Analysis of Clinical trial data

BY:
SRIRAM NARLA
STUDENT ID: @00646253
EMAIL ID: S.NARLA@EDU.SALFORD.AC.UK
MSC DATA SCIENCE

Introduction and Set up required.

In today's world, enormous amounts of data are generated on a daily basis in various forms such as text, speech, video, and photos. It is extremely difficult to analyse all of the data in a variety of formats. Various tools and techniques are employed to analyse this large data. As a part of this module, we have used Pyspark and HiveQl and Amazon Web Services for evaluating the big data.

In this assignment, for implementing the Pyspark and Hive Ql Databricks community edition has been used.

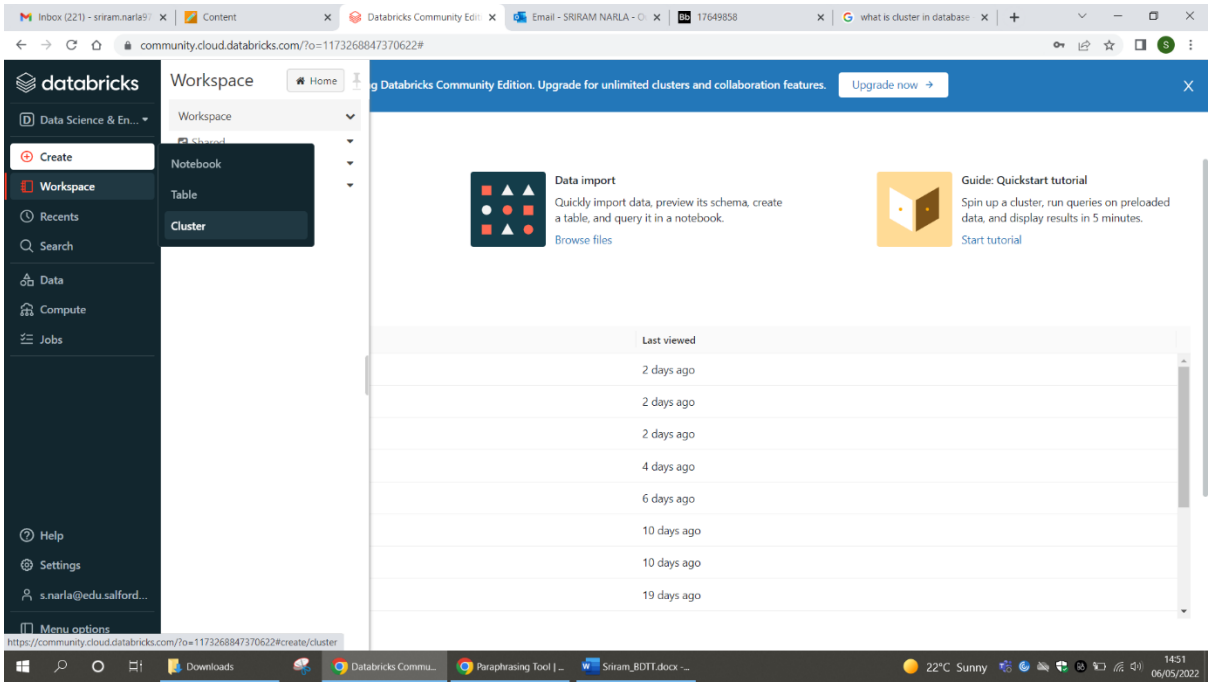
Link for Databricks community edition: [Login - Databricks Community Edition]

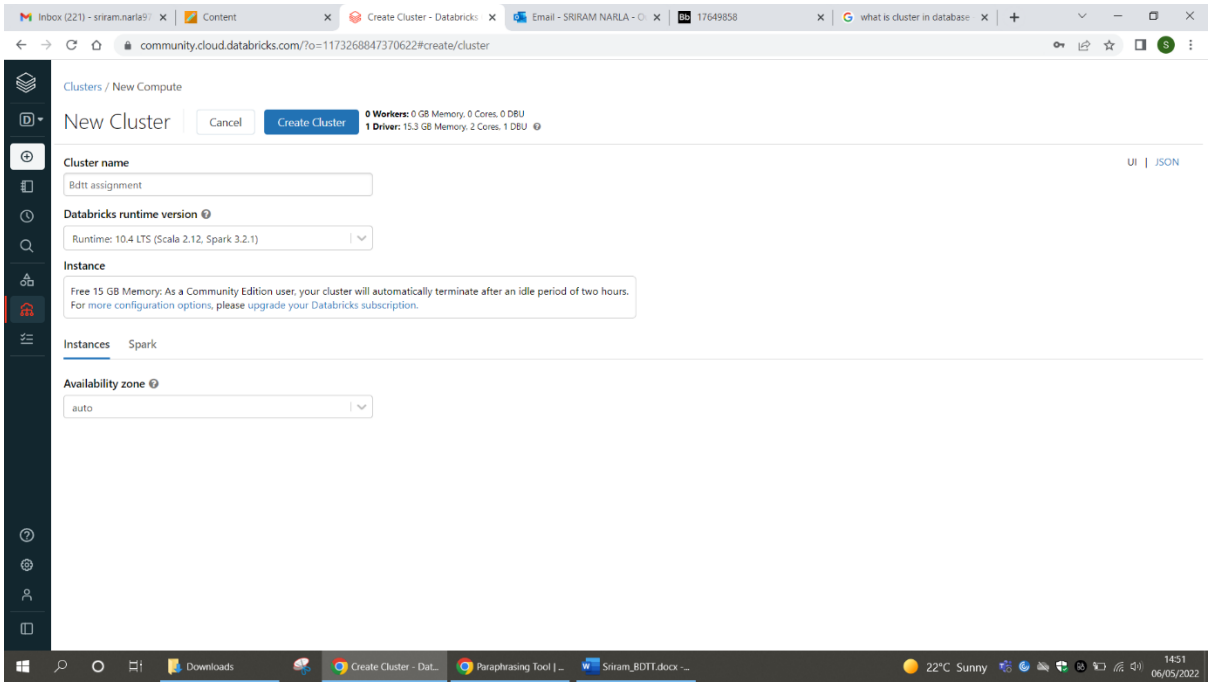
After logging into the Databricks, a cluster needs to be created to run the notebooks. As a part of this assignment, I have created three notebooks.

- Python Notebook for DataFrames Implementation.
- Python Notebook for RDD Implementation.
- SQL Notebook for HiveQl Implementation.

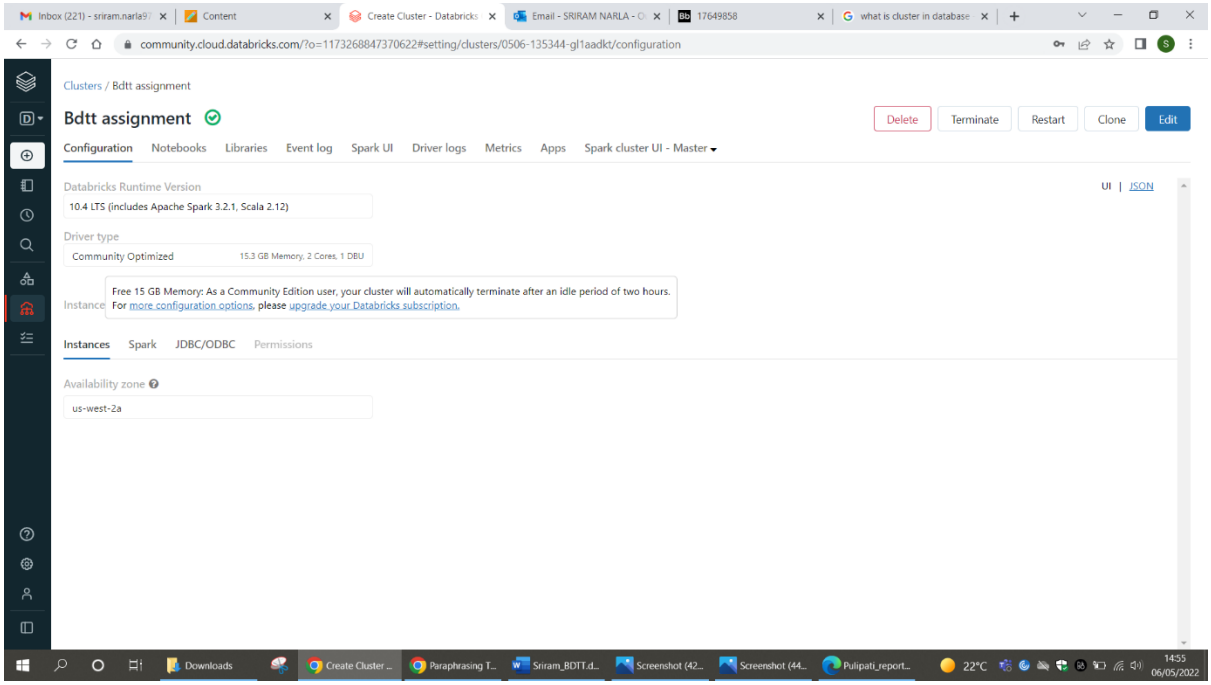
Creating a cluster:

On the left side, click create cluster and enter the name.

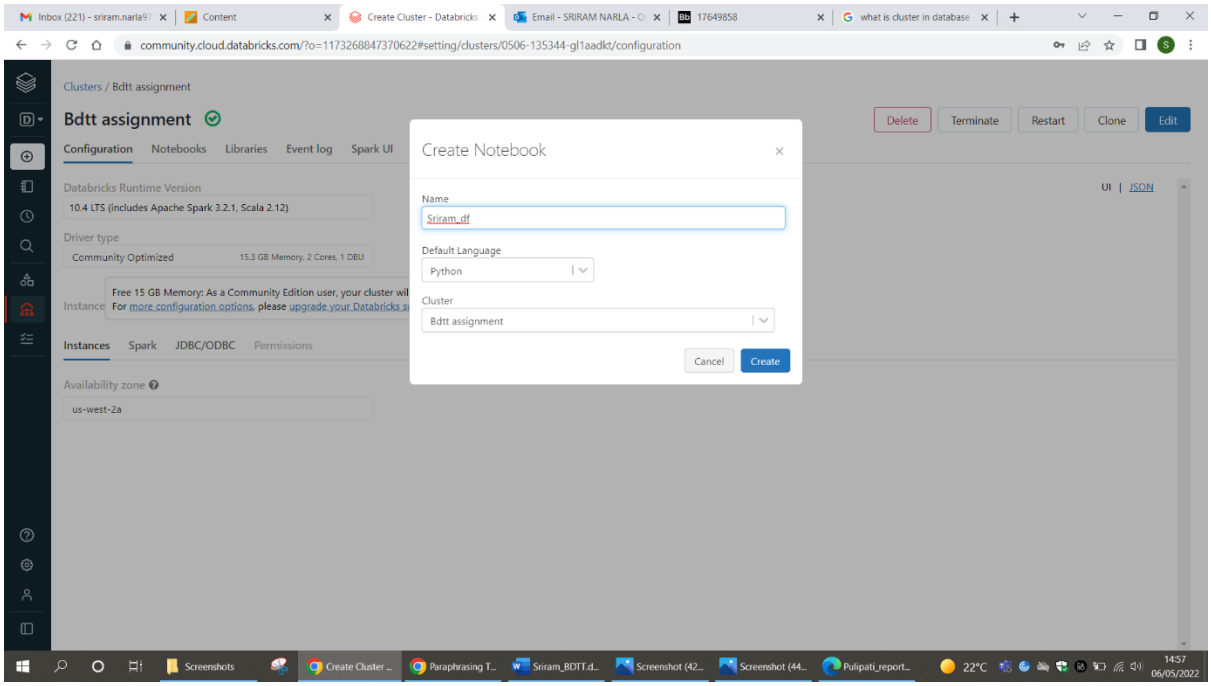




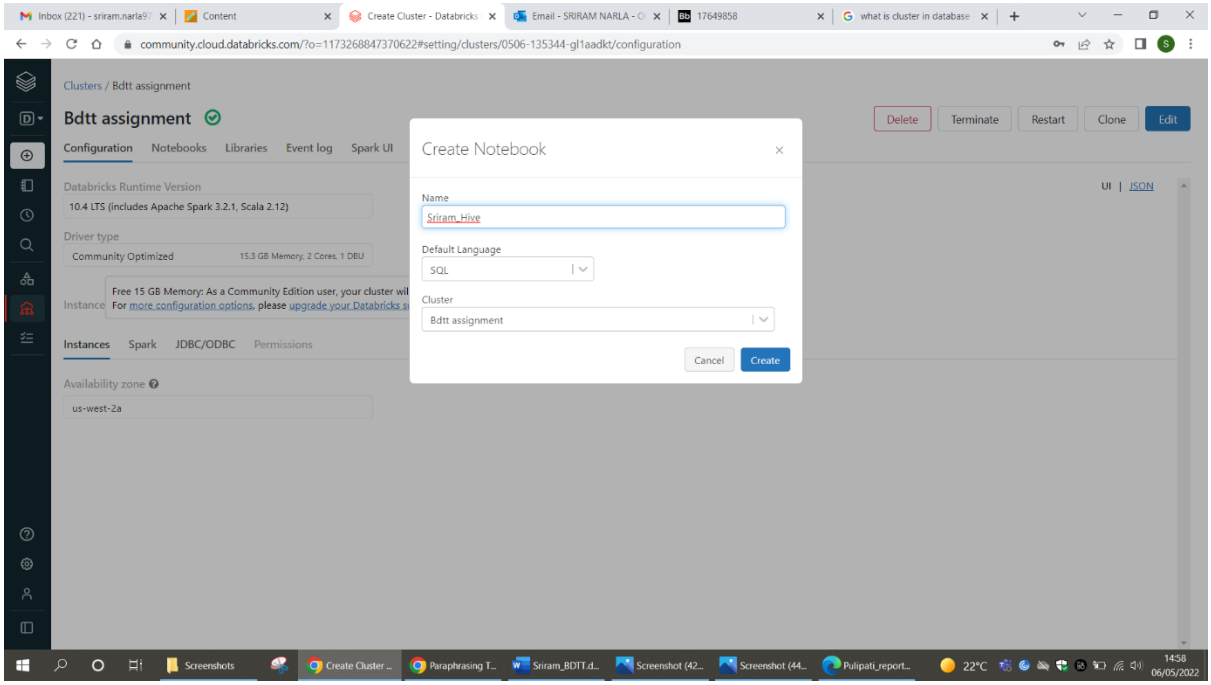
After Creating:



Next step is to create a notebook. For Dataframes and RDD Implementation, default language is Python.



Hive Notebook: SQL Language needs to be selected.

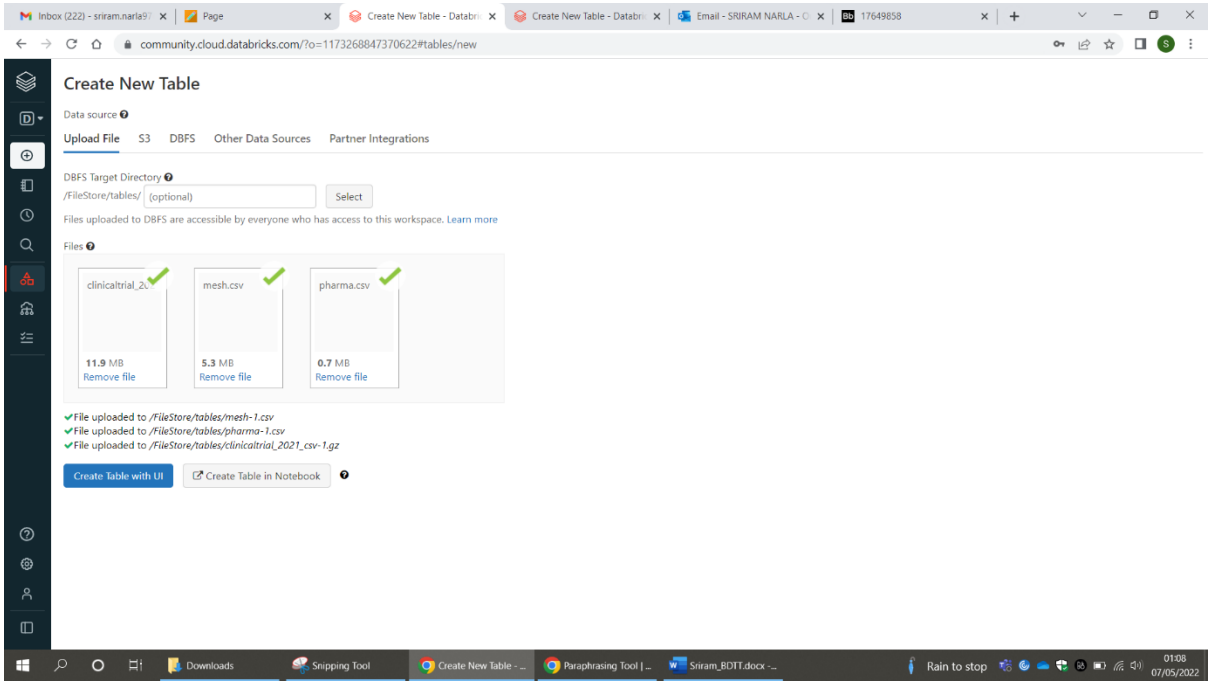


Data cleaning and preparation

Cleaning and preparing data are one of the most important aspects of data analysis. If the data is not properly prepared, the outcomes may be inconclusive, and clients may be misled. Although this is the first stage in the analysis, it is the most crucial because of the impact it has on the outcomes.

Pyspark Data Preparation:

Initially, I have uploaded clinicaltrial_2021.csv.gz, mesh.csv, pharma.csv files into the databricks. This is done using data -> create table -> upload selected files.



All these files are uploaded to location '/FileStore/tables/' in databricks file system(dbfs).

As a part of setup, I have created the notebook for pyspark implementation in Dataframe and RDD separately and data preparation is same in both.

I have imported the various functions/packages used during the implementation in the first two commands.

```
Cmd 1

#used for visualisation
%pip install bokeh

Python interpreter will be restarted.
Collecting bokeh
  Downloading bokeh-2.4.2-py3-none-any.whl (18.5 MB)
Collecting PyYAML>=3.10
  Downloading PyYAML-6.0-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (701 kB)
Requirement already satisfied: pillow>=7.1.0 in /databricks/python3/lib/python3.8/site-packages (from bokeh) (8.2.0)
Requirement already satisfied: tornado>=5.1 in /databricks/python3/lib/python3.8/site-packages (from bokeh) (6.1)
Collecting typing-extensions>=3.10.0
  Downloading typing_extensions-4.2.0-py3-none-any.whl (24 kB)
Requirement already satisfied: packaging>=16.8 in /databricks/python3/lib/python3.8/site-packages (from bokeh) (20.9)
Requirement already satisfied: numpy>=1.11.3 in /databricks/python3/lib/python3.8/site-packages (from bokeh) (1.20.1)
Requirement already satisfied: Jinja2>=2.9 in /databricks/python3/lib/python3.8/site-packages (from bokeh) (2.11.3)
Requirement already satisfied: MarkupSafe>=0.23 in /databricks/python3/lib/python3.8/site-packages (from Jinja2>=2.9->bokeh) (2.0.1)
Requirement already satisfied: pyparsing>=2.0.2 in /databricks/python3/lib/python3.8/site-packages (from packaging>=16.8->bokeh) (2.4.7)
Installing collected packages: typing-extensions, PyYAML, bokeh
Successfully installed PyYAML-6.0 bokeh-2.4.2 typing-extensions-4.2.0
Python interpreter will be restarted.

Command took 10.54 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:27:43 AM on Bdtt assignment
```

Cmd 2

```
import os
import pyspark.sql.functions
from pyspark.sql.functions import split,explode,from_unixtime,unix_timestamp,col,avg,window
import matplotlib.pyplot as plt
from bokeh.io import output_file, show
from bokeh.plotting import figure
from bokeh.embed import file_html
from bokeh.resources import CDN
```

```
Command took 0.59 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:30:24 AM on Bdtt assignment
```

File_path variable has the file path.

```
file_path = '/FileStore/tables/clinicaltrial_2021_csv.gz'
```

```
Command took 0.03 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 1:25:44 AM on Bdtt assignment
```

Function file_exists is created to check whether the given file exists or not.

```
def file_exists(path):
    try:
        dbutils.fs.ls(path)
        return True
    except Exception as e:
        if 'java.io.FileNotFoundException' in str(e):
            return False
        else:
            raise
```

```
Command took 0.03 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 1:25:46 AM on Bdtt assignment
```

If file exists, splitted the file_path to get the root name in order to make it an environment variable to make it accessible through the command line. I have copied the file to the local temp folder(file:/tmp/) to unzip the file else it will tell file donot exists.

```
if file_exists(file_path) == True:
    fileroot = file_path.split("/")[3].split(".")[0]
    dbutils.fs.cp(file_path,"file:/tmp/")
else:
    print("File is not found in dbfs")
```

```
Command took 0.71 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 1:25:47 AM on Bdtt assignment
```

In this command, I have made the above fileroot variable as an environment variable using the function os.environ

Cmd 4

```
os.environ['fileroot'] = fileroot
```

```
Command took 0.03 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:41:08 AM on Bdtt assignment
```

I have used the below Unix command to gzip the file and copy the extracted file to the same local temp folder.

