Big Data Tools and Techniques Coursework

Analysis of clinical trial datasets using Databricks and AWS platforms

Kamal Kiani
Student ID: @00588509

Contents

1. Platforms and required setup

1.1. Databricks – PySpark setup

PySpark is a Python API for Apache Spark, which is a distributed computing framework and open source. It has a set of libraries for processing large-scale data in real-time. In this coursework the community version of databricks platform is used to analyze the dataset and solve the required problems. The specifications of the image that is used to create the cluster is:

- databricks runtime version 10.4 LTS
- Scala 2.12
- Spark 3.2.1

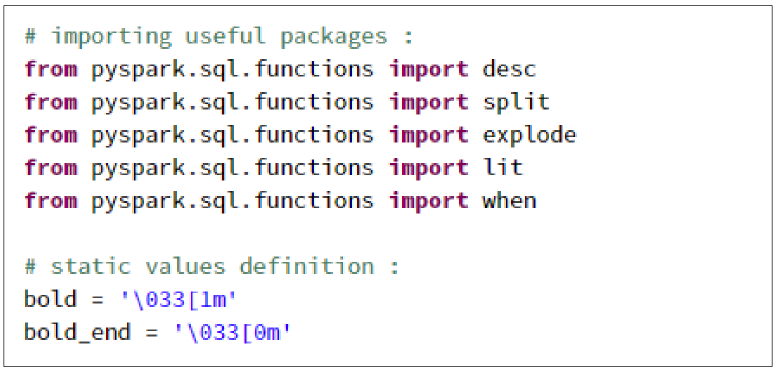
Using the free instance of databricks cluster and as a community edition user, we will have 15 GB Memory which the cluster will automatically terminate after an idle period of two hours.

The default environment variable for PySpark is set as below, and we will not change this value as the main path to python bin directory:

PYSPARK_PYTHON=/databricks/python3/bin/python3

In the next step of preparing the environment, we create the notebook and attach the cluster created in the previous step to this notebook. We must make sure that the language of the notebook is set to Python in case we want to solve the problems with PySpark tools.

Before starting to use the PySpark tools, it is necessary to import some useful packages. In this study we will use 5 packages which they are indicated in figure1.1. As it is shown in figure1.1, two global and static values are defined to enhance the coding.



```
# importing useful packages :
from pyspark.sql.functions import desc
from pyspark.sql.functions import split
from pyspark.sql.functions import explode
from pyspark.sql.functions import lit
from pyspark.sql.functions import when

# static values definition :
bold = '\033[1m'
bold_end = '\033[0m'
```

Figure1.1

1.2. Databricks – HiveQL setup

Hadoop is an open-source framework to store and process Big Data. It contains two main modules, one is HDFS (Hadoop Distributed File System) and the other one is MapReduce. On the other hand, we have "Hive" which resides on top of Hadoop to make querying and analyzing of big data. In fact, Hive is a platform used to develop

SQL type scripts to do MapReduce operations. The following component diagram figure1.2, illustrates the architecture of Hive.

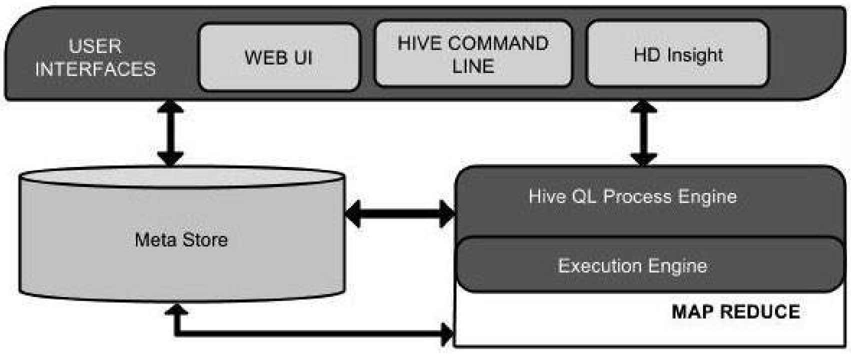
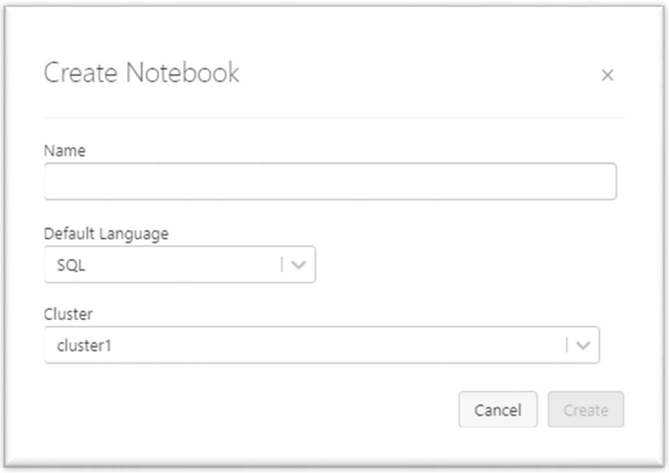


Figure1.2

The cluster creation and setup are the same as cluster creation explained in the previous section1.1. But we have to make sure the default language while creating a notebook is set "SQL" as it is shown in figure1.3.


