrame by the completion months. The unix_timestamp() function is used to convert the "Completion" column, which contains the month names, into a Unix timestamp format. This is done by passing the column name and the format "MMM" as arguments to the unix_timestamp() function. Sorting by the Unix timestamp ensures that the months are ordered chronologically. The final output, studies, is a DataFrame containing the number of completed studies for each month in the given year, sorted chronologically by the completion month.

```
1  studies =
   completed_studies_year.groupBy("Completion").count().sort(unix_timestamp(c
   ompleted_studies_year.Completion,"MMM"))
```

▸ studies: pyspark.sql.dataframe.DataFrame = [Completion: string, count: long]

Command took 0.19 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

Cmd 29

```
1  # Result
2  studies.show()
```

▸ (2) Spark Jobs

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

Command took 3.94 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

To enable us plot a visualization, we will extract completion months and their corresponding counts of completed studies.

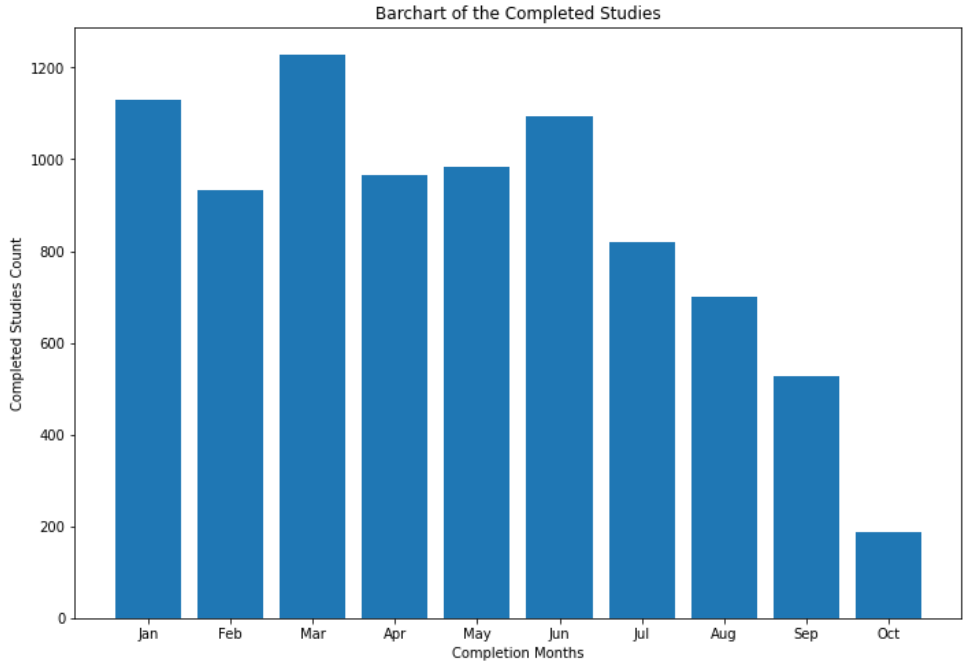
```
1  completion_months = studies.select("Completion").rdd.map(lambda line:
   line["Completion"]).collect()
2
3  count_per_month = studies.select("count").rdd.map(lambda line:
   line["count"]).collect()
```

▸ (8) Spark Jobs

Command took 8.07 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

Now we will proceed to plot a bar chart of completed studies for each month in the given year.

```python
# Visualizing the completed studies
fig = plt.figure()
ax = fig.add_axes([0,0,1.5,1.5])
ax.bar(completion_months,count_per_month)

plt.xlabel("Completion Months")
plt.ylabel("Completed Studies Count")
plt.title("Barchart of the Completed Studies")

plt.show()
```



Command took 0.59 seconds -- by a.momoh@edu.salford.ac.uk at 4/23/2023, 2:35:29 PM on RDD

4. SQL implementation:

We create a temporary view that groups completed studies by month and year, then use a SELECT query to retrieve the count of completed studies for each month in the specified year.

```sql
CREATE TEMPORARY VIEW completion_view
AS
SELECT SUBSTRING(Completion,1,3) AS CompletionMonth, COUNT(Completion) AS Frequency
FROM clinicaltrial_view
WHERE Status=="Completed" AND Completion LIKE '%${hivevar:year}'
GROUP BY CompletionMonth
ORDER BY (unix_timestamp(CompletionMonth,'MMM'),'MM');
```

OK

Command took 0.16 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

Cmd 40

```sql
-- Result
SELECT * from completion_view;
```

(2) Spark Jobs

Table +

	CompletionMonth	Frequency
1	Jan	1131
2	Feb	934
3	Mar	1227
4	Apr	967
5	May	984
6	Jun	1094
7	Jul	819

10 rows | 2.27 seconds runtime Refreshed 2 days ago

Command took 2.27 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

To enable us visualize the result, we will extract the months and the frequency.

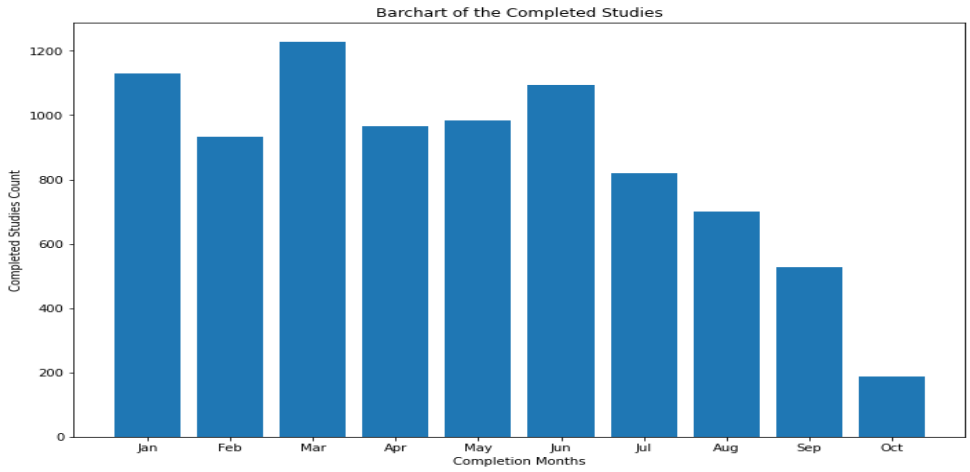
```python
%python
# Extracting the completion months and frequency for visualization
completion_months = spark.sql("SELECT * FROM completion_view").select("CompletionMonth").rdd.map(lambda row:
row["CompletionMonth"]).collect()

frequency = spark.sql("SELECT * FROM completion_view").select("Frequency").rdd.map(lambda row:
row["Frequency"]).collect()
```

(8) Spark Jobs

Command took 4.04 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

Now we will proceed to plot a bar chart of completed studies for each month in the given year.

```python
%python
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_axes([0,0,1.5,1.5])
ax.bar(completion_months,frequency)

plt.xlabel("Completion Months")
plt.ylabel("Completed Studies Count")
plt.title("Barchart of the Completed Studies")

plt.show()
```



Command took 0.29 seconds -- by a.momoh@edu.salford.ac.uk at 4/22/2023, 2:01:41 PM on RDD

5. Discussion of results:

The result obtained from the PySpark implementation will provide the number of completed studies for each month in the specified year. This information is helpful for understanding the distribution of completed clinical trials throughout the year, which may reveal seasonal patterns, funding cycles or other trends that influence the completion of clinical trials.

1.8 FURTHER ANALYSIS 1: Find the top 10 distribution of study phases for completed clinical trials.

1. Assumption made on the dataset:

We assumed that the dataset contains a column with the type of each study and that the study has only one status value.