Cmd 5

```
%sh
gzip -d /tmp/ /tmp/$fileroot.gz
```

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

Command took 0.47 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:44:01 AM on Bdtt assignment

Next, I have moved the file to required location('/FileStore/tables/) from temp folder.

Cmd 6

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

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

Command took 4.68 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:54:38 AM on Bdtt assignment

As the extracted file has the name ending with _csv, I have used below condition and changed it to the .csv format and moved the file to same location by renaming.

Cmd 7

```
if fileroot.endswith("_csv"):
    fileroot_new = fileroot.replace("_csv",".csv")
dbutils.fs.mv("/FileStore/tables/" + fileroot,"/FileStore/tables/" + fileroot_new)
```

```
Out[7]: True
```

Command took 4.13 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:54:48 AM on Bdtt assignment

Dataframes:

I have created a user defined function to read the files into a dataframe. It takes the input parameters such as file path, header, delimiter and returns appropriately.

Cmd 8

```
def read_file(file_path,header,delimiter):
    if header == True:
        return(spark.read.option("header",True).csv(file_path,sep=delimiter))
    else:
        return(spark.read.option("header",False).csv(file_path,sep=delimiter))
```

Command took 0.03 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:58:36 AM on Bdtt assignment

Now I have read the files into the respective dataframes by using the function read_file created above by giving the input parameters.

Cmd 9

```
file_path = "/FileStore/tables/"+fileroot_new
clinical_trail = read_file(file_path,True,"|")
mesh_df = read_file("/FileStore/tables/mesh.csv",True,",")
pharma_df = read_file("/FileStore/tables/pharma.csv",True,",")
```

▸ (3) Spark Jobs

Command took 8.68 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 2:02:30 AM on Bdtt assignment

RDD:

In RDD implementation, I have created similar function the read the data into respective Rdd's.

Cmd 6

```
def read_file(file_path,header):
    if header == True:
        rdd = sc.textFile(file_path)
        header = rdd.take(1)[0]
        return(rdd.filter(lambda line: line != header))
    else:
        return(sc.textFile(file_path))
```

```
Command took 0.02 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 2:06:32 AM on Bdtt assignment
```

Next, I have read the data into respective Rdd's. Clinical_trail file has a delimiter '|', I have used the map function and splitted the columns.

Cmd 7

```
file_path = "/FileStore/tables/"+fileroot_new
clinical_rdd = read_file(file_path,True)
mesh_rdd = read_file("/FileStore/tables/mesh.csv",True)
pharma_rdd = read_file("/FileStore/tables/pharma.csv",True)
clinical_trail = clinical_rdd.map(lambda s: s.split("|"))
```

▸ (3) Spark Jobs

```
Command took 3.75 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 2:06:38 AM on Bdtt assignment
```

Hive data preparation:

In this implementation also the file set up is same

I have dropped the tables if exists with the same name.

Cmd 1

```
drop table if exists clinicaltrail_2021;
drop table if exists pharma_data;
drop table if exists mesh_data;
```

OK

```
Command took 0.75 seconds -- by s.narla@edu.salford.ac.uk at 5/13/2022, 12:55:41 PM on Bdtt assignment
```

I have created tables using the below code.

Clinicaltrial table creation:

Cmd 9

```
CREATE TABLE if not exists clinicaltrail_2021(
ID STRING,
Sponsor STRING,
Status STRING,
Start STRING,
Completion STRING,
Type STRING,
Submission STRING,
Conditions STRING,
Interventions STRING)
USING csv OPTIONS ( 'multiLine' 'true', 'escape' '"', 'header' 'true', 'delimiter' '|')
LOCATION '/FileStore/tables/clinicaltrial_2021.csv';
```

OK

Command took 3.04 seconds -- by s.narla@edu.salford.ac.uk at 5/13/2022, 12:44:21 AM on bdtt assignment

Pharma table creation:

Cmd 10

```
CREATE TABLE if not exists pharma(
Company STRING,
Parent_Company STRING,
Penalty_Amount STRING,
Subtraction_From_Penalty STRING,
Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting STRING,
Penalty_Year STRING,
Penalty_Date STRING,
Offense_Group STRING,
Primary_Offense STRING,
Secondary_Offense STRING,
Description STRING,
Level_of_Government STRING,
Action_Type STRING,
Agency STRING,
Civil_Criminal STRING,
Prosecution_Agreement STRING,
Court STRING,
Case_ID STRING,
Private_Litigation_Case_Title STRING,
Lawsuit_Resolution STRING,
Facility_State STRING,
City STRING,
Address STRING,
Zip STRING,
NAICS_Code STRING,
NAICS_Translation STRING,
HQ_Country_of_Parent STRING,
HQ_State_of_Parent STRING,
Ownership_Structure STRING,
Parent_Company_Stock_Ticker STRING,
Major_Industry_of_Parent STRING,
Specific_Industry_of_Parent STRING,
Info_Source STRING,
Notes STRING)
USING csv OPTIONS ( 'multiLine' 'true', 'escape' '"', 'header' 'true', 'delimiter' ',')
LOCATION '/FileStore/tables/pharma.csv';
```

OK

Command took 1.05 seconds -- by s.narla@edu.salford.ac.uk at 5/13/2022, 12:44:21 AM on bdtt assignment

Mesh table creation:

Cmd 11

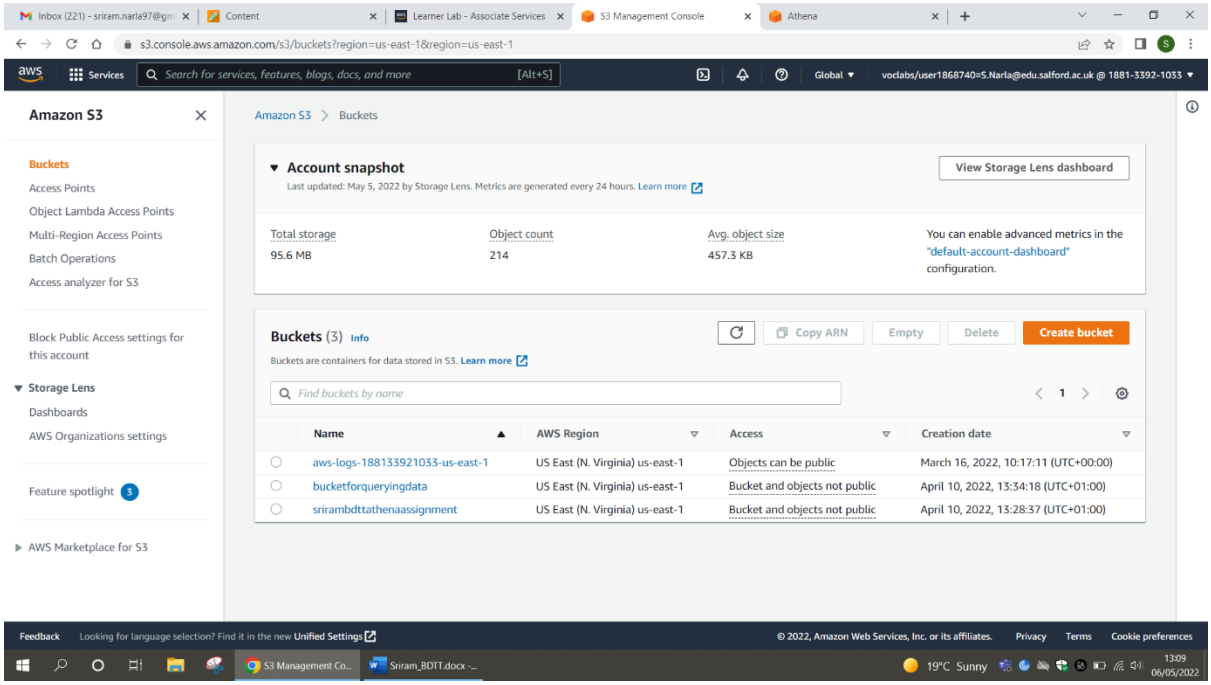
```
create table if not exists mesh(
term STRING,
tree STRING)
USING csv OPTIONS ( 'multiLine' 'true', 'escape' '"', 'header' 'true', 'delimiter' ',')
LOCATION '/FileStore/tables/mesh.csv';
```

OK

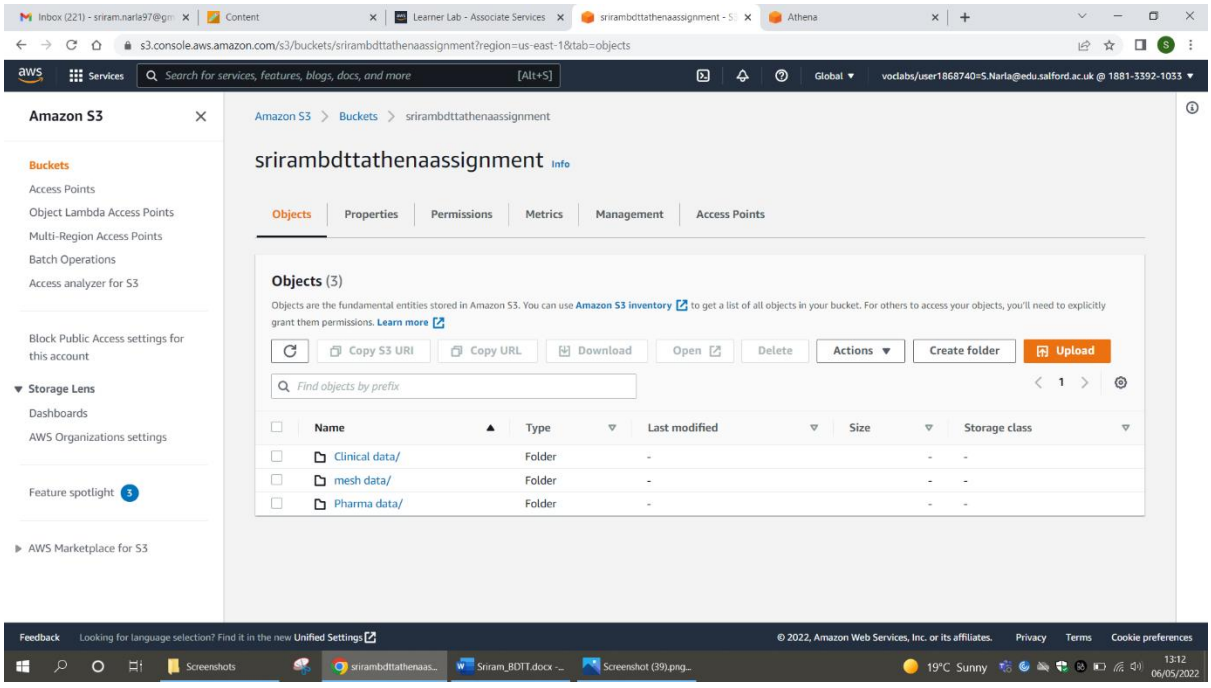
Command took 0.84 seconds -- by s.narla@edu.salford.ac.uk at 5/13/2022, 12:44:21 AM on bdtt assignment

AWS Data Preparation:

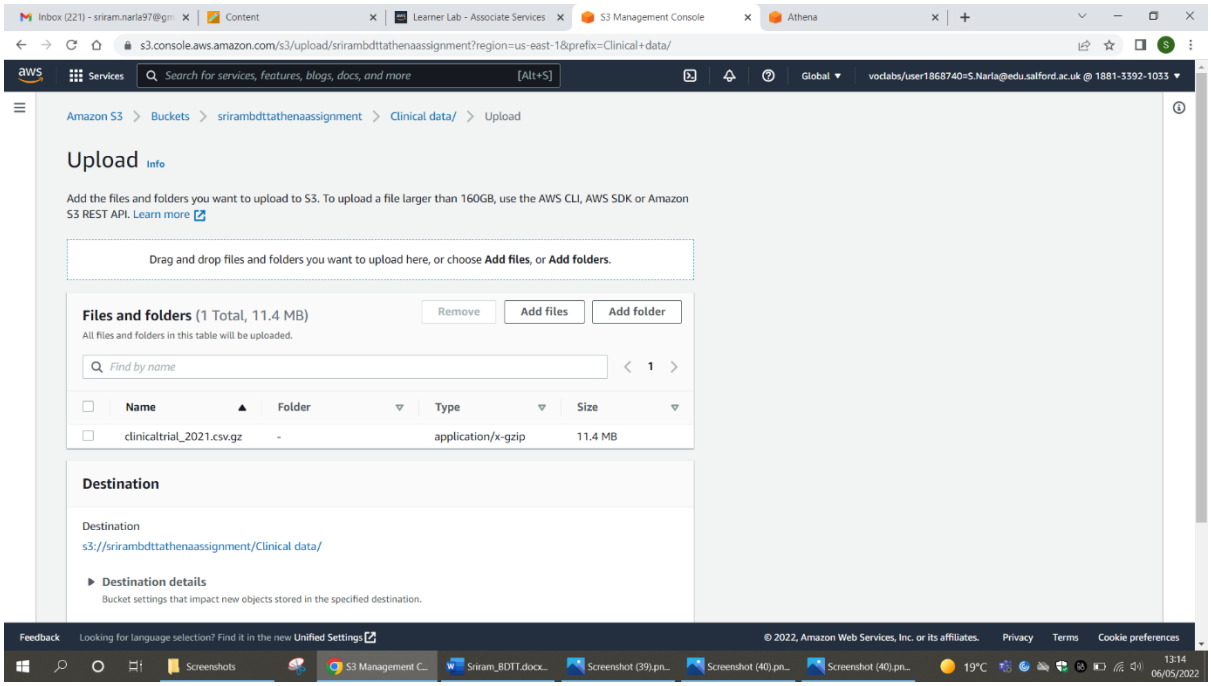
In S3, I have created the bucket using the Create bucket option.



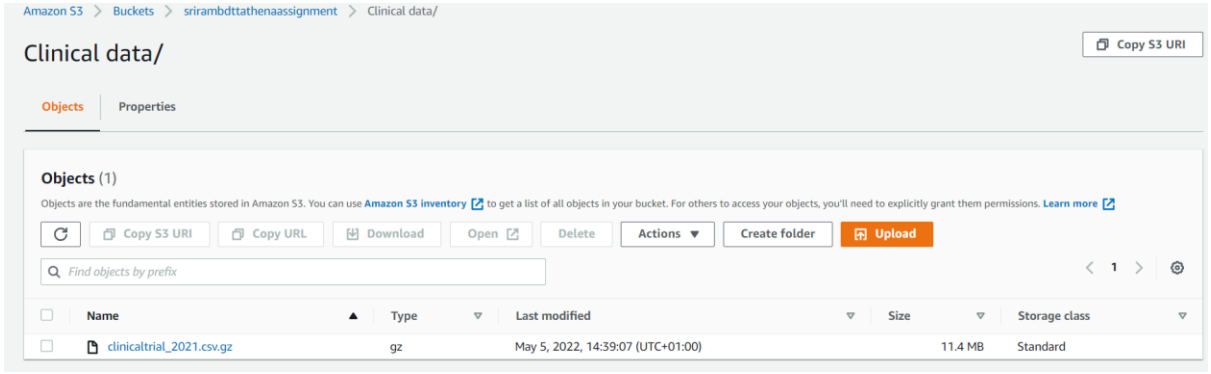
I have created a bucket srirambdttathenaassignment with three folders in it to store the data.



In each folder, I have clicked on the upload data and uploaded the required file as shown below.



File uploaded in clinical data is as shown below:

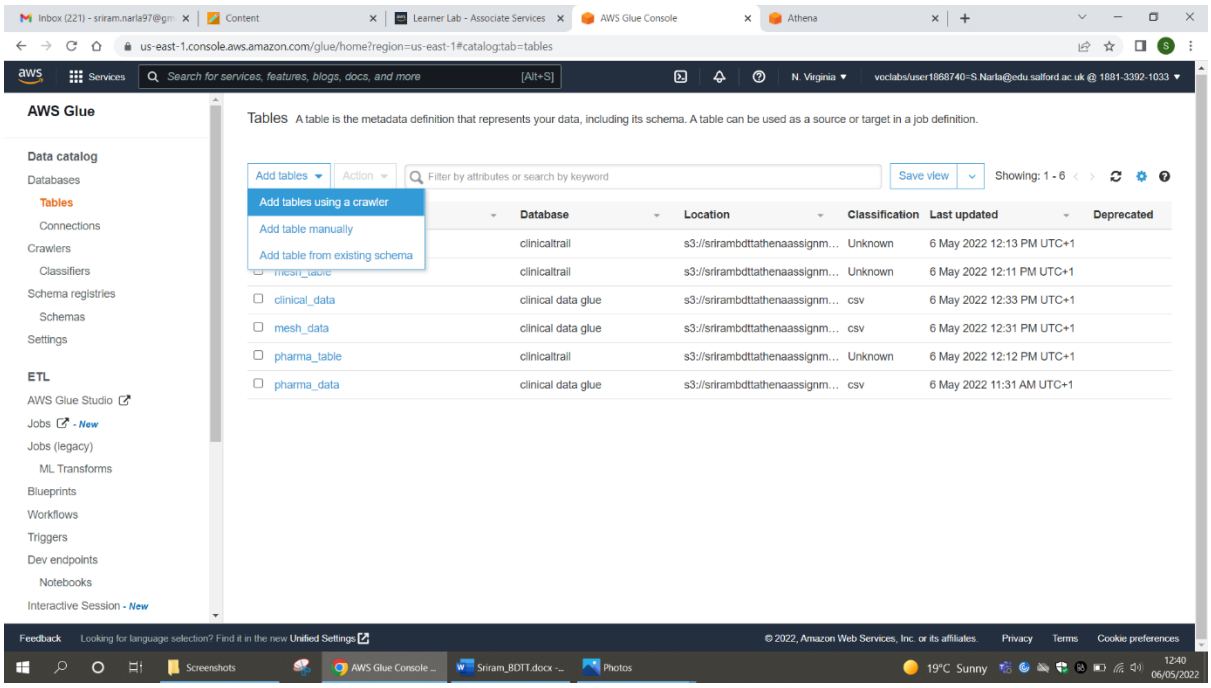


Similarly I have uploaded the mesh and the pharma data files in the respective folders.

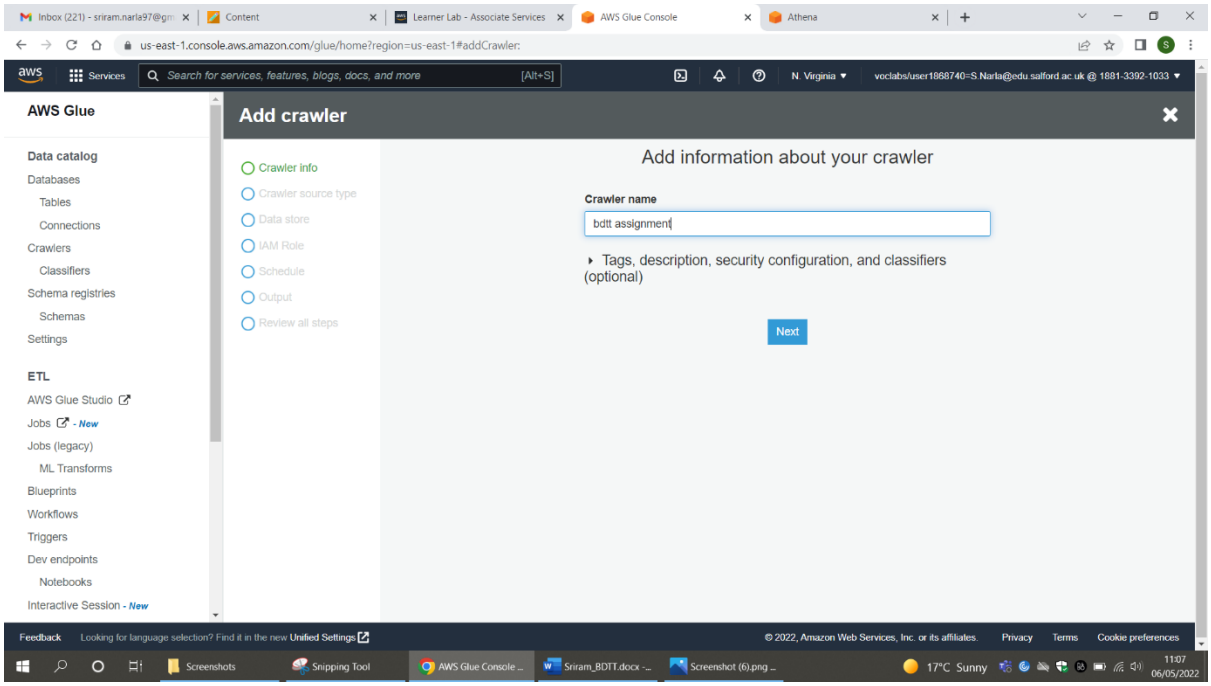
Data tables can be built in two ways when querying with AWS Athena. One is to use AWS Glue,which is an ETL service, and the other is to construct tables and query them directly in Athena. I've generated tables in both ways

Data Preparation AWS GLUE:

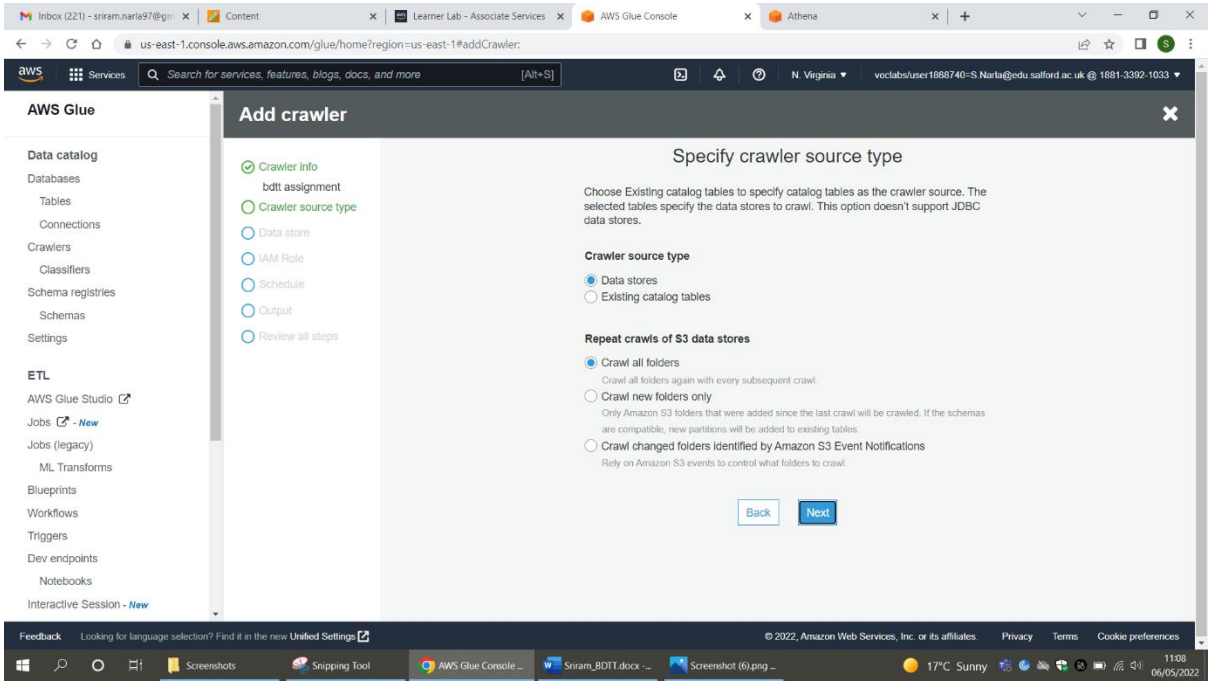
Goto Tables -> Add tables using crawler:



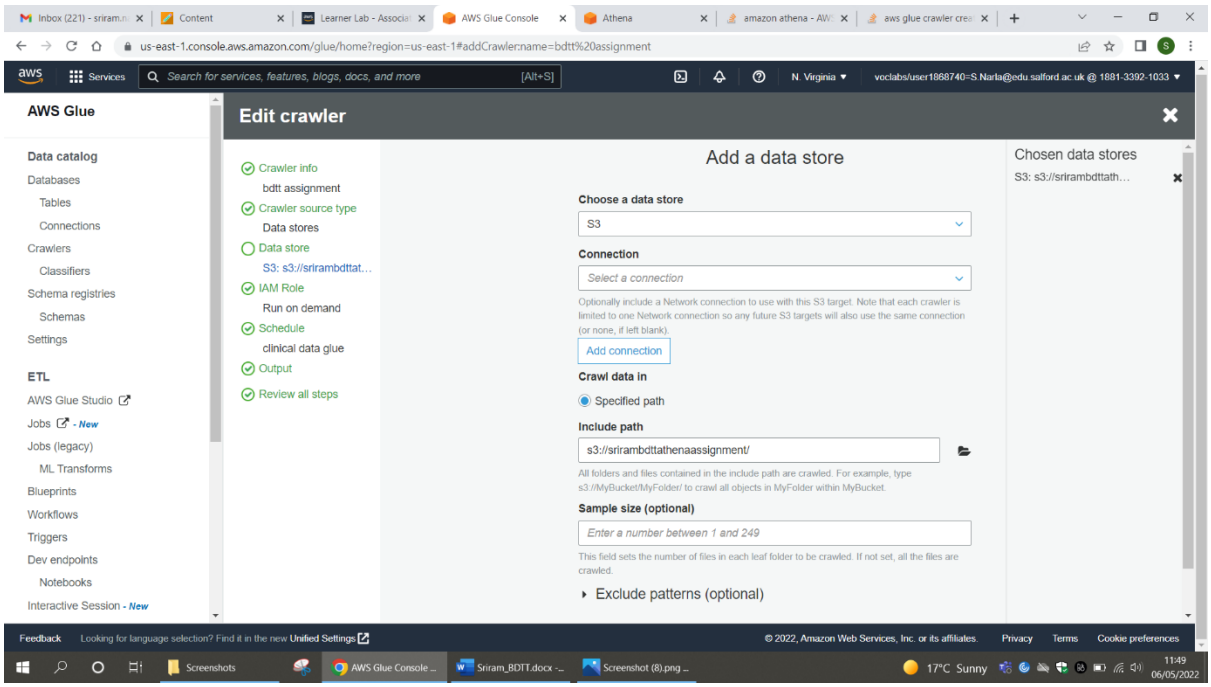
Add crawler-> Give name



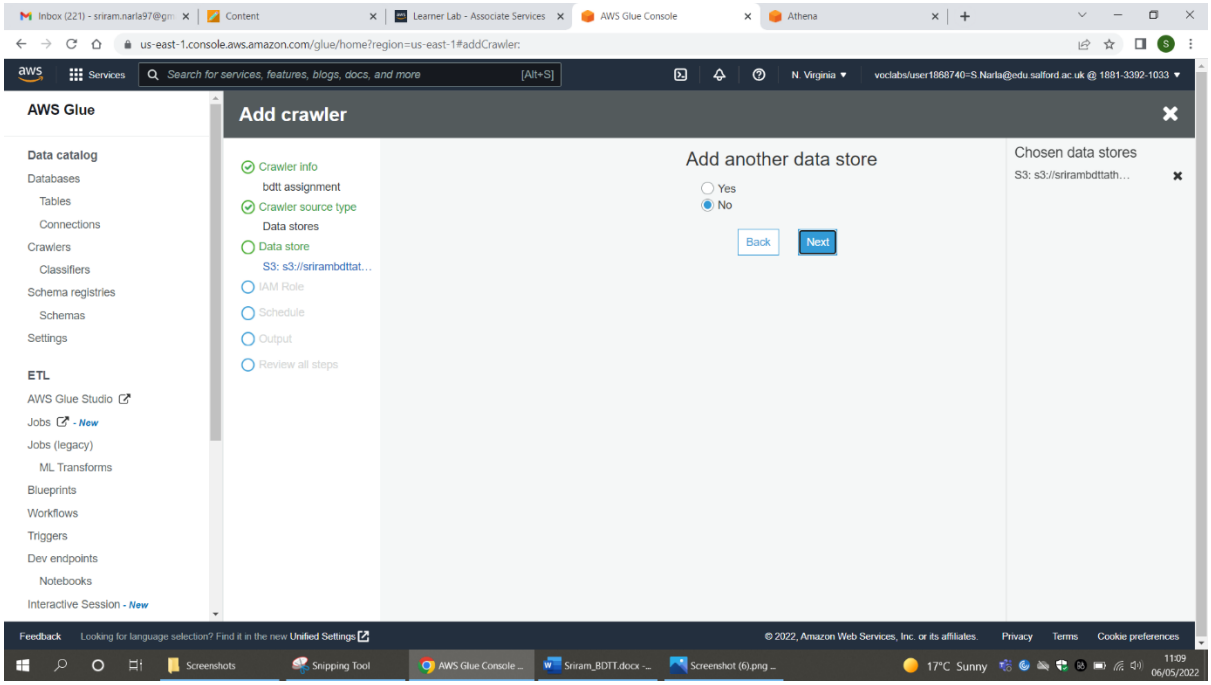
In Source Type, choose data stores and crawl all folders.



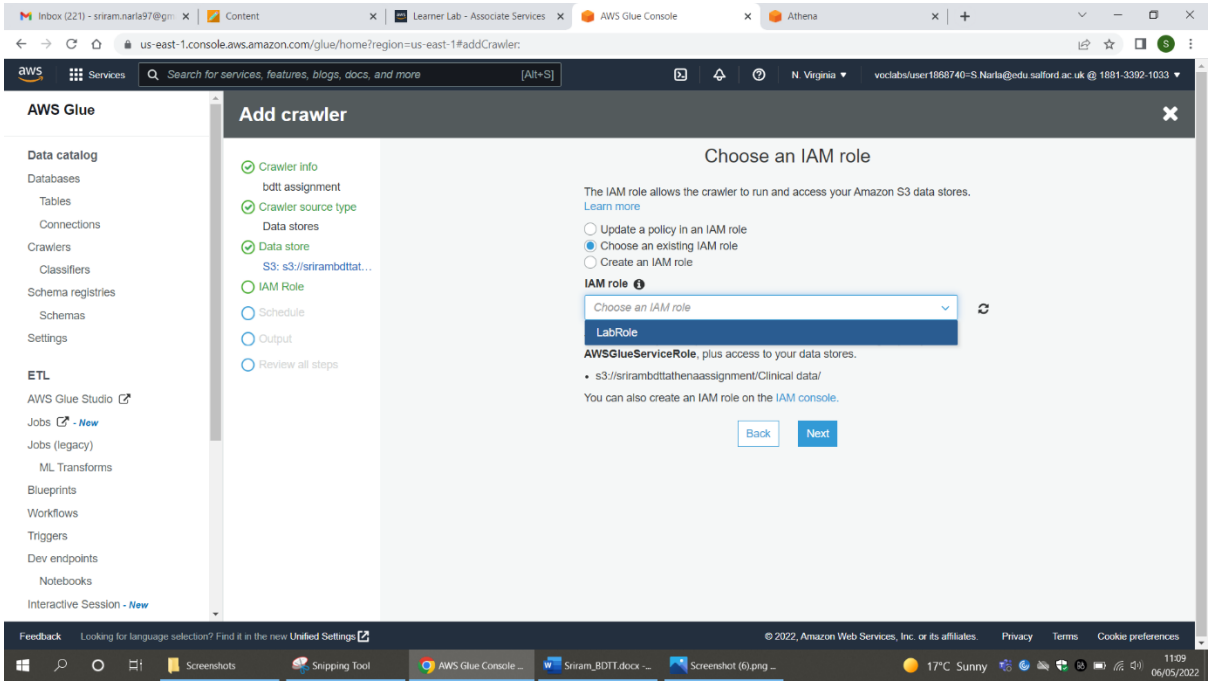
Then choose the S3 in data store and give required S3 path in include path.



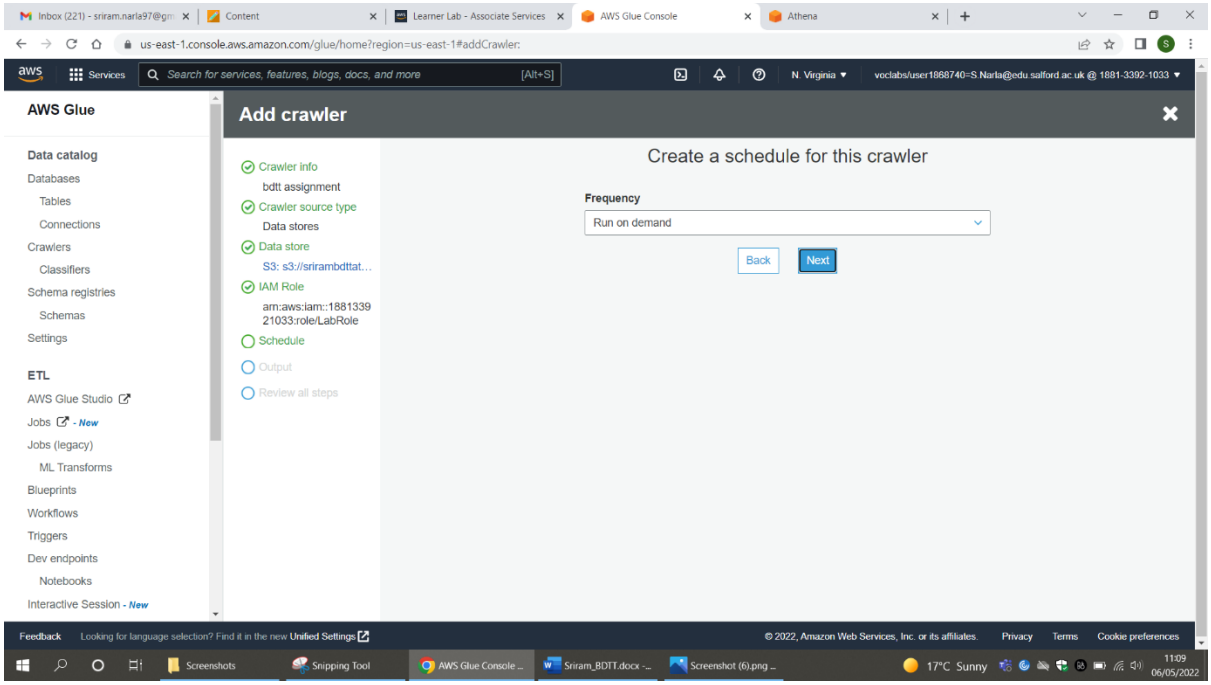
Donot add another data store



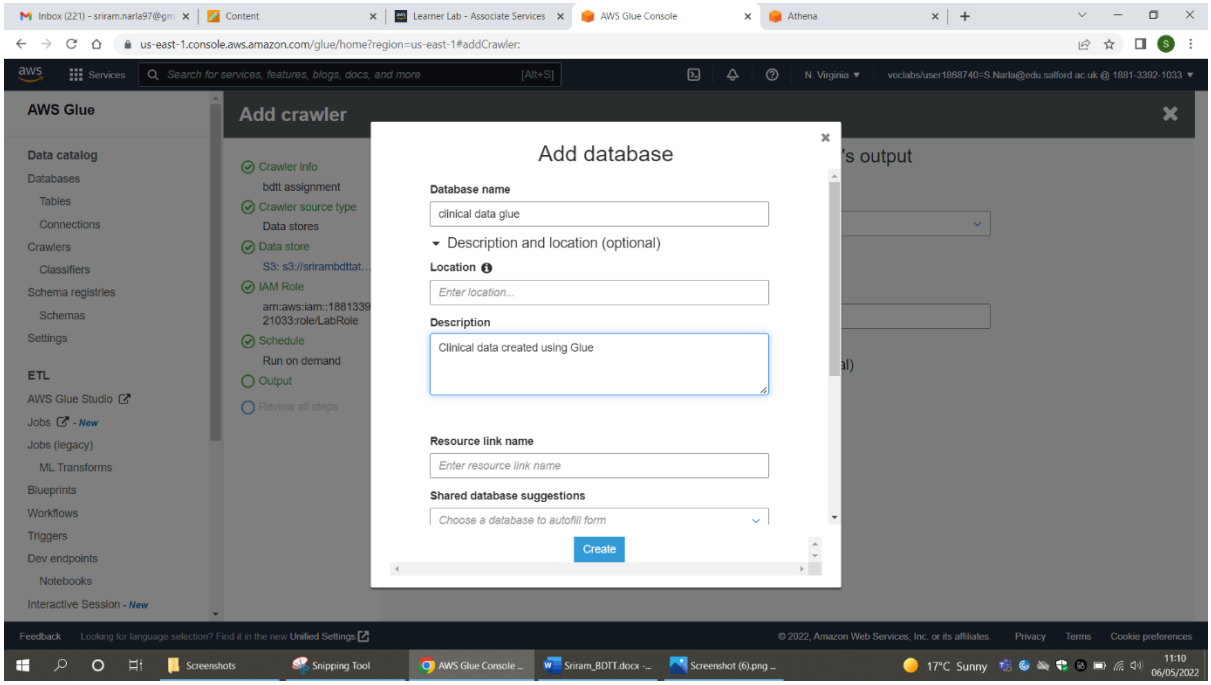
In the IAM role, choose the existing IAM role which is LabRole created for the Glue Assignment.

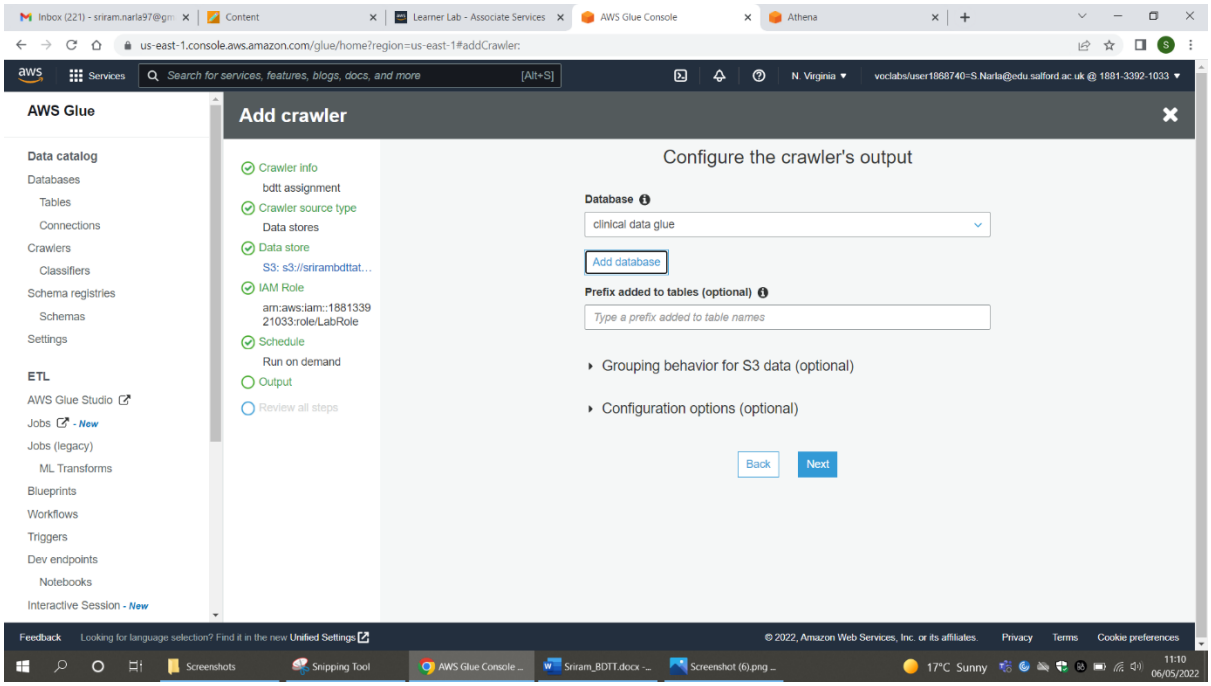


Frequency is run on demand

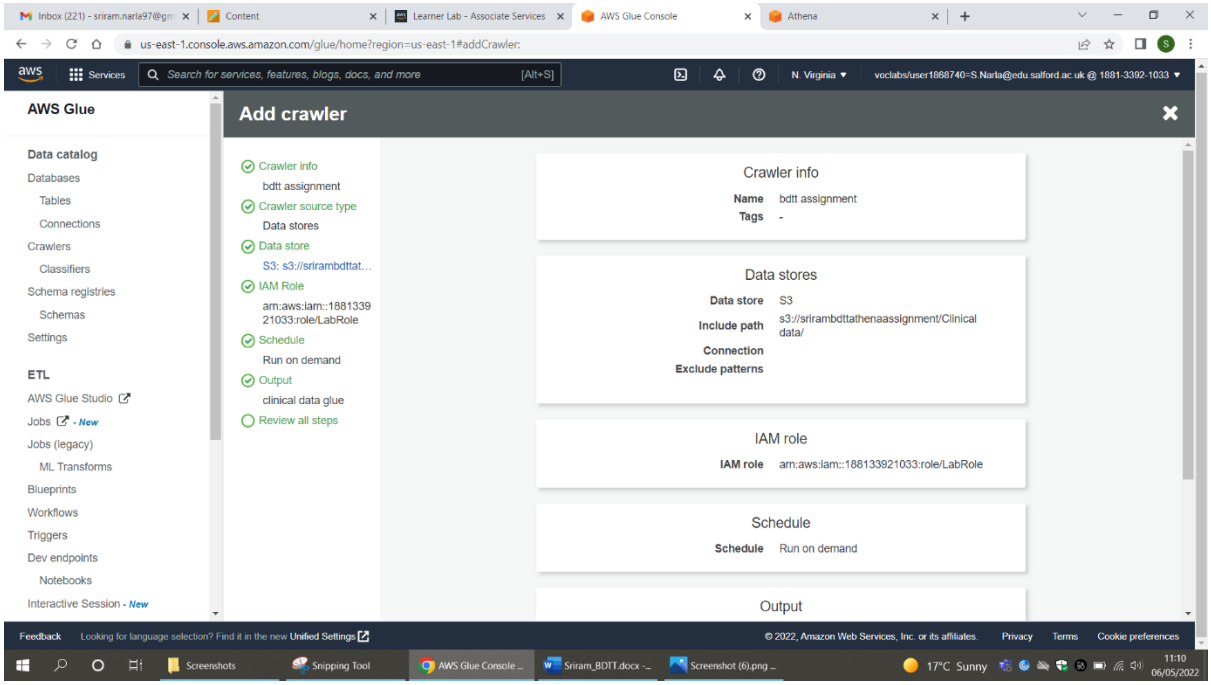


In Add database, created a database clinical data glue to store all the required tables.