Create Notebook

Name

Default Language: SQL

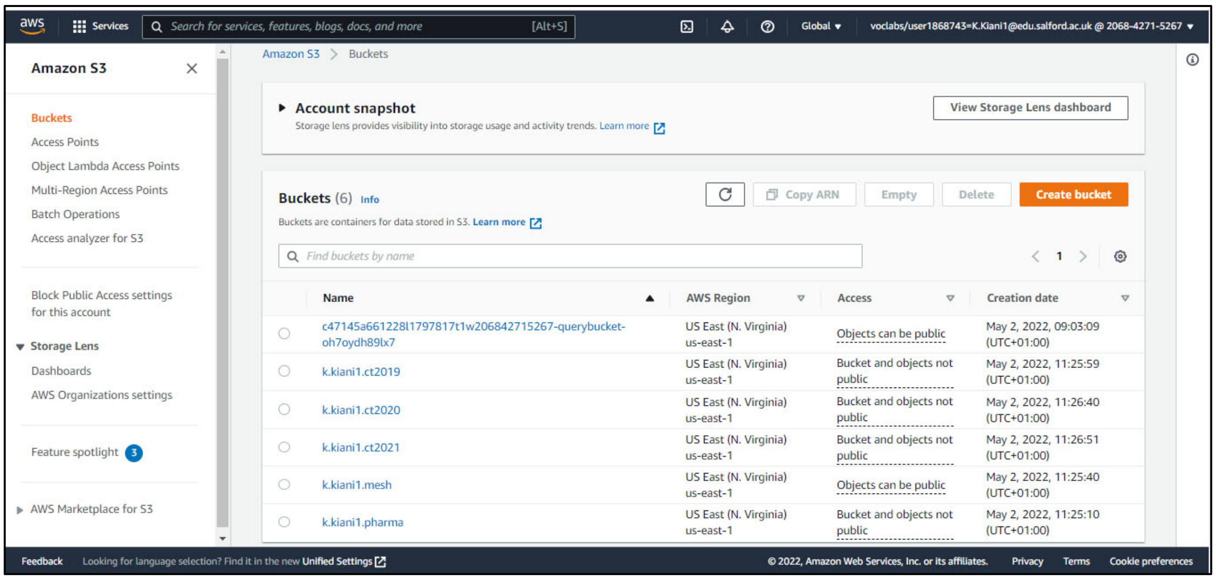
Cluster: cluster1

Figure1.3

1.3. AWS – Athena Glue setup

Amazon Web Services (AWS) provides us an integrated portfolio of cloud computing services to build, secure, and deploy big data applications. The first step for all big data solutions is storing data. Amazon Simple Storage Service (Amazon S3) is one of the most commonly used services for storing data and we will use this service for our application.

First, we create an IAM user account and add an "awsuser" to the "awsusers group". Then, we load data into Amazon S3 by creating S3 buckets and uploading the required objects into these buckets. As in the next steps we are going to use AWS Glue and define a crawler to create required tables, the objects must not be placed beside each other. So, there are 2 ways to achieve this. First way is to create separate buckets for each object and the second way is to define one bucket but separate folders in a bucket for each object. Figure1.4 shows the buckets created in this step and 5 objects loaded into these buckets.


aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
Global
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
Amazon S3
Buckets
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
Access analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Feature spotlight 3
AWS Marketplace for S3
Amazon S3 > Buckets
View Storage Lens dashboard
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Account snapshot

Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets (6) Info

Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
c47145a661228l1797817t1w206842715267-querybucket-oh7oydh89lx7	US East (N. Virginia) us-east-1	Objects can be public	May 2, 2022, 09:03:09 (UTC+01:00)
k.kiani1.ct2019	US East (N. Virginia) us-east-1	Bucket and objects not public	May 2, 2022, 11:25:59 (UTC+01:00)
k.kiani1.ct2020	US East (N. Virginia) us-east-1	Bucket and objects not public	May 2, 2022, 11:26:40 (UTC+01:00)
k.kiani1.ct2021	US East (N. Virginia) us-east-1	Bucket and objects not public	May 2, 2022, 11:26:51 (UTC+01:00)
k.kiani1.mesh	US East (N. Virginia) us-east-1	Objects can be public	May 2, 2022, 11:25:40 (UTC+01:00)
k.kiani1.pharma	US East (N. Virginia) us-east-1	Bucket and objects not public	May 2, 2022, 11:25:10 (UTC+01:00)

Figure1.4

After creating buckets and loading objects into them, we need to create AWS glue crawler and direct it to our data source. It can then infer a schema based on the data types that discovers. AWS Glue builds a catalog that contains metadata about the data sources and makes everything ready for "Amazon Athena" to query the files. Figure1.5 indicates the structure of our application and figure1.6 shows the created crawler and 5 separate data sources connected to this crawler.

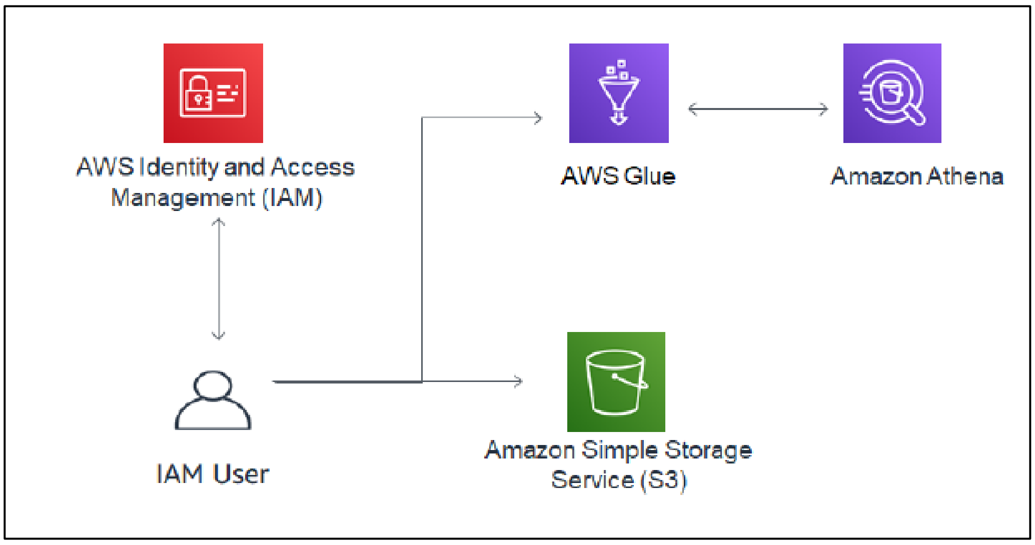
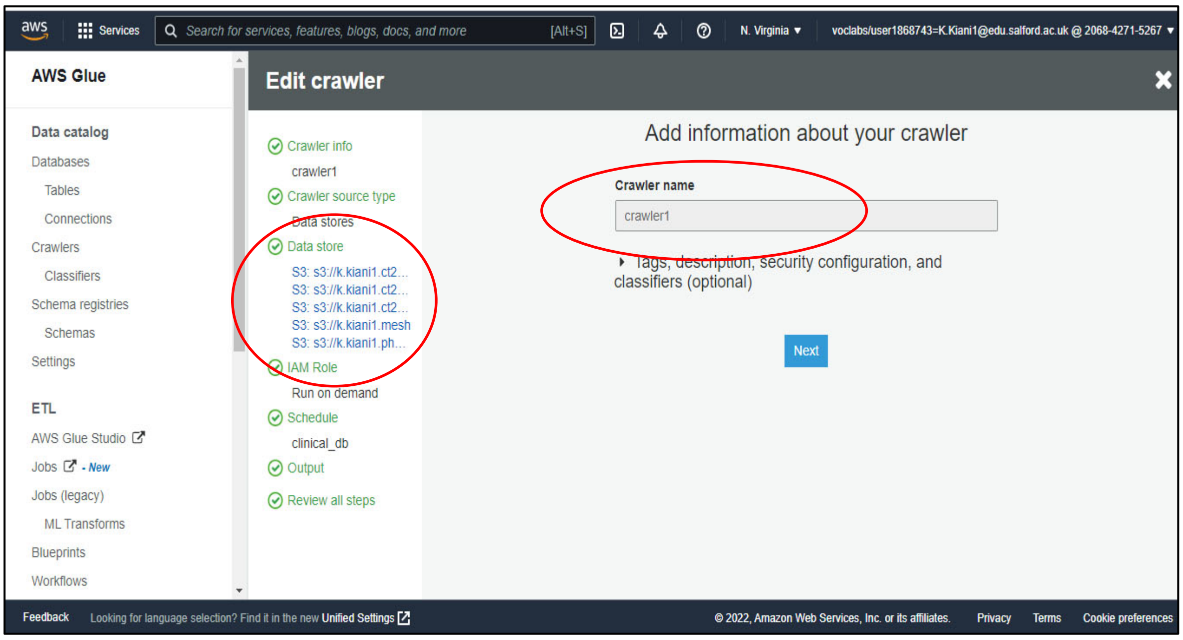

