Big Data Tools and Techniques

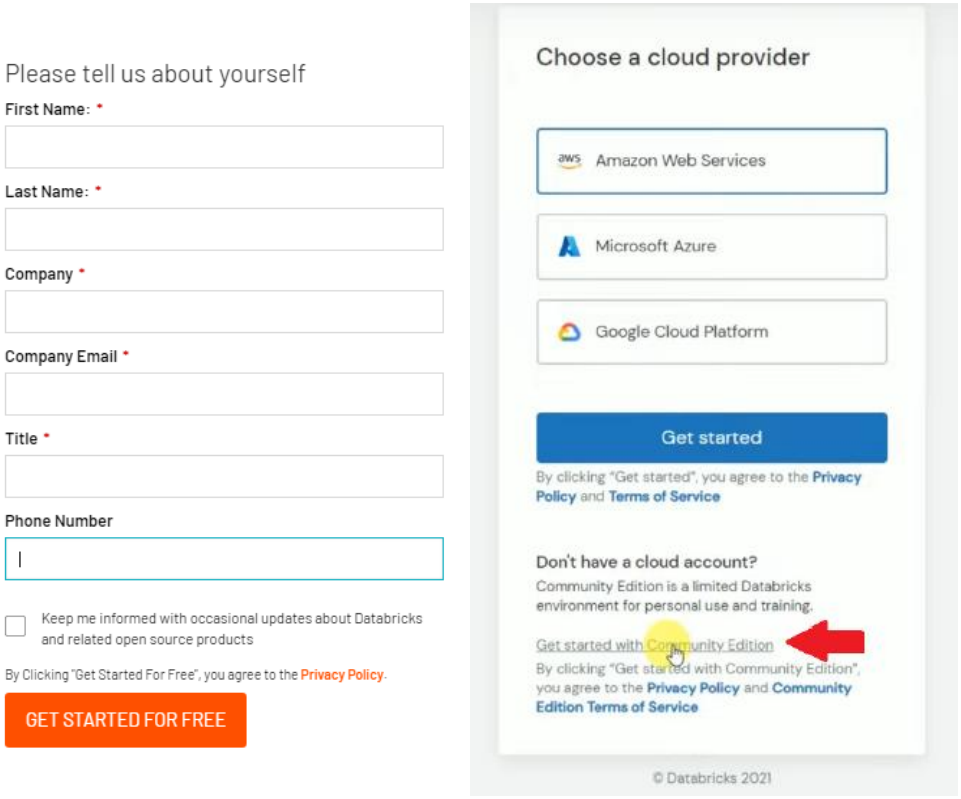
Assignment report

Roghayeh Asghari

1 Required Setup

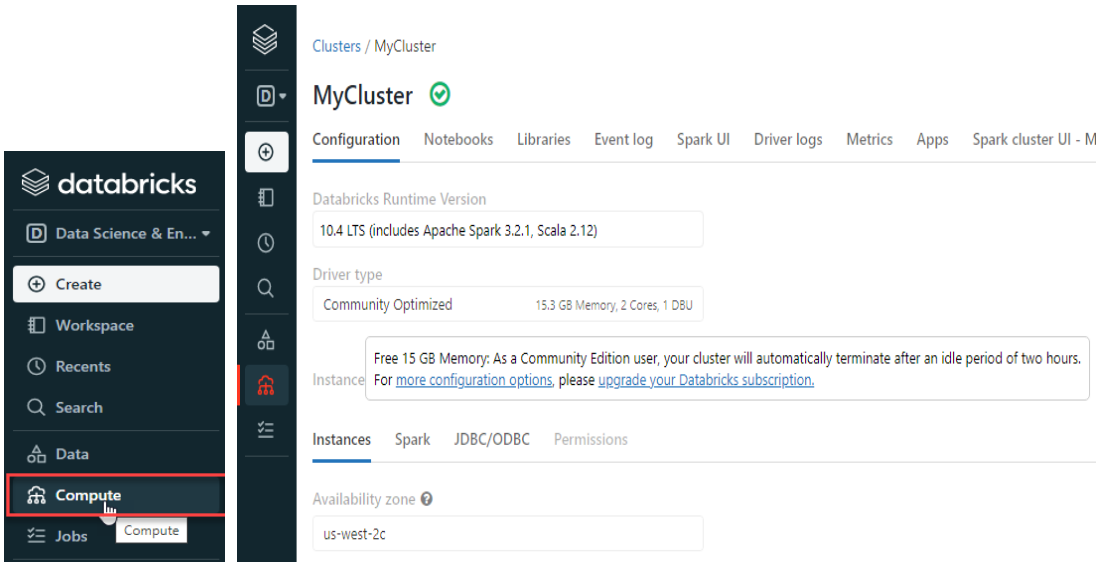
1.1 Creating a Databricks account [1]

Since this project has been implemented in Databricks a free Community Edition account has been created. Databricks free Community Edition account can be logged in via https://community.cloud.databricks.com.



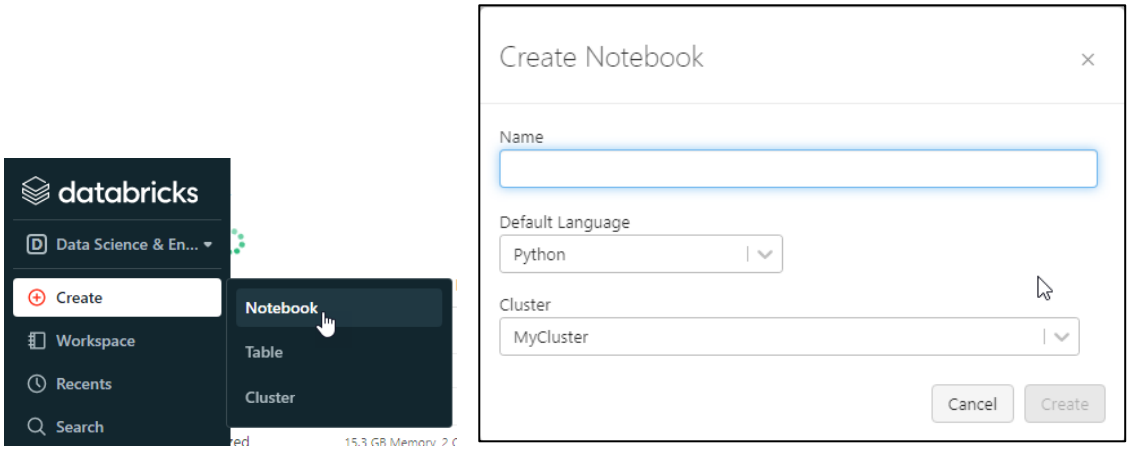
1.2 Creating a new cluster on Databricks [1]

To access to a machine, a new cluster have to be created. To do this, in the home page select the Compute in the left side panel. After clicking the "Create Cluster" button on the following page and giving the cluster a name, a new cluster will be created.



1.3 Creating a notebook

Databricks programs are written in Notebooks. A notebook is a collection of runnable cells which contain the commands. To create a new notebook, on the left-hand side, select Create and then select “Notebook". In this step, the name of notebook, its language (Python, Scala, SQL, R) and the name of the cluster have to be entered.

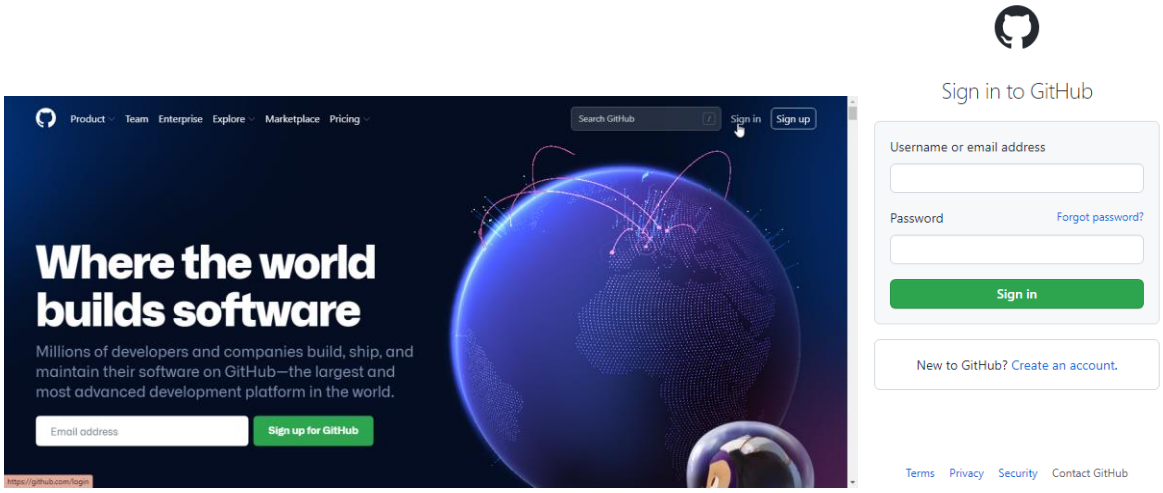


1.4 Using UNIX, Python, SQL commands in a notebook

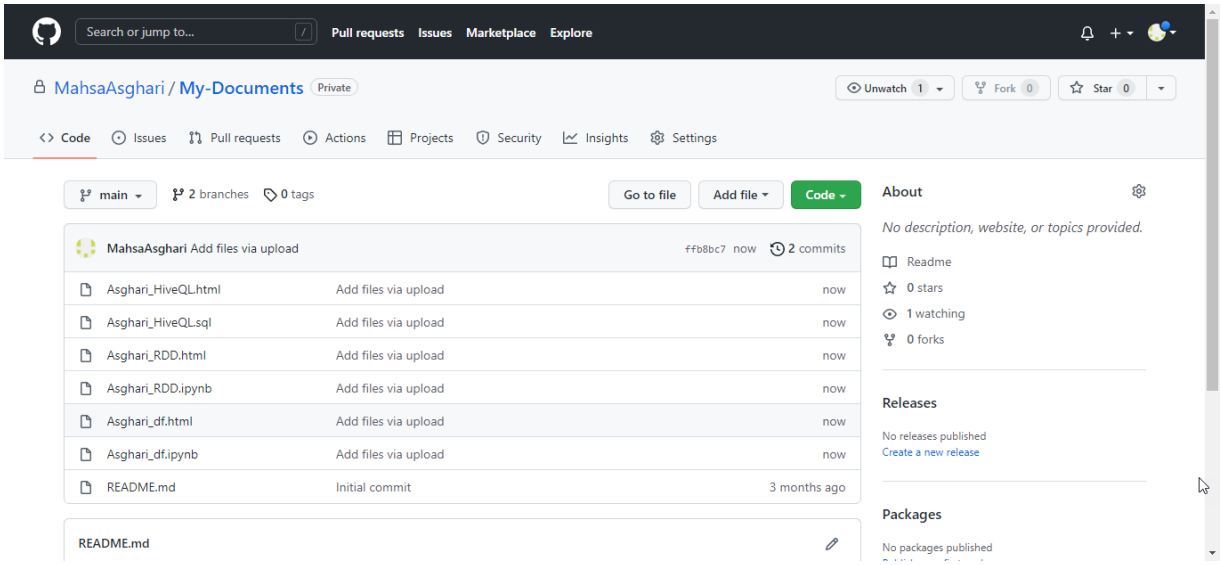
It is possible to write UNIX commands in a notebook by writing %sh at the beginning of the cell. %sql and %python can also be used to write a SQL or Python commands in a cell.

1.5 Version control

Since GitHub version control doesn't work with Community Edition, to have a GitHub repository an account has been created in https://github.com/.



After Signing up on this website, a GitHub repository will be given to you on your local machine. To have a backup of the Databricks notebooks, they have been exported and added to this repository.



2 Data Cleaning and Data Preparation

2.1 Datasets

This project consists of five CSV files listed as follows:

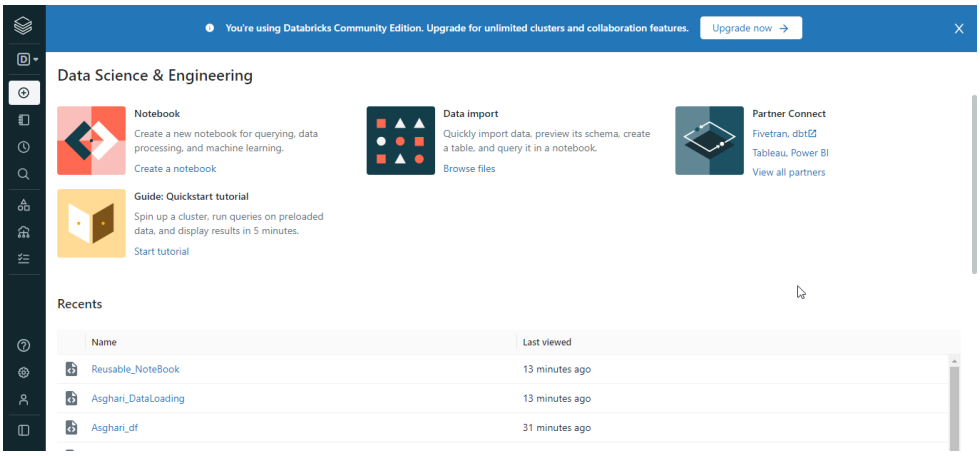
1. **clinicaltrial <year>.csv**: each row represents an individual clinical trial, identified by an Id, Sponsor, Status, Start and Completion date, the type of study, Submission date, and Conditions and the interventions. Individual conditions and interventions are separated by commas. [2]

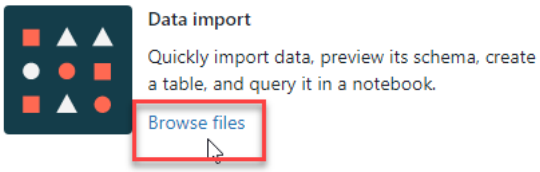
2. **mesh.csv**: the conditions from the clinical trial list may also appear in a number of hierarchies. The rows of this file contain condition (term), hierarchy identifier (tree) pairs. [2]

3. **pharma.csv**: the file contains a small number of a publicly available list of pharmaceutical violations. Parent Company contains the name of the pharmaceutical company. [2]

2.2 Data Loading

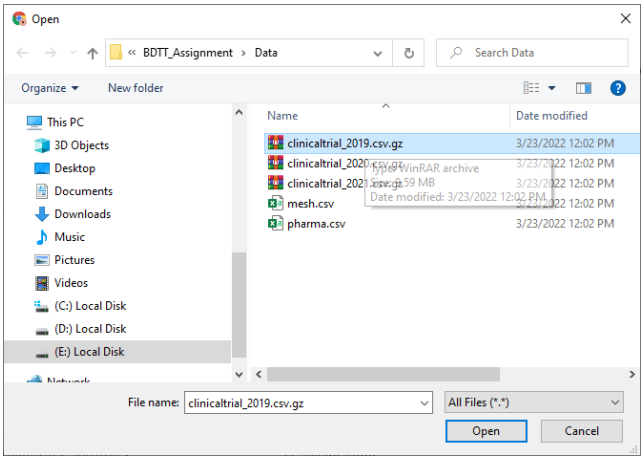
To import data onto Databricks, the Databricks user interface (UI) has been used.