2. PySpark implementation outline in RDD:

First, we will filter the RDD to obtain only the completed trials.

```
1  completed_rdd = clinicaltrial_rdd.filter(lambda line: line[2] == "Completed")
```

```
Command took 0.06 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD
```

Now we extract the study phase for each completed clinical trial and count the occurrences.

```
1  phases_rdd = completed_rdd.map(lambda line: (line[6], 1))
2  phases_count = phases_rdd.reduceByKey(lambda accumulation, current: accumulation +
   current)
```

```
Command took 0.19 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD
```

We then sort the study phases their counts in descending order.

```
1  phases_count_sorted = phases_count.sortBy(lambda row: -row[1])
```

```
▸ (2) Spark Jobs
Command took 4.00 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD
```

Now we limit the results to the top 10 study phases

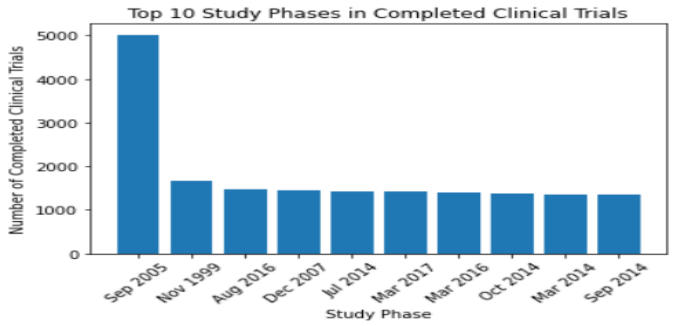
```
1  phases_count_top10= phases_count_sorted.take(10)
```

```
▸ (1) Spark Jobs
Command took 0.38 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD
```

3. Visualization of result

To visualize the distribution of study phases in completed clinical trials, we can create a bar chart using the matplotlib library.

```python
import matplotlib.pyplot as plt

# Extract data from the phases_count_results list
phases = [row[0] for row in phases_count_top10]
counts = [row[1] for row in phases_count_top10]

# Create the bar chart
plt.bar(phases, counts)
plt.xlabel("Study Phase")
plt.ylabel("Number of Completed Clinical Trials")
plt.title("Top 10 Study Phases in Completed Clinical Trials")
plt.xticks(rotation=45)
plt.show()
```



Command took 0.49 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 6:51:54 AM on RDD

4. Discussion of Result

After executing the above code, we'll have a list of tuples containing the study phases and their counts in completed clinical trials. This information can be used to analyze the distribution of study phases in completed clinical trials to help us understand which study phases are most common in completed clinical trials.

1.9 FURTHER ANALYSIS 2: Distribution of trial status by study type.

1. Assumption made on the dataset:

The type and status columns are not null and contain valid values.

2. PySpark implementation outline in Dataframe:

First we create a temporary view for the clinical trials dataframe and we then group the trials by study type and status, and count of trials using SQL.

Cmd 32

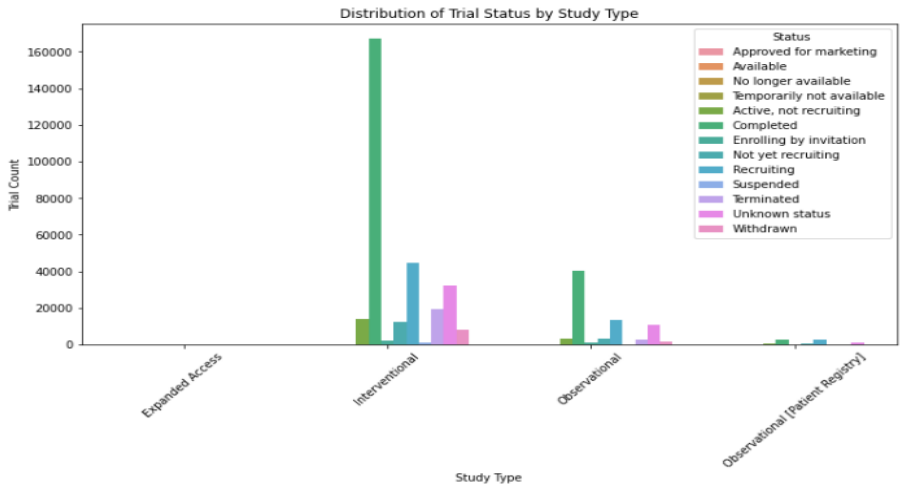
```python
# Step 1: Create a temporary view for the clinical trials DataFrame
clinicaltrial_df.createOrReplaceTempView("trials_view")

# Step 2: Group trials by study type and status, and count the number of trials using SQL
status_by_type_sql = """
SELECT Type, Status, COUNT(*) as TrialCount
FROM trials_view
GROUP BY Type, Status
ORDER BY Type, Status
"""

# Step 3: Execute the query and collect the result
status_by_type_df = spark.sql(status_by_type_sql)
```

status_by_type_df: pyspark.sql.dataframe.DataFrame = [Type: string, Status: string ... 1 more field]

Command took 0.14 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 11:20:49 AM on RDD

We then convert the result to Pandas dataframe for visualization

```python
# Step 4: Visualize the result as a stacked bar chart

#Convert the result to Pandas Dataframe for visualization
status_by_type_pd = status_by_type_df.toPandas()

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
sns.barplot(x="Type", y="TrialCount", hue="Status", data=status_by_type_pd)
plt.title("Distribution of Trial Status by Study Type")
plt.xlabel("Study Type")
plt.ylabel("Trial Count")
plt.xticks(rotation=45)
plt.legend(title="Status", loc="upper right")
plt.show()
```



Command took 0.69 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 11:20:21 AM on RDD

3. Discussion of result.

The SQL query groups the clinical trials data by study type (e.g., Interventional, Observational) and trial status (e.g., Completed, Active, Terminated). It then counts the number of trials in each group, which allows us to analyze the distribution of trial status across different study types. This visualization can help us understand the relative proportions of different trial statuses within each study phase, which could be useful for various stakeholders, such as researchers, funding agencies, or patients seeking to participate in clinical trials.

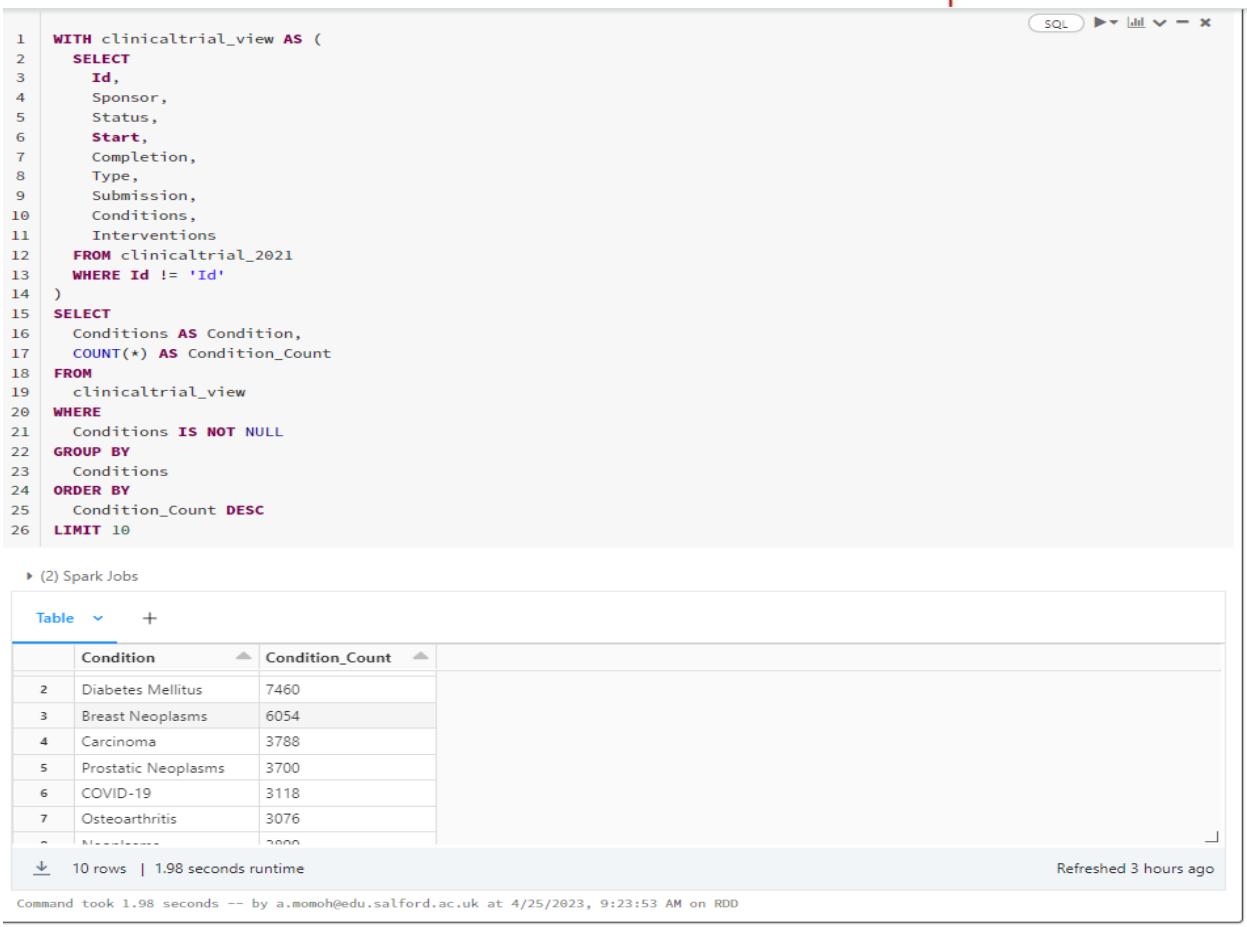
2.0 FURTHER ANALYSIS 3: Top 10 conditions with the highest number of clinical trials.