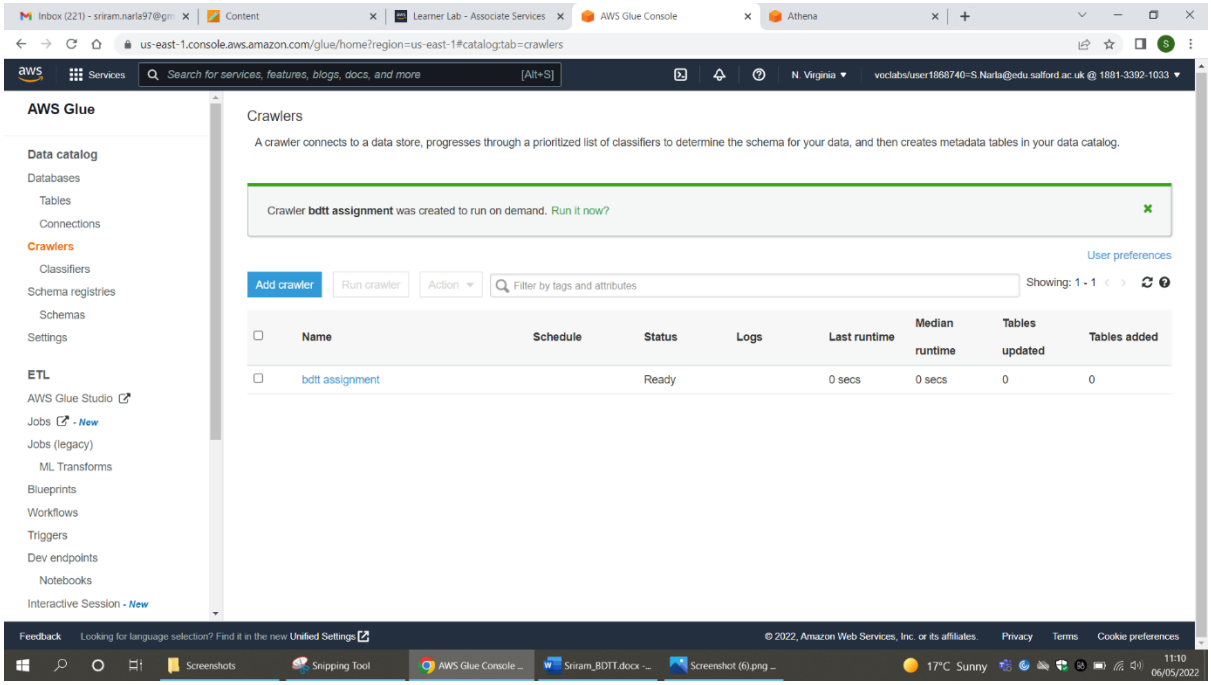




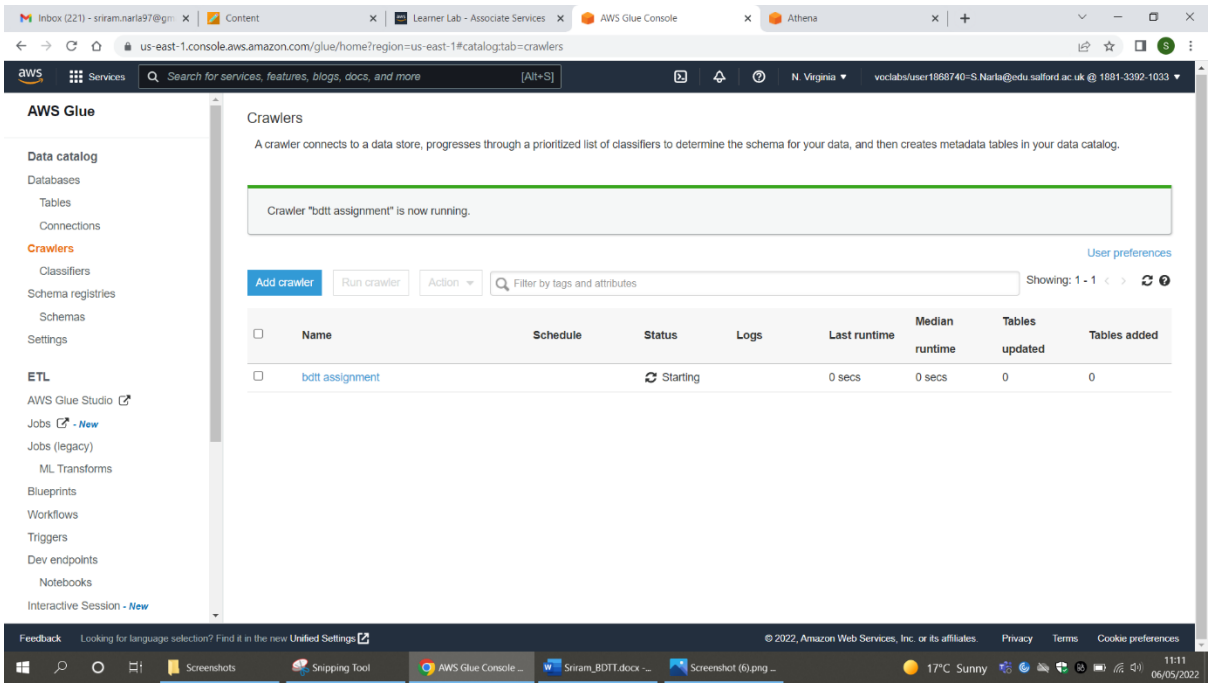
Reviewing all the steps:



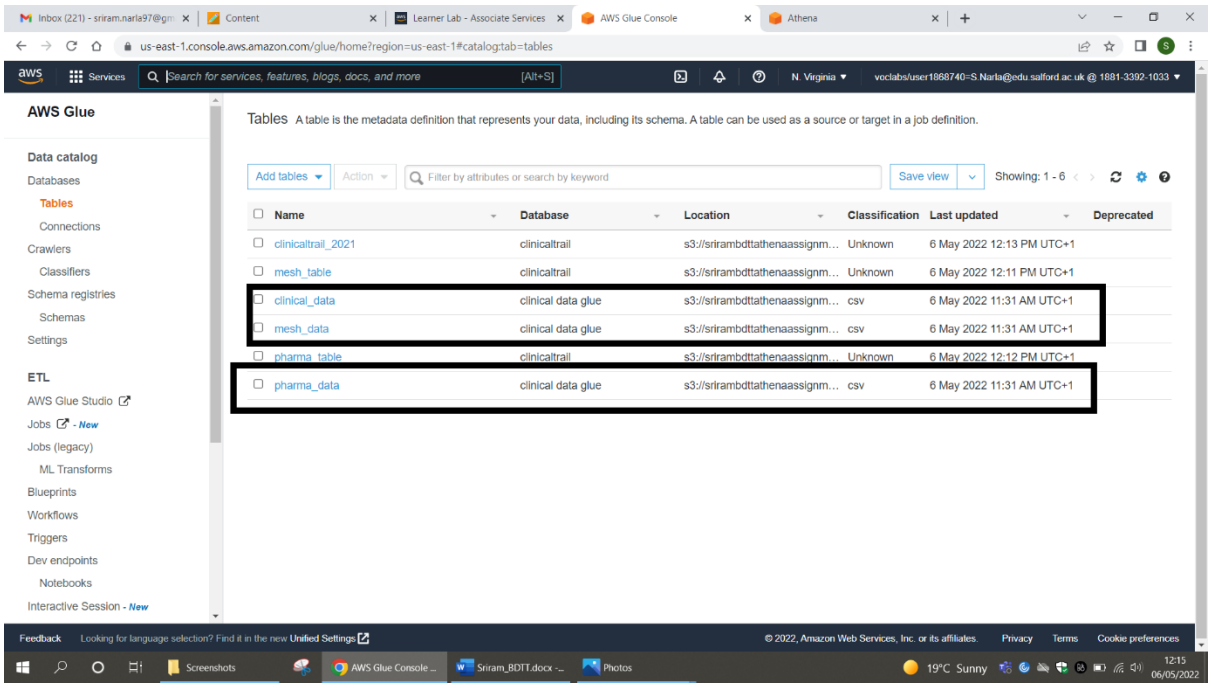
After creating, click on run the crawler



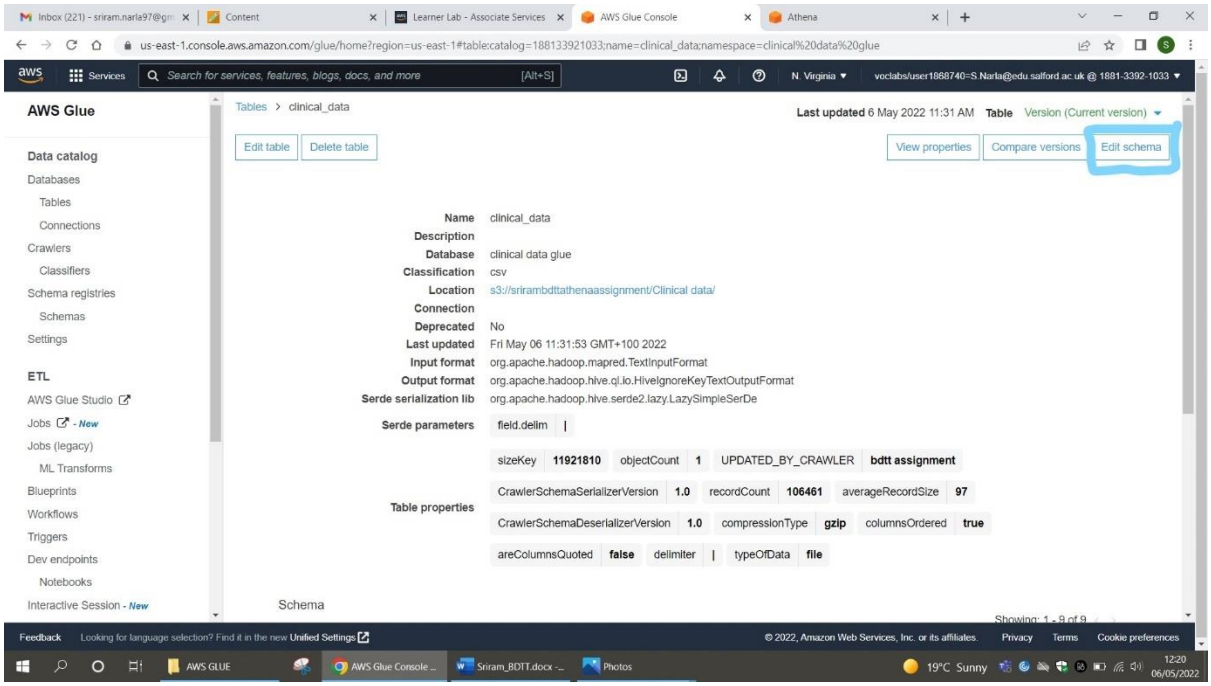
Crawler is running.



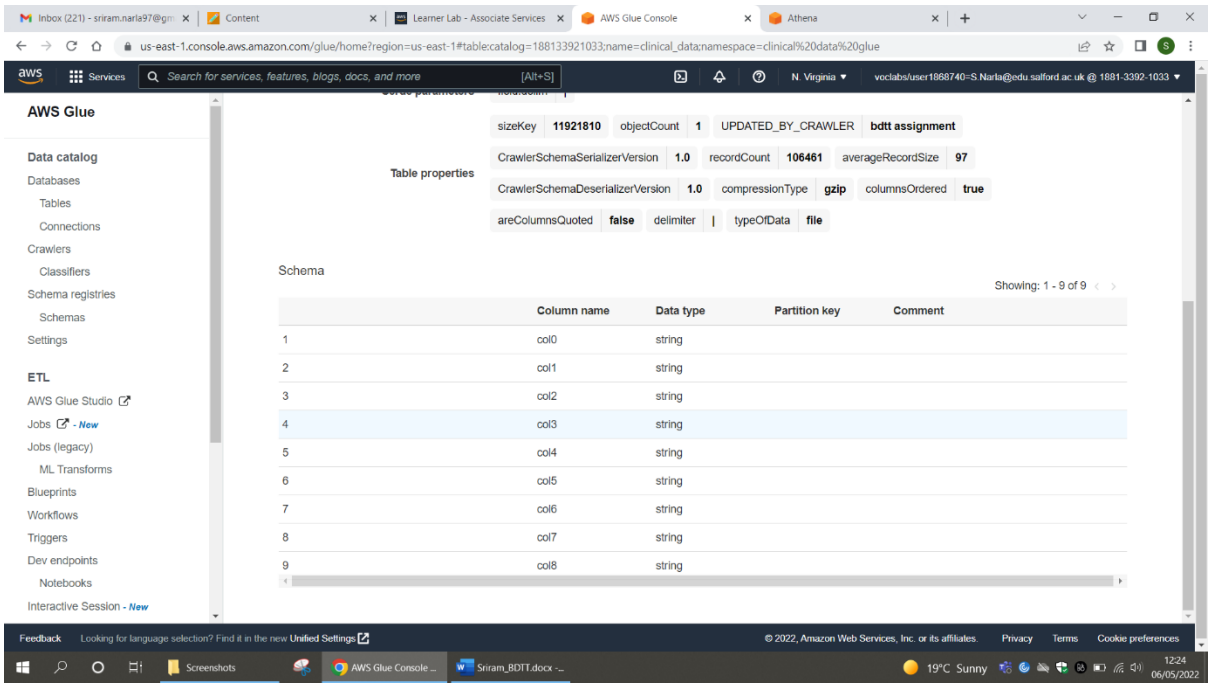
Next Go to tables and check if the tables are created. Three tables clinical_data, mesh_data, pharma_data are created in the above created database clinical data glue



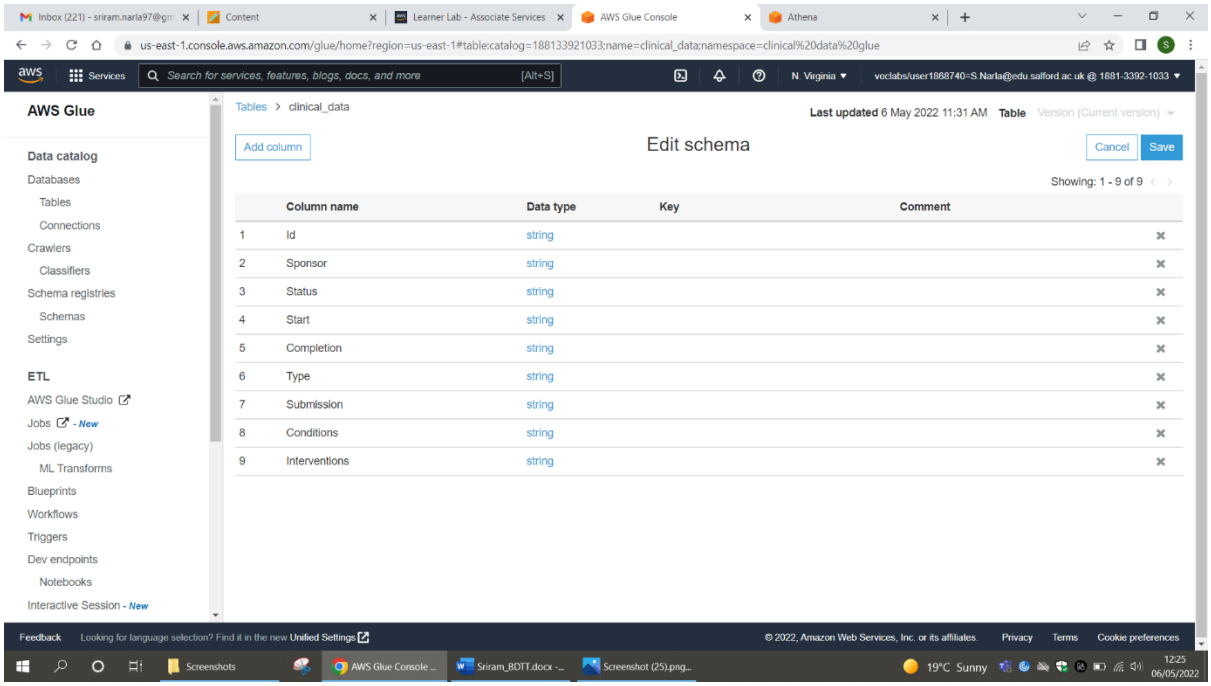
Open table to view properties, edit schema and exclude the header.



We can see that Glue has detected the total file size and delimiter, record count, compression type and all. On the right side, click on edit schema.

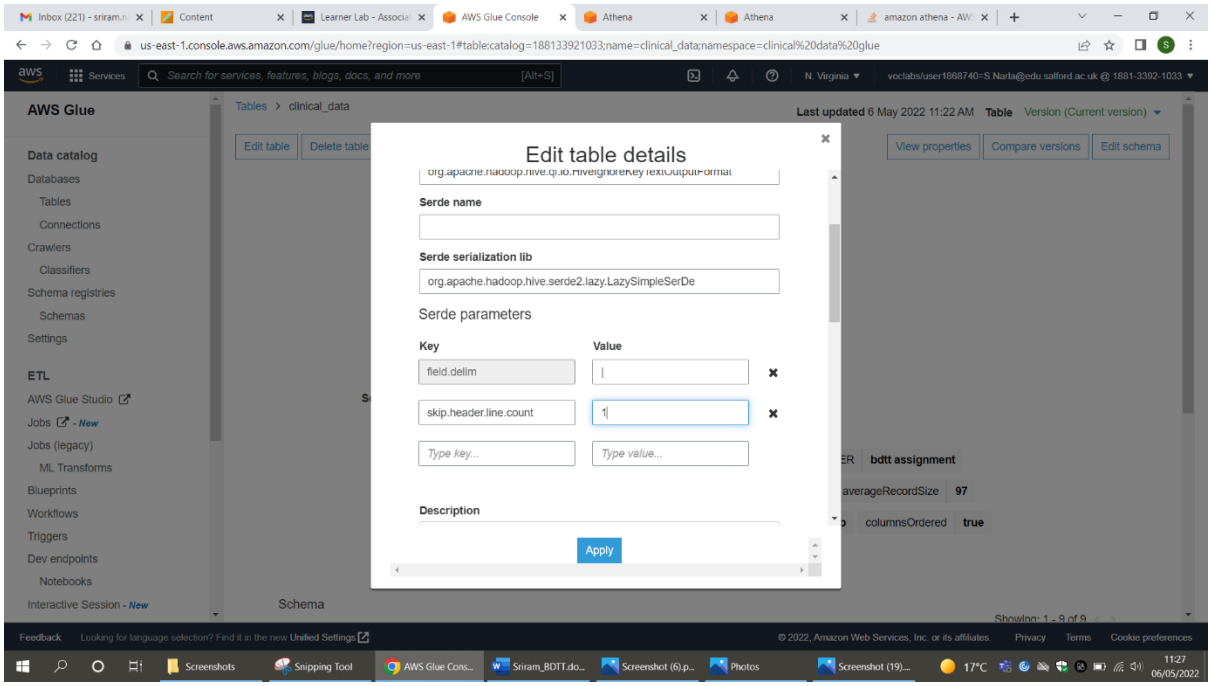


After changing the column names

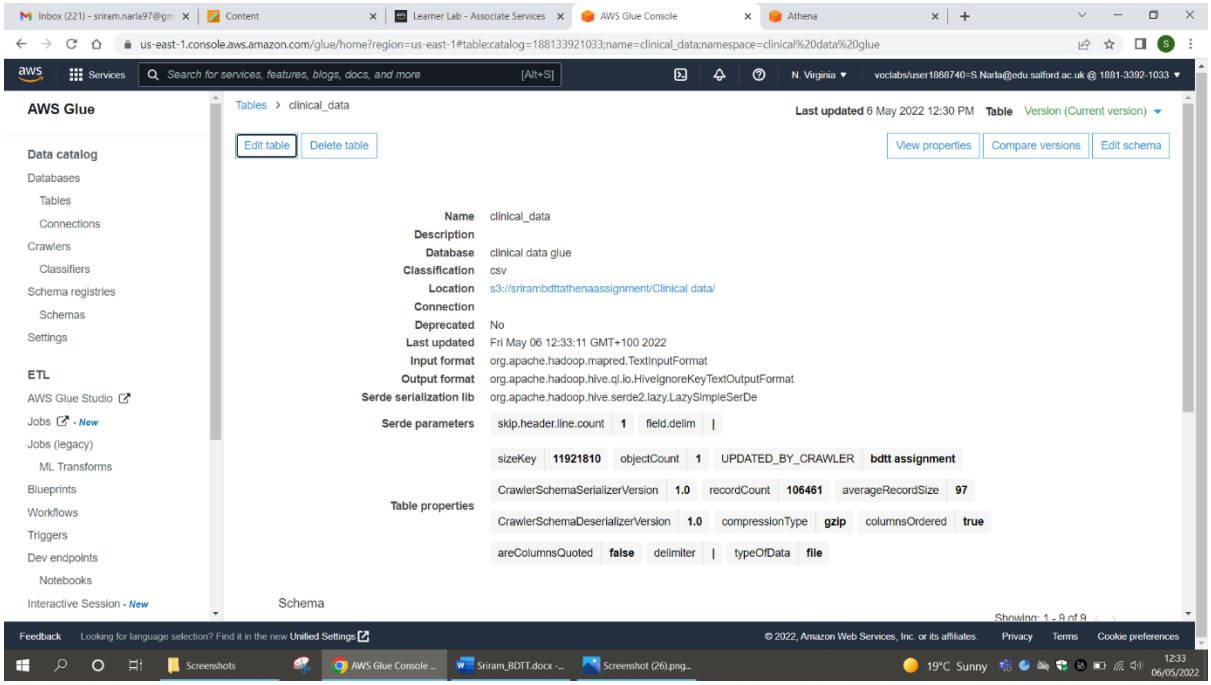


Then in the Edit table properties add the following to detect the header.