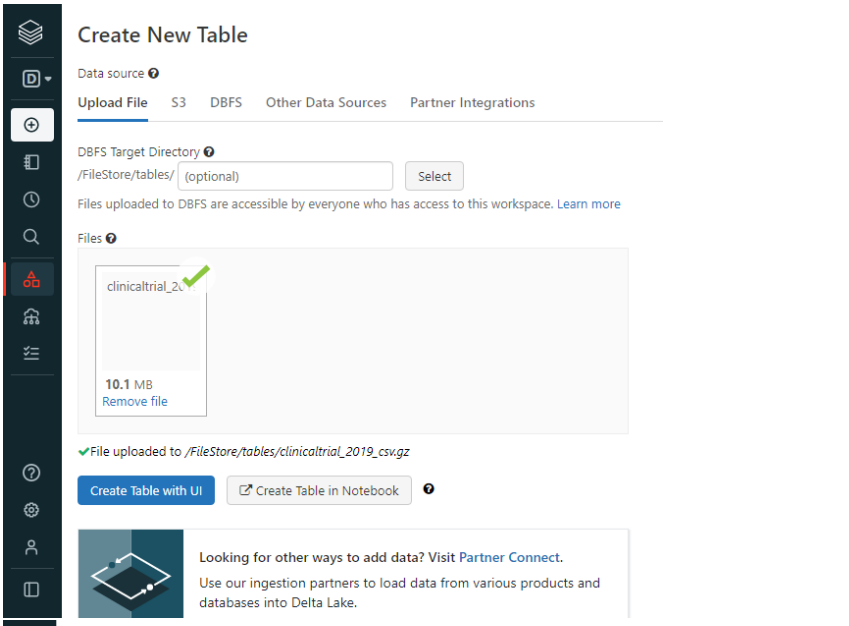


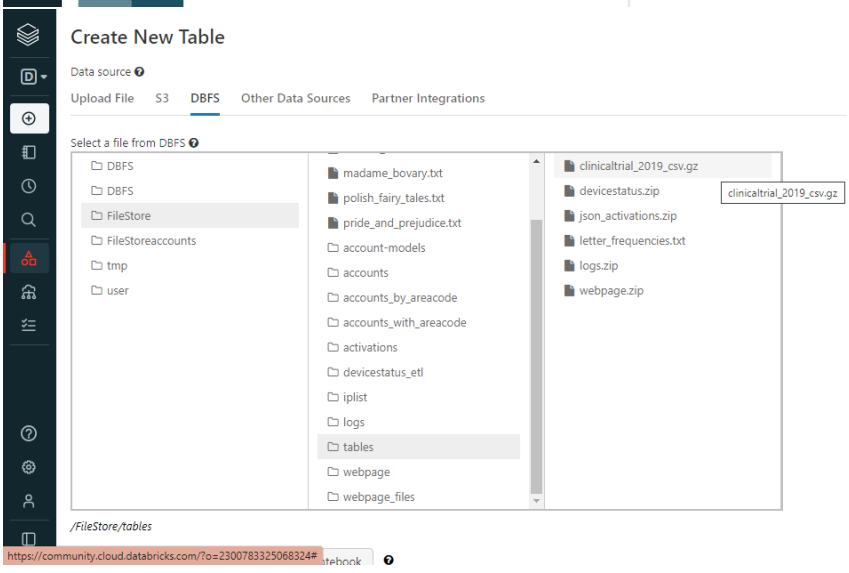


The datasets have been selected to import into Databricks during this step. Data has been stored in /FileStore/tables/.

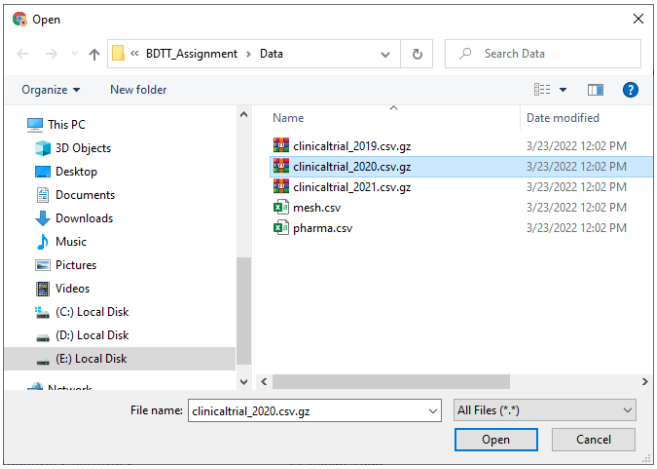
2.2.1 Importing Clinicaltrial_2019.csv.gz

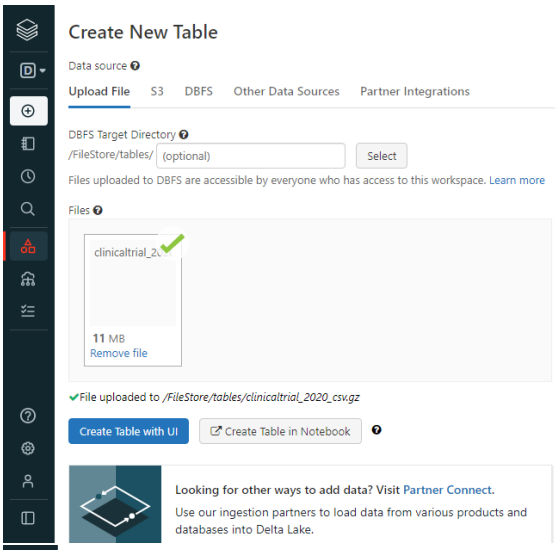


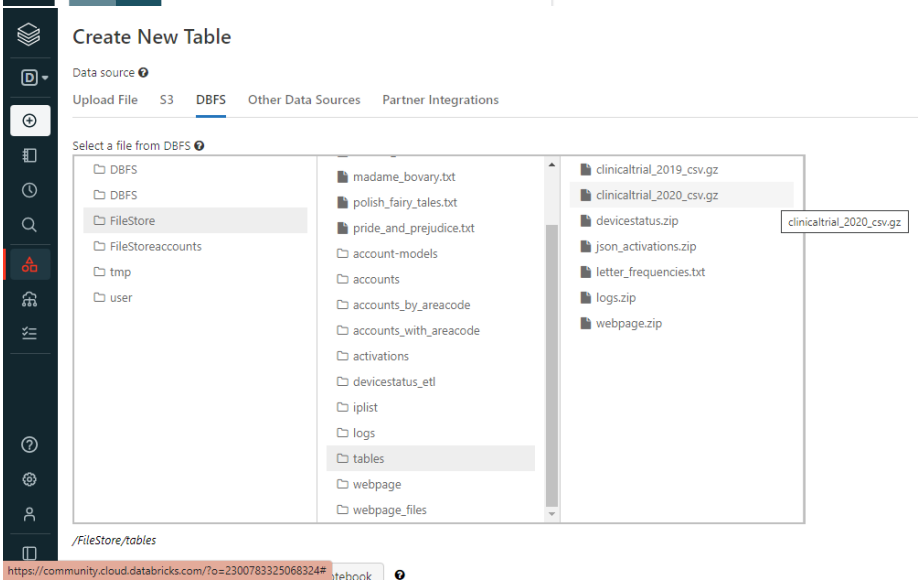

Create New Table
Data source
Upload File
S3
DBFS
Other Data Sources
Partner Integrations
DBFS Target Directory
/FileStore/tables/
(optional)
Select
Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more
Files
clinicaltrial_2
10.1 MB
Remove file
File uploaded to /FileStore/tables/clinicaltrial_2019_csv.gz
Create Table with UI
Create Table in Notebook
Looking for other ways to add data? Visit Partner Connect.
Use our ingestion partners to load data from various products and databases into Delta Lake.


Create New Table
Data source
Upload File
S3
DBFS
Other Data Sources
Partner Integrations
Select a file from DBFS
DBFS
DBFS
FileStore
FileStoreaccounts
tmp
user
madame_bovary.txt
polish_fairy_tales.txt
pride_and_prejudice.txt
account-models
accounts
accounts_by_areacode
accounts_with_areacode
activations
devicestatus_etl
iplist
logs
tables
webpage
webpage_files
clinicaltrial_2019_csv.gz
devicestatus.zip
json_activations.zip
letter_frequencies.txt
logs.zip
webpage.zip
clinicaltrial_2019_csv.gz
/FileStore/tables
https://community.cloud.databricks.com/?o=2300783325068324#

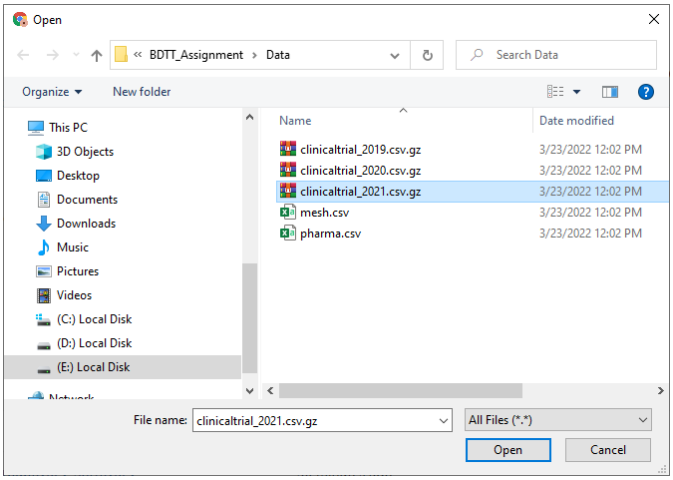
2.2.2 Importing Clinicaltrial_2020.csv.gz

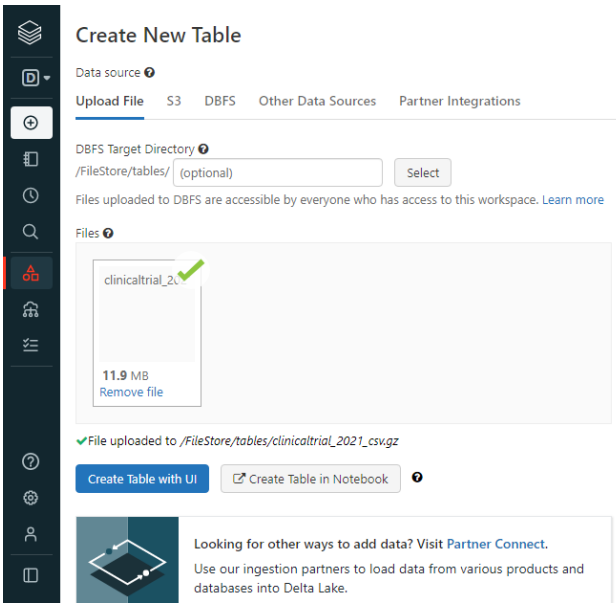


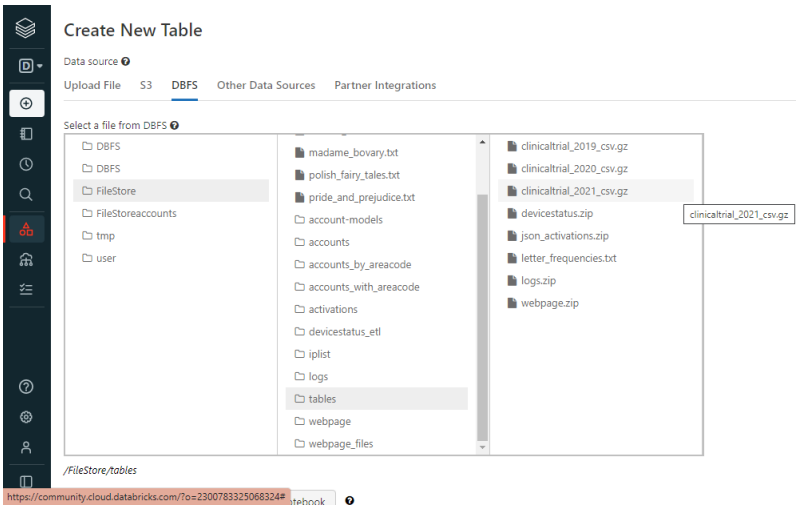




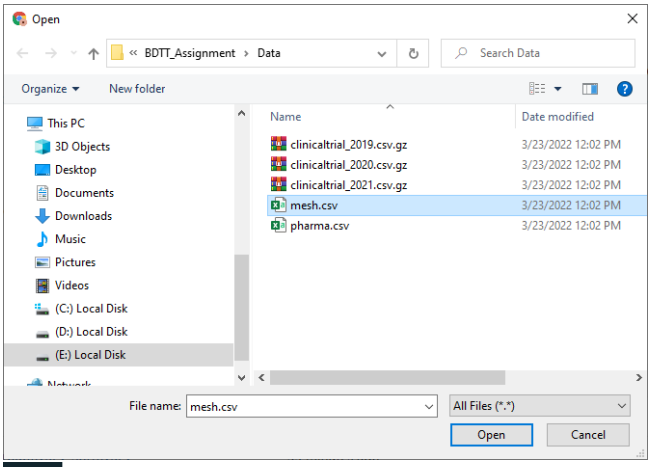
2.2.3 Importing Clinicaltrial_2021.csv.gz