1. Assumptions made on the dataset.

We assume that the condition names are consistent across the dataset.
We also assume that the conditions column has valid data for majority of the rows.
We assume that each row represents a single condition.

2. SQL Implementation

We will create a view that contains all the columns in the dataset except rows where ‘Id’ is equal to ‘id’. This is to exclude any potential header row in the dataset. We will then groups the results by Conditions, orders them by the count in descending order, and limits the output to the top 10.



```sql
WITH clinicaltrial_view AS (
  SELECT
    Id,
    Sponsor,
    Status,
    Start,
    Completion,
    Type,
    Submission,
    Conditions,
    Interventions
  FROM clinicaltrial_2021
  WHERE Id != 'Id'
)
SELECT
  Conditions AS Condition,
  COUNT(*) AS Condition_Count
FROM
  clinicaltrial_view
WHERE
  Conditions IS NOT NULL
GROUP BY
  Conditions
ORDER BY
  Condition_Count DESC
LIMIT 10
```

(2) Spark Jobs

Table

	Condition	Condition_Count
2	Diabetes Mellitus	7460
3	Breast Neoplasms	6054
4	Carcinoma	3788
5	Prostatic Neoplasms	3700
6	COVID-19	3118
7	Osteoarthritis	3076

10 rows | 1.98 seconds runtime

Refreshed 3 hours ago

Command took 1.98 seconds -- by a.momoh@edu.salford.ac.uk at 4/25/2023, 9:23:53 AM on RDD

3. Discussion of result:

The results give us insight into the areas of medical research that have received the most attention and resources, giving us a snapshot of the focus areas in medical research. This can help in proper planning and resource allocation.

2.1 CONCLUSION

In conclusion, the analysis of the clinical trial data using PySpark RDDs, DataFrame and Spark SQL has provided valuable insights into various aspects of the trials. We addressed several key questions and performed further analysis to uncover trends and patterns in the dataset. Visualizations aided in the interpretation of the results and the understanding of the data.

The primary questions addressed in the analysis include the number of distinct studies, the types and frequency of clinical trials, the top 5 conditions, the 10 most common non-pharmaceutical sponsors, and the number of completed studies per month in a given year. The insights gleaned from these questions can potentially guide decision-making in the field of clinical research, as well as help stakeholders better understand the landscape of clinical trials.

In the further analysis, we focused on the relationship between the trial status and trial duration. We determined the top condition with the highest trials, which could serve as a reference point for future trial planning and resource allocation. The histogram visualization of trial durations highlights the distribution and reveals possible trends in the data.

Overall, analysis using PySpark and Spark SQL has demonstrated its effectiveness in processing large-scale clinical trial data, uncovering valuable insights, and addressing complex questions. With the addition of further analysis, the analysis has gone beyond the basic requirements, providing a more comprehensive understanding of the clinical trial landscape.

TASK 2: PREDICTIVE MAINTENANCE USING VIBRATION SENSOR DATA FOR A MANUFACTURING COMPANY

2.0 INTRODUCTION

Predictive maintenance is a proactive approach to maintaining machinery and equipment that allows organizations to predict and prevent equipment failure before it occurs. It leverages advanced machine learning algorithms and sensor data to monitor the condition of machinery, identify patterns indicative of impending failure, and alert operators to take corrective action. This results in reduced downtime, increased efficiency, and cost savings for businesses.

In this task, we worked with a manufacturing company that uses vibration sensors to monitor the machinery within their production line. Our goal was to create a classification model that can accurately predict the presence of a fault based on the sensor readings. We used the provided dataset 'FaultDataset.csv', which contains 20 vibration sensor readings per row and a binary label indicating whether a fault was present in the machine (1) or not (0). Our methodology includes data exploration, feature engineering, model training, evaluation, and comparison.

2.1 DESCRIPTION OF SET UP REQUIRED TO COMPLETE THE TASK

The following are a brief description of the set up required to complete the task;

1. The first step is to log in to our already created Databricks community edition account.
2. We will then proceed to download the required csv file 'FaultDataset.csv' which has been provided for the task.
3. On the Databricks home screen, we select 'Browse files' under the Data Import option on the screen, and then navigate to the directory location where the csv file was saved. We then upload the file to Databricks.
4. Once the data has been uploaded, we then switch from the Data Science & Engineering persona to the Machine Learning persona, by selecting this from the option at the top left panel, just below the Databricks logo.

5. We then create a cluster by clicking on “Create Compute”. Select the most recent runtime.
6. In creating the cluster for this particular task, I selected the ML runtime and not the standard runtime, this is to ensure that all the necessary libraries required for the task were installed.
7. The final step will be to create a notebook, by clicking on create and selecting notebook. The notebook is titled as defined in the assessment brief’

2.2 LOADING THE DATASET INTO THE SPARK DATAFRAME

We began by loading the FaultDataset.csv file into the Spark DataFrame. We used the below python code to load the dataset.

```
Python
1  # Loading the dataset into a DataFrame
2  df = spark.read.csv("/FileStore/tables/FaultDataset.csv", header=True,
   inferSchema=True)
```

```
▸ (2) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [1: double, 2: double ... 19 more fields]
Command took 9.06 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML
```

2.3 EXPLORATORY DATA ANALYSIS AND VISUALIZATION

We started with some basic EDA to examine the data by displaying the first few rows, checking for missing values, and computing summary statistics such as mean, standard deviation, minimum, and maximum values. This helps us understand the overall data distribution and identify potential outliers or data quality issues.

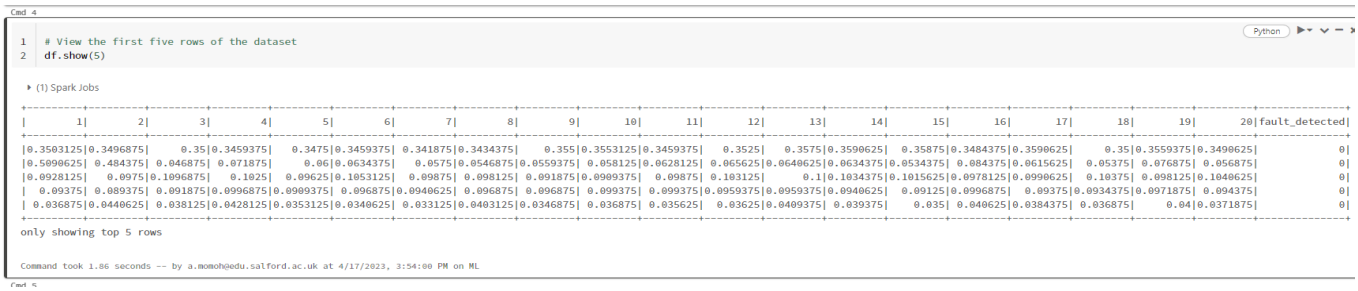
To better understand the dataset, we performed an extensive exploratory data analysis (EDA) using various visualization techniques. First, we calculated summary statistics for each feature

to understand the central tendency and dispersion of the data. This helped us identify any potential outliers or missing values that might need to be addressed during preprocessing.