AWS Identity and Access Management (IAM)
AWS Glue
Amazon Athena
IAM User
Amazon Simple Storage Service (S3)

Figure1.5



Add information about your crawler

crawler1
Data stores
S3: s3://k.kiani1.ct2...
S3: s3://k.kiani1.ct2...
S3: s3://k.kiani1.ct2...
S3: s3://k.kiani1.mesh
S3: s3://k.kiani1.ph...
Run on demand
clinical_db

Crawler name: crawler1

Tags, description, security configuration, and classifiers (optional)

Figure1.6

2. Data preparation

2.1. Databricks – PySpark data preparation

In this study we have 5 different dataset files in the .csv format which 3 of them are Clinical Trials for years 2019, 2020 and 2021. Each row of these datasets represents an individual clinical trial, idented by an Id, listing the sponsor, the status of the study, the start and completion dates, the type of study, the submission date, and the lists of conditions the trial concerns. It is remarkable that individual conditions are separated by commas. It is remarkable that, the clinical trials 2019, and 2020 are test datasets and as requested, clinical trial 2021 is the main dataset for results submission.

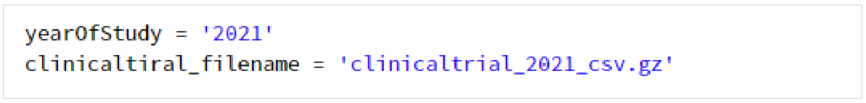
The two remaining datasets are pharma and mesh that accordingly contain list of publicly available pharmaceutical violations and the conditions from the clinical trial list.

These required .csv files have been read using *spark.read.csv()* and stored separately in 3 Dataframes:

1. clinicaltrial (Clinical Trial dataframe of the study year)
2. mesh (mesh dataframe)
3. pharma (pharma dataframe)

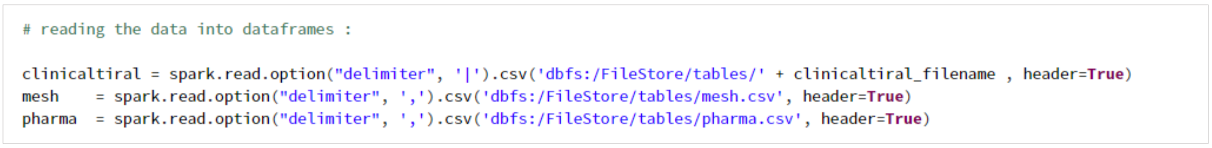
To run the code based on other years as input datasets, two variables named 'yearOfStudy' and 'clinicaltiral_filename' are defined in the first lines of the code. By simply changing the values for these two variables we can read and load the dataset for other years and find the outputs in a desired year.

These variables definition is shown in figure2.1 and the code in PySpark to read and build the dataframes is shown in figure2.2.



```
yearOfStudy = '2021'
clinicaltiral_filename = 'clinicaltrial_2021_csv.gz'
```

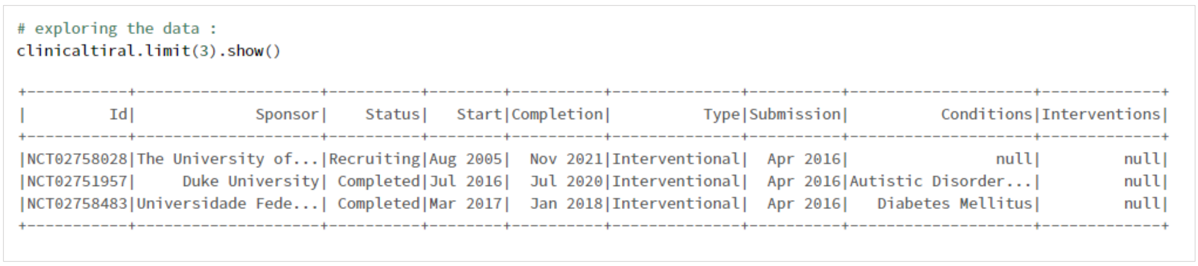
Figure2.1



```
# reading the data into dataframes :

clinicaltiral = spark.read.option("delimiter", '|').csv('dbfs:/FileStore/tables/' + clinicaltiral_filename , header=True)
mesh    = spark.read.option("delimiter", ',').csv('dbfs:/FileStore/tables/mesh.csv', header=True)
pharma  = spark.read.option("delimiter", ',').csv('dbfs:/FileStore/tables/pharma.csv', header=True)
```