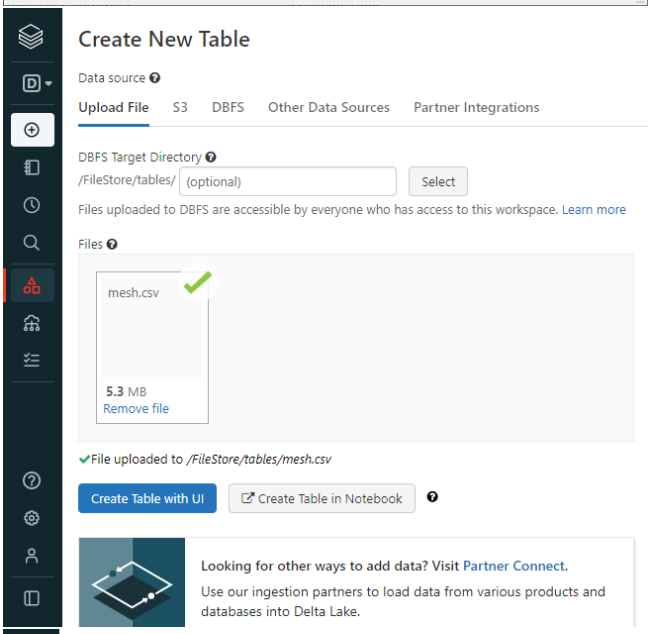


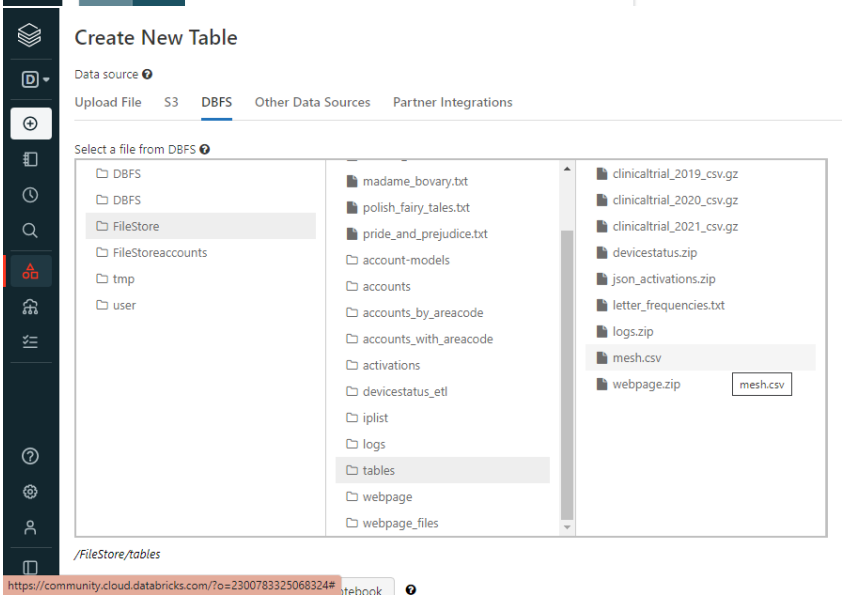




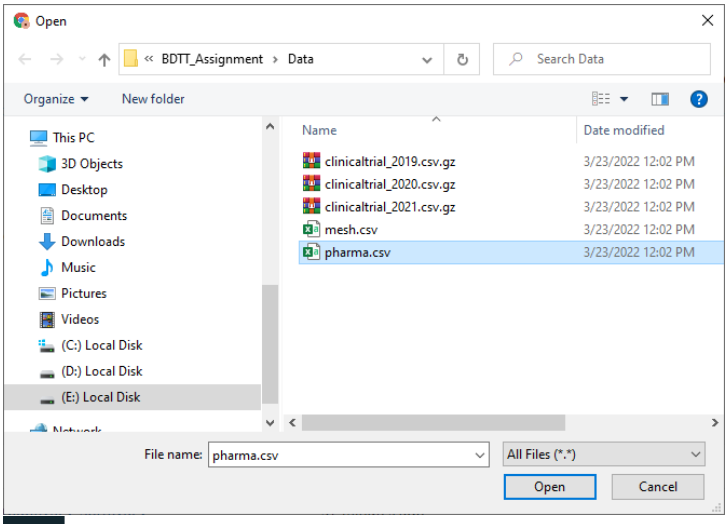
2.2.4 Importing mesh.csv

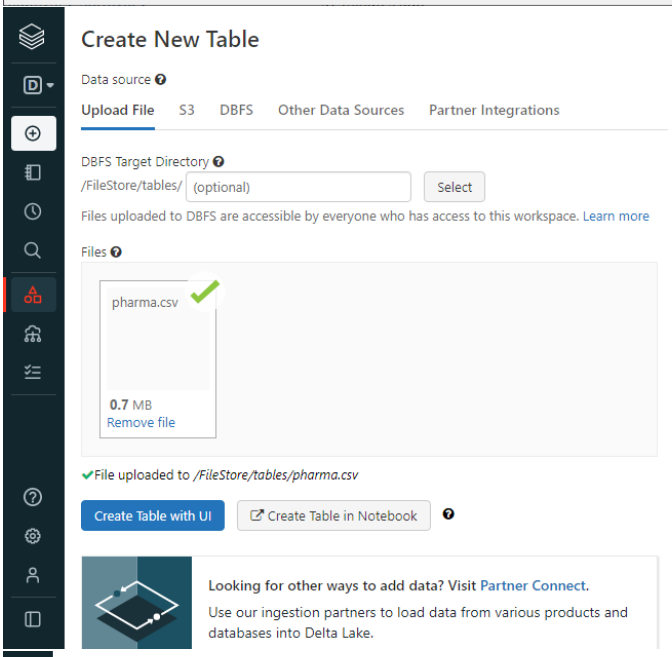


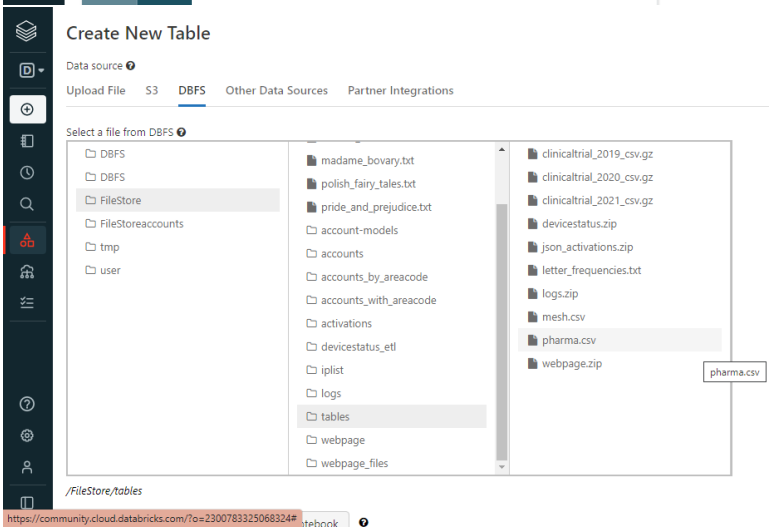




2.2.5 Importing pharma.csv



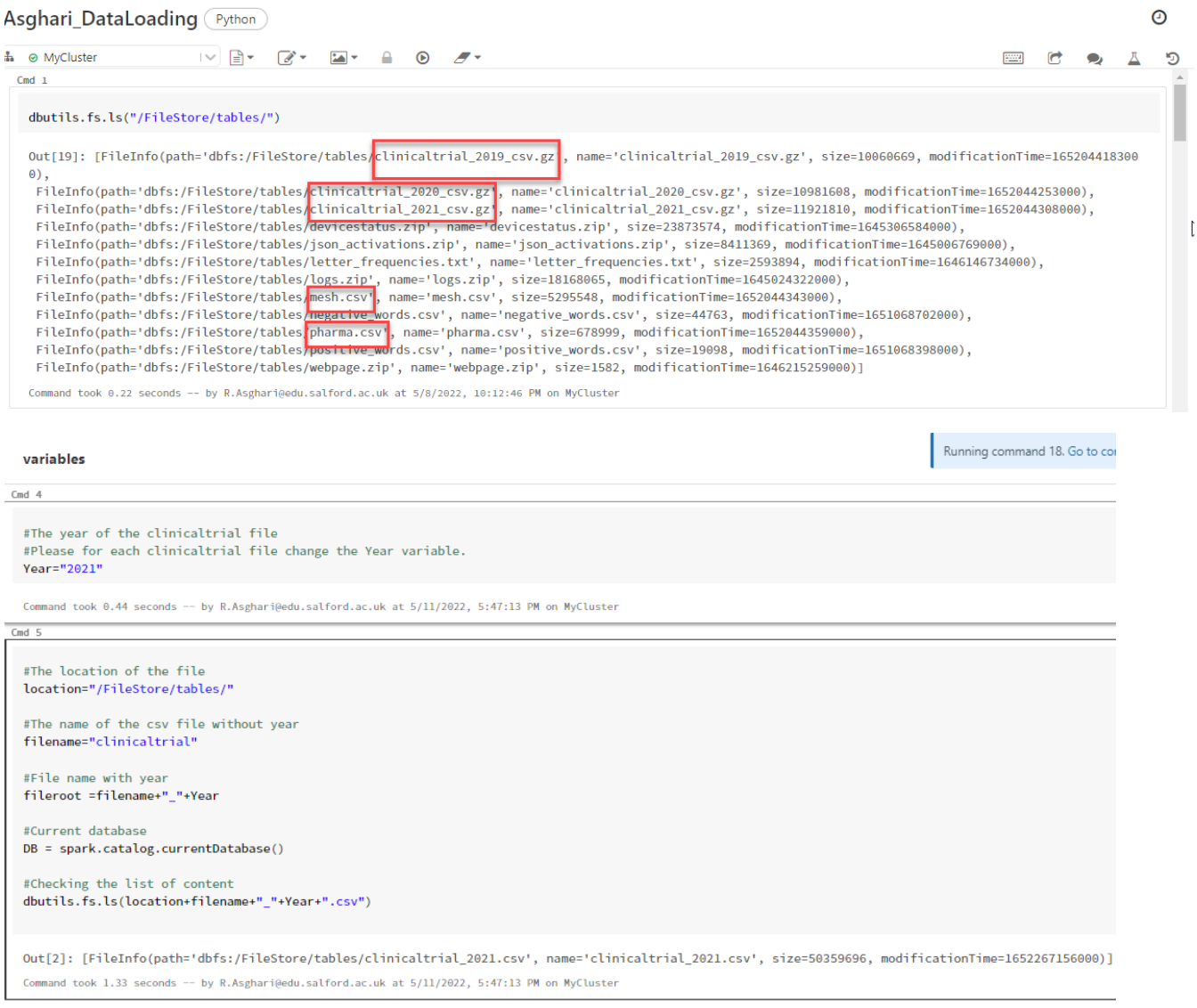




2.3 Renaming & Extracting Clinical Datasets

A reusable Python notebook named Asghari_Dataloading has been created to prepare datasets and HiveQL tables. Since there are three clinicaltrial datasets from 2019 to 2021 a variable named “Year” has been declared. By changing this variable and running the notebook, each file would be prepared. To clean and prepare data two steps are needed:

1. The name of clinicaltrial files needs to be changed from “_csv.gz” to “.csv.gz”.
2. As the dbutils do not support zip files, they have to be Extracted. Therefore, files are copied to a local drive, extracted there and then moved to DBFS.



Functions

Stop running all cells in this notebook.

Cmd 7

```
#Create a function to rename files from _csv.gz to .csv.gz
def Rename_Files(fileroot):
    #the location of file
    FilePath ="/FileStore/tables/" + fileroot+"_csv.gz"
    try:
        #if file exist:
        if len(dbutils.fs.ls(FilePath))==1:
            #rename file name
            dbutils.fs.mv(FilePath,"/FileStore/tables/"+fileroot+".csv.gz")
            dbutils.fs.ls("/FileStore/tables/" + fileroot+".csv.gz" )
    except:
        #if file does not exist:
        print("The file /FileStore/tables/" + fileroot+"_csv.gz does not exist.")
```

Command took 0.06 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:13 PM on MyCluster

Cmd 8

```
#Create function to copy .gz files from DBFS to /tmp directory
def Copy_Files_To_Tmp(fileroot):
    try:
        #Copy data from DBFS to file:/tmp/
        dbutils.fs.cp("/FileStore/tables/" + fileroot + ".csv.gz", "file:/tmp/")
        #Declare an environment variable to be usable in shell commands
        import os
        os.environ['fileroot']= fileroot
        dbutils.fs.ls("file:/tmp/" + fileroot + ".csv.gz")
    except:
        #If the file .csv.gz does not exist
        print("The file /FileStore/tables/" + fileroot + ".csv.gz does not exist.")
```

Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:13 PM on MyCluster

Rename a file from _csv.gz to .csv.gz

Cmd 7

```
#Control the file exists or not
try:
    print(dbutils.fs.ls("/FileStore/tables/" + fileroot+"_csv.gz" ))
except:
    print("The file /FileStore/tables/" + fileroot+"_csv.gz does not exist.")
```

```
[FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021_csv.gz', name='clinicaltrial_2021_csv.gz', size=11921810, modificationTime=1652044308000)]
```

Command took 0.10 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:03:41 PM on MyCluster

Cmd 8

```
#Execute the function
Rename_Files(fileroot)
```

Command took 1.50 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:03:51 PM on MyCluster

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

```
Out[55]: [FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2019.csv', name='clinicaltrial_2019.csv', size=42400056, modificationTime=1652047090000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2019.csv.gz', name='clinicaltrial_2019.csv.gz', size=10060669, modificationTime=1652045682000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2020.csv', name='clinicaltrial_2020.csv', size=46318151, modificationTime=1652047310000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2020.csv.gz', name='clinicaltrial_2020.csv.gz', size=10981608, modificationTime=1652047241000),
 FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv.gz', name='clinicaltrial_2021.csv.gz', size=11921810, modificationTime=1652047433000),
 FileInfo(path='dbfs:/FileStore/tables/devicestatus.zip', name='devicestatus.zip', size=23873574, modificationTime=1645306584000),
 FileInfo(path='dbfs:/FileStore/tables/json_activations.zip', name='json_activations.zip', size=8411369, modificationTime=1645006769000),
 FileInfo(path='dbfs:/FileStore/tables/letter_frequencies.txt', name='letter_frequencies.txt', size=2593894, modificationTime=1646146734000),
 FileInfo(path='dbfs:/FileStore/tables/logs.zip', name='logs.zip', size=18168065, modificationTime=1645024322000),
 FileInfo(path='dbfs:/FileStore/tables/mesh.csv', name='mesh.csv', size=5295548, modificationTime=1652044343000),
 FileInfo(path='dbfs:/FileStore/tables/negative_words.csv', name='negative_words.csv', size=44763, modificationTime=1651068702000),
 FileInfo(path='dbfs:/FileStore/tables/pharma.csv', name='pharma.csv', size=678999, modificationTime=1652044359000),
 FileInfo(path='dbfs:/FileStore/tables/positive_words.csv', name='positive_words.csv', size=19098, modificationTime=1651068398000),
 FileInfo(path='dbfs:/FileStore/tables/webpage.zip', name='webpage.zip', size=1582, modificationTime=1646215259000)]
```

Command took 0.15 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:03:59 PM on MyCluster

Copy .gz file from dbfs:/FileStore/tables/ to file:/tmp/

Cmd 11

```
#Execute Function for .csv.gz file to copy file from dbfs:/FileStore/tables/ to file:/tmp/
Copy_Files_To_Tmp(fileroot)
```

Command took 0.52 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:08:47 PM on MyCluster

Cmd 12

```
try:
    print(dbutils.fs.ls("file:/tmp/"+fileroot+".csv.gz"))
except:
    print("The file file:/tmp/"+fileroot+".csv.gz does not exist.")
```

```
[FileInfo(path='file:/tmp/clinicaltrial_2021.csv.gz', name='clinicaltrial_2021.csv.gz', size=11921810, modificationTime=1652047728214)]
```

Command took 0.04 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:08:52 PM on MyCluster

Extract a gz file:

Cmd 15

```
%sh
gunzip -d /tmp /tmp/$fileroot.csv.gz
```

```
gzip: /tmp is a directory -- ignored
```

Command took 0.38 seconds -- by R.Asghari@edu.salford.ac.uk at 4/1/2022, 1:21:10 PM on MyCluster

Cmd 15

```
dbutils.fs.ls("file:/tmp/"+fileroot+".csv")
```

```
Out[59]: [FileInfo(path='file:/tmp/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1652047728214)]
```

Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:12:14 PM on MyCluster

Move the file from local /tmp directory into DBFS

Cmd 17

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

```
Out[60]: True
```

```
Command took 2.42 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:15:24 PM on MyCluster
```

Cmd 18

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

```
Out[61]: [FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1652048126000)]
```

```
Command took 0.08 seconds -- by R.Asghari@edu.salford.ac.uk at 5/8/2022, 11:15:29 PM on MyCluster
```

2.4 Creating Hive tables

The second section of DataLoading notebook is for creating HiveQL tables. A function named createTable_From_CSV has been defined to create a hive table from a csv file. Using this function, a dataframe has been created from a csv file and then this dataframe converted to a hive table in the “default” database.

```
#A function to create a Hive table from a csv file.
def createTable_From_CSV (location, filename,delimiter):
    # File type
    file_type = "csv"
    #Declaring file location based on the file name
    if filename=="clinicaltrial":
        #Append the Year variable to a clinicaltrial file
        FilePath= location+filename+"_"+Year+"."+file_type
    else:
        FilePath = location+filename+"."+file_type #file path
    # CSV options
    infer_schema = True
    first_row_is_header = True

    #creating a dataframe from a csv file
    df = spark.read.format(file_type) \
    .option ("inferSchema", infer_schema) \
    .option ("header", first_row_is_header) \
    .option ("sep", delimiter) \
    .load (FilePath)

    try:
         #Removing the directory if it exists.By default, data is stored in the /user/hive/warehouse directory.
           if len(dbutils.fs.ls("dbfs:/user/hive/warehouse/"+filename))>=1:
               dbutils.fs.rm("dbfs:/user/hive/warehouse/"+filename,True)
               print("dbfs:/user/hive/warehouse/"+filename+ " was removed.")
    except:
            pass
    DB = spark.catalog.currentDatabase()#Current database
    tables = spark.catalog.listTables(DB)#List of tables existed in current database
    table_list= [table.name for table in tables]  #a list of the name of tables
    exists = filename in table_list  #If filename exists in the above list or not
    #if the table doesn not exists convert dataframe to a table
    if exists:
        print("A Table named "+filename+" exist.")
    else:
        df.write.saveAsTable(filename)
        print("A Table named "+filename+" was created.")
```

Creating Hive Table from Clinicaltrial files

Cmd 25

```
%sql
--Drop table if the table exists
drop table if exists Clinicaltrial
```

OK

Command took 0.56 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

Cmd 26

```
#Create a table from Clinicaltrial file which is a pip delimited file.
createTable_From_CSV(location, filename,"|")

#Controlling the creation of the table
spark.catalog.listTables(DB)
```

▸ (7) Spark Jobs

```
dbfs:/user/hive/warehouse/clinicaltrial was removed.
A Table named clinicaltrial was created.
Out[15]: [Table(name='clinicaltrial', database='default', description=None, tableType='MANAGED', isTemporary=False)]
```

Command took 50.38 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

Creating Hive Table from mesh.csv

Cmd 28

```
%sql
--Dropping the table if the table exists
drop table if exists mesh
```

OK

Command took 0.34 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

Cmd 29

```
#creating mesh table fron mesh.csv
createTable_From_CSV("/FileStore/tables/","mesh",",")
spark.catalog.listTables(DB)
```

▸ (9) Spark Jobs

```
dbfs:/user/hive/warehouse/mesh was removed.
A Table named mesh was created.
Out[16]: [Table(name='clinicaltrial', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='mesh', database='default', description=None, tableType='MANAGED', isTemporary=False)]
```

Command took 11.96 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

Creating Hive Table from Pharma.csv

Cmd 31

```
%sql
--Dropping the table if the table exists
drop table if exists pharma
```

OK

Command took 0.15 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

Cmd 32

```
#creating pharma table fron pharma.csv
createTable_From_CSV("/FileStore/tables/","pharma",",")
spark.catalog.listTables(DB)
```

▸ (11) Spark Jobs

```
dbfs:/user/hive/warehouse/pharma was removed.
A Table named pharma was created.
Out[17]: [Table(name='clinicaltrial', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='mesh', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='pharma', database='default', description=None, tableType='MANAGED', isTemporary=False)]
```

Command took 10.39 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 5:47:14 PM on MyCluster

```
#Controlling the default hive directory
dbutils.fs.ls("dbfs:/user/hive/warehouse/")
```

```
Out[18]: [FileInfo(path='dbfs:/user/hive/warehouse/clinicaltrial/', name='clinicaltrial/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/user/hive/warehouse/mesh/', name='mesh/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/user/hive/warehouse/pairdf/', name='pairdf/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/user/hive/warehouse/pairtable/', name='pairtable/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/user/hive/warehouse/pharma/', name='pharma/', size=0, modificationTime=0),
 FileInfo(path='dbfs:/user/hive/warehouse/zip_level_risk_v1_1/', name='zip_level_risk_v1_1/', size=0, modificationTime=0),
```

3 Problem answers

3.1 Question 1

3.1.1 Assumptions

- The Dataset is Clinicaltrial_2021.csv located in “dbfs:/FileStore/tables/clinicaltrial_2021.csv”.
- clinicaltrial_2021.csv is a pip-delimited file.
- Each row represents an individual clinical trial.