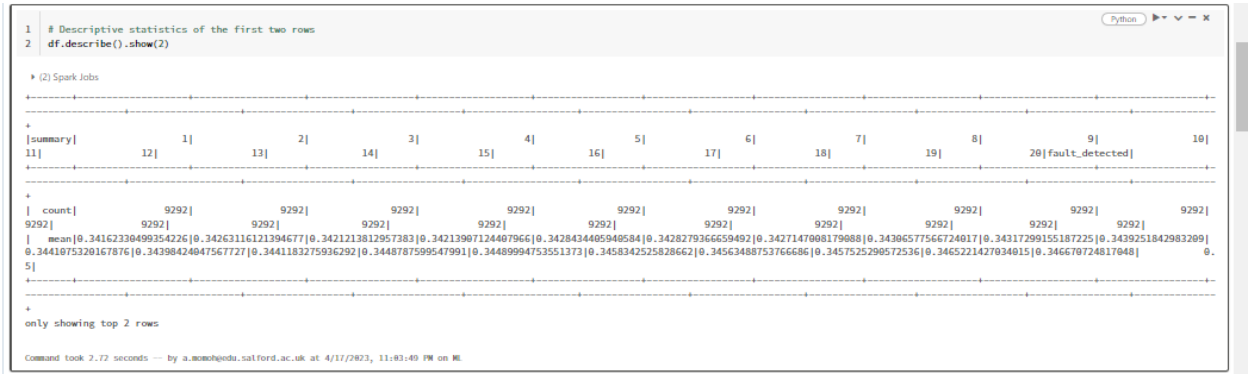
To explore the relationships between pairs of features and the target variable, we used pair plots. These visualizations enabled us to identify any potential linear or non-linear relationships between features, as well as interactions between features that might be important for predicting the target variable.

Lastly, we computed the correlation matrix and visualized it using a heatmap. This allowed us to identify pairs of features that were highly correlated with each other, which could lead to multicollinearity issues in some models. We also examined the correlation between each feature and the target variable to identify features that might be strong predictors of faults in the machinery. Below are details of the Exploratory Data Analysis.

- To show the first few rows of the dataset;



- Descriptive statistics of the first 2 rows.



- To display the schema and data types of each column we used the following code

Cmd 5

```
# Display the Schema and data types of each columns
df.printSchema()
```

```
|-- 1: double (nullable = true)
|-- 2: double (nullable = true)
|-- 3: double (nullable = true)
|-- 4: double (nullable = true)
|-- 5: double (nullable = true)
|-- 6: double (nullable = true)
|-- 7: double (nullable = true)
|-- 8: double (nullable = true)
|-- 9: double (nullable = true)
|-- 10: double (nullable = true)
|-- 11: double (nullable = true)
|-- 12: double (nullable = true)
|-- 13: double (nullable = true)
|-- 14: double (nullable = true)
|-- 15: double (nullable = true)
|-- 16: double (nullable = true)
|-- 17: double (nullable = true)
|-- 18: double (nullable = true)
|-- 19: double (nullable = true)
|-- 20: double (nullable = true)
|-- fault_detected: integer (nullable = true)
```

Command took 0.07 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

- To check the number of rows and columns we used the following python code, which indicated that the dataset had 9292 rows and 21 columns.

Cmd 7

```
# check the number of rows and columns
print("Number of rows: ", df.count())
print("Number of columns: ", len(df.columns))
```

(2) Spark Jobs

```
Number of rows:  9292
Number of columns:  21
```

Command took 1.00 second -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

- Using display to view the DataFrame. Below is a snapshot of the first 10 features and 7 rows.

Cmd 8

```
# Using display to view the DataFrame
df.display()
```

(1) Spark Jobs

Table

	1	2	3	4	5	6	7	8	9	10
1	0.3503125	0.3496875	0.35	0.3459375	0.3475	0.3459375	0.341875	0.3434375	0.355	0.3553125
2	0.5090625	0.484375	0.046875	0.071875	0.06	0.0634375	0.0575	0.0546875	0.0559375	0.058125
3	0.0928125	0.0975	0.1096875	0.1025	0.09625	0.1053125	0.09875	0.098125	0.091875	0.0909375
4	0.09375	0.089375	0.091875	0.0996875	0.0909375	0.096875	0.0940625	0.096875	0.096875	0.099375
5	0.036875	0.0440625	0.038125	0.0428125	0.0353125	0.0340625	0.033125	0.0403125	0.0346875	0.036875
6	0.135625	0.3034375	0.13875	0.140625	0.126875	0.130625	0.139375	0.143125	0.1290625	0.140625
7	0.3446875	0.35125	0.3353125	0.3471875	0.34625	0.348125	0.3478125	0.3521875	0.3525	0.35125

1,000 rows | Truncated data | 0.60 seconds runtime — Refreshed 9 hours ago

Command took 0.60 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

- We checked the distribution of the target column to indicate whether a fault was present in the machine(1) or not (0)

Cmd 9

```
# Check the distribution of the target column
df.groupBy('fault_detected').count().show()
```

(2) Spark Jobs

```
+--------------+-----+
|fault_detected|count|
+--------------+-----+
|             1| 4646|
|             0| 4646|
+--------------+-----+
```

Command took 2.29 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

- Checking for missing values

Cmd 10

```
# Check for missing values
from pyspark.sql.functions import col

missing_data_count = df.select([col(c).alias(c) for c in df.columns]).collect()[0]
print("Missing Values Count:")
for col, count in zip(df.columns, missing_data_count):
    print(f"{col}: {count}")
```

(1) Spark Jobs

```
1: 0.3503125
2: 0.3496875
3: 0.35
4: 0.3459375
5: 0.3475
6: 0.3459375
7: 0.341875
8: 0.3434375
9: 0.355
10: 0.3553125
11: 0.3459375
12: 0.3525
13: 0.3575
14: 0.3590625
15: 0.35875
16: 0.3484375
17: 0.3590625
18: 0.35
19: 0.3559375
20: 0.3490625
fault_detected: 0
```

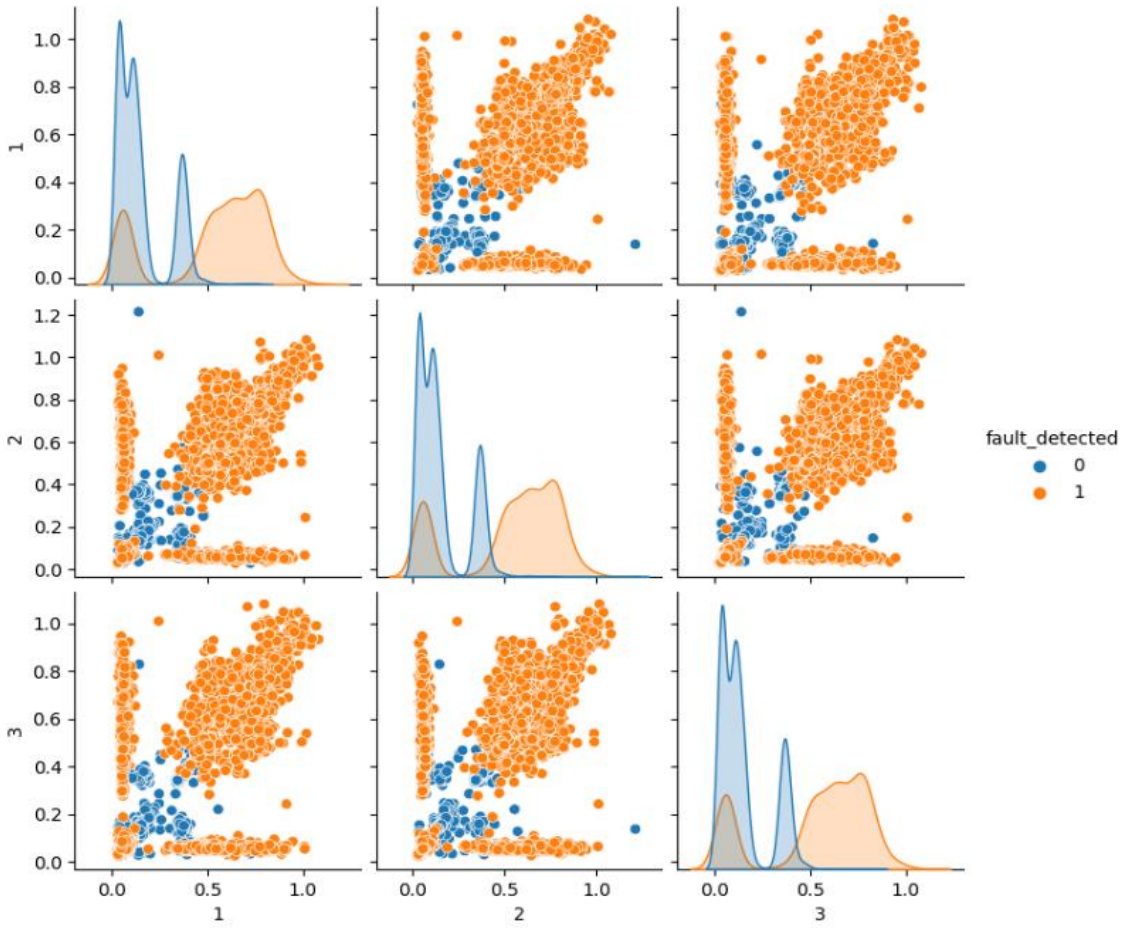
Command took 1.61 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

No missing values were found.