In serde parameters, give Key as skip.header.line.count and value as 1.

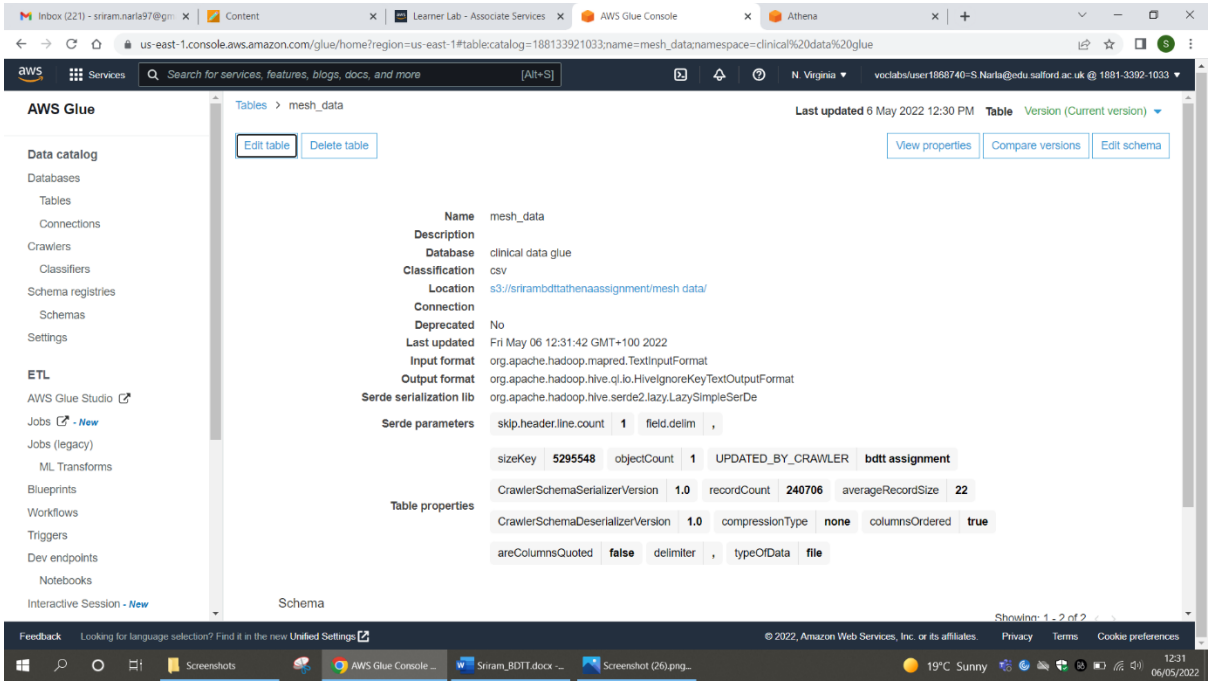


Again, view the properties to see whether they are changed.

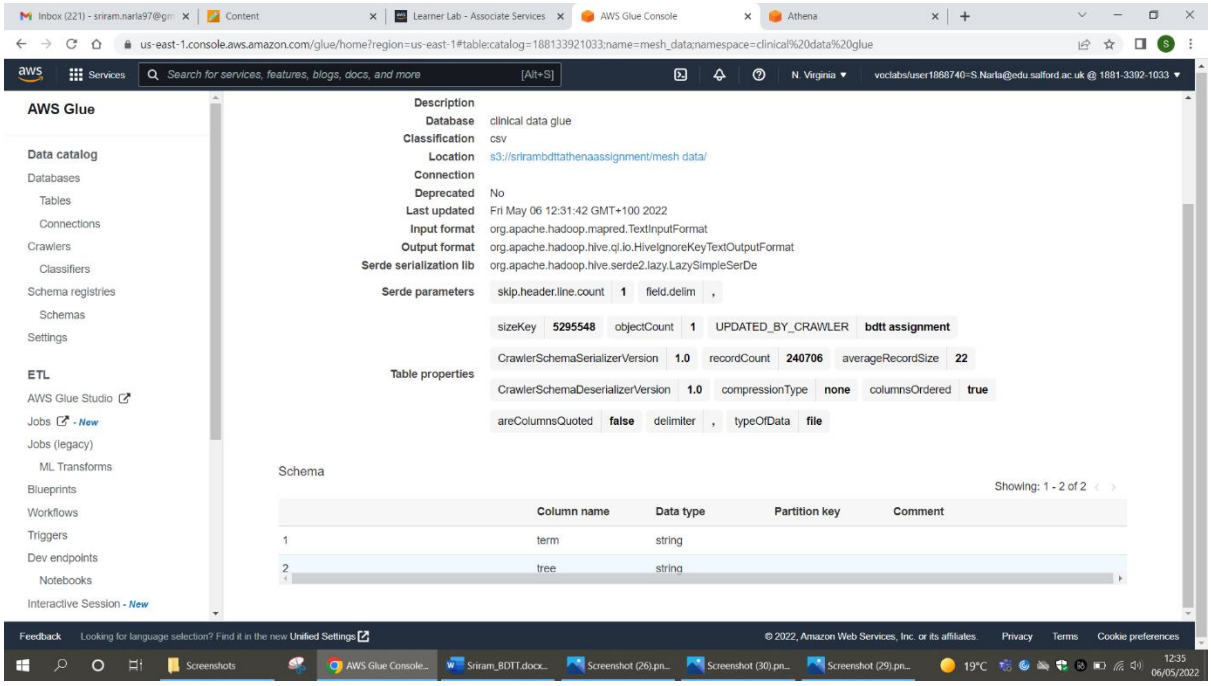


Same approach is taken for mesh data, pharma data tables as well.

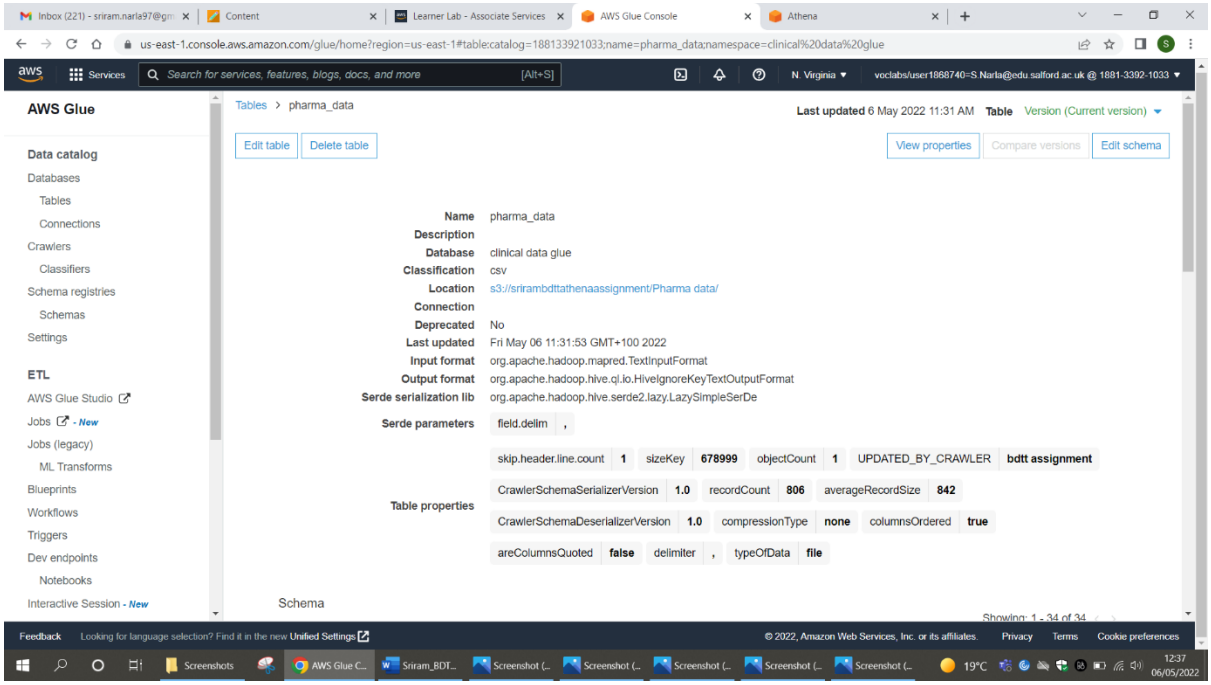
Mesh data Table properties



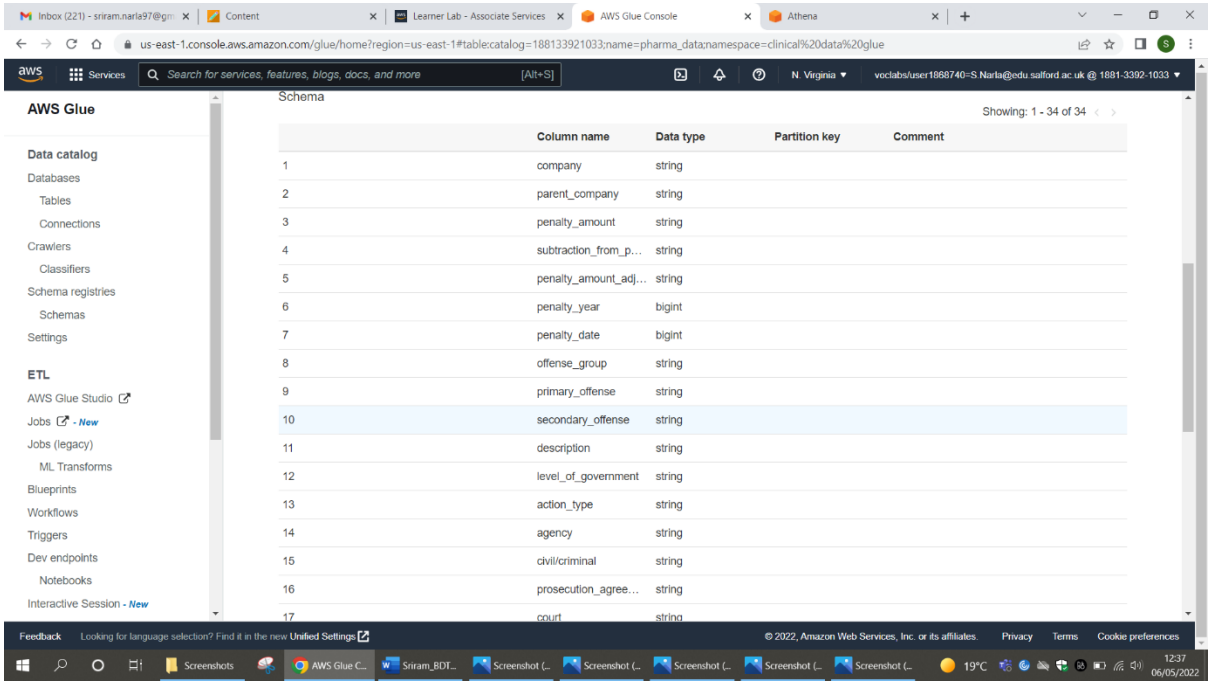
Schema:



Pharma table properties:

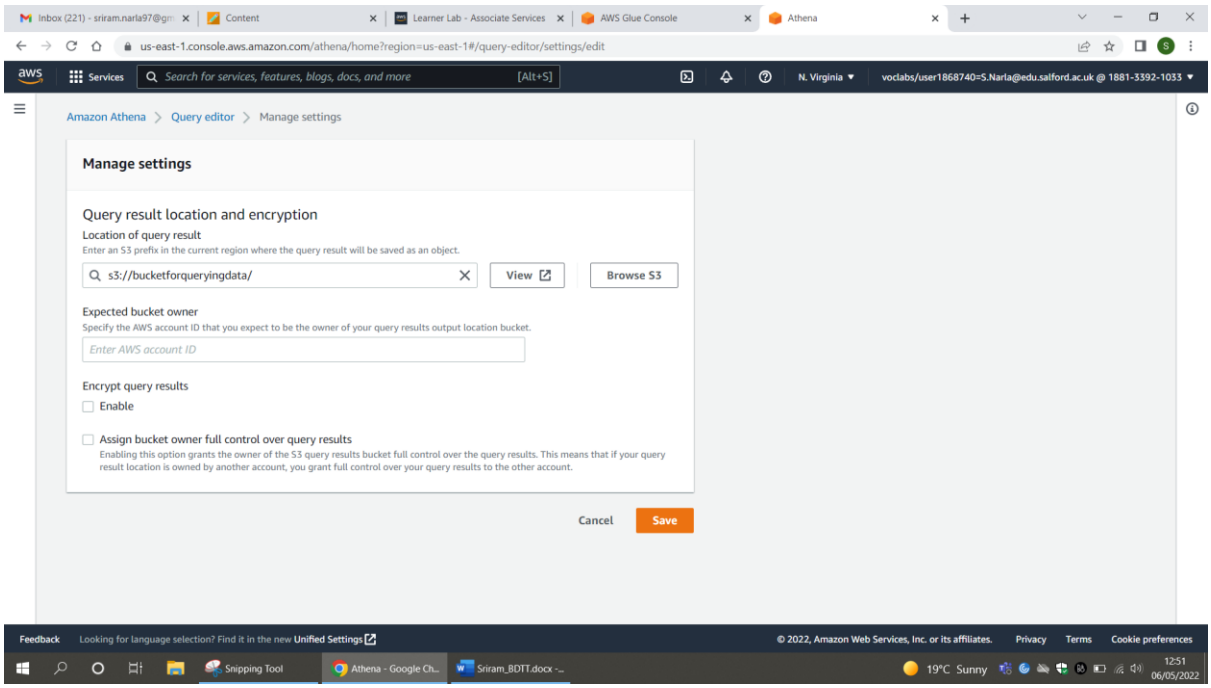


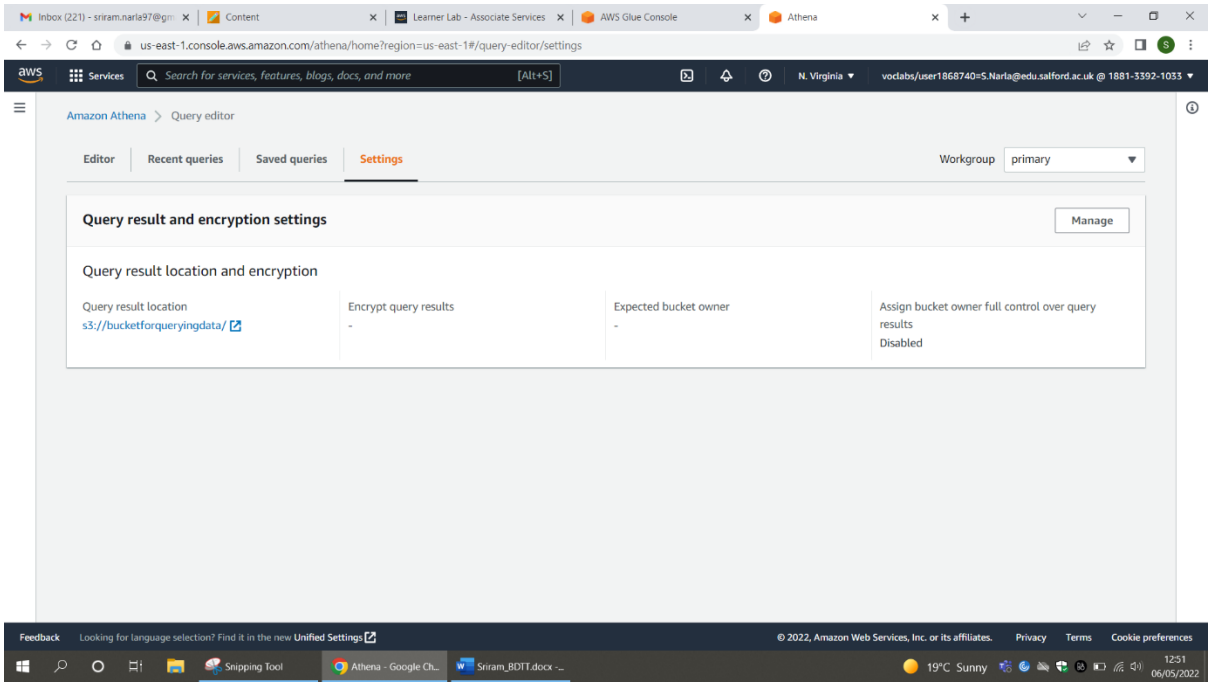
Schema for pharma table:



Data Preparation in Athena:

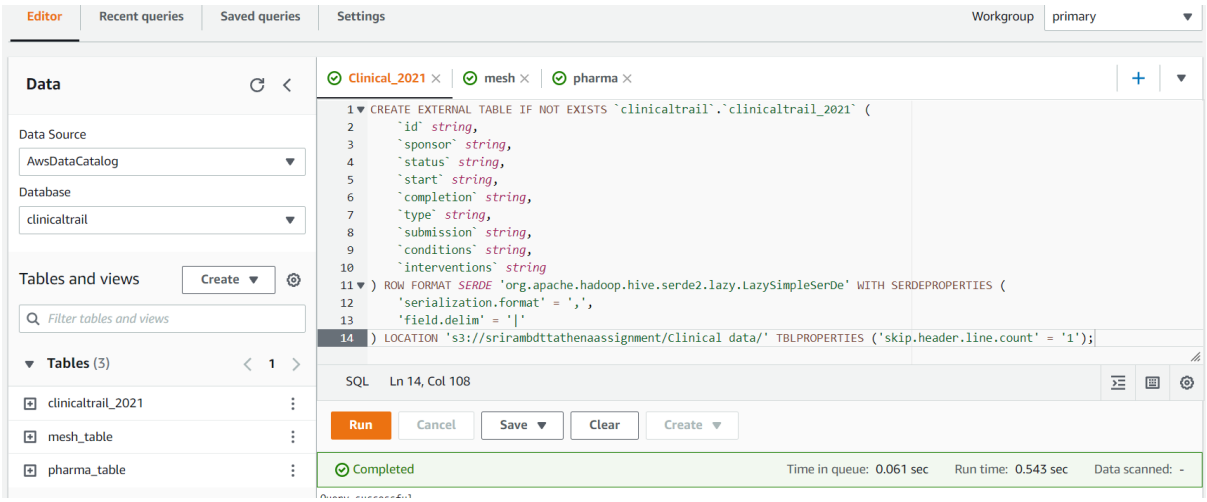
To do the Querying in Athena, in settings give a query result location. I have already created a bucket with name bucketforqueryingdata for this purpose.