Figure2.2

Figure2.3 is the result of exploring clinical trial 2021 dataset. Now the dataframes are built and ready for next steps.



```
# exploring the data :
clinicaltiral.limit(3).show()

+-----------+--------------------+---------+--------+----------+--------------+----------+--------------------+-------------+
|         Id|             Sponsor|   Status|   Start|Completion|          Type|Submission|          Conditions|Interventions|
+-----------+--------------------+---------+--------+----------+--------------+----------+--------------------+-------------+
|NCT02758028|The University of...|Recruiting|Aug 2005|  Nov 2021|Interventional|  Apr 2016|                null|         null|
|NCT02751957|     Duke University| Completed|Jul 2016|  Jul 2020|Interventional|  Apr 2016|Autistic Disorder...|         null|
|NCT02758483|Universidade Fede...| Completed|Mar 2017|  Jan 2018|Interventional|  Apr 2016|   Diabetes Mellitus|         null|
+-----------+--------------------+---------+--------+----------+--------------+----------+--------------------+-------------+
```

Figure2.3

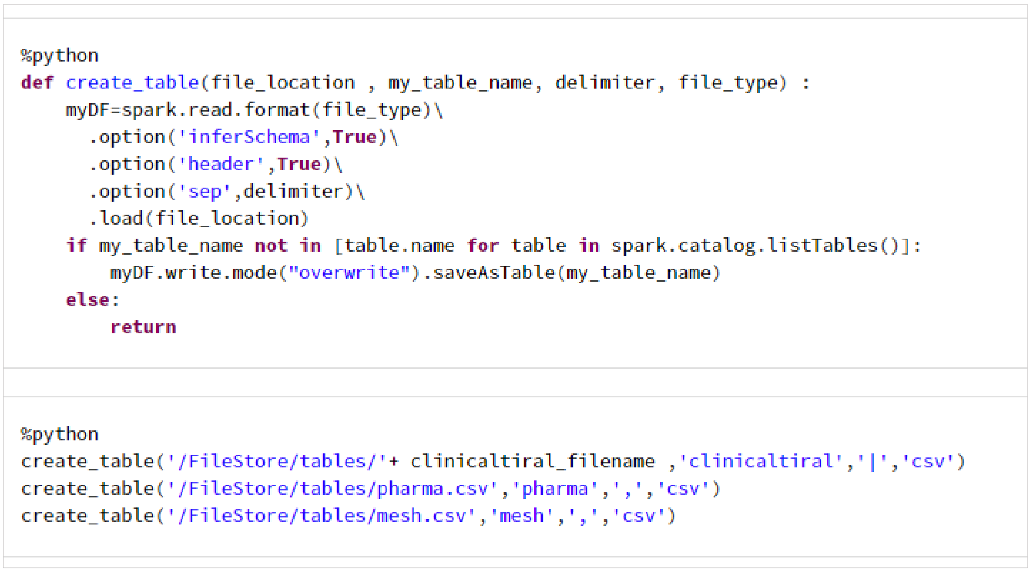
2.2. Databricks – HiveQL data preparation

In this section we are going to prepare the datasets to use with HiveQL. First, 3 tables are created using a function written in python. The 3 separate tables are as follows:

1. clinicaltrial (for a given year)
2. mesh
3. pharma

Figure2.4 is a function written in python to read the dataset and build the associated table. The 4 input parameters for this function are:

1. file_location: the location of dataset file.
2. my_table_name: the table name which we want to create.
3. delimiter: for clinical trial dataset this value is '|' and for the other 2 datasets the value is ','
4. file_type: '.csv' is the file type for each dataset file.

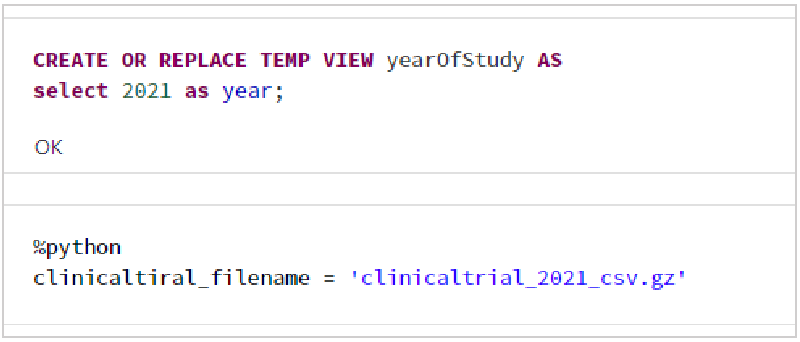


```
%python
def create_table(file_location , my_table_name, delimiter, file_type) :
    myDF=spark.read.format(file_type)\
      .option('inferSchema',True)\
      .option('header',True)\
      .option('sep',delimiter)\
      .load(file_location)
    if my_table_name not in [table.name for table in spark.catalog.listTables()]:
        myDF.write.mode("overwrite").saveAsTable(my_table_name)
    else:
        return
```

```
%python
create_table('/FileStore/tables/'+ clinicaltiral_filename ,'clinicaltiral','|','csv')
create_table('/FileStore/tables/pharma.csv','pharma',',','csv')
create_table('/FileStore/tables/mesh.csv','mesh',',','csv')
```

Figure2.4

To run the code based on other years as input datasets, as shown in figure2.5, one variable named 'clinicaltiral_filename' and one temp view named 'yearOfStudy' are defined in the first lines of the code. By simply changing their values, we can read and load the dataset for other years and find the outputs in a desired year.



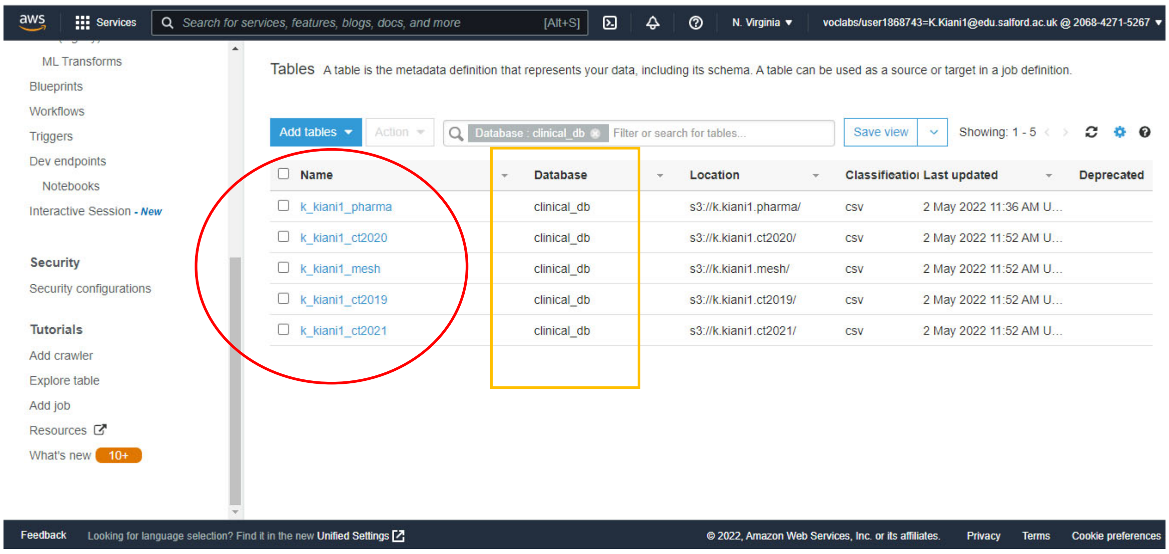
```
CREATE OR REPLACE TEMP VIEW yearOfStudy AS
select 2021 as year;

OK
```

```
%python
clinicaltiral_filename = 'clinicaltrial_2021_csv.gz'
```