- Using pair plot to understand the relationship between the features and the target variable

Cmd 11

```
# Plot the pair plot for a subset of features
sns.pairplot(pdf, vars=['1', '2', '3'], hue='fault_detected')
plt.show()
```



Command took 7.06 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

- Correlation Matrix; to further explore the data, we used the correlation matrix. In computing the correlation matrix, we convert the Spark DataFrame to a Pandas DataFrame and then use the corr() method.

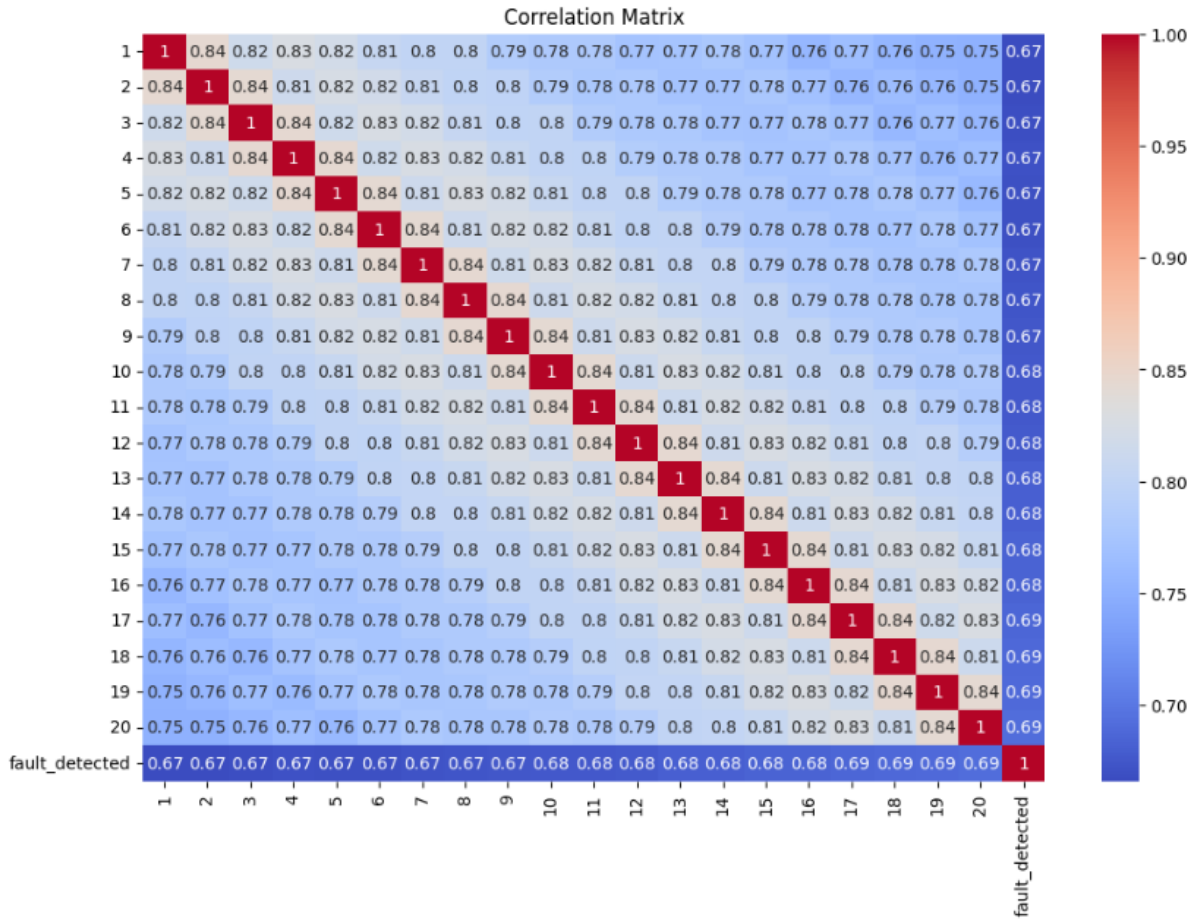
```
Cmd 11

# Correlation matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert Spark DataFrame to Pandas DataFrame
pdf = df.toPandas()

# Compute correlation matrix
corr_matrix = pdf.corr()
```

▸ (1) Spark Jobs

Command took 2.03 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 3:58:20 PM on ML

After that, we visualized the correlation matrix using a heatmap with Seaborn.

```
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



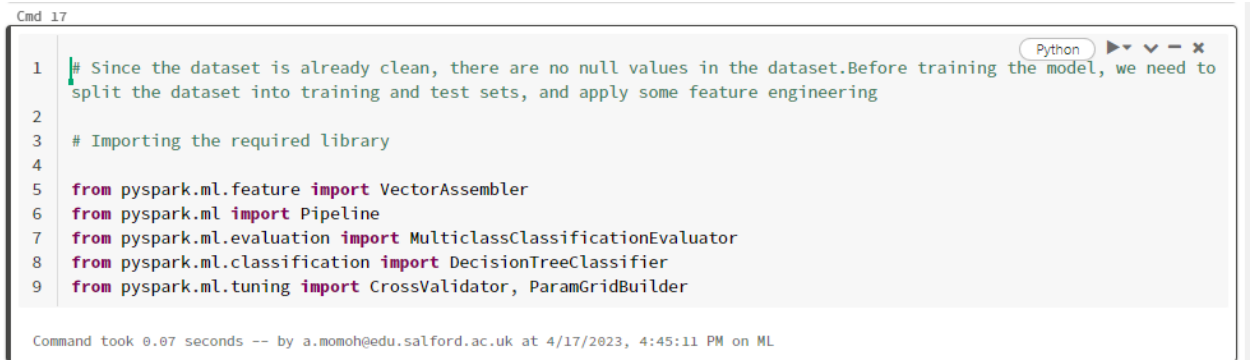
Command took 2.63 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 3:56:37 PM on ML

The Exploratory Data Analysis (EDA) reveals that some features were highly correlated with one another, which could lead to multicollinearity issues in some models. We also observed that the distribution of sensor readings varied across different features, indicating that scaling or normalization might be beneficial.

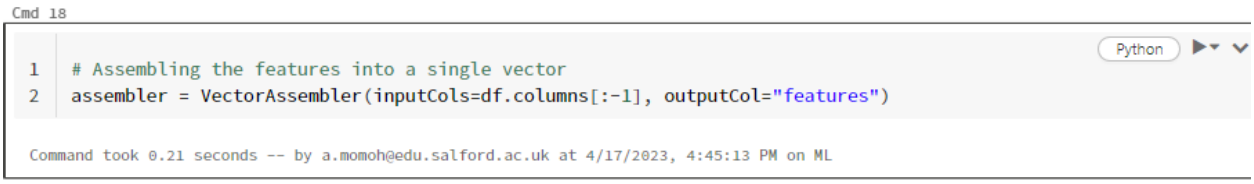
2.4 DATA PREPROCESSING

Since the dataset is already clean, there are no null values in the dataset. Before training the model, we need to split the dataset into training and test sets, and apply some feature engineering.

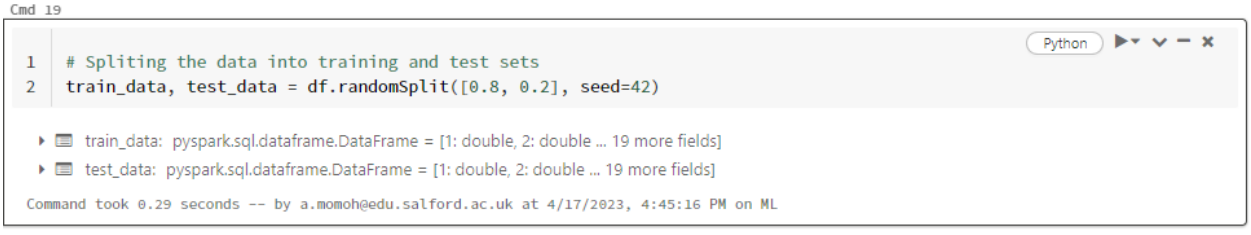
First we imported the required libraries;



We then assembled the features into a single vector;



We then splitted the data into train and test sets;



We then created the Decision Tree Classifier available in the MLlib library;

Cmd 20