3.1.2 DataFrame implementation outline

- The clinicaltrial_2021.csv has been converted to a dataframe. Then each unique record has been counted.
- A reusable notebook with a variable named Year has been created. This variable allows us to change the file name automatically. When the "Year" is changed, the notebook will run for a different clinicaltrial file.
- The second variable is "File_Path", which stores the location of the file.
- A function named "create_DataFrame_From_CSV" has been defined to create a dataframe from a CSV file.

```
#please change the Year for each clinicaltrial file
#The year of the clinicaltrial file
Year="2021"
#The location of the file
File_Path="/FileStore/tables/clinicaltrial_"+Year+".csv"
# to see the list of content
dbutils.fs.ls(File_Path)
```

```
Out[1]: [FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1652048126000)]
Command took 0.10 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:01:16 PM on MyCluster
```

Functions:

Cmd 5

```
#A function to create a dataframe from csv file.
def create_DataFrame_From_CSV (File_Path,delimiter):
    csvDF= spark.read.format("csv") \
    .option("inferSchema", True) \
    .option("header", True) \
    .option("sep", delimiter) \
    .load(File_Path)
    return csvDF
```

```
Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 2:24:13 PM on MyCluster
```

```
#Creating a dataframe called clinicaltrialDF from the clinical trials dataset.
clinicalDF=create_DataFrame_From_CSV(File_Path,"|")
clinicalDF.distinct().count()
```

```
▸ (5) Spark Jobs
Out[25]: 387261
Command took 10.98 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 7:18:20 PM on MyCluster
```

3.1.3 HiveQL implementation outline

Hive tables has been created in the Dataloading notebook. A reusable SQL notebook with a variable named “Year” has been created. The count of distinct records of table has been computed.

Hive tables create in Asghari_DataLoading.ipynb.

Please make sure Asghari_DataLoading.ipynb has been executed before running this notebook.

Cmd 2

```
--The year of clinicaltrial file
set Year=2021
```

	key	value
1	Year	2021

Showing all 1 rows.

Command took 0.30 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:13 PM on MyCluster

Cmd 3

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

Cmd 4

```
--The number of unique studies in the clinicaltrial table
select count(distinct id) Frequency from  clinicaltrial
```

(3) Spark Jobs

	Frequency
1	387261

Showing all 1 rows.

Command took 3.04 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:24:00 PM on MyCluster

Cmd 5

3.1.4 RDD implementation outline

A reusable notebook with the variables named Year and File_Path has been created. A function named "create_RDD_From_CSV_without_Header" has been defined to convert a csv file to an RDD without a header. Inside this function, another function is called named "Remove_Header_From_RDD". This function removes the first element of RDD which is the header of CSV file. Using create_RDD_From_CSV_without_Header, an RDD has been created from the clinicaltrial_2021.CSV. As this file is pip-delimited, the elements of RDD have been split by pip. The results are kept in an RDD named clinical_RDD. Finally, the number of distinct items of RDD has been counted.

Variables:

Cmd 3

```
#Please change Year for each clinical data
#The year of the clinicaltrial file
Year="2021"

#The location of the file
File_Path="/FileStore/tables/clinicaltrial_"+Year+".csv"

# to see the list of content
dbutils.fs.ls(File_Path)
```

```
Out[1]: [FileInfo(path='dbfs:/FileStore/tables/clinicaltrial_2021.csv', name='clinicaltrial_2021.csv', size=50359696, modificationTime=1652048126000)]
```

Command took 0.29 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:04:09 PM on MyCluster

```
#Difining a function to remove the header of a RDD
def Remove_Header_From_RDD (RDD):

    #Taking the Header of clinical_RDD
    Header= RDD.first()

    #Removing the header of RDD
    RDD_withoutHeader = RDD.filter(lambda l : l != Header)

    return RDD_withoutHeader
```

Command took 0.02 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:04:09 PM on MyCluster

Cmd 7

```
#Defining a function to creat an RDD from a csv file and remove the header
def create_RDD_From_CSV_without_Header (File_Path):
    rdd = Remove_Header_From_RDD(sc.textFile(File_Path))
    return rdd
```

Command took 0.02 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:04:09 PM on MyCluster

```
#Executing function to create RDD from CSV file
#split clinical_RDD by pip (|)
clinical_RDD =create_RDD_From_CSV_without_Header(File_Path)\
.map(lambda l : l.split('|'))#Creating a list of elements

clinical_RDD.take(10)
```

▸ (2) Spark Jobs

```
Out[7]: [['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',
```

Command took 2.60 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:04:09 PM on MyCluster

```
#Counting the distinc of elements
clinical_RDD.map(lambda l : l[0]).distinct().count()
#2019:326348
#2020:356466
#2021:387261
```

▸ (1) Spark Jobs

```
Out[38]: 387261
Command took 3.81 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 5:12:01 PM on MyCluster
```

3.1.5 Result on the submission (final) data

Dataframe Implementation:

```
Out[25]: 387261
```

HiveQL Implementation:

	Frequency
1	387261

RDD Implementation:

```
Out[8]: 387261
```

3.1.6 Discussion of result

According to the result, there is 387261 unique study id in clinicaltrial_2021.csv. In addition, without using the distinct, we have 387261 studies as well. If the number of studies is computed based on different columns, the result can be different because some fields have null values like Submission, Completion, Conditions and Interventions. Therefore, to count the number of studies, Id is the best column that is unique and not null. Although for each study there is just one record in the dataset, some studies have more than one condition and Interventions.

```
--The count of records based on different columns
select 'id' column_name, count(id) cnt from clinicaltrial
union all
select 'Sponsor',count(Sponsor) from clinicaltrial
union all
select 'Status' , count(Status) from clinicaltrial
union all
select 'Start' , count(Start) from clinicaltrial
union all
select 'Type' , count(Type) from clinicaltrial
union all
select 'Submission' , count(Submission) from clinicaltrial
union all
select 'Completion' , count(Completion) from clinicaltrial
union all
select 'Conditions', count(Conditions) from clinicaltrial
union all
select 'Interventions', count(Interventions) from clinicaltrial
```

▸ (18) Spark Jobs

	column_name	cnt
1	id	387261
2	Sponsor	387261
3	Status	387261
4	Start	387261
5	Type	387261
6	Submission	387261
7	Completion	374001

Showing all 9 rows.

column_name	cnt
id	387261
Sponsor	387261
Status	387261
Start	387261
Type	387261
Submission	387261
Completion	374001
Conditions	322130
Interventions	133424

3.2 Question 2

3.2.1 Assumptions made

- The dataset is clinicaltrial_2021.csv which is a pip-delimited file.

- The sixth column of dataset is “Type", which shows the type of studies.
- There are four types of studies, including Observational [Patient Registry], Expanded Access, Interventional and Observational

3.2.2 Dataframe implementation outline

The data have to be grouped by "Type" and then the number of records within each type has to be counted. Finally, the frequency of each type has to be sorted in descending order. To count the frequency of each column in a dataframe a function has been defined named “Frequencies_Of_Column”.

```
#A function to calculate the frequencies of a column in a dataframe. The result is sorted in descending order
from pyspark.sql.functions import col
def Frequencies_Of_Column (DF,Column):
    return DF.groupBy(Column).count()\
    .sort(col("count").desc())#The result is sorted in descending order
```

```
Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/9/2022, 7:55:34 PM on MyCluster
```

```
#Using the function to count the frequency of each study type and sorting frequencies in a descending order
display(Frequencies_Of_Column(clinicalDF,"Type"))
```

▸ (2) Spark Jobs

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

Showing all 4 rows.

```
Command took 4.41 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster
```

3.2.3 HiveQL implementation outline

By grouping data by Type and using the count function in the clinicaltrial table, we can find the frequency of each type of study. Based on the frequency of each study type, the results were sorted in descending order.

```
--The frequency of each study type
select type, count(*) cnt from clinicaltrial
group by type
order by cnt desc;
```

▸ (2) Spark Jobs

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

Showing all 4 rows.

Command took 2.33 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

3.2.4 RDD implementation outline

A pair-rdd has been created from the fifth item (type) of the clinical_RDD. Using Frequency_Of_pair_RDD function, the frequency of each type has been computed. By using a function named print_Pair_RDD the specific number of the elements of an RDD will be printed.

```
def print_Pair_RDD (rdd,num):
    for l in rdd.take(num):
        print(l[0] ,' ', l[1])
```

Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:58 PM on MyCluster

Cmd 9

```
#using reduceByKey to have the frequency of each key
#using sortBy to sort the pair RDD on its values which is the frequency of each key.
#This function take a pair-rdd as a parameter. Finally, return a pair-rdd which is sorted by the frequency of keys.

def Frequency_Of_pair_RDD (rdd):
    F_rdd=rdd.reduceByKey(lambda x,y : x+y)\
    .sortBy(lambda x : x[1],False)
    return F_rdd
```

Command took 0.03 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:58 PM on MyCluster

```
#TYPE is the 5th element of clinical_RDD.
#Creating a pair RDD in which the key is TYPE and the value is one.
#Using Frequency_Of_pair_RDD to compute the frequency of each type.

Type_frequency_RDD = Frequency_Of_pair_RDD(clinical_RDD.map(lambda l : (l[5],1)))

#print result
print_Pair_RDD(Type_frequency_RDD,10)
```

▸ (4) Spark Jobs

```
Interventional   301472
Observational   77540
Observational [Patient Registry]   8180
Expanded Access   69
```

Command took 3.24 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:58 PM on MyCluster

3.2.5 Result on the submission (final) data

Dataframe Implementation:

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

HiveQL Implementation:

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

RDD Implementation:

```
Interventional    301472
Observational    77540
Observational [Patient Registry]    8180
Expanded Access    69
```

3.2.6 Discussion of result

In the clinicaltrial_2021.csv, the most common type of study is the "Interventional" studies with 301472 case studies. Observational studies are the second most popular with 77540 studies. Observational [Patient Registry] with 8180 is ranked the third frequent type of study. Expanded Access studies are the last types of studies with 69 studies. Some Interventional studies have been completed without any interventions. In this analysis, the frequency of these studies has been listed by the completion year. The year 2019 has the most such data.

```
select right(Completion,4) Year, Count(*) cnt from clinicaltrial
where type like "Interventional"-- Interventional Studies
and Status ="Completed"--Completed Studies
and Interventions is null-- without any Interventions
group by Year
order by cnt desc
limit (5)
```

▸ (2) Spark Jobs

	Year	cnt
1	2019	9508
2	2018	8958
3	2017	8409
4	2020	8103
5	2016	7726

Showing all 5 rows.

Command took 1.67 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 2:24:25 PM on MyCluster

In addition, there are no completed Expanded Access studies is the table.

```
--The frequency of completed studied bu study type
select type, count(*) cnt from clinicaltrial
where status="Completed"
group by type
order by cnt desc;
```

▸ (2) Spark Jobs

	type	cnt
1	Interventional	166951
2	Observational	40276
3	Observational [Patient Registry]	2522

Showing all 3 rows.

Command took 2.79 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 8:42:46 PM on MyCluster

3.3 Question 3

3.3.1 Assumptions made

- The conditions associated with a study are separated by commas. The conditions column should be organized into distinct rows.
- After producing a dataset with separated condition in each row, we need to group dataset by conditions column and then count the number of studies in each condition. Finally, it is needed to sort the result in a descending order.

3.3.2 Dataframe implementation outline

Using the split and explode function, the conditions column has been separated into different rows. Using the Frequencies_Of_Column, the number of conditions can be computed and sorted.

```
display(clinicalDF.select(split(col("Conditions"),",").alias("Conditions")))
```

▸ (1) Spark Jobs

	Conditions
1	null
2	["Autistic Disorder", "Autism Spectrum Disorder"]
3	["Diabetes Mellitus"]
4	["Tuberculosis", "Lung Diseases", "Pulmonary Disease"]
5	["Diverticular Diseases", "Diverticulum", "Diverticulosis"]
6	["Asthma"]
7	["Hypoventilation"]

Truncated results, showing first 1000 rows.

Click to re-execute with maximum result limits.

Command took 0.35 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 7:33:24 PM on MyCluster

Cmd 14

```
#Importing sql.functions  to use sql functions like explode,split and col.
from pyspark.sql.functions import *
#Using split command, a list of conditions is created.
#The explode command separates the list of conditions into different rows.
Splited_Conditions_DF=clinicalDF.select(explode(split(col("Conditions"),",")).alias("Conditions"))

#Using Frequencies_Of_Column function, to count the frequency of Conditions in descending order
display(Frequencies_Of_Column (Splited_Conditions_DF,"Conditions").limit(5))
```

▸ (2) Spark Jobs

	Conditions	count
1	Carcinoma	13389
2	Diabetes Mellitus	11080
3	Neoplasms	9371
4	Breast Neoplasms	8640
5	Syndrome	8032

Showing all 5 rows.

3.3.3 HiveQL implementation outline

A view has been created named VW_exploded_conditions in which using split and explode function, each row has a condition. Then the frequency of each condition has been computed.

```
--Creating a view in which each record has a condition
--Split function splits create a list of conditions
--Explode function spreads list of conditions into differrent rows(like flatMap in RDD)

CREATE OR REPLACE TEMP VIEW VW_exploded_conditions AS
SELECT Id,Sponsor,
Status,
Start,
Completion,
Type,
Submission,
explode(split(conditions , ',')) as conditions ,
Interventions
FROM clinicaltrial
```

OK

Command took 0.37 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

Cmd 9

```
--Frequency of top 5 conditions
select  conditions ,count(*) frequency from VW_exploded_conditions
group by conditions
order by frequency desc
limit(5)
```

▸ (2) Spark Jobs

	conditions	frequency
1	Carcinoma	13389
2	Diabetes Mellitus	11080
3	Neoplasms	9371
4	Breast Neoplasms	8640
5	Syndrome	8032

Showing all 5 rows.

Command took 2.67 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 7:05:45 PM on MyCluster

3.3.4 RDD implementation outline

The seventh item of the clinicaltrial dataset contains the conditions. With a flatMap and split function, each element has one condition. Then a pair-rdd with no null conditions has been created. The frequency of conditions has been computed using Frequency_Of_pair_RDD.

```
#Conditions is the 7th element of clinical_RDD .In some records, each condition contains more than one value which needs to be split by comma.
#Some conditions are null so we need to Filter the items which is not null.
Splited_Conditions_RDD= clinical_RDD.flatMap(lambda l : l[7].split(','))\
.filter(lambda x : x != '')\
.map(lambda x : (x,1))#creating a pair RDD . The Key is a condition and the value is one.

Splited_Conditions_RDD.take(10)
```

▸ (1) Spark Jobs

```
Out[11]: [('Autistic Disorder', 1),
 ('Autism Spectrum Disorder', 1),
 ('Diabetes Mellitus', 1),
 ('Tuberculosis', 1),
 ('Lung Diseases', 1),
 ('Pulmonary Disease', 1),
 ('Diverticular Diseases', 1),
 ('Diverticulum', 1),
 ('Diverticulosis', 1),
 ('Asthma', 1)]
```

Command took 1.00 second -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

Cmd 17

```
#Using Frequency_Of_pair_RDD to have the frequency of each condition
#printing The top 5 conditions with their frequencies
print_Pair_RDD(Frequency_Of_pair_RDD(Splited_Conditions_RDD),5)
```

▸ (3) Spark Jobs

```
Carcinoma   13389
Diabetes Mellitus   11080
Neoplasms   9371
Breast Neoplasms   8640
Syndrome   8032
```

Command took 3.87 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

3.3.5 Result on the submission (final) data

Dataframe Implementation

	Conditions	count
1	Carcinoma	13389
2	Diabetes Mellitus	11080
3	Neoplasms	9371
4	Breast Neoplasms	8640
5	Syndrome	8032

HiveQL Implementation:

	conditions	frequency
1	Carcinoma	13389
2	Diabetes Mellitus	11080
3	Neoplasms	9371
4	Breast Neoplasms	8640
5	Syndrome	8032

RDD Implementation

```
Carcinoma   13389
Diabetes Mellitus   11080
Neoplasms   9371
Breast Neoplasms   8640
Syndrome   8032
```

3.3.6 Discussion of result

In the study, the most prevalent condition was Carcinoma with 13389 cases studied. Diabetes Mellitus ranked second with 11080 cases studied. Neoplasms and Breast Neoplasms are placed in the third and fourth frequent studies with 9371 and 8640 cases studied respectively. Finally, Syndrome is ranked as the fifth most common condition.