Figure2.5

2.3. AWS – Athena Glue data preparation

As explained in section1.3 the 5 datasets are loaded into S3 buckets and an AWS Athena glue crawler is defined to discover the schema. After running the crawler and creating a database named "clinical_db", as shown in figure 2.6 the 5 tables have been created based on the discovered schema.

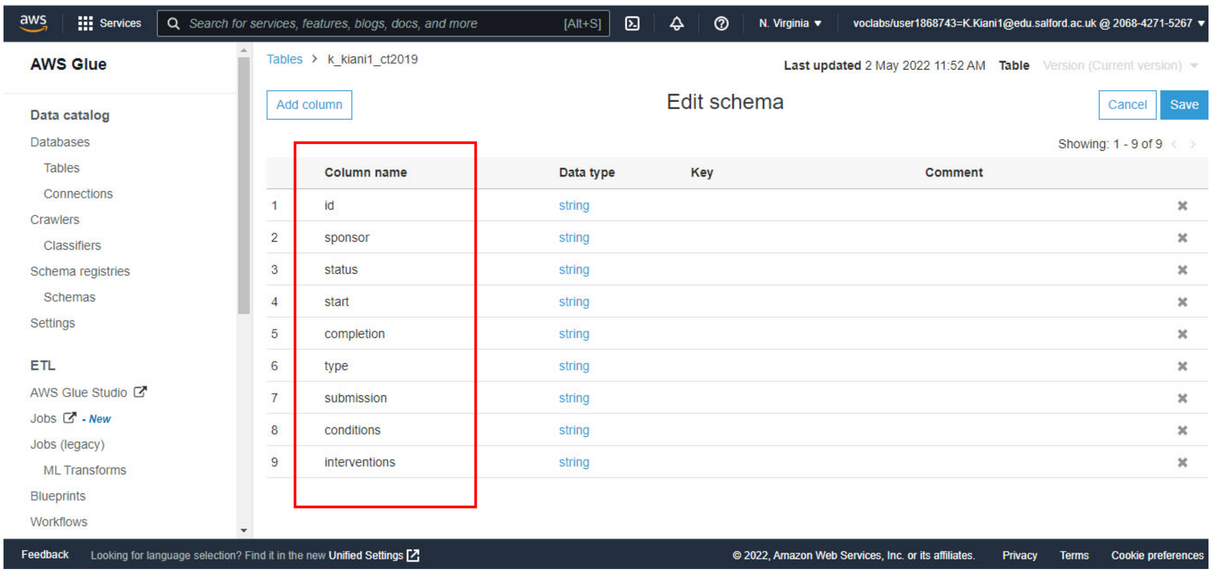

aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
N. Virginia
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
ML Transforms
Blueprints
Workflows
Triggers
Dev endpoints
Notebooks
Interactive Session - New
Security
Security configurations
Tutorials
Add crawler
Explore table
Add job
Resources
What's new 10+
Add tables
Action
Database : clinical_db
Filter or search for tables...
Save view
Showing: 1 - 5
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Tables A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Name	Database	Location	Classification	Last updated	Deprecated
k_kiani1_pharma	clinical_db	s3://k.kiani1.pharma/	csv	2 May 2022 11:36 AM U...	
k_kiani1_ct2020	clinical_db	s3://k.kiani1.ct2020/	csv	2 May 2022 11:52 AM U...	
k_kiani1_mesh	clinical_db	s3://k.kiani1.mesh/	csv	2 May 2022 11:52 AM U...	
k_kiani1_ct2019	clinical_db	s3://k.kiani1.ct2019/	csv	2 May 2022 11:52 AM U...	
k_kiani1_ct2021	clinical_db	s3://k.kiani1.ct2021/	csv	2 May 2022 11:52 AM U...	

Figure2.6

The AWS Athena glue has the option to view the table meta data and even alter the tables names and attributes. Using this option, as shown in figure2.7 we change the column names to the appropriate names.



AWS Glue

Tables > k_kiani1_ct2019

Last updated 2 May 2022 11:52 AM Table Version (Current version)