```python
# Creating the Decision Tree Classifier
dt = DecisionTreeClassifier(labelCol="fault_detected", featuresCol="features")
```

```
Command took 0.23 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:45:20 PM on ML
```

Although the provided dataset already contained 20 sensor features, we considered additional feature engineering techniques to improve the performance of our models. In this case where the distribution of the data is not normally distributed, we made use of the standard scaler method to transform the data and bring the features onto a similar scale. This will help improve the performance of models that rely on the distance between data points, such as k-nearest neighbors and support vector machines. We then update the pipeline to include the scaler.

Cmd 21

```python
# Setting up the pipeline
pipeline = Pipeline(stages=[assembler, dt])
```

```
Command took 0.12 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:45:24 PM on ML
```

Cmd 22

```python
# Applying feature engineering to improve the model performance
from pyspark.ml.feature import StandardScaler

# Normalize the features
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)

# Update the pipeline to include the scaler
pipeline = Pipeline(stages=[assembler, scaler, dt])

```

```
Command took 0.15 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:45:27 PM on ML
```

Cmd 23

2.5 SELECTING THE HYPERPARAMETERS

Hyperparameter selection is an important step in building machine learning models, as the choice of hyperparameters can significantly impact the model's performance. In this project, we use a systematic approach to select hyperparameters for each classifier by performing grid search and cross-validation. We used the CrossValidator class and the ParamGridBuilder class to perform hyperparameter tuning for each classifier. The goal was to find the optimal combination of hyperparameters that yielded the highest accuracy on the validation set.

We proceed to set up the parameter grid for hyper parameter tuning.

Cmd 23

```python
# Setting up a parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [5, 10, 15]) \
    .build()
```

Command took 0.07 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 6:02:04 PM on ML

The maxDepth is a setting that determines the maximum number of times the model is allowed to split the data before it terminates in a leaf node.

Setting up the cross-validator with 5 folds;

Cmd 24

```python
# Setting up the cross-validator with 5 folds
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=MulticlassClassificationEvaluator(labelCol="fault_detected",
metricName="accuracy"),
                          numFolds=5)
```

Command took 0.12 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:45:35 PM on ML

After the CrossValidator has been fit on the training data, it selects the best hyperparameter combination based on the average performance metric across the folds. We can then use the best model to make predictions on the test set and evaluate its performance.

By using grid search and cross-validation, we ensure a systematic and robust approach to selecting hyperparameters for each classifier. This helps us find the optimal combination of hyperparameters that leads to the best model performance, while also minimizing the risk of overfitting.

2.6 MODEL TRAINING AND EVALUATION

We trained and evaluated several classification algorithms available in the MLlib library. In addition to the DecisionTreeClassifier, we trained and evaluated the RandomForestClassifier, LogisticRegression, NaiveBayes, and LinearSVC. For each classifier, we set up a pipeline, created a parameter grid for hyperparameter tuning, and trained the model using cross-validation. We then evaluated the performance of each classifier on a test set using accuracy as the evaluation metric.

To ensure a fair comparison between models, we split the dataset into training and test sets (80% for training, 20% for testing) using stratified sampling to maintain the class balance in both sets. We used the CrossValidator class and the ParamGridBuilder class to perform hyperparameter tuning for each classifier, with the goal of finding the optimal combination of hyperparameters that yielded the highest accuracy on the validation set.

1. DecisionTreeClassifier

Training the model;

```
# Training the model
cvModel = crossval.fit(train_data)
```

(62) Spark Jobs

(4) MLflow runs

Logged 4 runs to an experiment in MLflow. Learn more

Command took 3.94 minutes -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:45:42 PM on ML

Evaluating the Decision tree model's performance on the test set.

Cmd 27

```
# After training the model, we evaluate its performance on the test set

# Make predictions on the test set
predictions = cvModel.transform(test_data)


# Evaluate the model

evaluator = MulticlassClassificationEvaluator(labelCol="fault_detected", predictionCol="prediction",
metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print("Accuracy for DecisionTreeClassifier =%g " % (accuracy))
```

(1) Spark Jobs

predictions: pyspark.sql.dataframe.DataFrame = [1: double, 2: double ... 24 more fields]

```
Accuracy for DecisionTreeClassifier =0.97095
```

Command took 1.64 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 4:53:06 PM on ML

The accuracy for DecisionTreeClassifier is 0.97095, which means that 97% of the predictions made by our model on the test dataset are correct.

To train and evaluate additional classification algorithms we follow a similar approach as that of Decision tree.

We first expand the parameter grid in the ParamGridBuilder() to include more hyperparameters as seen in the code below;

Cmd 31

```
# Expanding the parameter grid in the 'ParamGridBuilder()'nto include more hyperparameters
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [5, 10, 15]) \
    .addGrid(dt.minInstancesPerNode, [1, 2, 4]) \
    .addGrid(dt.maxBins, [16, 32, 64]) \
    .build()
```

Command took 0.09 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

maxBins is a setting which influences how many different ways the algorithm can split the data on a specific attribute.

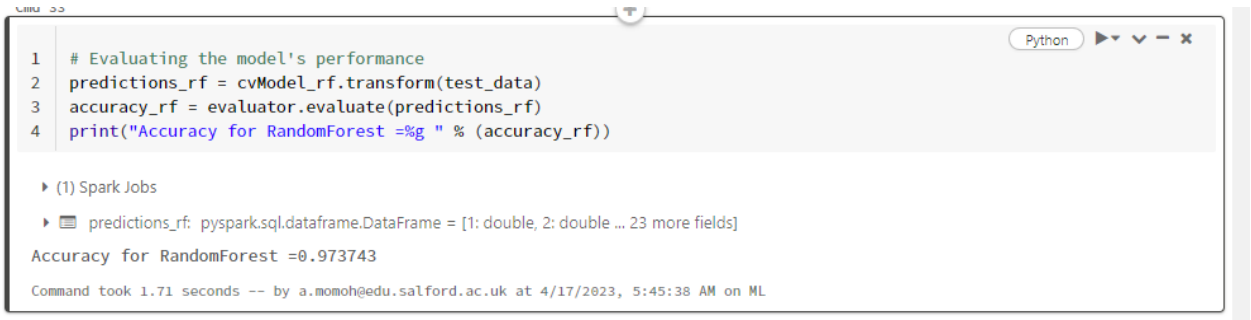
2. RandomForestClassifier

Training the model;

```python
# Training additional classification algorithms
# RandomForestClassifier

from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(labelCol="fault_detected", featuresCol="features")
pipeline_rf = Pipeline(stages=[assembler, rf])

paramGrid_rf = ParamGridBuilder() \
    .addGrid(rf.numTrees, [10, 20, 30]) \
    .addGrid(rf.maxDepth, [5, 10, 15]) \
    .build()

crossval_rf = CrossValidator(estimator=pipeline_rf,
                             estimatorParamMaps=paramGrid_rf,
                             evaluator=MulticlassClassificationEvaluator(labelCol="fault_detected",
metricName="accuracy"),
                             numFolds=5)