Moreover, there are 65131 studies with null condition. When conditions are speared in different rows, null conditions are removed. For example, in the VW_exploded_conditions (in HiveQL implementation), the number of distinct studies (id) is 322130 while the number of studies in dataset is 387261.

```
--Discussion of result
--The number of distinct studies is 322130 while the number of studies in dataset is 387261.
select count(distinct(id)) from VW_exploded_conditions
```

▸ (3) Spark Jobs

	count(DISTINCT id)
1	322130

Showing all 1 rows.

Command took 3.60 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 3:24:05 PM on MyCluster

Cmd 13

```
select count(*) from clinicaltrial
where conditions is null
```

▸ (2) Spark Jobs

	count(1)
1	65131

Showing all 1 rows.

Command took 1.13 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 3:11:46 PM on MyCluster

3.4 Question 4

3.4.1 Assumptions made

- Mesh.csv is located in "/FileStore/tables/mesh.csv" which is a comma-delimited file.

- Mesh.csv contain condition (term), hierarchy identifier (tree) pairs. [2]
- We need to extract first three letters of “tree” as a root .
- We need to join clinicaltrial dataset and mesh dataset on conditions and term and then count the number of each root in the joined dataset.
-

3.4.2 Dataframe implementation outline

At first, a dataframe is created from mesh.csv then the root is extracted from “tree”. In the next step, the clinicaltrial dataframe with separated conditions in each row (Splited_Conditions_DF) is joined with a mesh dataframe on conditions and terms. The frequency of roots has been computed.

```
#Creating dataframe from a csv file via a function named "create_DataFrame_From_CSV"
#Extracting the 3 letters of the left side of "tree" as a root of each conditions
meshDF = create_DataFrame_From_CSV("/FileStore/tables/mesh.csv",",")\
.select("term",substring("tree",0,3).alias("root"))
display(meshDF)
```

▸ (3) Spark Jobs

	term	root
1	Calcimycin	D03
2	A-23187	D03
3	Temefos	D02
4	Temefos	D02
5	Temefos	D02
6	Abate	D02
7	Abate	D02

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 3.54 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

```
#inner join Splited_Conditions_DF and meshDF on conditions and term
joinDF=Splited_Conditions_DF.join(meshDF, Splited_Conditions_DF.Conditions == meshDF.term, "inner")
#display(joinDF)

#selecting conditions and root
joinDF_with_root=joinDF.select ("Conditions","root")
#display(joinDF_with_root)

#calculating the frequencies of each root code
display(Frequencies_Of_Column (joinDF_with_root,"root").limit(10))
```

▸ (2) Spark Jobs

	root	count
1	C04	143994
2	C23	136079
3	C01	106674
4	C14	94523
5	C10	92310
6	C06	85646
7	C08	70720

Showing all 10 rows.

Command took 7.25 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

3.4.3 HiveQL implementation outline

The root of conditions has been kept in a view named VW_mesh_root. By combining VW_exploded_conditions, which contains separated conditions in each row, and VW_mesh_root, the frequency of each root has been computed and sorted.

```
--Creating a view to extract root from tree column
CREATE OR REPLACE TEMP VIEW VW_mesh_root AS
select term,
left(tree,3) root --extract root of each tree
from mesh
```

OK

Command took 0.37 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

Cmd 12

```
select * from VW_mesh_root
```

▸ (1) Spark Jobs

	term	root
1	Calcimycin	D03
2	A-23187	D03
3	Temefos	D02
4	Temefos	D02
5	Temefos	D02
6	Abate	D02
7	Abate	D02

Truncated results, showing first 1000 rows.

Click to re-execute with maximum result limits.

Command took 1.01 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

```
--Frequency of 10 most frequent roots
--In the past result pdf file, instead of 5 most frequent roots, 10 most frequent roots had been written.
select root, count(*) frequency from VW_exploded_conditions c
inner join VW_mesh_root m
on c.conditions = m.term
group by root
order by frequency desc
limit(10);
```

▸ (2) Spark Jobs

	root	frequency
1	C04	143994
2	C23	136079
3	C01	106674
4	C14	94523
5	C10	92310
6	C06	85646
7	C08	70720

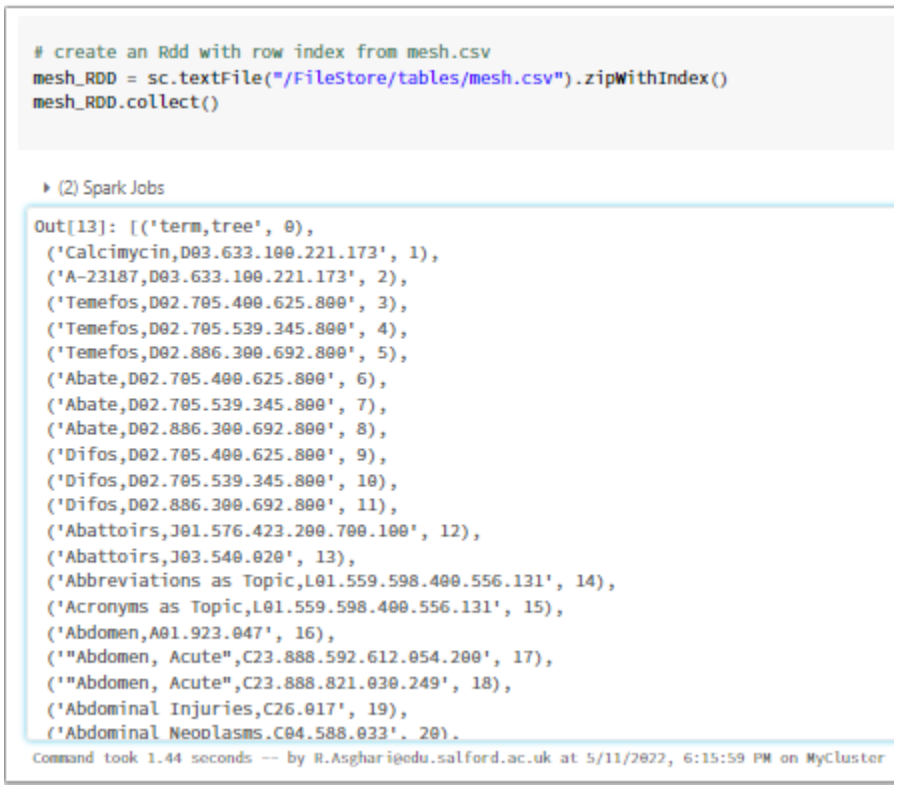
Showing all 10 rows.

Command took 5.21 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

3.4.4 RDD implementation outline

Since mesh.csv is a comma separated file and inside the file, term column is a comma separated list. Converting the file to RDD is complicated. To create an RDD from mesh.csv, two implementation with sc. textFile() and dataframe has been deployed. After creating a pair-rdd from mesh.csv, the root has been

extracted. Then mesh_root_rdd joined with Splited_Conditions_RDD. Finally, the frequency of each root has been computed.



```
# create an Rdd with row index from mesh.csv
mesh_RDD = sc.textFile("/FileStore/tables/mesh.csv").zipWithIndex()
mesh_RDD.collect()
```

▸ (2) Spark Jobs

```
Out[13]: [('term,tree', 0),
 ('Calcimycin,D03.633.100.221.173', 1),
 ('A-23187,D03.633.100.221.173', 2),
 ('Temefos,D02.705.400.625.800', 3),
 ('Temefos,D02.705.539.345.800', 4),
 ('Temefos,D02.886.300.692.800', 5),
 ('Abate,D02.705.400.625.800', 6),
 ('Abate,D02.705.539.345.800', 7),
 ('Abate,D02.886.300.692.800', 8),
 ('Difos,D02.705.400.625.800', 9),
 ('Difos,D02.705.539.345.800', 10),
 ('Difos,D02.886.300.692.800', 11),
 ('Abattoirs,J01.576.423.200.700.100', 12),
 ('Abattoirs,J03.540.020', 13),
 ('Abbreviations as Topic,L01.559.598.400.556.131', 14),
 ('Acronyms as Topic,L01.559.598.400.556.131', 15),
 ('Abdomen,A01.923.047', 16),
 ('"Abdomen, Acute",C23.888.592.612.054.200', 17),
 ('"Abdomen, Acute",C23.888.821.030.249', 18),
 ('Abdominal Injuries,C26.017', 19),
 ('Abdominal Neoplasms,C04.588.033', 20),
```

Command took 1.44 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
# Getting first row and create mesh_Header
#l[1]=>row index
#collect()[0]=> take ['term', 'tree'] from [['term', 'tree']]
mesh_Header = mesh_RDD.filter(lambda l: l[1] == 0) \
.map(lambda l: l[0].split(",")).collect()[0]
mesh_Header
```

▸ (1) Spark Jobs

```
Out[35]: ['term', 'tree']
```

Command took 0.87 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 8:08:55 PM on MyCluster

Cmd 22

```
#Removing header from mesh_RDD
#split data by comma
mesh_RDD_splited = mesh_RDD.filter(lambda l: l[1] > 0)\
.map(lambda l: l[0].split(","))
mesh_RDD_splited.collect()
```

▸ (1) Spark Jobs

```
Out[15]: [['Calcimycin', 'D03.633.100.221.173'],
 ['A-23187', 'D03.633.100.221.173'],
 ['Temefos', 'D02.705.400.625.800'],
 ['Temefos', 'D02.705.539.345.800'],
 ['Temefos', 'D02.886.300.692.800'],
 ['Abate', 'D02.705.400.625.800'],
 ['Abate', 'D02.705.539.345.800'],
 ['Abate', 'D02.886.300.692.800'],
 ['Difos', 'D02.705.400.625.800'],
 ['Difos', 'D02.705.539.345.800'],
 ['Difos', 'D02.886.300.692.800'],
 ['Abattoirs', 'J01.576.423.200.700.100'],
 ['Abattoirs', 'J03.540.020'],
 ['Abbreviations as Topic', 'L01.559.598.400.556.131'],
 ['Acronyms as Topic', 'L01.559.598.400.556.131'],
 ['Abdomen', 'A01.923.047'],
 ['"Abdomen', ' Acute"', 'C23.888.592.612.054.200'],
 ['"Abdomen', ' Acute"', 'C23.888.821.030.249'],
 ['Abdominal Injuries', 'C26.017'],
 ['Abdominal Neoplasms', 'C04.588.033'],
 ['Abdominal Muscles', 'A02.633.567.050'],
```

Command took 1.04 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
#some of terms have more than one values which are sepetrated by comma.
#So after spliting mesh_RDD_splited by comma it is needed to join those items together.
#We join comma with the elements of each row from 0 to -1 (except the last item)
#l[-1][0:3] => Extracting 3 character of trees
mesh_root_rdd = mesh_RDD_splited.map(lambda l: (",".join(l[0:-1]), l[-1][0:3]))
mesh_root_rdd.collect()
```

▸ (1) Spark Jobs

```
Out[16]: [('Calcimycin', 'D03'),
 ('A-23187', 'D03'),
 ('Temefos', 'D02'),
 ('Temefos', 'D02'),
 ('Temefos', 'D02'),
 ('Abate', 'D02'),
 ('Abate', 'D02'),
 ('Abate', 'D02'),
 ('Difos', 'D02'),
 ('Difos', 'D02'),
 ('Difos', 'D02'),
 ('Abattoirs', 'J01'),
 ('Abattoirs', 'J03'),
 ('Abbreviations as Topic', 'L01'),
 ('Acronyms as Topic', 'L01'),
 ('Abdomen', 'A01'),
 ('"Abdomen, Acute"', 'C23'),
 ('"Abdomen, Acute"', 'C23'),
 ('Abdominal Injuries', 'C26'),
 ('Abdominal Neoplasms', 'C04'),
 ('Abdominal Muscles', 'A02'),
```

Command took 1.09 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

4-2.Create RDD from Mesh.csv with dataframe

Cmd 25

```
#Creat dataframe from mesh.csv
from pyspark.sql.functions import *
meshDF_root = create_DataFrame_From_CSV ("/FileStore/tables/mesh.csv",",")
display(meshDF_root)
```

▸ (3) Spark Jobs

	term	tree
1	Calcimycin	D03.633.100.221.173
2	A-23187	D03.633.100.221.173
3	Temefos	D02.705.400.625.800
4	Temefos	D02.705.539.345.800
5	Temefos	D02.886.300.692.800
6	Abate	D02.705.400.625.800
7	Abate	D02.705.539.345.800

Truncated results, showing first 1000 rows.
Click to re-execute with maximum result limits.

Command took 1.76 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
#converting Dataframe to RDD
#Extracting root from first item (the three first letters of tree)
mesh_root_rdd=meshDF_root.rdd.map(lambda l : (l[0],l[1][0:3]))
mesh_root_rdd.collect()
```

▸ (1) Spark Jobs

```
Out[18]: [('Calcimycin', 'D03'),
 ('A-23187', 'D03'),
 ('Temefos', 'D02'),
 ('Temefos', 'D02'),
 ('Temefos', 'D02'),
 ('Abate', 'D02'),
 ('Abate', 'D02'),
 ('Abate', 'D02'),
 ('Difos', 'D02'),
 ('Difos', 'D02'),
 ('Difos', 'D02'),
 ('Abattoirs', 'J01'),
 ('Abattoirs', 'J03'),
 ('Abbreviations as Topic', 'L01'),
 ('Acronyms as Topic', 'L01'),
 ('Abdomen', 'A01'),
 ('Abdomen, Acute', 'C23'),
 ('Abdomen, Acute', 'C23'),
 ('Abdominal Injuries', 'C26'),
 ('Abdominal Neoplasms', 'C04'),
 ('Abdominal Muscles', 'A02'),
```

Command took 2.28 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
# Joining Splited_Conditions_RDD(that is a paired Rdd which contains seperated conditions in each elements)
#with mesh_RDD_splited_final (that is a pair rdd that the key is term and the value is root)
Join_Condition_mesh = Splited_Conditions_RDD.join(mesh_root_rdd)
Join_Condition_mesh.collect()
```

▸ (1) Spark Jobs

```
Out[20]: [('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
 ('Autistic Disorder', (1, 'F03')),
```

Command took 8.62 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
#Extractung the value of join_condition_mesh_rdd
#reversing the keys and values

Join_Condition_mesh_frequency = Join_Condition_mesh.map(lambda l : l[1]) \
.map(lambda l : (l[1],l[0]))

#use Frequency_Of_pair_RDD to count the frequency of each root
#Print the 10 the most frequent roots
print_Pair_RDD(Frequency_Of_pair_RDD (Join_Condition_mesh_frequency),10)
```

▸ (3) Spark Jobs

```
C04   143994
C23   136079
C01   106674
C14   94523
C10   92310
C06   85646
C08   70720
C13   42599
C18   41276
C12   40161
```

Command took 3.00 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyC

3.4.5 Result on the submission (final) data

Dataframe Implementation

root	count
C04	143994
C23	136079
C01	106674
C14	94523
C10	92310
C06	85646
C08	70720
C13	42599
C18	41276
C12	40161

HiveQL Implementation

root	frequency
C04	143994
C23	136079
C01	106674
C14	94523
C10	92310
C06	85646
C08	70720
C13	42599
C18	41276
C12	40161

RDD Implementation

```
C04    143994
C23    136079
C01    106674
C14    94523
C10    92310
C06    85646
C08    70720
C13    42599
C18    41276
C12    40161
```

3.4.6 Discussion of result

In this study, the first most popular root is C04 which is the root of 143994 conditions. C23 is the second frequent root with 136079 conditions. C01 is ranked the third frequent root. C14 and C10 is placed in fourth and fifth frequent roots of conditions with 94523 and 92310 conditions respectively.

Furthermore, C04 which is the most frequent root, is the root of three frequent conditions (Breast Neoplasms, Carcinoma and Neoplasms). C23 is the root of Syndrome.