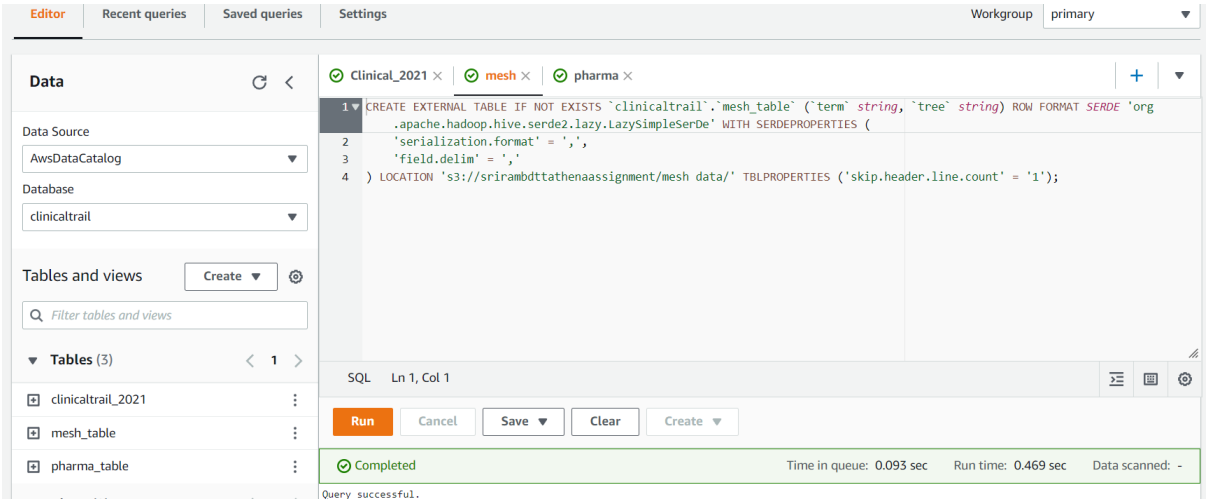


In Athena, I have used the below code to create the tables:

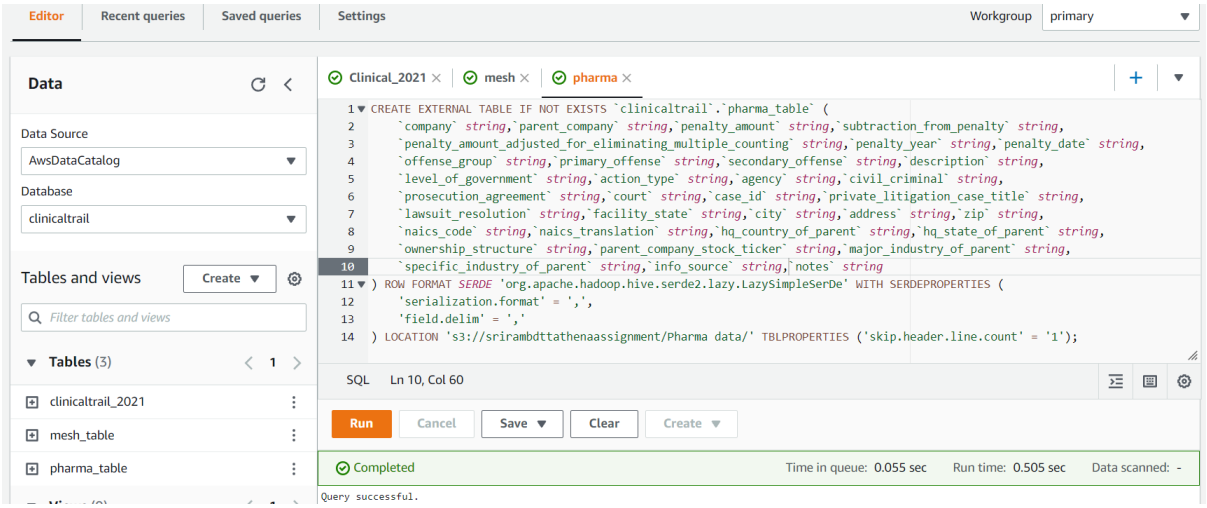
Clinical data table creation:



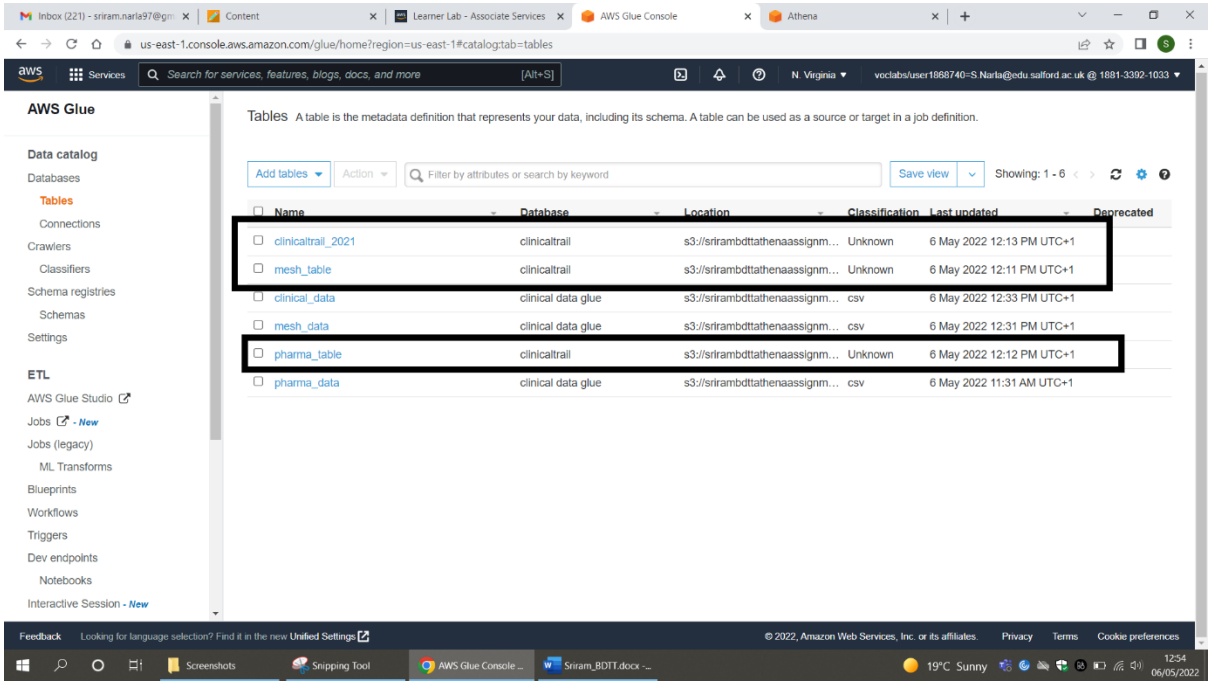
Mesh data table creation:



Pharma data table creation:



The tables created can also be viewed in the AWS Glue tables.



Problem Answers

Question 1:

Assumptions made:

ID column is selected assuming that it is unique as it is different for each clinical trial.

PySpark(Dataframe) implementation:

Outline:

In this I have selected the Id column and applied the distinct function to check the arguments with different values, then took the count of them using count function.

```
Cmd 13

clinical_trail.select(clinical_trail.Id).distinct().count()
```

HiveQL implementation:

Outline:

Selected Id column by distinct and counted them.

```
Cmd 13

select distinct count(Id) as distinct_count from clinicaltrail_2021;
```

PySpark(RDD) implementation:

Outline:

In this I have filtered the clinical_rdd based on Id column and used distinct function and counted them.

Cmd 9

```
clinical_rdd.filter(lambda x: x[0]).distinct().count()
```

AWS implementation commands:

Selected Id column by distinct and counted them as distinct_count.

Clinical_2021 × | mesh × | pharma × | Q1&2&3 × | Question 1 ×

```
1 SELECT distinct count(Id) as distinct_count FROM "clinicaltrail"."clinicaltrail_2021"
```

Result on the submission data:

Dataframe:

```
clinical_trail.select(clinical_trail.Id).distinct().count()
```

▸ (3) Spark Jobs

```
Out[16]: 387261
```

Command took 5.41 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 12:14:08 PM on Bdtt assignment

Hive:

Cmd 11

```
select distinct count(Id) as distinct_count from clinical_trail;
```

▸ (2) Spark Jobs

	distinct_count
1	387261

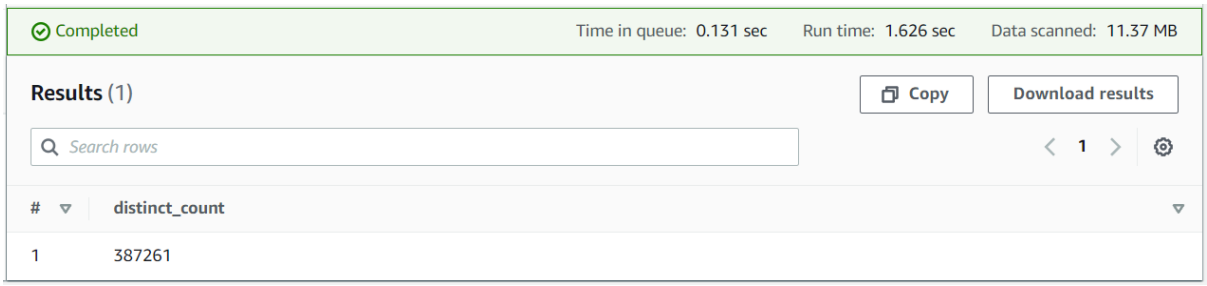
Showing all 1 rows.

Command took 2.15 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 12:29:27 PM on Bdtt assignment

RDD:

```
Cmd 9

clinical_rdd.filter(lambda x: x[0]).distinct().count()

▸ (1) Spark Jobs

Out[8]: 387261

Command took 5.23 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 12:20:38 PM on Bdtt assignment
```

AWS:



Discussion of result:

The total number of distinct studies in the dataset is 387261 and results are identical in all implementations.

Question 2:

Assumptions made:

Type column is assumed to be complete without inconsistences in data.

PySpark(Dataframe) implementation:

Outline:

Clinical_trail has been grouped with Type column, took the count of them to get the frequency which is sorted in descending.

```
Cmd 16

clinical_trail.groupBy('Type').count().orderBy('count', ascending=False).show()
```

HiveQL implementation:

Outline:

Selected the Type, count of Type as frequncy and grouped by Type, sorted by frequency in descending.

```
Cmd 15

select Type,count(Type) as frequency from clinicaltrail_2021 group by Type order by frequency desc
```

PySpark(RDD) implementation:

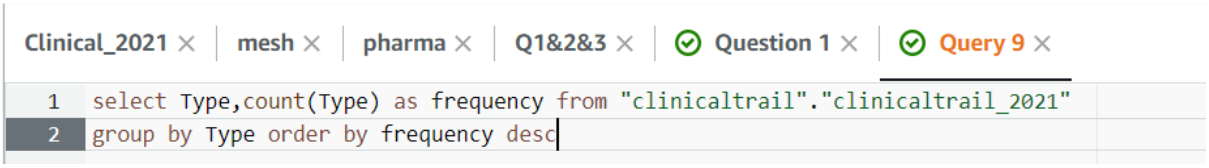
Outline:

Type column has been selected and converted it into a set form / pair rdd using the map function. Reducebykey function is used to merge the values with same key value and sortby is used to sort the data based on the column. Negative of column is used to sort in descending order.

```
Cmd 11

clinical_trail.map(lambda x: (x[5],1))\
              .reduceByKey(lambda x,y : x+y)\
              .sortBy(lambda x: -x[1]).collect()
```

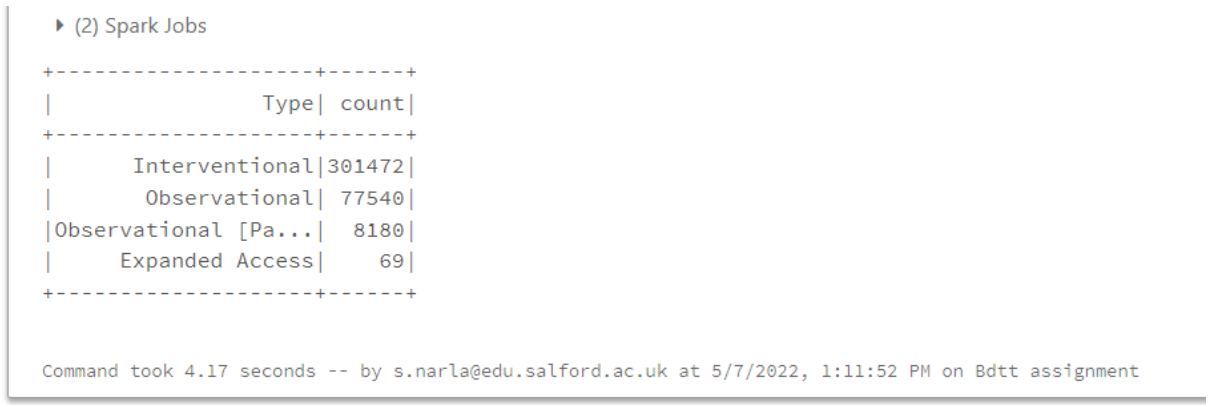
AWS implementation commands:

Same as HiveQl part as above.



Result on the submission data:

Dataframes:



Hive:

(2) Spark Jobs

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

Showing all 4 rows.

Command took 2.10 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:14:21 PM on Bdtt assignment

RDD:

(3) Spark Jobs

```
Out[9]: [('Interventional', 301472),
 ('Observational', 77540),
 ('Observational [Patient Registry]', 8180),
 ('Expanded Access', 69)]
```

Command took 3.27 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 1:15:39 PM on Bdtt assignment

AWS:

Completed | Time in queue: 0.232 sec | Run time: 1.869 sec | Data scanned: 11.37 MB

Results (4) | Copy | Download results

Search rows | 1

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

Discussion of result:

Interventional Type is the highest with 301472 clinical trails and Expanded Access Type is the least with 69 clinical trials. The sum of trails of all the other three types is very less than the highest type Interventional. Results are identical in all the implementations.

Question 3:

Assumptions made:

Conditions column has been assumed to be having no null values and data is completely accurate.

PySpark(Dataframe) implementation:

Outline:

Split_clinical_trail dataframe is obtained by removing the empty conditions using a where clause and then adding a column splitconditions by using the split function based on delimiter ‘,’.

Exploded_clinical_trail is formed by adding a column conditions by using the explode function based on the column splitconditions created in earlier step.

Conditions column is then grouped and counted to get the frequency and sorted in descending order

Cmd 18

```
split_clinical_trail = clinical_trail.where(clinical_trail.Conditions!= ' ')\
                       .withColumn('splitConditions', split(clinical_trail.Conditions,","))
exploded_clinical_trail = split_clinical_trail.withColumn('Conditions', explode(split_clinical_trail.splitConditions))
exploded_clinical_trail.groupBy('Conditions').count().orderBy('count', ascending=False).show(5)
```

HiveQL implementation:

Outline:

Sub query will choose the conditions_split, which is created by splitting the conditions with ',' and exploding them, resulting in a virtual table with one or more rows, which lateral view will apply to the output row. The conditions_split is chosen as conditions, counted as frequency, grouped by conditions, and arranged by frequency in descending order, with the values limited to 5.

```
select conditions_split as conditions,count(conditions_split) as frequency from (
      select conditions_split from clinicaltrail_2021 lateral view explode(split(conditions,','))
      conditions AS conditions_split)
      group by conditions_split order by frequency desc limit 5;
```

PySpark(RDD) implementation:

Outline:

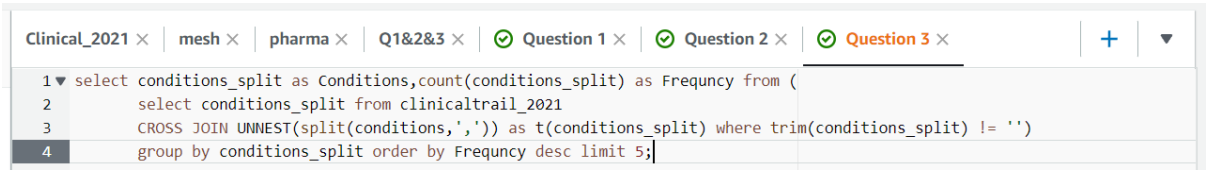
Clinical_trail has been mapped with Conditions column has been splitted based on ‘,’ and a filter is put up to the remove the empty values, then flat mapping the values, convert it into a set form or pair rdd form and using the reducebykey transformation to count the frequency with is sorted in the descending order, with values limiting to 5.

Cmd 13

```
clinical_trail.map(lambda x: x[7].split(","))\
              .filter(lambda x : x[0]!= '')\
              .flatMap(lambda x :x)\
              .map(lambda x: (x,1))\
              .reduceByKey(lambda x,y : x+y)\
              .sortBy(lambda x: -x[1]).take(5)
```

AWS implementation commands:

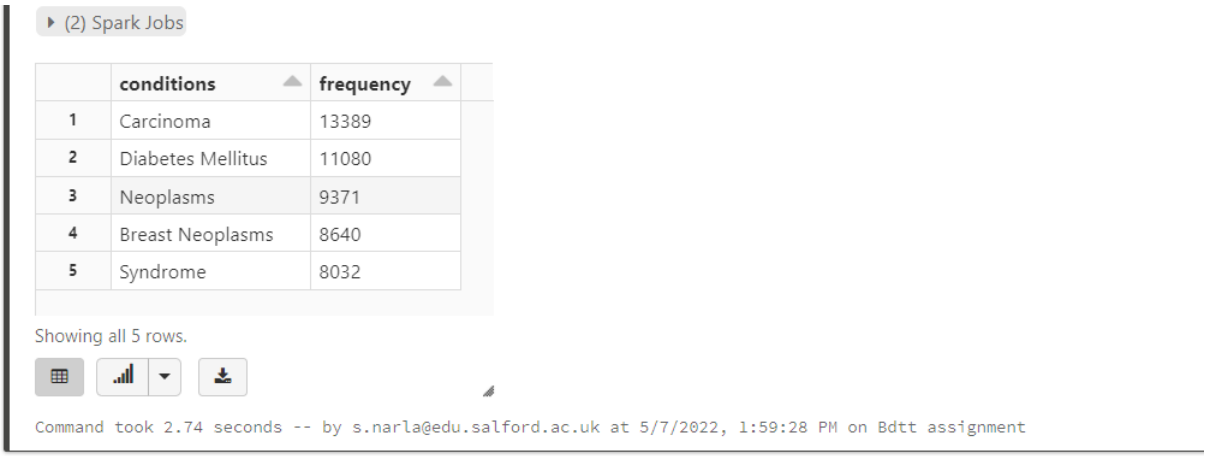
Same as HiveQl but used unnest function as explode function is not supported in Athena



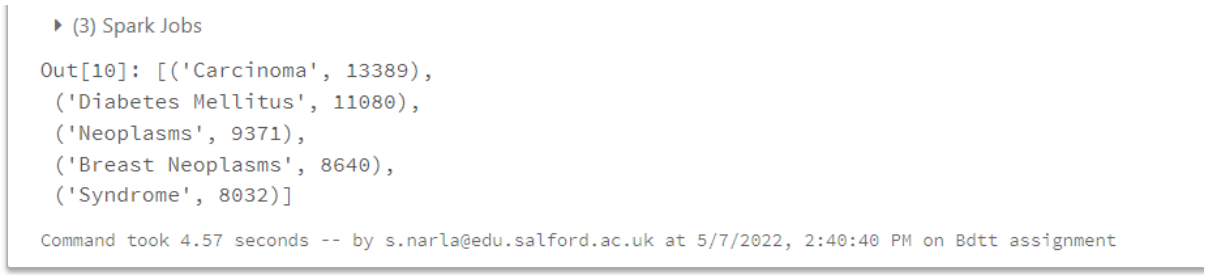
Result on the submission data:

Dataframe:

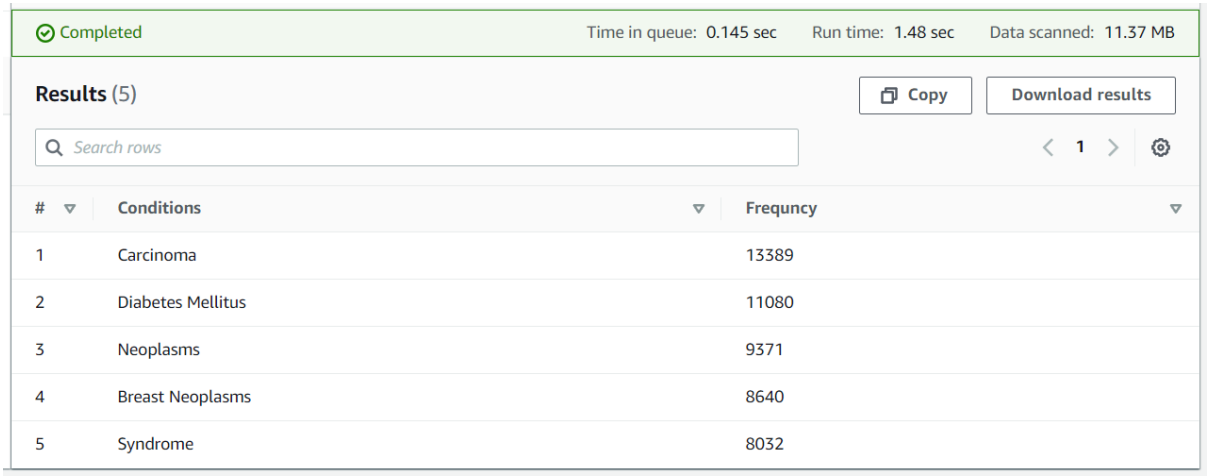
HiveQl:



RDD:



AWS:



Completed | Time in queue: 0.145 sec | Run time: 1.48 sec | Data scanned: 11.37 MB

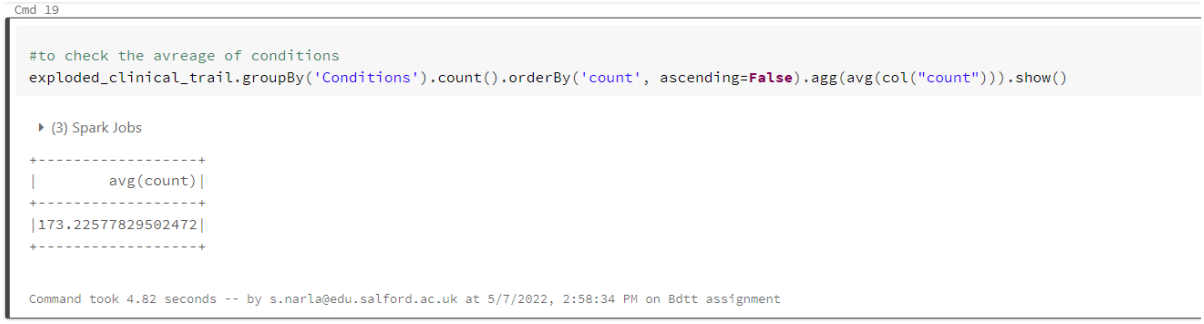
Results (5) — Copy | Download results

Search rows — 1

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

Discussion of result:

Carcinoma is the condition with highest frequency of 13389. Results obtained are same in all implementations.



Cmd 19

```
#to check the avreage of conditions
exploded_clinical_trail.groupBy('Conditions').count().orderBy('count', ascending=False).agg(avg(col("count"))).show()
```

▸ (3) Spark Jobs

```
+------------------+
|        avg(count)|
+------------------+
|173.22577829502472|
+------------------+
```

Command took 4.82 seconds -- by s.narla@edu.salford.ac.uk at 5/7/2022, 2:58:34 PM on Bdtt assignment

Average of frequency of all conditions is 173.22. Thus, frequency obtained is very high for top 5 conditions.