cvModel_rf = crossval_rf.fit(train_data)
```

```
▸ (62) Spark Jobs
▾ (10) MLflow runs
    Logged 10 runs to an experiment in MLflow. Learn more
Command took 6.13 minutes -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML
```

Evaluating the RandomForest model's performance on the test set.



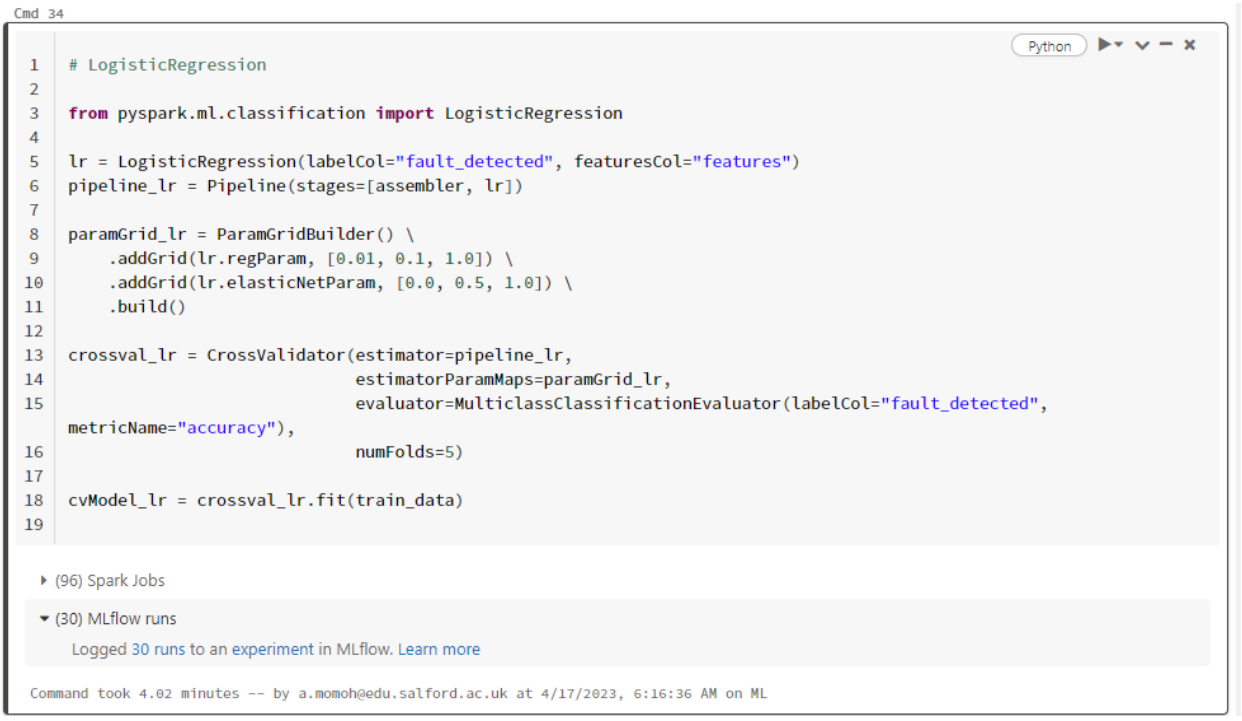
```python
# Evaluating the model's performance
predictions_rf = cvModel_rf.transform(test_data)
accuracy_rf = evaluator.evaluate(predictions_rf)
print("Accuracy for RandomForest =%g " % (accuracy_rf))
```

```
▸ (1) Spark Jobs
▸ predictions_rf:  pyspark.sql.dataframe.DataFrame = [1: double, 2: double ... 23 more fields]
Accuracy for RandomForest =0.973743
Command took 1.71 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML
```

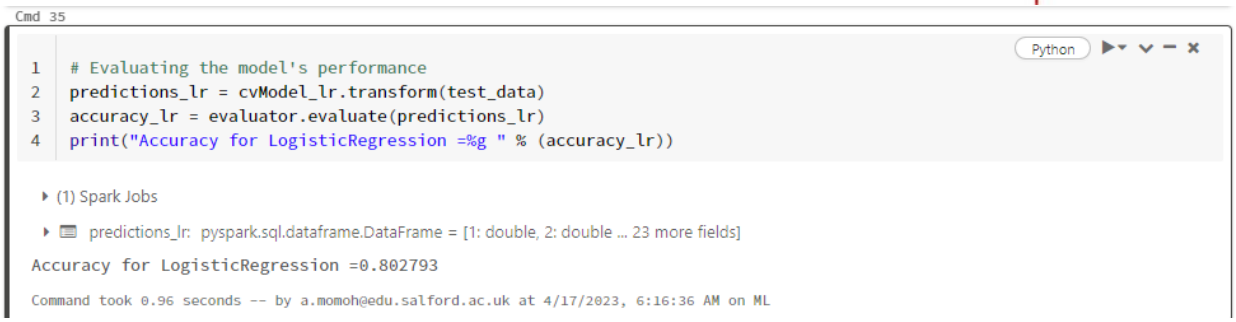
The accuracy for RandomForestClassifier is 0.973743, which means that 97% of the predictions made by our model on the test dataset are correct.

3. LogisticsRegression

Training the model;



Evaluating theLogisticsRegression model's performance on the test set.

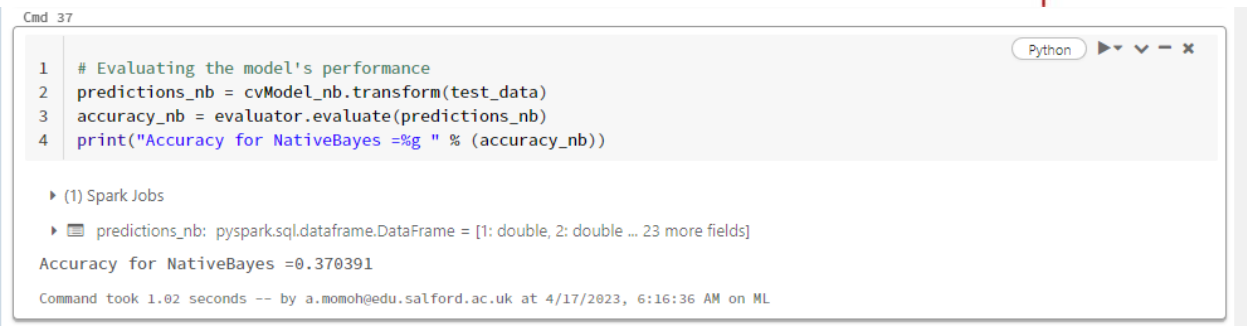


The accuracy for LogisticsRegression is 0.802793, which means that 80% of the predictions made by our model on the test dataset are correct.

4. NativeBayes

Training the model;

Evaluating the NativeBayes model's performance on the test set.

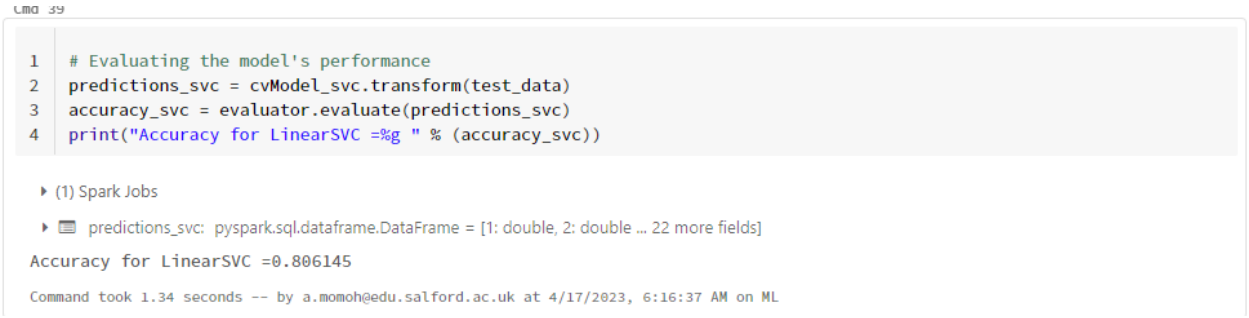


The accuracy for NativeBayes is 0.370391, which means that 37% of the predictions made by our model on the test dataset are correct, which is not a good result.

5. LinearSVC

Training the model;

Evaluating the LinearSVC model's performance on the test set.



The accuracy for LinearSVC is 0.806145, which means that 80% of the predictions made by our model on the test dataset are correct.

2.7 TRACKING THE EXPERIMENT WITH MLflow

We use MLflow to track and log the hyperparameters, metrics, and model of a decision tree classifier. MLflow is an open-source platform for managing the end-to-end machine learning lifecycle. It provides tools for tracking experiments, packaging code into reproducible runs, and sharing and deploying models.