```
--Discission of result(Question 4)
--The root of the 5 most frequent conditions
select  distinct conditions,root from (select Conditions from VW_exploded_conditions
where Conditions in ("Carcinoma","Diabetes Mellitus","Neoplasms","Breast Neoplasms","Syndrome")--the 5 most frequent conditions
)c
left join VW_mesh_root m--root of conditions
on c.Conditions=m.term
order by conditions
```

▸ (2) Spark Jobs

	conditions	root
1	Breast Neoplasms	C04
2	Breast Neoplasms	C17
3	Carcinoma	C04
4	Diabetes Mellitus	C18
5	Diabetes Mellitus	C19
6	Neoplasms	C04
7	Syndrome	C23

Showing all 7 rows.

Command took 2.52 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 3:31:11 PM on MyCluster

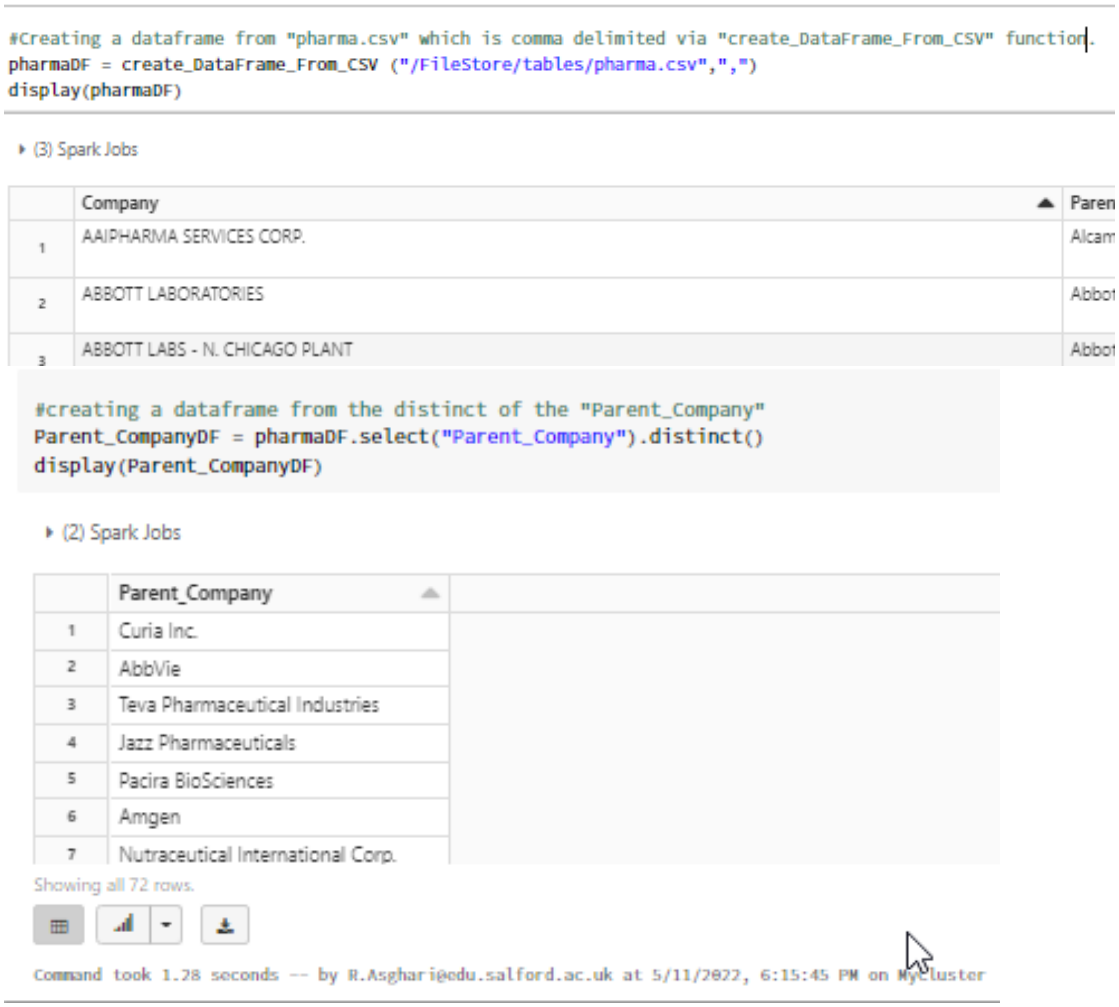
3.5 Question 5

3.5.1 Assumptions made

- The second column of clinicaltrial dataset is Sponsor.
- The location of pharma.csv is "/FileStore/tables/pharma.csv" which is a comma-delimited file.
- The second column of pharma.csv is Parent Company shows the pharmaceutical companies.[2]
- We need to extract a unique list of Parent Company from pharma.csv.

3.5.2 Dataframe implementation outline

A dataframe has been created from pharma.csv. Another dataframe that consist of a distinct list of Parent Company has been created. By left joining between clinicaltrial dataframe and distinct parent company, we can get the sponsors that their Parent_Company is null which means that they are not a pharmaceutical company. using Frequencies_Of_Column can count the frequency of each non-pharmaceutical company.



```
#Creating a dataframe from "pharma.csv" which is comma delimited via "create_DataFrame_From_CSV" function.
pharmaDF = create_DataFrame_From_CSV ("/FileStore/tables/pharma.csv",",")
display(pharmaDF)
```

▸ (3) Spark Jobs

	Company	Paren
1	AAIPHARMA SERVICES CORP.	Alcam
2	ABBOTT LABORATORIES	Abbot
3	ABBOTT LABS - N. CHICAGO PLANT	Abbot

```
#creating a dataframe from the distinct of the "Parent_Company"
Parent_CompanyDF = pharmaDF.select("Parent_Company").distinct()
display(Parent_CompanyDF)
```

▸ (2) Spark Jobs

	Parent_Company
1	Curia Inc.
2	AbbVie
3	Teva Pharmaceutical Industries
4	Jazz Pharmaceuticals
5	Pacira BioSciences
6	Amgen
7	Nutraceutical International Corp.

Showing all 72 rows.

Command took 1.28 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

```
 #Some sponsores are not pharmaceutical companies. So we have to use Left join.
clinical_farma_joinDF = clinicalDF.join(Parent_CompanyDF, clinicalDF.Sponsor==pharmaDF.Parent_Company, "left")\
.select("id","Sponsor","Parent_Company")

#The count of new dataframe have to be the same as original dataframe(clinicalDF) because we use left join.
clinical_farma_joinDF.count()

#When a Parent_companies is null, the sponsor is not a pharmaceutical companies.
No_Farma_sponsorDF = clinical_farma_joinDF\
.filter(clinical_farma_joinDF.Parent_Company.isNull())
No_Farma_sponsorDF.count()

#Calculating the frequency of each sponsors
No_Farma_sponsorDF_Frequencies = Frequencies_Of_Column (No_Farma_sponsorDF,"Sponsor")\
.withColumnRenamed("count","cnt")\
.limit(10)

#Printing the name of the sponsor along with its frequency
for row in No_Farma_sponsorDF_Frequencies.collect():
   print(row.Sponsor + ":" +  str(row.cnt))
```

▸ (9) Spark Jobs

```
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 11.30 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 8:31:01 PM on MyCluster

3.5.3 HiveQL implementation outline

Using createTable_From_CSV a table has been created from pharma.csv named pharma. At first, a list of distinct parent_company has been computed from pharma table. Clinicaltrial table consists all the sponsores has been joined with the distinct list of parent_company. Finally, the frequency of each sponsor has been computed and sorted.

```
--Frequency of 10 most common sponsors
select cp.Sponsor,count(*) number_of_clinical_trials
from
(
  select c.id,c.Sponsor,p.parent_company
  from clinicaltrial c
  left join
  (select distinct(parent_company) parent_company from pharma  ) p--distinct of parent_company
  on c.Sponsor=p.Parent_Company
where parent_company is null--sponsors that are not pharmaceutical companies
) cp
group by cp.Sponsor
order by number_of_clinical_trials desc
limit(10);
```

▸ (3) Spark Jobs

	Sponsor	number_of_clinical_trials
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

Showing all 10 rows.

Command took 4.92 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

3.5.4 RDD implementation outline

Due to the commas inside the comma-separated pharma.csv, the file has been converted to a dataframe and then convered to a rdd. A pair-rdd has been created from Parent_Company and another pair-rdd has been created from the sponsors. By left joining both rdds, the sponsors without parent company have been selected. Finally, the frequency of those companies has been computed.

```
#Creating a dataframe from pharma.csv
pharma_DF = create_DataFrame_From_CSV("/FileStore/tables/pharma.csv",",")
display(pharma_DF)
```

```
#converting dataframe to RDD
pharma_DF_rdd = pharma_DF.rdd
pharma_DF_rdd.take(10)
```

```
#convert dataframe to RDD
#pharma_DF_rdd = pharma_DF.select("Parent_Company").distinct().rdd
#Create pair RDD to use in join
Parent_Company_rdd = pharma_DF_rdd.map(lambda l : (l[1],1))

Parent_Company_rdd.collect()
```

▸ (1) Spark Jobs

```
Out[24]: [('Abbott Laboratories', 1),
 ('AbbVie', 1),
 ('AbbVie', 1),
 ('Abbott Laboratories', 1),
 ('Johnson & Johnson', 1),
 ('Abbott Laboratories', 1),
 ('Abbott Laboratories', 1),
 ('Johnson & Johnson', 1),
 ('Johnson & Johnson', 1),
 ('Abbott Laboratories', 1),
 ('AbbVie', 1),
```

```
#creating Pair RDD from Sponsores of clinical_RDD
Sponsore_RDD = clinical_RDD.map(lambda l : (l[1],1))
Sponsore_RDD.collect()
```

▸ (1) Spark Jobs

```
Out[26]: [('The University of Hong Kong', 1),
 ('Duke University', 1),
 ('Universidade Federal do Rio de Janeiro', 1),
 ('Istanbul Medeniyet University', 1),
 ('University of Roma La Sapienza', 1),
 ('Consorzio Futuro in Ricerca', 1),
 ('Ankara University', 1),
 ('Ruijin Hospital', 1),
 ('Washington University School of Medicine', 1),
 ('Orphazyme', 1),
 ('Novo Nordisk A/S', 1),
 ('Daniel Alexandre Bottino', 1),
 ('Bulent Ecevit University', 1),
 ('Institut fÜr Pharmakologie und Präventive Medizin', 1),
 ('The Third Xiangya Hospital of Central South University', 1),
 ('Tel Aviv Medical Center', 1),
 ('Medicines for Malaria Venture', 1),
 ('James Cook University, Queensland, Australia', 1),
 ('Soonchunhyang University Hospital', 1),
 ('Member Companies of the Opioid PMR Consortium', 1),
 ('Marmara University', 1),
```

Command took 2.60 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

Cmd 36

```
#using leftOuterJoin to join Sponsore_RDD and  Parent_Company_rdd.
#Usig left join to have all the sponsores because some of them are not pharmaceutical companies.
Join_Sponsore_Parent_Company_rdd = Sponsore_RDD.leftOuterJoin(Parent_Company_rdd)
Join_Sponsore_Parent_Company_rdd.collect()
```

▸ (1) Spark Jobs

```
Out[27]: [('Duke University', (1, None)),
 ('Duke University', (1, None)),
 ('Duke University', (1, None)),
 ('Duke University', (1, None)),
 ('Duke University', (1, None)),
 ('Duke University', (1, None)),
```

```
#Filtering sponsores that are not pharmaceutical companies (None)
#Creating pair rdd from Sponsore,(keys=Sponsores  and values=one)

Sponsore_Frequency_rdd= Join_Sponsore_Parent_Company_rdd.filter(lambda l : l[1][1]==None )\
.map(lambda l : (l[0], 1))\

#using Frequency_Of_pair_RDD to count the frequency of each sponsore and sort by the frequencies in descending order
#Using function to print pair rdd
print_Pair_RDD(Frequency_Of_pair_RDD (Sponsore_Frequency_rdd),10)
```

▸ (3) Spark Jobs

```
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 2.08 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

3.5.5 Result on the submission (final) data

Dataframe Implementation

```
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
```

HiveQL Implementation

Sponsor	number_of_clinical_trials
National Cancer Institute (NCI)	3218
M.D. Anderson Cancer Center	2414
Assistance Publique - HÃ´pitaux 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

RDD Implementation

```
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
```

3.5.6 Discussion of result

A company named “National Cancer Institute (NCI)” is a non-pharmaceutical company that have supported 3218 clinical trial studies. M.D. Anderson Cancer Center, Assistance Publique - Hôpitaux de Paris, Mayo Clinic, Merck Sharp & Dohme Corp, Assiut University and Novartis Pharmaceuticals are the companies that have supported almost between 2000 and 2500 studies. Massachusetts General Hospital,

Cairo University and Hoffmann-La Roche are non-pharmaceutical company that have supported less than 2000 studies.

Furthermore, pharmaceutical companies have funded more studies than non-pharmaceutical companies. For this analysis, two views are created and then the frequency of studies based on them is computed.

```
--Create a view for companies are not pharmaceutical
Create or replace temp view VW_Non_pharmaceutical_companies as
select * from clinicaltrial c
left join pharma p
on c.Sponsor=p.Parent_Company
where p.Parent_Company is null;


--Create a view for companies are pharmaceutical
Create or replace temp view VW_pharmaceutical_companies as
select * from clinicaltrial c
left join pharma p
on c.Sponsor=p.Parent_Company
where p.Parent_Company is not null;
```

OK

Command took 0.78 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 2:02:50 PM on MyCluster

```
--The frequency of studies supported by pharmaceutical and non-pharmaceutical companies
select 'pharmaceutical_companies' Company_Type,count(distinct id) cnt from VW_pharmaceutical_companies
union all
select 'pharmaceutical_companies'Company_Type,count(distinct id) cnt from VW_Non_pharmaceutical_companies
```

▸ (5) Spark Jobs

	Company_Type	cnt
1	pharmaceutical_companies	16260
2	pharmaceutical_companies	371001

Showing all 2 rows.

Command took 5.37 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 4:00:21 PM on MyCluster

3.6 Question 6

3.6.1 Assumptions made

- Selecting studies that their status is completed.
- The Month and year of completion date can be extracted from Completion column in clinicaltrial dataset.
- Filtering the studies that completed in the year 2021.
- To visualize the result bokeh and matplotlib and built-in charts have been used.

3.6.2 Dataframe implementation outline

The studies with completed status have been filtered. Then month and year have been extracted from the completion column. The studies that have been completed in 2021 have been found. Finally, the number of completed studies in 2021 by month has been computed.

```
#Filtering completed studies
completedDF = clinicalDF.where(clinicalDF.Status=="Completed" )

#Extracting Month and Year from Completion Date.
completedDF_Date = completedDF.select("Sponsor","Status","Completion",substring("Completion",0,3)\
                                      .alias("Month"),substring("Completion",4,7).alias("Year"))

#selecting the completed studies of the given year
completedDF_year = completedDF_Date.where(completedDF_Date.Year == int(Year))

#Frequency of each Month
completedDF_year_Frequency=Frequencies_Of_Column(completedDF_year,"Month")

display(completedDF_year_Frequency)
```

▸ (2) Spark Jobs

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

Showing all 10 rows.

Command took 3.69 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 8:44:13 PM on MyCluster

```
#Calculating the frequency of completion studies each month
#using "when" to assign an id to each month and then sort the result by id , then just show Month and year column.
completedDF_Fre_Month = completedDF_year_Frequency.select(col("*"),\
when(col("Month") == "Jan", 1) \
.when(col("Month") == "Feb", 2) \
.when(col("Month") == "Mar", 3) \
.when(col("Month") == "Apr", 4) \
.when(col("Month") == "May", 5) \
.when(col("Month") == "Jun", 6) \
.when(col("Month") == "Jul", 7) \
.when(col("Month") == "Aug", 8) \
.when(col("Month") == "Sep", 9) \
.when(col("Month") == "Oct", 10) \
.when(col("Month") == "Nov", 11) \
.when(col("Month") == "Dec", 12) \
.otherwise("unkwon") \
.alias("Id")).\
sort(col("Id").cast("int"))\
.select("Month","count")

#Does not Show Null values of November and December
display(completedDF_Fre_Month)
```

▸ (2) Spark Jobs

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

Showing all 10 rows.

Command took 3.93 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

The results of the previous implementation are not sorted by month and There is no result for November and December. To sort the result by Month and show November and December in results, a dataframe with the id of months has been created named Month_Desc_DF. By Joining this dataframe with the dataframe consisting of the frequency of the month, we can sort the results by the id of each month.