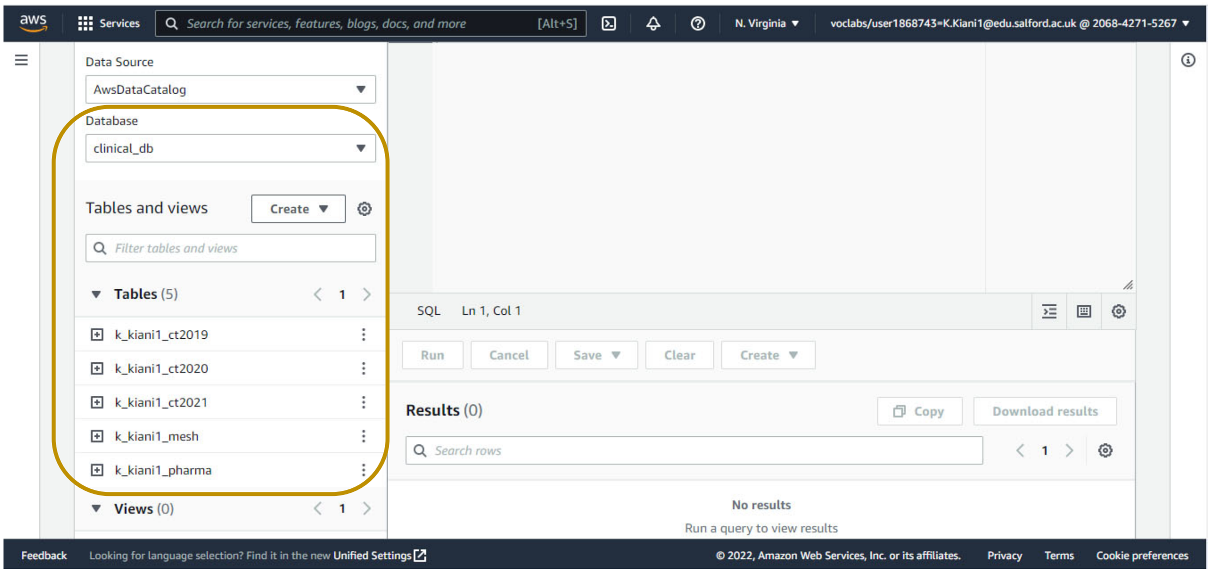
Edit schema

Showing: 1 - 9 of 9

	Column name	Data type	Key	Comment
1	id	string		
2	sponsor	string		
3	status	string		
4	start	string		
5	completion	string		
6	type	string		
7	submission	string		
8	conditions	string		
9	interventions	string		

Figure2.7

Finally, the database "clinical_db" and 5 tables are ready to make the queries and further analysis of data. Figure2.8 shows the final database and tables created.



Data Source: AwsDataCatalog

Database: clinical_db

Tables and views

Tables (5)
- k_kiani1_ct2019
- k_kiani1_ct2020
- k_kiani1_ct2021
- k_kiani1_mesh
- k_kiani1_pharma

Views (0)

Results (0)

No results

Run a query to view results

Figure2.8

3. Problems and Results

3.1. Question 1

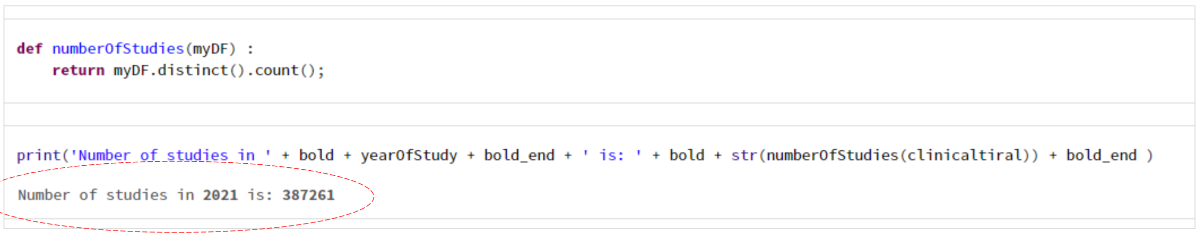
3.1.1. Assumptions

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

To find the answer for this question we need to refer to the clinical trials 2021 and count the number of studies. We need to make sure that distinct studies will be counted, so, while using each tool and platforms we have to explicitly use a command to check this.

3.1.2. PySpark implementation – Dataframe

Figure3.1.1 illustrates the function defined to count the number of different studies and the code to call this function. As the input parameter we send the Dataframe to the function and using the distinct() and count() functions, it calculates and returns the number of studies. Also, the output is shown in this figure. In section 3.1.5 the results will be discussed.



```
def numberOfStudies(myDF) :
    return myDF.distinct().count();
```

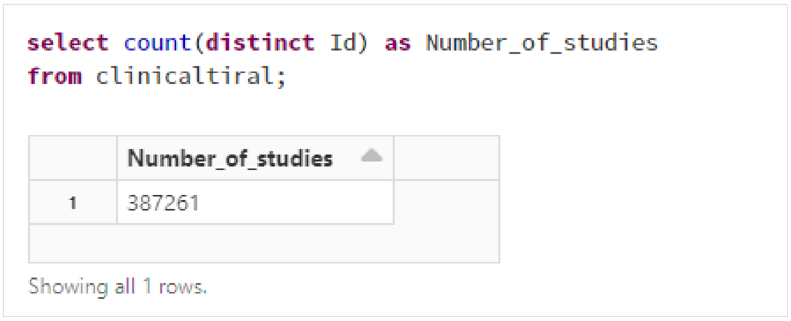
```
print('Number of studies in ' + bold + yearOfStudy + bold_end + ' is: ' + bold + str(numberOfStudies(clinicaltiral)) + bold_end )
```

```
Number of studies in 2021 is: 387261
```

Figure3.1.1

3.1.3. HiveQL implementation

To find the answer to the question1 using the HiveQL, as it is clear in figure3.1.2, we count distinct studies based on clinicaltrial table in a given year (here is 2021).



```
select count(distinct Id) as Number_of_studies
from clinicaltiral;
```

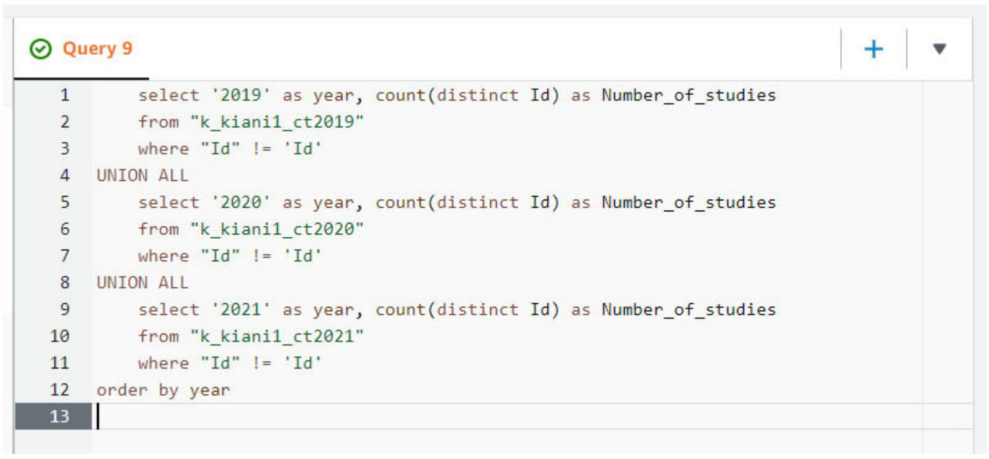
	Number_of_studies
1	387261

Showing all 1 rows.