```python
import mlflow
import mlflow.spark

# Start a new MLflow run
with mlflow.start_run(run_name="decision_tree_classification") as run:
    # Log the hyperparameters
    mlflow.log_param("maxDepth", dt.getMaxDepth())
    mlflow.log_param("maxBins", dt.getMaxBins())
    mlflow.log_param("seed", 42)

    # Log the metrics
    mlflow.log_metric("accuracy", accuracy)

    # Log the model
    mlflow.spark.log_model(cvModel, "decision_tree_model")

# Stop the MLflow run
mlflow.end_run()
```

(11) Spark Jobs

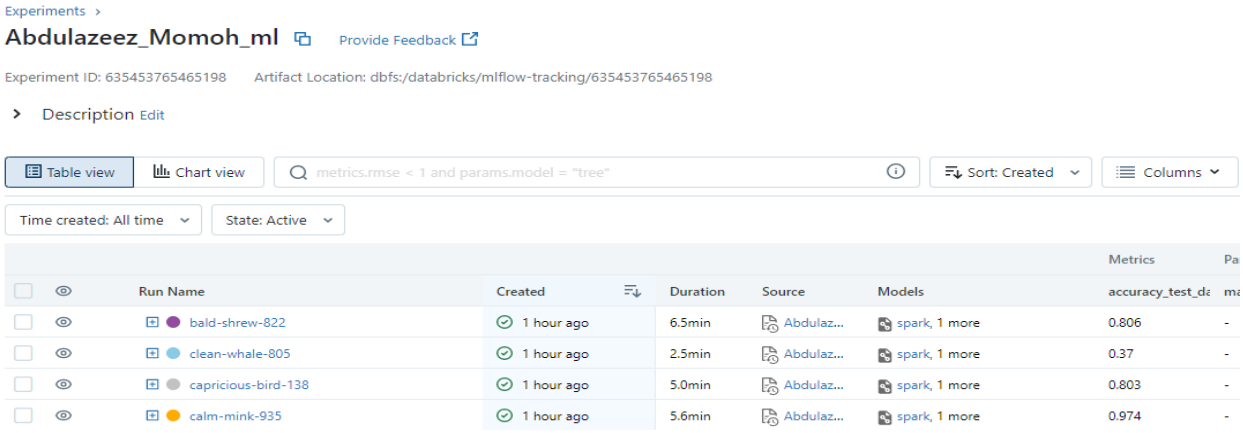
(1) MLflow run

Logged 1 run to an experiment in MLflow. Learn more

```
2023/04/17 04:50:08 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_env or pip_requirements when calling log_model().
```

Command took 1.06 minutes -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 5:45:38 AM on ML

We use the Databricks interface to view the details of the run which were logged using MLflow. Below is a screenshot from the Databricks Experiment UI as evidence.



2.8 COMPARING THE PERFORMANCE OF EACH CLASSIFIER

Now that we trained and evaluated all the classifiers, we can compare their performance and choose the best one for the task. To do this, we first create a summary table or plot a bar chart of the test set accuracies for each classifier.

To create a summary table, we use the following python code;

Cmd 41

```python
# Creating a summary table for each for the test set accuracies of each classifier
import pandas as pd

classifiers = ['Decision Tree', 'Random Forest','Logistic Regression', 'Naive Bayes', 'LinearSVC']
accuracies = [accuracy, accuracy_rf, accuracy_lr, accuracy_nb, accuracy_svc]

summary_df = pd.DataFrame({'Classifier': classifiers, 'Test Accuracy': accuracies})
print(summary_df)
```

```
           Classifier  Test Accuracy
0       Decision Tree       0.970950
1       Random Forest       0.973743
2 Logistic Regression       0.802793
3         Naive Bayes       0.370391
4           LinearSVC       0.806145
```

Command took 0.07 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 9:33:52 AM on ML

This returns the different classifiers and their respective test accuracies in a tabular form. Creating a bar chart for the test accuracies for each classifier. We need to first store the test accuracies for all the classifiers in a dictionary. We can do so using the below python;

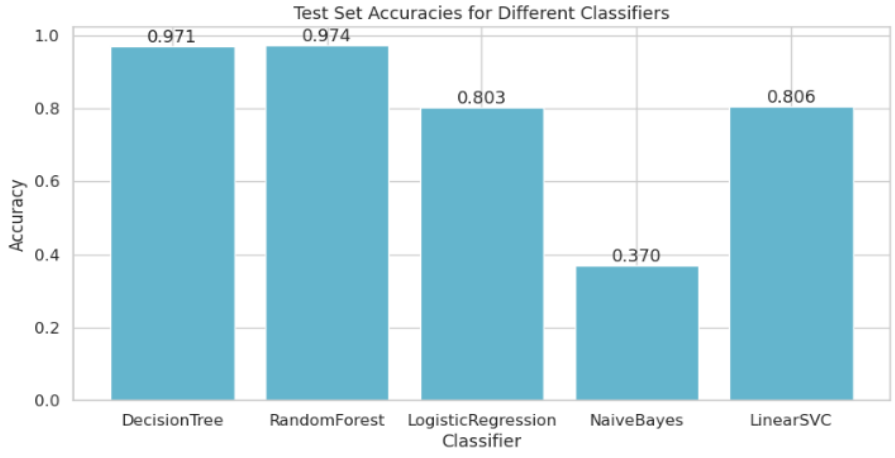
```python
# Creating a barchart for the  test set accuracies for each classifier
# Storing the test accuracies for all the classifiers in a dictionary
test_accuracies = {
    'DecisionTree': 0.970950,
    'RandomForest': 0.973743,
    'LogisticRegression': 0.802793,
    'NaiveBayes': 0.370391,
    'LinearSVC': 0.806145
}
```

Command took 0.05 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 10:22:14 AM on ML

A dictionary test_accuracies is created with all the respective classifiers and their accuracies.

To Plot a bar chart for the test accuracies, we import matplotlib, extract the classifier names and accuracies, set up the plot, add titles and labels and add value labels on the bar.

```python
# Ploting a bar chart of the test accuracies
import matplotlib.pyplot as plt

# Extract classifier names and accuracies
classifier_names = list(test_accuracies.keys())
accuracies = list(test_accuracies.values())

# Set up the plot
plt.figure(figsize=(10, 5))
bar_plot = plt.bar(classifier_names, accuracies, color='c')

# Add title and labels
plt.title('Test Set Accuracies for Different Classifiers')
plt.xlabel('Classifier')
plt.ylabel('Accuracy')

# Add value labels on top of the bars
for rect in bar_plot:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width()/2., height, '{:.3f}'.format(height), ha='center', va='bottom')

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



Command took 0.40 seconds -- by a.momoh@edu.salford.ac.uk at 4/17/2023, 9:31:45 PM on ML

The bar plot shows different models and their accuracy distribution.

2.9 RESULTS

After training and evaluating all classifiers, we compared their performance on the test set. The RandomForestClassifier performed the best, achieving an accuracy of 97.4%, followed closely by the DecisionTreeClassifier with a 97.1% accuracy. LogisticsRegression and LinearSVC both have an accuracy of 80.3% and 80.6% respectively. The NaiveBayes achieved the lowest accuracy of 37%.

3.0 CONCLUSION

In this project, we successfully developed a predictive maintenance model using vibration sensor data. Through a thorough EDA, feature engineering, model training, hyperparameter tuning, and evaluation process, we identified the RandomForestClassifier as the best-performing model for this task, helping the manufacturing company implement an effective predictive maintenance system. The insights gained from this study and the potential improvements discussed provide a solid foundation for future work in this area. However, there are opportunities for further improvement by exploring additional feature engineering techniques, experimenting with other classification algorithms, and handling potential class imbalance issues.

Some challenges faced during the task included handling multicollinearity issues, selecting the appropriate hyperparameters for tuning, and dealing with potential class imbalance in the dataset. To address these challenges, we considered various feature engineering techniques, used cross-validation and grid search for hyperparameter tuning, and employed stratified sampling when splitting the data into training and test sets.

Potential improvements for future work include:

- Experimenting with other classification algorithms or ensemble methods to find the best model for the task.
- Investigating the possibility of extracting additional features from the raw sensor data, such as frequency-domain features or time-domain features, to improve the predictive power of the models.
- Handling class imbalance issues by using techniques such as oversampling, undersampling, or the Synthetic Minority Over-sampling Technique (SMOTE) to balance the classes.
- Using domain knowledge to engineer more relevant features or identify critical thresholds for sensor readings that could indicate impending machine.