```
##To Show Null values of November and December, create an RDD to Sort data by the number of months
Month_Desc=[("Jan",1),
            ("Feb",2),
            ("Mar",3),
            ("Apr",4),
            ("May",5),
            ("Jun",6),
            ("Jul",7),
            ("Aug",8),
            ("Sep",9),
            ("Oct",10),
            ("Nov",11),
            ("Dec",12)]
Month_Desc_RDD= sc.parallelize(Month_Desc)

Month_Desc_DF=Month_Desc_RDD.toDF()
Month_Desc_DF=Month_Desc_DF.withColumnRenamed("_1","Month_")
Month_Desc_DF=Month_Desc_DF.withColumnRenamed("_2","Id")
display(Month_Desc_DF)
```

▸ (4) Spark Jobs

	Month_	Id
1	Jan	1
2	Feb	2
3	Mar	3
4	Apr	4
5	May	5
6	Jun	6
7	Jul	7

Showing all 12 rows.

Command took 1.77 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

```
#Using left Join  between completedDF_year_Frequency to have all 12 months
completedDF_Fre_Month=c.join(completedDF_year_Frequency,completedDF_year_Frequency.Month==Month_Desc_DF.Month_,"left")
completedDF_Fre_Month=completedDF_Fre_Month.select(completedDF_Fre_Month.Month_.alias("Month"),"count")

display(completedDF_Fre_Month)
```

▸ (4) Spark Jobs

	Month	count
6	Jun	1094
7	Jul	819
8	Aug	700
9	Sep	528
10	Oct	187
11	Nov	null
12	Dec	null

Showing all 12 rows.

Command took 4.07 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

3.6.2.1 Visualization By Bokeh

To visualize result in bokeh the dataframe converted to an RDD.

```
#Converting dataframe to RDD
RDD = completedDF_Fre_Month.rdd
RDD.take(10)

#Extracting Months from RDD
MonthRDD= RDD.map(lambda x:x[0])
MonthRDD.collect()
```

▸ (5) Spark Jobs

```
Out[21]: ['Jan',
 'Feb',
 'Mar',
 'Apr',
 'May',
 'Jun',
 'Jul',
 'Aug',
 'Sep',
 'Oct',
 'Nov',
 'Dec']
```

Command took 5.02 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

Cmd 33

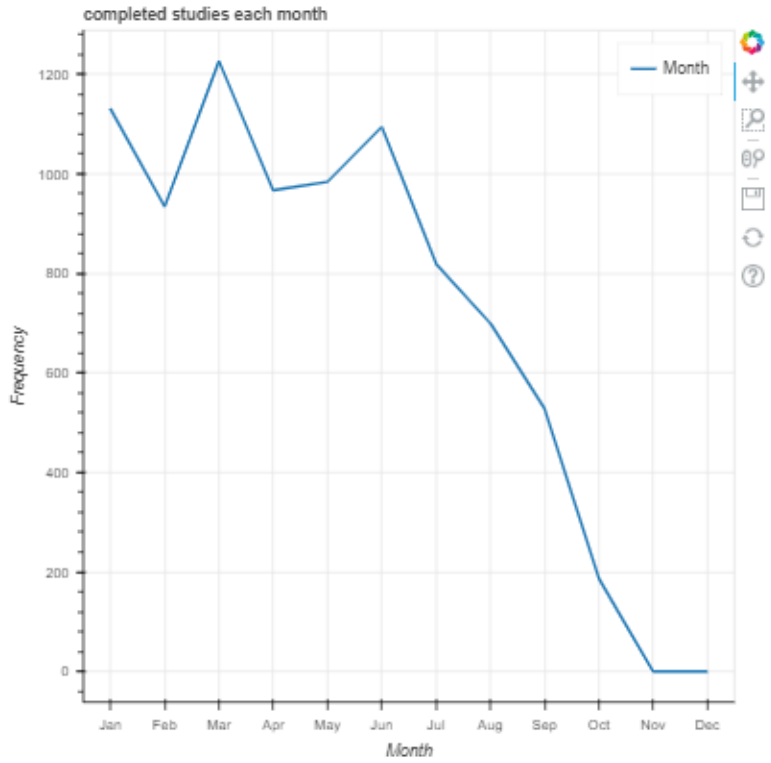
```
#Extracting frequency of each month
frequencyRDD= RDD.map(lambda x:x[1])
frequencyRDD.collect()
```

▸ (1) Spark Jobs

```
Out[22]: [1131, 934, 1227, 967, 984, 1094, 819, 700, 528, 187, None, None]
```

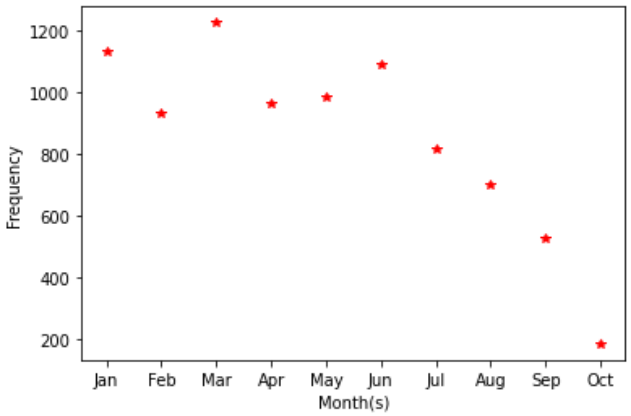
Command took 0.38 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:45 PM on MyCluster

```
from bokeh.plotting import figure
from bokeh.embed import components, file_html
from bokeh.resources import CDN

# prepare data
Month = MonthRDD.collect()
Frequency = frequencyRDD.collect()

# creating a new plot with a title and axis labels
p = figure(title="completed studies each month", x_axis_label='Month', y_axis_label='Frequency',x_range=MonthRDD.collect() )

# adding a line renderer with legend and line thickness
p.line(Month,Frequency, legend_label="Month", line_width=2)

# creating an html document that embeds the Bokeh plot
html = file_html(p, CDN, "my plot")

# display this html
displayHTML(html)
```



3.6.2.2 *Visualization by matplotlib*

```
import matplotlib.pyplot as plt
# prepare data
Month = MonthRDD.collect()
Frequency = frequencyRDD.collect()
#plt.bar(Month,Frequency)
plt.plot(Month,Frequency,'r*')
plt.xlabel ("Month(s)")
plt.ylabel ("Frequency")
plt.savefig ('MyPlot.pdf')
```



3.6.3 HiveQL implementation outline

The studies with completed status have been filtered. In addition, using right (Completion,4), the Year extracted from Completion date column. The records which completed in the year 2021 have been filtered. Using left (Completion,3), the Month of completion date has been computed for each record. Finally, the frequency of completed studies in each month in the year 2021 have been computed and sorted in a descending order. To use the Year variable which declared at the beginning of the notebook, ${hiveconf:Year} has been used.

```
--Creating a view to Sort the number of completed studies in each month of a given year
CREATE OR REPLACE TEMP VIEW VW_Month_Frequency AS
select Month_Id id,
Month,
count(*) frequency
from
(select Sponsor,
Status,
Completion,
case
when left(Completion,3) = 'Jan' then 1
when left(Completion,3) = 'Feb' then 2
when left(Completion,3) = 'Mar' then 3
when left(Completion,3) = 'Apr' then 4
when left(Completion,3) = 'May' then 5
when left(Completion,3) = 'Jun' then 6
when left(Completion,3) = 'Jul' then 7
when left(Completion,3) = 'Aug' then 8
when left(Completion,3) = 'Sep' then 9
when left(Completion,3) = 'Oct' then 10
when left(Completion,3) = 'Nov' then 11
when left(Completion,3) = 'Dec' then 12
else 'Unkhown'
end Month_Id,--to sort data by the id of each Month

left(Completion,3) Month,--extract month from Completion date
right(Completion,4)Year--extract year from Completion date
from clinicaltrial c
where c.Status ='Completed'--Completed studies
and right(Completion,4)=${hiveconf:Year}-- the given year

)k
group by Month_Id,k.Month
order by cast(Month_Id as int)
```

```
OK
```

```
--The number of completed studies  in each month of a given year in a table format
select Month, frequency  from VW_Month_Frequency
```

```
--In the previous implementation, the November and december are not shown in the result
--because there are no completed study in those month.
--To see this month with zero completed study, the VW_Month has been created.
CREATE OR REPLACE TEMP VIEW VW_Month
AS
select 1 id,'Jan' Month
union all
select 2 id,'Feb'Month
union all
select 3 id,'Mar'Month
union all
select 4 id,'Apr'Month
union all
select 5 id,'May'Month
union all
select 6 id,'Jun'Month
union all
select 7 id,'Jul'Month
union all
select 8 id,'Aug'Month
union all
select 9 id,'Sep'Month
union all
select 10 id,'Oct'Month
union all
select 11 id,'Nov'Month
union all
select 12 id,'Dec'Month
```

OK

Command took 0.16 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

```
--Creating a view to show the frequency of the completed studies in each Monhs Sorted ny Month
CREATE OR REPLACE TEMP VIEW VW_Month_Frequency AS
select Month_Id, Month, count(id)frequency from (select m.id Month_Id,m.Month , k.id from VW_Month m
left join
(select id,
Sponsor,
Status,
Completion,
left(Completion,3) Month,--extract month from Completion date
right(Completion,4)Year--extract year from Completion date
from clinicaltrial c
where c.Status ='Completed'--Completed studies
and right(Completion,4)=${hiveconf:Year}-- the given year
)k
on k.Month= m.Month
)k
group by Month_Id, Month
order by Month_Id
```

OK

Command took 0.49 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

```
select Month, frequency from VW_Month_Frequency
```

▸ (2) Spark Jobs

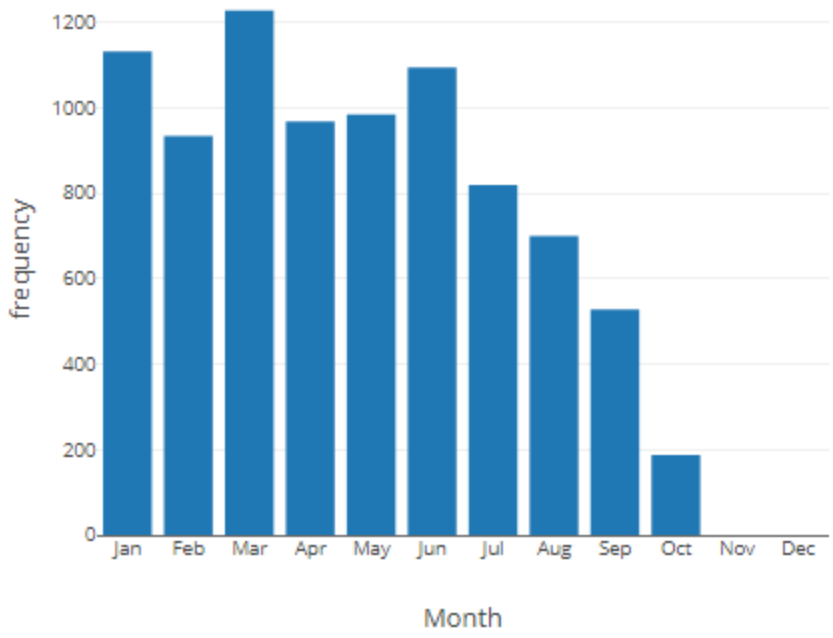
	Month	frequency
1	Jan	1131
2	Feb	934
3	Mar	1227
4	Apr	967
5	May	984
6	Jun	1094
7	Jul	819

Showing all 12 rows.

Command took 4.77 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:23:14 PM on MyCluster

```
select Month, frequency from VW_Month_Frequency
```

▸ (2) Spark Jobs



3.6.4 RDD implementation outline

In the clinical_RDD the completed study (l[2]=="Completed") has been filtred. The empty completion date has been removed. Completion month and year has been extracted. The completed studies in 2021

selected. The frequency of completed studied in each month has been computed.

```
#filtering status=Completed (l[2]=="Completed")
#Removing empty completion date and Extract completion date (l[4]) and
#Splitting Completion date by space to extract month and year seperately (l.split(" "))
#Filtering year equal to given year. given year is stored in a variable named "Year".
#Creating a pair rdd from month with value=1
#Using reduceByKey to have the frequency of each month
completed_RDD = clinical_RDD.filter(lambda l : l[2]=="Completed")\
.filter(lambda l : l[4]!= '')\
.map(lambda l : l[4].split(" "))\
.filter(lambda l : l[1]==Year)\
.map(lambda l: (l[0],1))\
.reduceByKey(lambda x,y : x+y)
# the result is not sorted by Month
completed_RDD.collect()
```

▸ (1) Spark Jobs

```
Out[30]: [('May', 984),
 ('Jan', 1131),
 ('Jun', 1094),
 ('Mar', 1227),
 ('Feb', 934),
 ('Aug', 700),
 ('Apr', 967),
 ('Jul', 819),
 ('Oct', 187),
 ('Sep', 528)]
```

Command took 2.42 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
#creating an RDD to Sort data by the id of months
Month_Desc=[("Jan",1),
            ("Feb",2),
            ("Mar",3),
            ("Apr",4),
            ("May",5),
            ("Jun",6),
            ("Jul",7),
            ("Aug",8),
            ("Sep",9),
            ("Oct",10),
            ("Nov",11),
            ("Dec",12)]
Month_Desc_RDD= sc.parallelize(Month_Desc)
Month_Desc_RDD.collect()
```

▸ (1) Spark Jobs

```
Out[31]: [('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 0.10 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM or

```
#Joining completed_RDD with Month_Desc_RDD
#Sorting data by the number of month (l[1][0])
#Extracting month and its frequency from sorted data l[0]=name of month and l[1][1] is its frequency
Month_frequency_RDD = Month_Desc_RDD.leftOuterJoin(completed_RDD)\
.sortBy(lambda l : l[1][0])\
.map(lambda l :(l[0],l[1][1]))

print_Pair_RDD(Month_frequency_RDD,12)
```

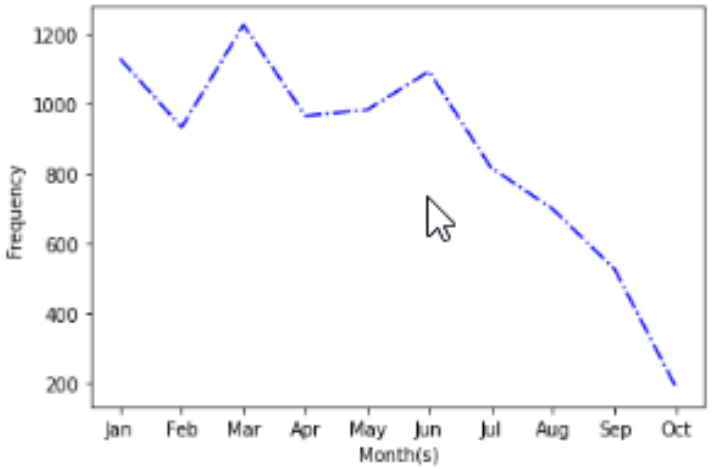
▸ (5) Spark Jobs

```
Jan   1131
Feb   934
Mar   1227
Apr   967
May   984
Jun   1094
Jul   819
Aug   700
Sep   528
Oct   187
Nov   None
Dec   None
```

Command took 1.67 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6:15:59 PM on MyCluster