Figure3.1.2

3.1.4. AWS implementation

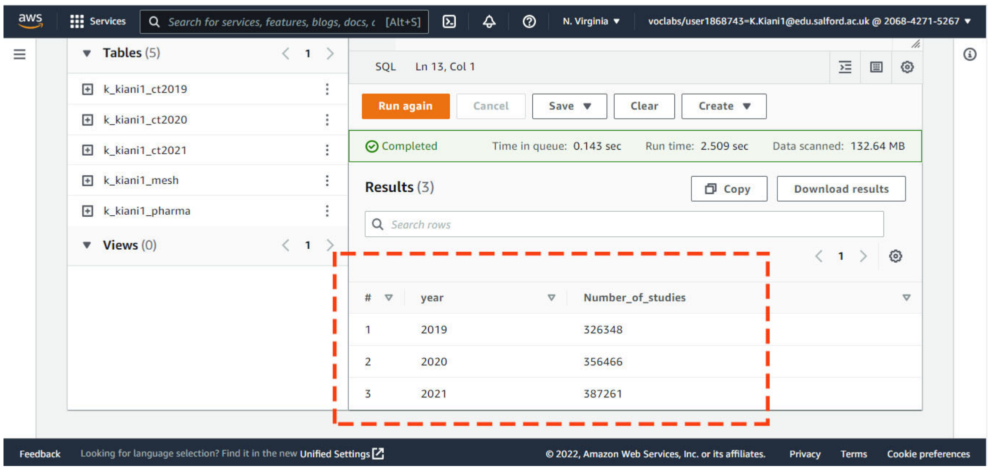
Using the AWS Athena, a new query is created to run the SQL code which is shown in figure3.1.3, and figure3.1.4 is the output of the mentioned code. In AWS implementation, it is decided to run the code for all 3 years, so the "union all" command is used to union the 3 different results in one single table. In the resulting table, each row indicates number of distinct studies in a specific year.



Query 9

```
    select '2019' as year, count(distinct Id) as Number_of_studies
    from "k_kiani1_ct2019"
    where "Id" != 'Id'
UNION ALL
    select '2020' as year, count(distinct Id) as Number_of_studies
    from "k_kiani1_ct2020"
    where "Id" != 'Id'
UNION ALL
    select '2021' as year, count(distinct Id) as Number_of_studies
    from "k_kiani1_ct2021"
    where "Id" != 'Id'
order by year
```

Figure3.1.3


aws
Services
Search for services, features, blogs, docs, [Alt+S]
N. Virginia
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
Tables (5)
k_kiani1_ct2019
k_kiani1_ct2020
k_kiani1_ct2021
k_kiani1_mesh
k_kiani1_pharma
Views (0)
SQL Ln 13, Col 1
Run again
Cancel
Save
Clear
Create
Copy
Download results
Search rows
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Completed   Time in queue: 0.143 sec   Run time: 2.509 sec   Data scanned: 132.64 MB

Results (3)

#	year	Number_of_studies
1	2019	326348
2	2020	356466
3	2021	387261

Figure3.1.4

3.1.5. Results and Discussion

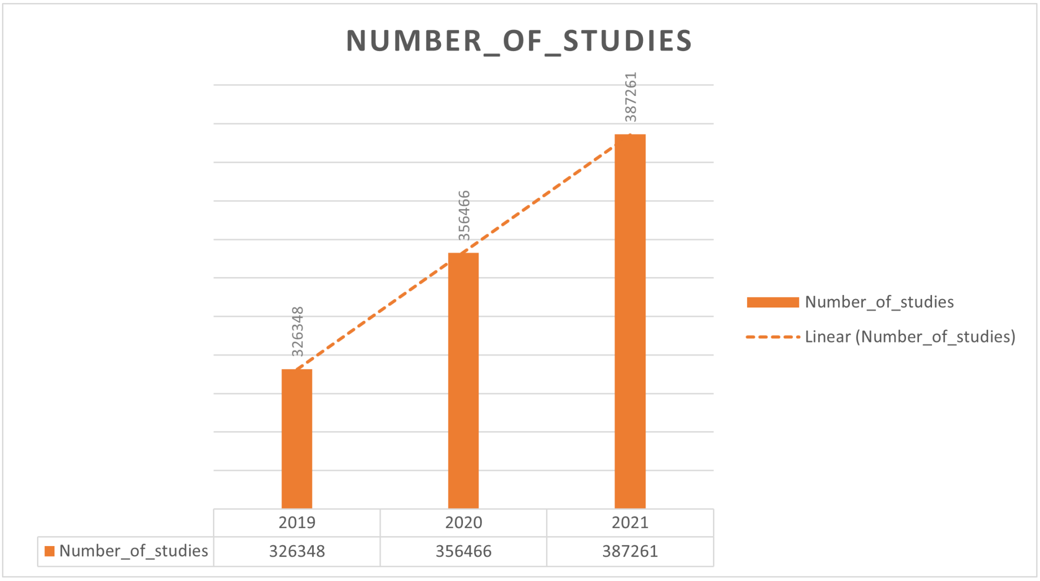
According to previous sections, all 3 different implementations using AWS Athena, PySpark and HiveQL resulted the same output values for the year 2021.

To have a comparison between three years, the final results are summarized in table3.1.1. These values are number of different studies in 3 different years.

year	Number_of_studies
2019	326348
2020	356466
2021	387261

Table3.1.1

After exporting the results to the "Microsoft Excel", and using the plot tools in this software, the results are compared in a bar chart. Figure3.1.5 illustrates this comparative bar chart. As it is clear in this figure the most studies were in year 2021 while the least studies belong to year 2019. It is remarkable that there is a linear growth in number of studies between 2019 and 2021.