Question 4:

Assumptions made:

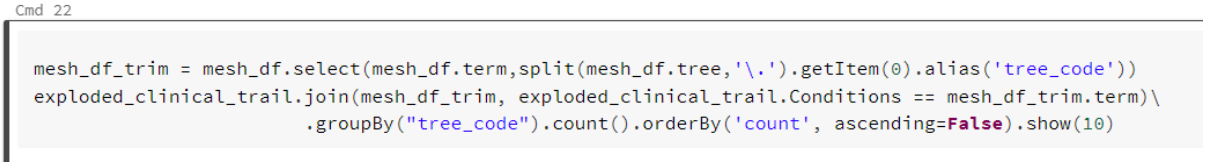
I have assumed that for each condition in clinical_trail data has the unique tree in the mesh data and the matching is upto data. I have used the exploded_clinical_trial from the Question 3.

PySpark(Dataframe) implementation:

Outline:

I have selected the term and splitted the tree based on '.'(escape character is used) to get the tree_code which is first 3 letters in dataframe mesh_df_trim.

Now I have joined with exploded_clinical_trial based on conditions and term, grouped by tree_code and counted to get the frequency which is sorted in descending order.



Cmd 22

```
mesh_df_trim = mesh_df.select(mesh_df.term,split(mesh_df.tree,'\.').getItem(0).alias('tree_code'))
exploded_clinical_trail.join(mesh_df_trim, exploded_clinical_trail.Conditions == mesh_df_trim.term)\
                       .groupBy("tree_code").count().orderBy('count', ascending=False).show(10)
```

HiveQL implementation:

Outline:

Selected the tree_code, counted it as frequency obtained using two subqueries and joined them. The first subquery is same as question3. In the second subquery, I have selected the term, tree_code which is split part of tree from mesh_data. Then it is grouped by tree_code and ordered by frequency in descending limiting the values to 10.

```
select tree_code,count(tree_code) as frequency from
       (select conditions_split from clinicaltrail_2021
        lateral view explode(split(conditions,',')) conditions AS conditions_split ) as clinc
        left outer join
        (select term,SPLIT(tree,'[\.]')[0] as tree_code from mesh) as mesh
        on clinc.conditions_split=mesh.term
        group by tree_code order by frequency desc limit 10;
```

PySpark(RDD) implementation:

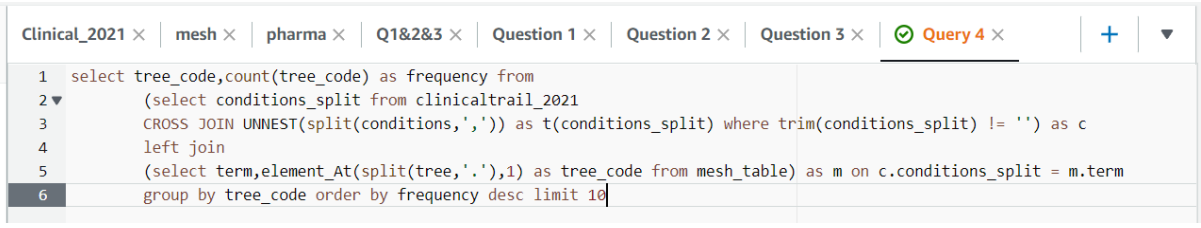
Outline:

Clinical_trail is mapped with split of conditions, filtered to remove empty values, flatmapping the values, converting it to pair rdd form which is joined with the mesh_rdd which is mapped with split based on ‘,’. Then it is mapped with the conditions, split of tree based on ‘.’ to get the code, again mapped to convert to set form. By using the reducebykey transformation frequency is obtained which is sorted in descending limiting to 10 values.

```
Cmd 17

clinical_trail.map(lambda x: x[7].split(","))\
              .filter(lambda x : x[0]!= '')\
              .flatMap(lambda x :x)\
              .map(lambda x: (x,1))\
              .join(mesh_rdd.map(lambda x: x.split(',')))\
              .map(lambda x: (x[0],x[1].split('.')[0]))\
              .map(lambda x: (x[1][1],x[1][0]))\
              .reduceByKey(lambda x,y : x+y)\
              .sortBy(lambda x: -x[1]).take(10)
```

AWS implementation commands:

This is similar to HiveQL but used unnest instead of explode, left join instead of left outer join as Athena is not supported.



Result on the submission data:

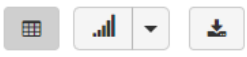
Dataframes:

```
+---------+------+
|tree_code| count|
+---------+------+
|      C04|143994|
|      C23|136079|
|      C01|106674|
|      C14| 94523|
|      C10| 92310|
|      C06| 85646|
|      C08| 70720|
|      C13| 42599|
|      C18| 41276|
|      C12| 40161|
+---------+------+
only showing top 10 rows


Command took 6.64 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 11:45:33 AM on Bdtt assignment
```

Hive:

	tree_code	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 3.77 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 12:12:48 PM on Bdtt assignment

RDD:

▸ (3) Spark Jobs

```
Out[13]: [('C04', 143994),
 ('C23', 136079),
 ('C01', 106674),
 ('C14', 94523),
 ('C10', 92310),
 ('C06', 85646),
 ('C08', 70720),
 ('C13', 42599),
 ('C18', 41276),
 ('C12', 40161)]
```

Command took 8.69 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 12:15:53 PM on Bdtt assignment

AWS:

Completed | Time in queue: 0.19 sec | Run time: 2.355 sec | Data scanned: 16.42 MB

Results (10) | Copy | Download results

Search rows | 1

#	tree_code	frequency
1	C04	143994
2	C23	136079
3	C01	106674
4	C14	94523
5	C10	92310
6	C06	85646
7	C08	70720
8	C13	42599
9	C18	41276
10	C12	40161

Discussion of result:

Results obtained are same for all the implementations. Code C04 has the highest frequency with 143994 conditions. I tried to investigate the unique conditions which has the highest frequency code. There are total 382 unique conditions with code C04.

Cmd 23

```
#to find out the unique conditions for the top most tree code C04
exploded_clinical_trail.join(mesh_df_trim.where(mesh_df_trim.tree_code == 'C04'), exploded_clinical_trail.Conditions == mesh_df_trim.term)\
                       .select(exploded_clinical_trail.Conditions).distinct().display()
```

(2) Spark Jobs

	Conditions
1	Adenocarcinoma of Lung
2	Stomach Neoplasms
3	Lymphomatoid Granulomatosis
4	Hereditary Breast and Ovarian Cancer Syndrome
5	Adrenal Gland Neoplasms
6	Bowens Disease
7	Pancreatic Neoplasms

Showing all 382 rows.

Command took 6.28 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 12:32:15 PM on Bdtt assignment

Question 5:

Assumptions made:

Parent_Company column has all the pharmaceutical companies has been assumed.

PySpark(Dataframe) implementation:

Outline:

Clinical_trail has been left anti joined with pharma_df based on sponsor and parent_company and grouped by sponsor, counted and sorted in descending order limiting values to 10.

```
clinical_trail.join(pharma_df, clinical_trail.Sponsor == pharma_df.Parent_Company, "leftanti")\
              .groupby("Sponsor").count().orderBy('count', ascending=False).show(10)
```

HiveQL implementation:

Outline:

I have used two types of queries in Hive. One is Subquery and other is joins.

Selected the sponsor, counted as frequency from clinical_trail by applying where condition in which sponsor not in Selected the parent_company in subquery and grouped the whole by sponsor and ordered by frequency in descending.

```
select sponsor, count(sponsor) as frequency
      from clinicaltrail_2021 where sponsor not in (select parent_company from pharma)
      group by sponsor order by frequency desc limit 10;
```

Selected the sponsor, counted as frequency from clinical_trail left anti joined with pharma based on sponsor and parent company which is grouped by sponsor and ordered by frequency in descending.

```
select sponsor, count(sponsor) as frequency
      from clinicaltrail_2021 left anti join pharma
      on clinicaltrail_2021.sponsor = pharma.parent_company
      group by sponsor order by frequency desc limit 10;
```

Advantage of join is it executes faster but subqueries are easy to read. Joins takes more time in retrieving the data from all the tables and execute faster. Subqueries will decrease the complexity in code and easy to read.

PySpark(RDD) implementation:

Outline:

Parent_Company list has all the parent company names obtained by mapping with pharma_rdd by splitting and mapping with the required index.

Clinical_trail is mapped by changing sponsor to pair rdd and reducebykey transformation is used to find frequency which is sorted in descending then filter applied based on sponsor not in parent_company.

Cmd 19

```
parent_company = pharma_rdd.map(lambda x: x.split("\"")).map(lambda x: x[3]).collect()
clinical_trail.map(lambda s: (s[1],1))\
              .reduceByKey(lambda x,y : x+y)\
              .sortBy(lambda x: -x[1])\
              .filter(lambda x: x[0] not in parent_company).take(10)
```

AWS implementation commands:

It is same as HiveQl implementation using the subquery.

Clinical_2021 | mesh | pharma | Question 1 | Question 2 | Question 3 | Query 4 | Question 5

```
1  select Sponsor,count(Sponsor) as count from "clinicaltrail"."clinicaltrail_2021"
2         where Sponsor not in (select REPLACE(parent_company,'"','') as parent from "clinicaltrail"."pharma_table")
3         group by Sponsor order by count desc limit 10
```

Result on the submission data:

Dataframe:

(2) Spark Jobs

```
+--------------------+-----+
|             Sponsor|count|
+--------------------+-----+
|National Cancer I...| 3218|
|M.D. Anderson Can...| 2414|
|Assistance Publiq...| 2369|
|         Mayo Clinic| 2300|
|Merck Sharp & Doh...| 2243|
|   Assiut University| 2154|
|Novartis Pharmace...| 2088|
|Massachusetts Gen...| 1971|
|    Cairo University| 1928|
|   Hoffmann-La Roche| 1828|
+--------------------+-----+
only showing top 10 rows


Command took 5.22 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 1:27:53 PM on Bdtt assignment
```

HiveQL:

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

Showing all 10 rows.

Command took 2.52 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 12:47:10 PM on Bdtt assignment

	sponsor	frequency
4	Mayo Clinic	2300
5	Merck Sharp & Dohme Corp.	2243
6	Assiut University	2154
7	Novartis Pharmaceuticals	2088
8	Massachusetts General Hospital	1971
9	Cairo University	1928
10	Hoffmann-La Roche	1828

Showing all 10 rows.

Command took 2.52 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 12:47:10 PM on Bdtt assignment

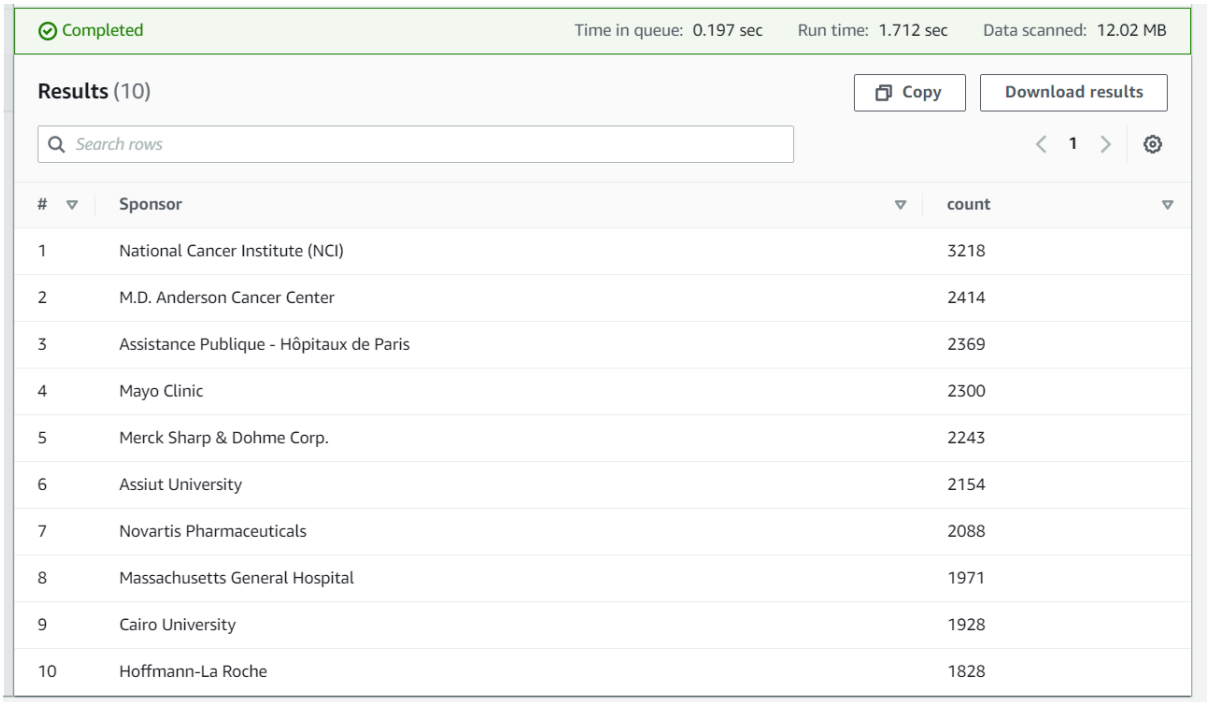
RDD:

▸ (4) Spark Jobs

```
Out[22]: [('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 3.39 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 1:19:32 PM on Bdtt assignment

AWS:



Completed | Time in queue: 0.197 sec | Run time: 1.712 sec | Data scanned: 12.02 MB

Results (10) — Copy | Download results

Search rows — 1

#	Sponsor	count
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
8	Massachusetts General Hospital	1971
9	Cairo University	1928
10	Hoffmann-La Roche	1828

Discussion of result:

Results obtained in all the implementations are same. National Cancer Institute has sponsored 3218 clinical trials in non-pharmaceutical companies.

Question 6:

Assumptions made:

Status is assumed to be completed and completion date column is assumed to be having accurate data.

PySpark(Dataframe) implementation:

Outline:

Clinical_trail is filtered with status as completed, added columns for completion_ year and completion_month by split on ‘ ‘ completion.It has been filtered with year 2021 and grouped by completion_montn and counted.

To Sort the data based on calendar month, I have used function from_unixtime and added month number based on calendar month and sorted it based on month number to get frequency per month.

Cmd 27

```
clinical_trail_filter = clinical_trail.filter("Status == 'Completed'")\
                                      .withColumn('Completion_year',f.split(clinical_trail['Completion'], ' ').getItem(1))\
                                      .withColumn('Completion_month',f.split(clinical_trail['Completion'], ' ').getItem(0))\
                                      .filter("Completion_year == 2021").groupBy('Completion_month').count()
clinical_trail_sort = clinical_trail_filter.withColumn("mnth_number",from_unixtime(unix_timestamp(clinical_trail_filter["Completion_month"],'MMM'),'MM'))\
                                           .orderBy("mnth_number").select("Completion_month","count")
clinical_trail_sort.show()
```

HiveQL implementation:

Outline:

Selected completed_month and counted it by split of completion column on ‘ ‘ from clinical_trail applied where condition for status which is grouped by completed_month, used having condition to check year 2021 which is sorted based on calendar month number by using from_unixtime function.

```
select split(Completion," ")[0] as completed_month, count(split(Completion," ")[0]) as frequency
       from clinicaltrail_2021
       where Status == "Completed"
       group by completed_month,split(Completion," ")[1]
       having split(Completion," ")[1] == "2021"
       order by from_unixtime(to_unix_timestamp(completed_month,'MMM'),'MM')
```

PySpark(RDD) implementation:

Outline:

I have created a month_rdd with calendar_month and their numbers in it to sort my final RDD

```
Cmd 21

month_list = [('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_rdd = sc.parallelize(month_list)

Command took 0.04 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 3:19:08 PM on Bdtt assignment
```

Clinical_trail has been filtered with status completed and mapped by completion split on ‘ ‘ which is filtered to remove empty values and checked for year 2021 by converting it to a paird rdd form. Reducebykey transformation is used to find frequency which is joined by month_rdd created above and sorted it based on calendar months and mapped the required columns to print.

```
clinical_trail_completed = clinical_trail.filter(lambda s: s[2] == 'Completed')\
                                         .map(lambda x : x[4].split(' '))
clinical_trail_completed.filter(lambda x : x[0] != '')\
                        .filter(lambda x: (x[1] == '2021'))\
                        .map(lambda x: (x[0],1))\
                        .reduceByKey(lambda x,y : x+y)\
                        .join(month_rdd)\
                        .sortBy(lambda x: x[1][1])\
                        .map(lambda x: (x[0],x[1][0])).collect()
```

AWS implementation commands:

This is done like hive ql. I have used function date_parse in order to sort the column based on calendar month. I have added 01 to the completion and converted it into a upper case without any spaces to match the format of the function date_parse.

Question 6 | Question 4 | Question 5

```
select element_at(split(completion,' '),1) as completed_month,count(element_at(split(Completion,' '),1)) as count
from clinicaltrail_2021
where status='Completed'
group by element_at(split(completion,' '),1),element_at(split(Completion,' '),2),concat('01 ',completion)
having element_at(split(Completion,' '),2) = '2021'
order by date_parse(upper(replace(concat('01 ',completion),' ','')),'%d%b%Y')
```

Result on the submission data:

Dataframe:

```
+----------------+-----+
|Completion_month|count|
+----------------+-----+
|             Jan| 1131|
|             Feb|  934|
|             Mar| 1227|
|             Apr|  967|
|             May|  984|
|             Jun| 1094|
|             Jul|  819|
|             Aug|  700|
|             Sep|  528|
|             Oct|  187|
+----------------+-----+
```

1

Command took 3.27 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 2:55:28 PM on Bdtt assignment

Hive:

(2) Spark Jobs

	completed_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 10 rows.

1

Command took 2.94 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 3:14:19 PM on Bdtt assignment

▸ (2) Spark Jobs

	completed_month	frequency
4	Apr	967
5	May	984
6	Jun	1094
7	Jul	819
8	Aug	700
9	Sep	528
10	Oct	187

Showing all 10 rows.

Command took 2.94 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 3:14:19 PM on Bdtt assignment

RDD:

```
Out[24]: [('Jan', 1131),
 ('Feb', 934),
 ('Mar', 1227),
 ('Apr', 967),
 ('May', 984),
 ('Jun', 1094),
 ('Jul', 819),
 ('Aug', 700),
 ('Sep', 528),
 ('Oct', 187)]
```

Command took 4.33 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 3:19:10 PM on Bdtt assignment

AWS:

Completed | Time in queue: 0.125 sec | Run time: 1.193 sec | Data scanned: 11.37 MB

Results (10) | Copy | Download results

Search rows | 1

#	completed_month	count
1	Jan	1131
2	Feb	934
3	Mar	1227
4	Apr	967
5	May	984
6	Jun	1094
7	Jul	819
8	Aug	700
9	Sep	528
10	Oct	187

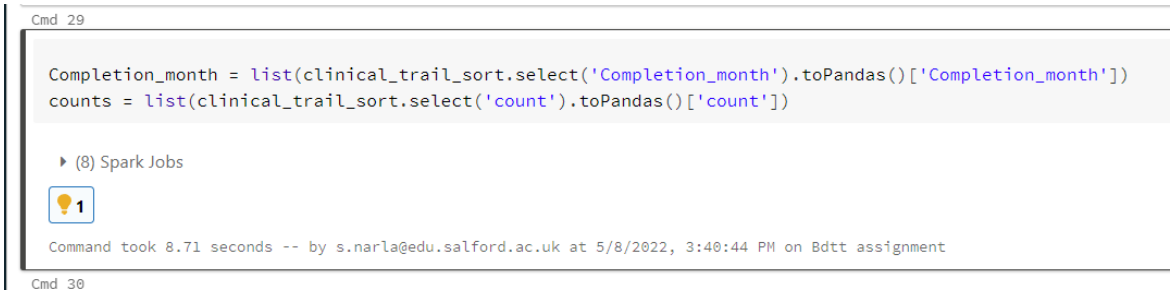
Discussion of result:

Result obtained is same in all implementations. In March month, highest number of clinical trials are completed and in October month least number of clinical trails have been completed.

Visualisation for Question 6:

I have done visualisation using bokeh, matplotlib, databricks.

Created two lists from the Question6 dataframe for plotting. These are used in visualisation.



Using Bokeh:

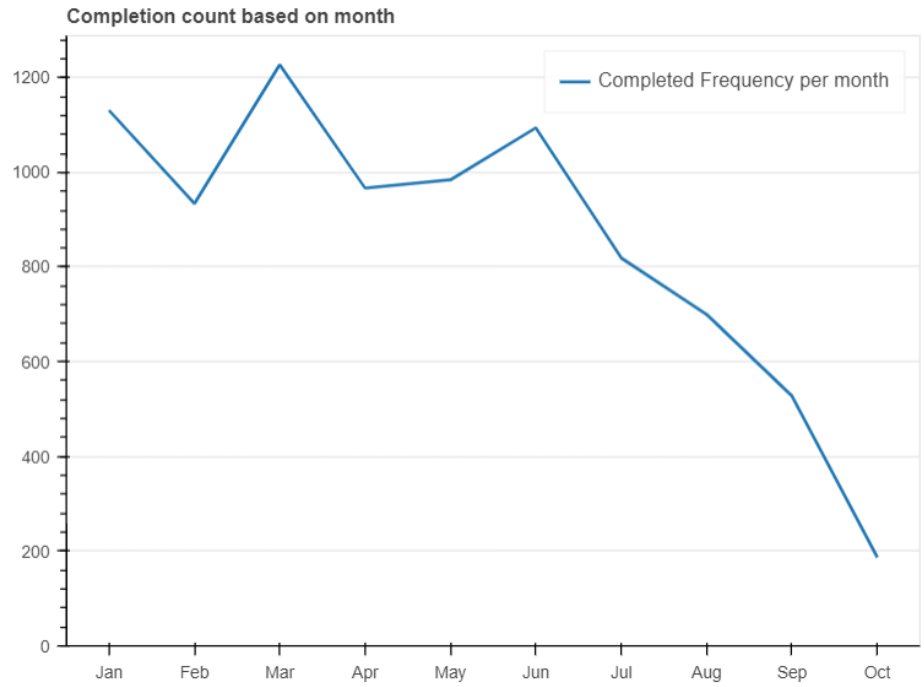
Functions required are imported initially. line plot has been plotted by giving required parameters.

Code:

```
Cmd 30
```

```
output_file("completion_line.html")
p = figure(x_range=Completion_month, height=450, title="Completion count based on month",
           toolbar_location=None, tools="")
p.line(Completion_month,counts,legend_label = 'Completed Frequency per month', width=2)
p.xgrid.grid_line_color = None
p.y_range.start = 0
html=file_html(p,CDN,"plot")
displayHTML(html)
```

Output:



Command took 0.09 seconds -- by s.narla@edu.salford.ac.uk at 5/8/2022, 3:44:31 PM on Bdtt assignment

Using Matplotlib:

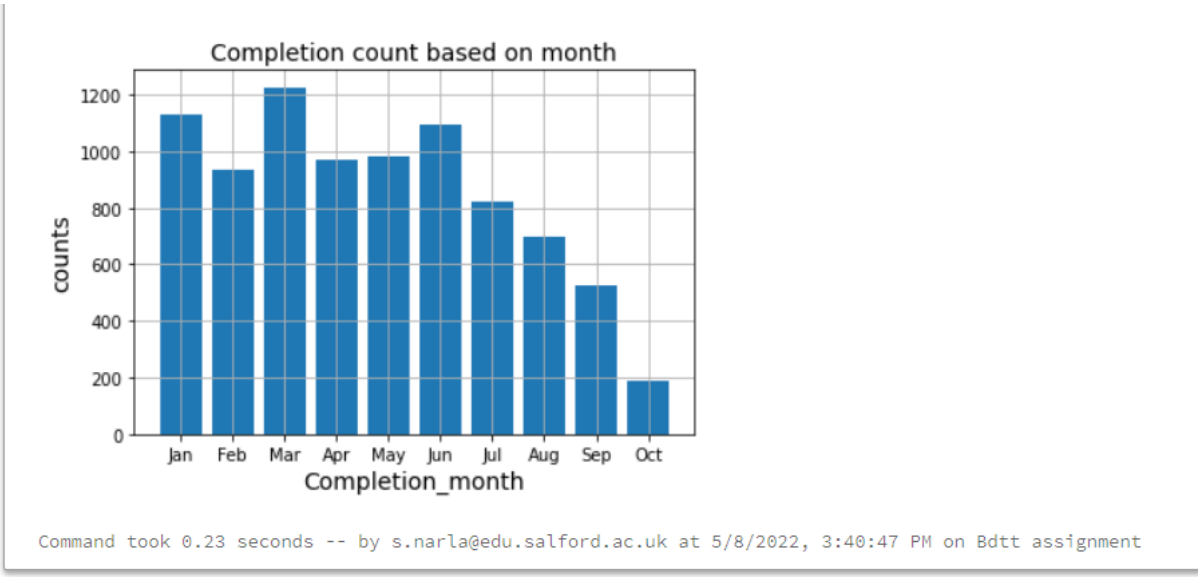
Bar graph is plotted based on input parameters.

Code:

Cmd 32

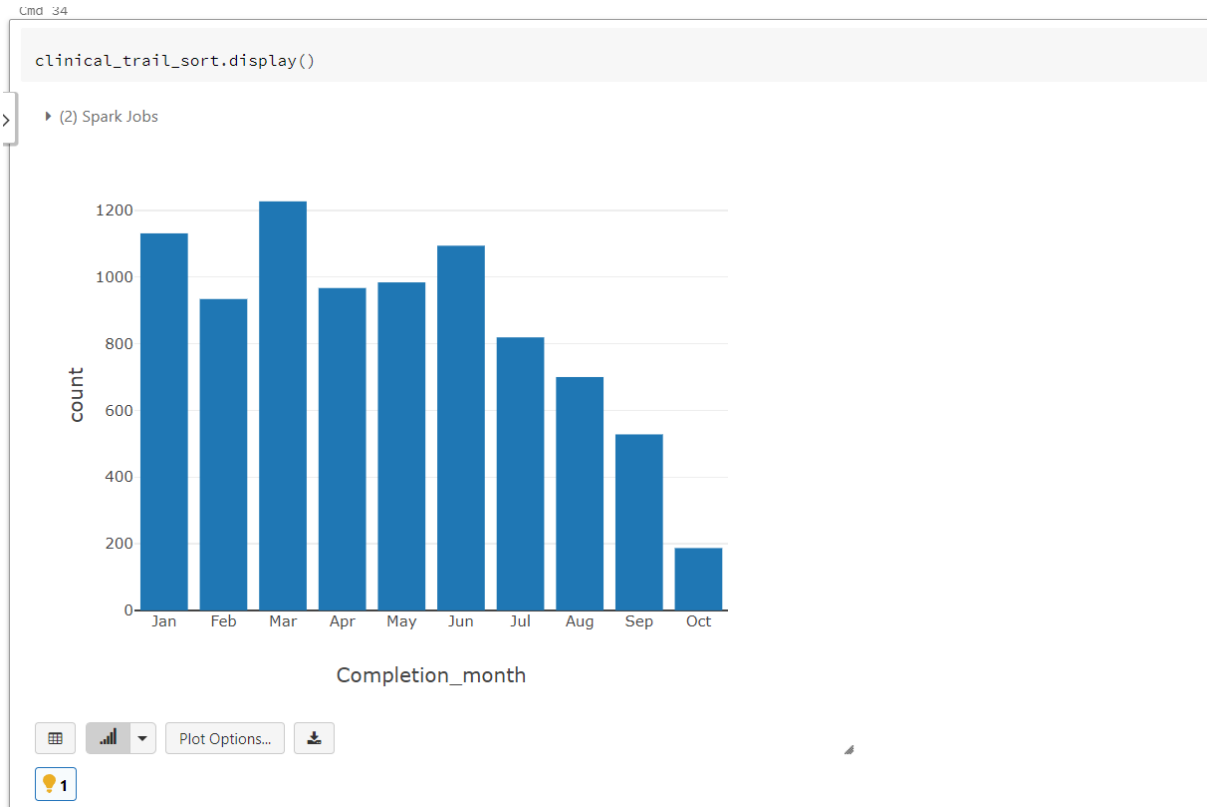
```
plt.bar(Completion_month, counts)
plt.title('Completion count based on month', fontsize=14)
plt.xlabel('Completion_month', fontsize=14)
plt.ylabel('counts', fontsize=14)
plt.grid(True)
plt.show()
```

Output:



Using Databricks:

By using the view below the result, this graph is obtained.



Further Analysis 1:

To find the percentage of clinical trials based on status.

Assumptions made:

Assuming data in the status column is upto date.

PySpark(Dataframe) implementation:

Outline:

Clinical_percent is formed by grouping status in Clinical_trail, counted and percent column is added by calculating the count of status by total sum of count. Window function has been used here with over clause. This is ordered by percent in descending order and required columns are selected. Using a round function percentage column has been rounded of to 3 digit.

```
Cmd 36

clinical_percent = clinical_trail.groupBy('Status').count()\
                                 .withColumn('percent', f.col('count')*100.00/f.sum('count').over(Window.partitionBy()))
clinical_percent.orderBy('percent', ascending=False)\
                .select(clinical_percent.Status,f.col('count'),f.round(clinical_percent.percent,3).alias('percentage of clinical trials')).show()
```

Hive Implementation:

Outline:

Selected Status, counted as frequency, find percentage and rounded it by using the over clause and grouped by status and ordered by frequency in descending order.

```
select Status,count(Status) as frequency,round(count(Status)*100/SUM(COUNT(Status)) OVER(),3) AS percentage_of_clinical_trails
             from clinicaltrail_2021 group by Status order by frequency desc;
```

Result on the submission data:

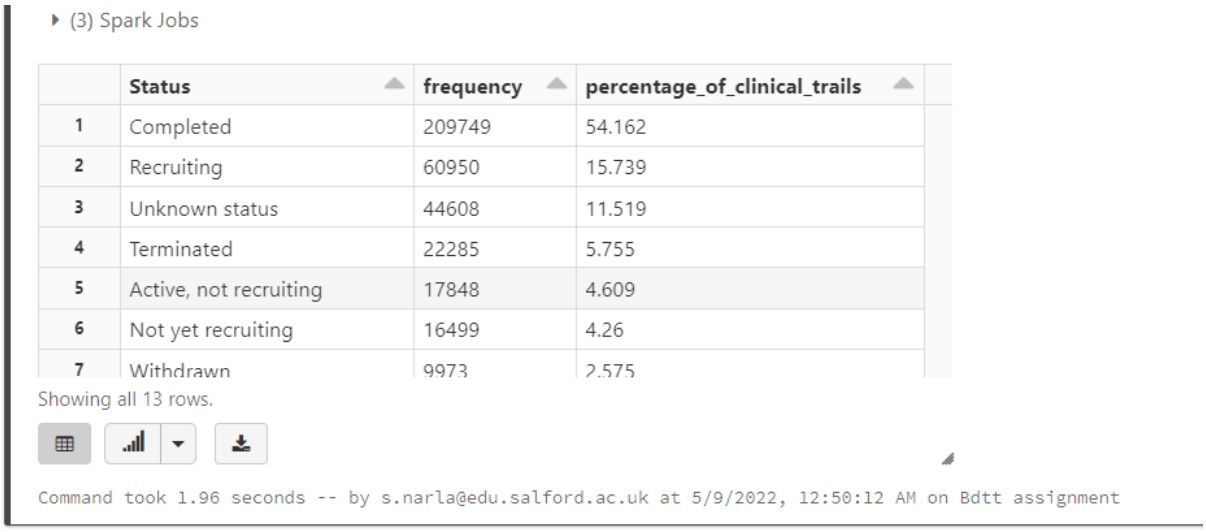
Dataframe:

▸ (3) Spark Jobs

Status	count	percentage of clinical trials
Completed	209749	54.162
Recruiting	60950	15.739
Unknown status	44608	11.519
Terminated	22285	5.755
Active, not recru...	17848	4.609
Not yet recruiting	16499	4.26
Withdrawn	9973	2.575
Enrolling by invi...	3682	0.951
Suspended	1598	0.413
No longer available	39	0.01
Approved for mark...	24	0.006
Available	5	0.001
Temporarily not a...	1	0.0

Command took 3.25 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 12:26:23 AM on Bdtt assignment

Hive:

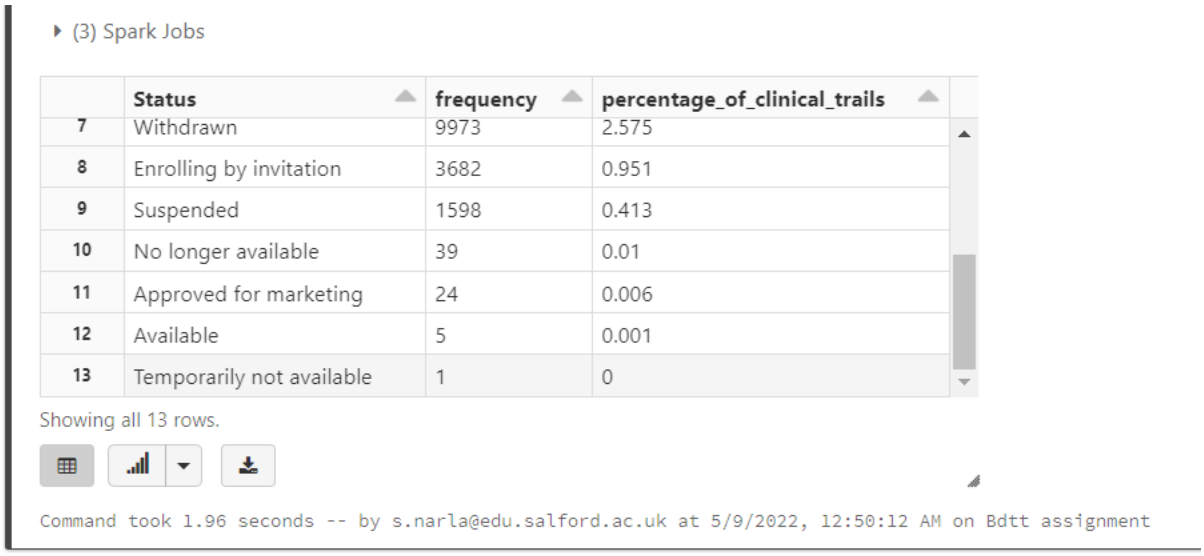


(3) Spark Jobs

	Status	frequency	percentage_of_clinical_trails
1	Completed	209749	54.162
2	Recruiting	60950	15.739
3	Unknown status	44608	11.519
4	Terminated	22285	5.755
5	Active, not recruiting	17848	4.609
6	Not yet recruiting	16499	4.26
7	Withdrawn	9973	2.575

Showing all 13 rows.

Command took 1.96 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 12:50:12 AM on Bdtt assignment



(3) Spark Jobs

	Status	frequency	percentage_of_clinical_trails
7	Withdrawn	9973	2.575
8	Enrolling by invitation	3682	0.951
9	Suspended	1598	0.413
10	No longer available	39	0.01
11	Approved for marketing	24	0.006
12	Available	5	0.001
13	Temporarily not available	1	0

Showing all 13 rows.

Command took 1.96 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 12:50:12 AM on Bdtt assignment

Discussion of result:

Result obtained is same in all implementations. Total 54.16% of the clinical trials have been completed and a significant amount of 11.4% are in unknown status. The withdrawn percent of clinical trials is 2.575%.

Further Analysis 2:

To find the number of months for completion.

Assumptions made:

Start and Completion date columns are accurate.

PySpark(Dataframe) implementation:

Outline:

I have added 4 columns based on split of two columns completion, start with each year and month separately, calendar month number is calculated using unix_timestamp. Thus the difference between them is calculated and selected the required columns.

```
Cmd 38

clinical_trail_filter = clinical_trail.filter("Status == 'Completed'")\
                                      .withColumn('Completion_year',f.split(clinical_trail.Completion, ' ').getItem(1))\
                                      .withColumn('Completion_month',f.split(clinical_trail.Completion, ' ').getItem(0))\
                                      .withColumn('Start_year',f.split(clinical_trail.Start, ' ').getItem(1))\
                                      .withColumn('Start_month',f.split(clinical_trail.Start, ' ').getItem(0))
clinical_trail_filter = clinical_trail_filter.withColumn("comp_mnth_number",from_unixtime(unix_timestamp(clinical_trail_filter.Completion_month,'MMM'),'MM'))\
                                             .withColumn("start_mnth_number",from_unixtime(unix_timestamp(clinical_trail_filter.Start_month,'MMM'),'MM'))
clinical_trail_filter = clinical_trail_filter.withColumn('Number_of_Months_for_Completion',((clinical_trail_filter.Completion_year - clinical_trail_filter.Start_year)*12-
                                             clinical_trail_filter.start_mnth_number+clinical_trail_filter.comp_mnth_number).cast('int'))
clinical_trail_filter.select(clinical_trail_filter.Start_year,clinical_trail_filter.start_mnth_number,clinical_trail_filter.Completion_year,clinical_trail_filter.comp_mnth_number,c
linical_trail_filter.Number_of_Months_for_Completion).display()
```

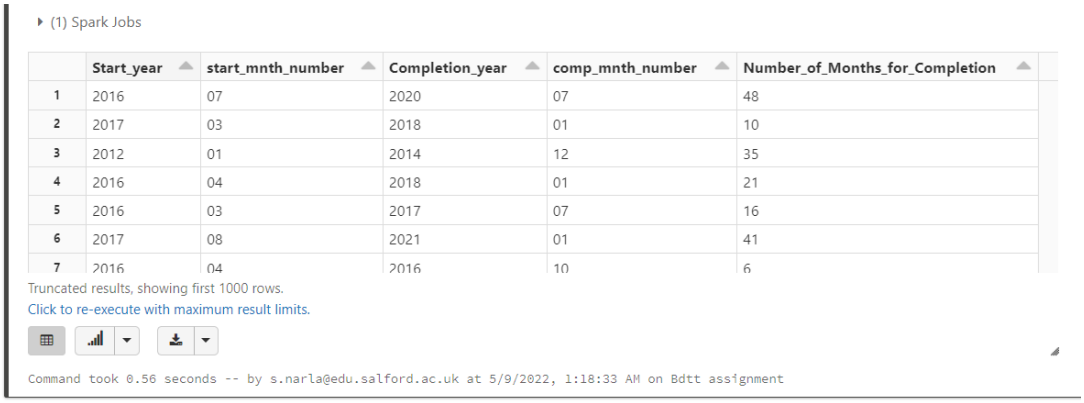
HiveQL implementation:

Selected year and month separately based on completion and Start columns split. Using from_unixtime function month number is derived. Calculated the number of months by simple formula based on above columns.

```
select split(Completion," ")[1] as completed_year,split(Start," ")[1] as start_year,
        from_unixtime(to_unix_timestamp(split(Start," ")[0],'MMM'),'MM') as start_month,
        from_unixtime(to_unix_timestamp(split(Completion," ")[0],'MMM'),'MM') as completed_month,
        (split(Completion," ")[1] - split(Start," ")[1])*12-
        from_unixtime(to_unix_timestamp(split(Start," ")[0],'MMM'),'MM') +
        from_unixtime(to_unix_timestamp(split(Completion," ")[0],'MMM'),'MM') as number_of_months
       from clinicaltrail_2021 where Status = 'Completed'
```

Result on the submission data:

Dataframe:



(1) Spark Jobs

	Start_year	start_mnth_number	Completion_year	comp_mnth_number	Number_of_Months_for_Completion
1	2016	07	2020	07	48
2	2017	03	2018	01	10
3	2012	01	2014	12	35
4	2016	04	2018	01	21
5	2016	03	2017	07	16
6	2017	08	2021	01	41
7	2016	04	2016	10	6

Truncated results, showing first 1000 rows.

Click to re-execute with maximum result limits.

Command took 0.56 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 1:18:33 AM on Bdtt assignment

Hive:

(1) Spark Jobs

	completed_year	start_year	start_month	completed_month	number_of_months
1	2020	2016	07	07	48
2	2018	2017	03	01	10
3	2014	2012	01	12	35
4	2018	2016	04	01	21
5	2017	2016	03	07	16
6	2021	2017	08	01	41
7	2016	2016	04	10	6

Truncated results, showing first 1000 rows.

Click to re-execute with maximum result limits.

Command took 0.41 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 1:35:31 AM on Bdtt assignment

Discussion of result:

Result obtained is same in all implementations.

Further Analysis 3:

To find the average number of months completed based on Type.

Dataframes:

Clinical_trial_filter has been grouped by Type and aggregate function is used to calculate average.

```
Cmd 40

clinical_trail_filter.groupby('Type').agg(avg(col("Number_of_Months_for_Completion")).cast('float').alias('Avg time for completion')).show()
```

HiveQL:

Selected the type and calculated the average based on number of months calculated similar to earlier question which is rounded to 3 digits where status is completed which is grouped by Type.

```
select Type,round(avg((split(Completion," ")[1] - split(Start," ")[1])*12-
        from_unixtime(to_unix_timestamp(split(Start," ")[0],'MMM'),'MM') +
        from_unixtime(to_unix_timestamp(split(Completion," ")[0],'MMM'),'MM')),3) as completed_month
       from clinicaltrail_2021 where Status = 'Completed' group by Type
```

Result:

Dataframe:

```
▸ (2) Spark Jobs
+--------------------+-----------------------+
|                Type|Avg time for completion|
+--------------------+-----------------------+
|Observational [Pa...|              31.887775|
|      Interventional|              30.046839|
|       Observational|               35.94214|
+--------------------+-----------------------+

Command took 5.37 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 1:53:58 AM on Bdtt assignment
```

Hive:

▸ (2) Spark Jobs

	Type	completed_month
1	Observational [Patient Registry]	31.888
2	Interventional	30.047
3	Observational	35.942

Showing all 3 rows.

Command took 3.84 seconds -- by s.narla@edu.salford.ac.uk at 5/9/2022, 1:34:20 AM on Bdtt assignment

Discussion of Result:

Interesting aspect found here is that Type Expanded access doesn't have any completed clinical trials. Observational took 35.9 months on average to complete. To validate this, I have done the following analysis.

Cmd 31

```
/*To validate that Type Expnaded access doesnt have any status Completed*/
Select Type,Count(Type) as frequency from clinicaltrail_2021 where Status = 'Completed' group by Type
```

Thus, it is confirmed that Expanded access donot have any clinical trials that are completed.