```
import matplotlib.pyplot as plt
# preparng data
Month = Month_frequency_RDD.map(lambda l : l[0]).collect()
Frequency = Month_frequency_RDD.map(lambda l : l[1]).collect()
plt.plot(Month,Frequency,'b-.')
plt.xlabel ("Month(s)")
plt.ylabel ("Frequency")
plt.savefig ('MyPlot.pdf')
```

▸ (2) Spark Jobs



Command took 0.90 seconds -- by R.Asghari@edu.salford.ac.uk at 5/11/2022, 6

3.6.5 Result on the submission (final) data

Dataframe Implementation

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

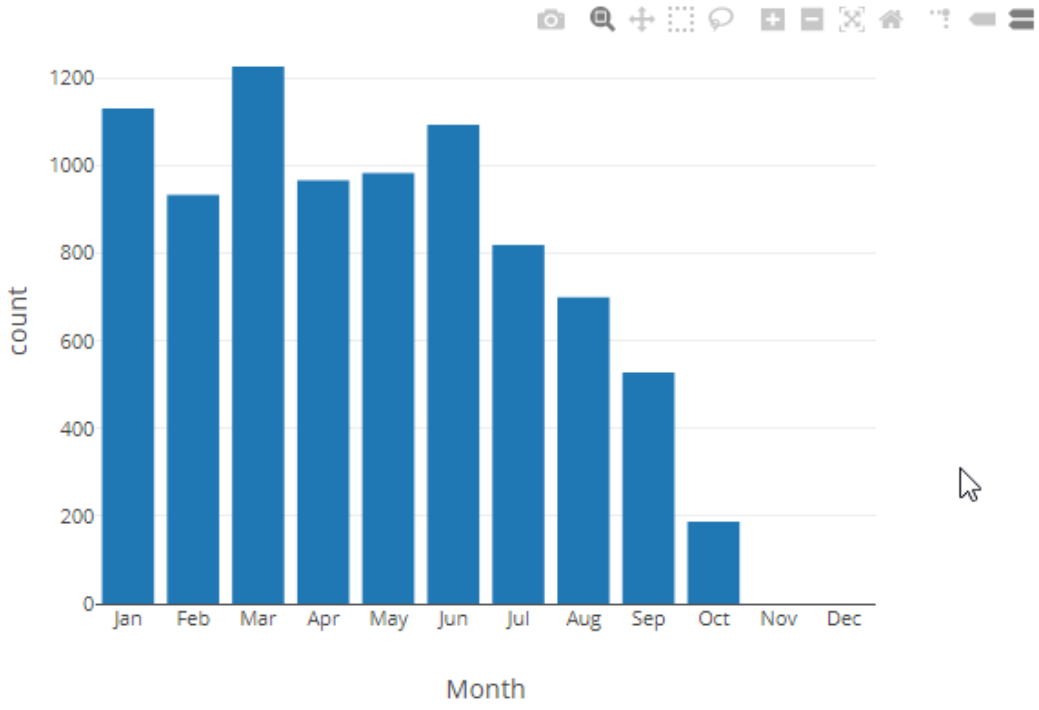
HivQL Implementation

Month	frequency
Jan	1131
Feb	934
Mar	1227
Apr	967
May	984
Jun	1094
Jul	819
Aug	700
Sep	528
Oct	187
Nov	0
Dec	0

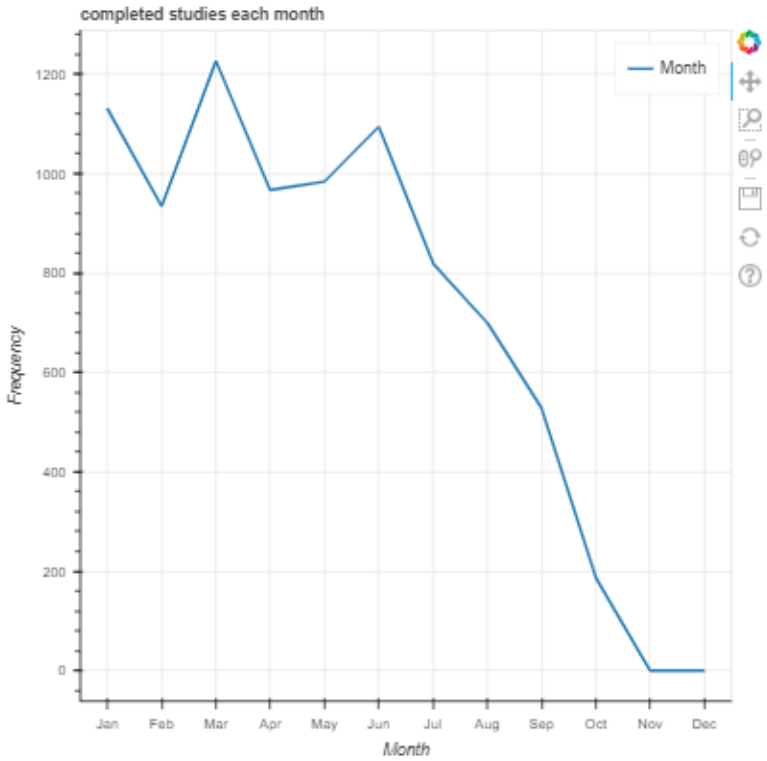
RDD Implementation

```
Jan	1131
Feb	934
Mar	1227
Apr	967
May	984
Jun	1094
Jul	819
Aug	700
Sep	528
Oct	187
Nov	None
Dec	None
```

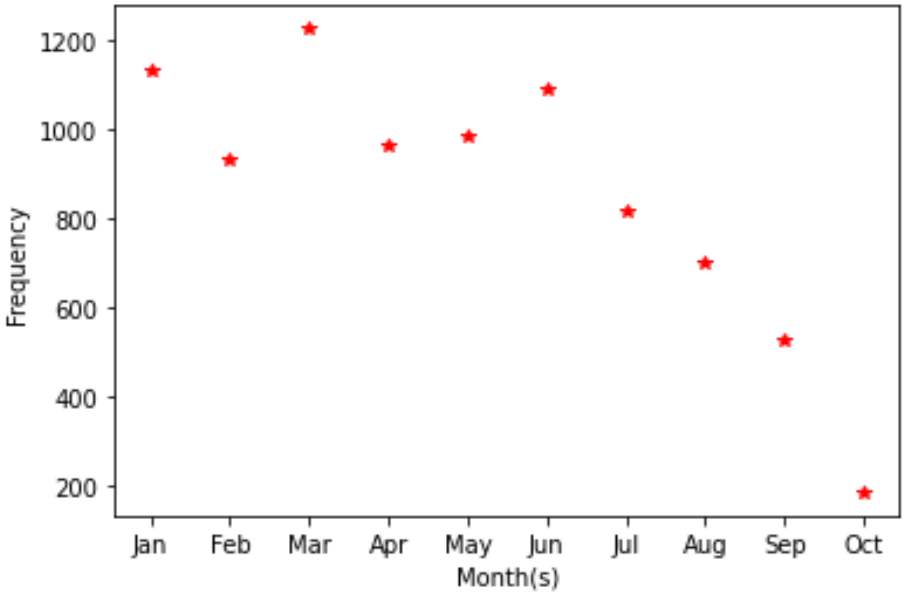
3.6.5.1 Visualization by Built-in Charts

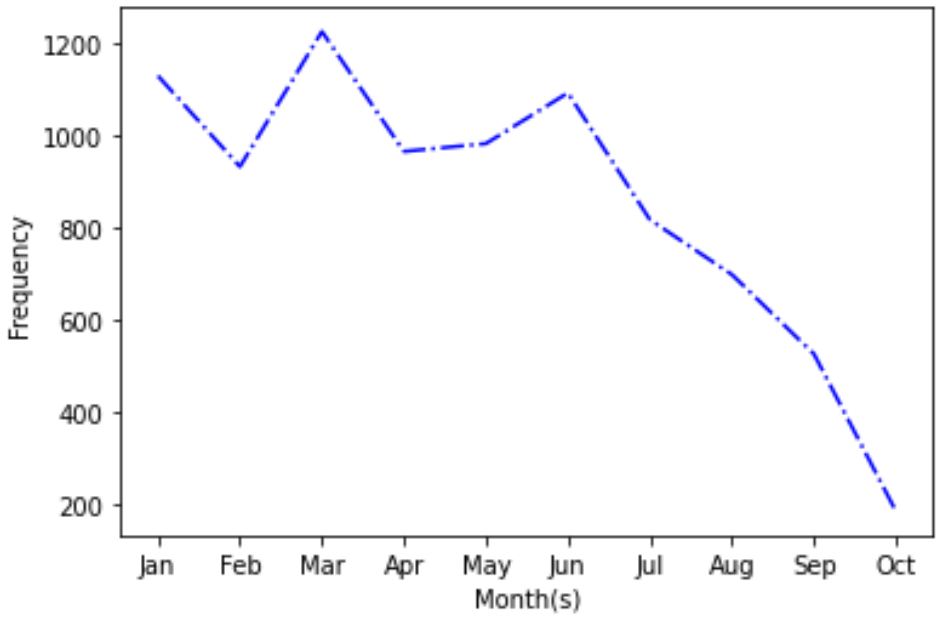


3.6.5.2 Visualization by Bokeh



3.6.5.3 Visualization by matplotlib





3.6.6 Discussion of result

1227 studies have been completed in March of 2021, which is the most throughout the entire year. With 1131 and 1094 completed studies, January and June rank second and third, respectively. Study completions decreased in the year 2021 after Jun.

In November and December, there are no completed studies, possibly due to null values in completion dates or maybe because no completed studies were available for download at the time the dataset was downloaded.

3.7 Further analysis 1: The 5 most frequent Interventions with their frequencies.

Interventions column is similar to Conditions to extract the frequency of each intervention a view has been created. Using the split and explode function, each intervention spreads in a different record. Then the frequency of them has been computed. As results show, the 5 most frequent intervention are Paclitaxel, Cyclophosphamide, Dexamethasone, Carboplatin and Antibodies.

As some studies include more than one condition and intervention , it is not possible to determine which intervention corresponds to which condition. If each clinical trial file contained a unique code based on the id, condition and intervention, the most frequent intervention for each condition could be extracted.

```
--Creating a view in which each record has an Interventions
--Split function creats a list of Interventions
--Explode function spreads list of Interventions into differrent rows(like flatMap in RDD)
CREATE OR REPLACE TEMP VIEW VW_exploded_Interventions_1 AS
SELECT Id,Sponsor,
Status,
Start,
Completion,
Type,
Submission,
Conditions,
explode(split(Interventions , ',')) as Interventions
FROM clinicaltrial
```

OK

Command took 0.47 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 7:22:02 PM on MyCluster

```
--The 5 most common Interventions
select Interventions,count(*) cnt from VW_exploded_Interventions_1
group by Interventions
order by cnt desc
limit(5)
```

▸ (2) Spark Jobs

	Interventions	cnt
1	Paclitaxel	3225
2	Cyclophosphamide	3012
3	Dexamethasone	2516
4	Carboplatin	2392
5	Antibodies	2335

Showing all 5 rows.

Command took 2.47 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 7:22:05 PM on MyCl

3.8 further analysis 2: The most frequent Interventions for the 5 most common conditions

Assuming all interventions correspond to all conditions in each record, this analysis has computed the five most common interventions for the five most common conditions. A temp view named VW_exploded_Interventions has been created based on VW_exploded_conditions (which has a condition in each record). In this view, each record has a condition and an intervention.

For each condition of the 5 most frequent conditions, a temp view has been created. The most frequently used intervention has been extracted for each condition. Using union all, the result has been generated. As results show, the most common intervention for Diabetes Mellitus is Insulin. This figure for Carcinoma is Carboplatin. Dexamethasone, Paclitaxel, and Fludarabine are the most commonly prescribed interventions for Neoplasms, Breast Neoplasms, and Syndrome, respectively.

```
--This view is based on the VW_exploded_conditions which has a condition in each record.
--Creating a view in which each record has an Interventions
--Split function creats a list of Interventions
--Explode function spreads list of Interventions into differrent rows(like flatMap in RDD)
CREATE OR REPLACE TEMP VIEW VW_exploded_Interventions AS
SELECT Id,Sponsor,
Status,
Start,
Completion,
Type,
Submission,
Conditions,
explode(split(Interventions , ',')) as Interventions
FROM VW_exploded_conditions
```

```
OK
```

```
Command took 0.40 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 7:27:44 PM on MyCluster
```

```
CREATE OR REPLACE TEMP VIEW VW_Carcinoma_Interventions  as
select "Carcinoma" conditions ,Interventions,count(*) cnt from VW_exploded_Interventions
where conditions ="Carcinoma"
group by Interventions
order by cnt desc
limit(1);
CREATE OR REPLACE TEMP VIEW VW_DiabetesMellitus_Interventions  as
select "Diabetes Mellitus" conditions ,Interventions,count(*) cnt from VW_exploded_Interventions
where conditions ="Diabetes Mellitus"
group by Interventions
order by cnt desc
limit(1);
CREATE OR REPLACE TEMP VIEW VW_BreastNeoplasms_Interventions  as
select "Breast Neoplasms" conditions ,Interventions,count(*) cnt from VW_exploded_Interventions
where conditions ="Breast Neoplasms"
group by Interventions
order by cnt desc
limit(1);
CREATE OR REPLACE TEMP VIEW VW_Syndrome_Interventions  as
select "Syndrome" conditions ,Interventions,count(*) cnt from VW_exploded_Interventions
where conditions ="Syndrome"
group by Interventions
order by cnt desc
limit(1);
CREATE OR REPLACE TEMP VIEW VW_Neoplasms_Interventions  as
select "Neoplasms" conditions ,Interventions,count(*) cnt from VW_exploded_Interventions
where conditions ="Neoplasms"
group by Interventions
order by cnt desc
limit(1);
```

```
OK
```

```
--The most common Interventions for the 5 most common conditions
select * from VW_Carcinoma_Interventions
union all
select * from VW_DiabetesMellitus_Interventions
union all
select * from VW_Neoplasms_Interventions
union all
select * from VW_BreastNeoplasms_Interventions
union all
select * from VW_Syndrome_Interventions
order by cnt desc
```

▸ (8) Spark Jobs

	conditions	Interventions	cnt
1	Diabetes Mellitus	Insulin	1104
2	Carcinoma	Carboplatin	1103
3	Neoplasms	Dexamethasone	726
4	Breast Neoplasms	Paclitaxel	717
5	Syndrome	Fludarabine	183

Showing all 5 rows.

Command took 13.04 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 10:59:39 AM

3.9 Further Analysis 3: The frequency of study types supported by pharmaceutical companies and non-pharmaceutical companies.

In order to analyze the frequency of study types between pharmaceutical and non-pharmaceutical companies, two views have been created. Based on the results, pharmaceutical companies have supported more studies in each study type. The interventional studies have the most frequent type of studies.

```
--Create a view for companies are not pharmaceutical
Create or replace temp view VW_Non_pharmaceutical_companies as
select * from clinicaltrial c
left join pharma p
on c.Sponsor=p.Parent_Company
where p.Parent_Company is null;


--Create a view for companies are pharmaceutical
Create or replace temp view VW_pharmaceutical_companies as
select * from clinicaltrial c
left join pharma p
on c.Sponsor=p.Parent_Company
where p.Parent_Company is not null;
```

OK

Command took 0.78 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 2:02:50 PM on MyCluster

```
--Computing the frequency of study types between pharmaceutical companies and non_pharmaceutical companies
select k.Type, k.pharmaceutical_companies_cnt, k2.Non_pharmaceutical_companies_cnt from
(select  Type,count(*) pharmaceutical_companies_cnt from VW_pharmaceutical_companies
group by Type)k--pharmaceutical companies
left join
(
select  Type,count(*) Non_pharmaceutical_companies_cnt from VW_Non_pharmaceutical_companies
group by Type)k2--non_pharmaceutical
on k.type=k2.type
order by pharmaceutical_companies_cnt desc
```

▸ (3) Spark Jobs

	Type	pharmaceutical_companies_cnt	Non_pharmaceutical_companies_cnt
1	Interventional	475725	287519
2	Observational	77513	75323
3	Observational [Patient Registry]	1989	8105
4	Expanded Access	613	54

Showing all 4 rows.

Command took 5.64 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 2:11:10 PM on MyCluster

3.10 Further Analysis 4: The frequency of Completed Studies without Completion date

Although some studies have been completed, they do not have any completion dates. Therefore, completion needs to be edited. The most frequent of such errors are the studies started in the year 2006.

```
select right(Start,4) Year, Count(*) cnt from clinicaltrial
where Completion is null---Completion date is null
and Status ="Completed"--Completed Studies
group by Year--start date
```

▸ (2) Spark Jobs

	Year	cnt
1	2006	532
2	2004	492
3	2005	487
4	2008	425
5	2003	410
6	2009	399
7	2007	386

Showing all 42 rows.

Command took 1.56 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022, 12:4

3.11 Further Analysis 5: The frequency of status in the given year

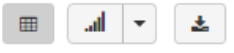
In this analysis, the frequency of each study status has been computed. The greatest number of studies are in the Recruiting status and Completed in the year 2021.

```
--The frequency of each study status
select Status,count(*)cnt from clinicaltrial
where right(Completion,4)=${hiveconf:Year}--the given year
group by Status
order by cnt desc
```

▸ (2) Spark Jobs

	Status	cnt
1	Recruiting	12902
2	Completed	8571
3	Active, not recruiting	6127
4	Not yet recruiting	2958
5	Unknown status	2289
6	Enrolling by invitation	918
7	Terminated	803

Showing all 9 rows.



Command took 2.19 seconds -- by R.Asghari@edu.salford.ac.uk at 5/12/2022,

Status	cnt
Recruiting	12902
Completed	8571
Active, not recruiting	6127
Not yet recruiting	2958
Unknown status	2289
Enrolling by invitation	918
Terminated	803
Withdrawn	753
Suspended	306

4 References

[1] "lab_databricks_shell.pdf".

[2] "assignment_brief.pdf".