	2019	2020	2021
🟧 Number_of_studies	326348	356466	387261

Figure3.1.5

3.2. Question 2

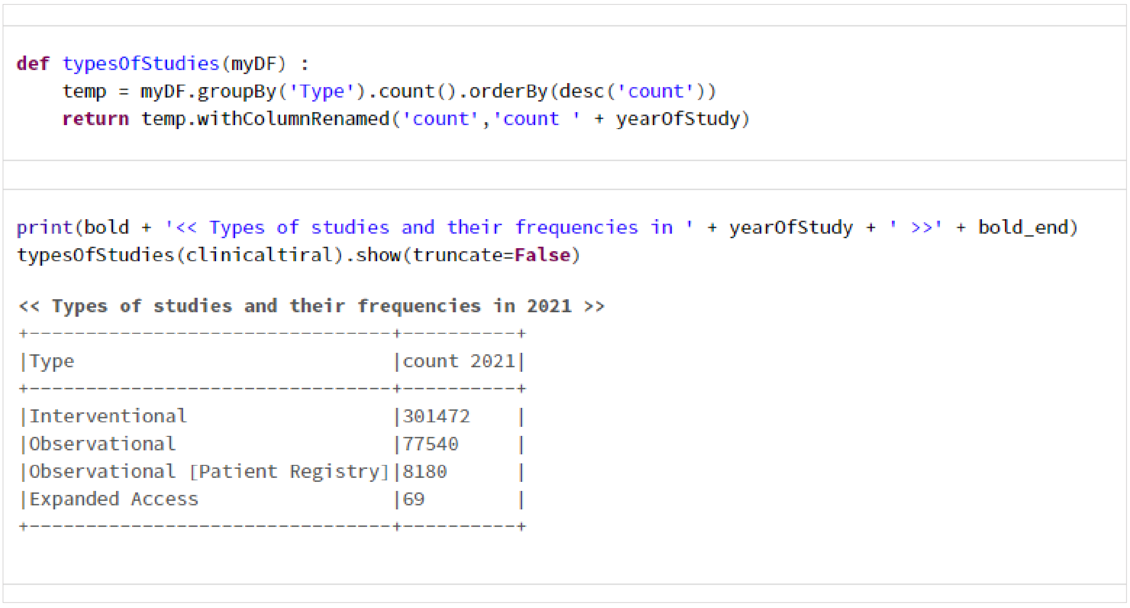
3.2.1. Assumptions

Question2: *You should list all the types (as contained in the Type column) of studies in the dataset along with the frequencies of each type. These should be ordered from most frequent to least frequent.*

To find the answer to this question we will refer to clinical trial 2021 dataset and will use the grouping and ordering techniques in each individual platform.

3.2.2. PySpark implementation – Dataframe

For this part, as it is clear in figure3.2.1, a function is defined to count the frequencies of each type. Within this function we use groupBy() to group rows based on the "Type" and then using count() function we count the frequency of each Type. The final results are ordered by DESC. The second part of the code calls the function and generates the output.



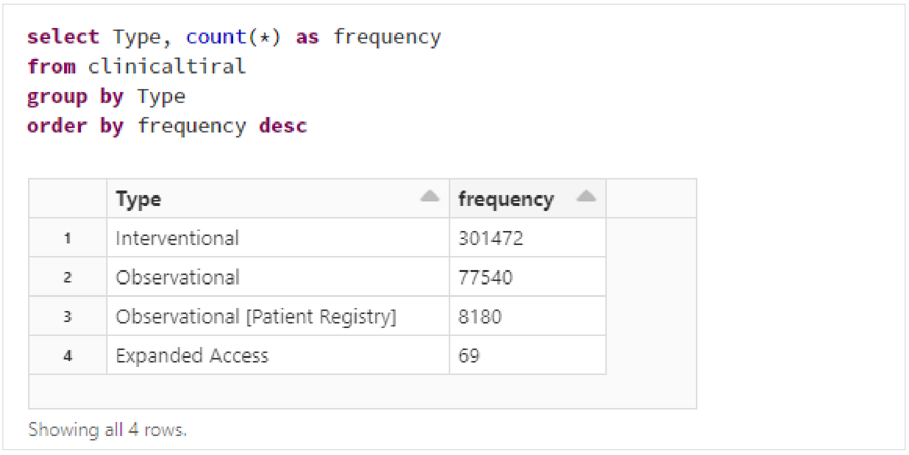
```
def typesOfStudies(myDF) :
    temp = myDF.groupBy('Type').count().orderBy(desc('count'))
    return temp.withColumnRenamed('count','count ' + yearOfStudy)
```

```
print(bold + '<< Types of studies and their frequencies in ' + yearOfStudy + ' >>' + bold_end)
typesOfStudies(clinicaltiral).show(truncate=False)

<< Types of studies and their frequencies in 2021 >>
+--------------------------------+----------+
|Type                            |count 2021|
+--------------------------------+----------+
|Interventional                  |301472    |
|Observational                   |77540     |
|Observational [Patient Registry]|8180      |
|Expanded Access                 |69        |
+--------------------------------+----------+
```

Figure3.2.1

3.2.3. HiveQL implementation

To find the answer with HiveQL, "group by" and "order by" have been used on clinical trial table. The SQL syntax and the outputs are shown in figure3.2.2.



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

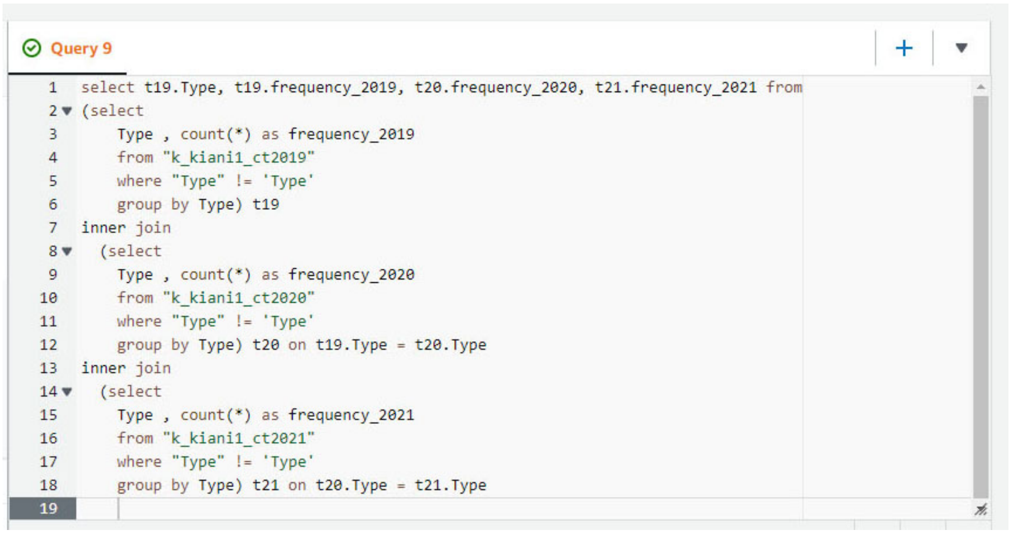
	Type	frequency
1	Interventional	301472
2	Observational	77540
3	Observational [Patient Registry]	8180
4	Expanded Access	69

Showing all 4 rows.

Figure3.2.2

3.2.4. AWS implementation

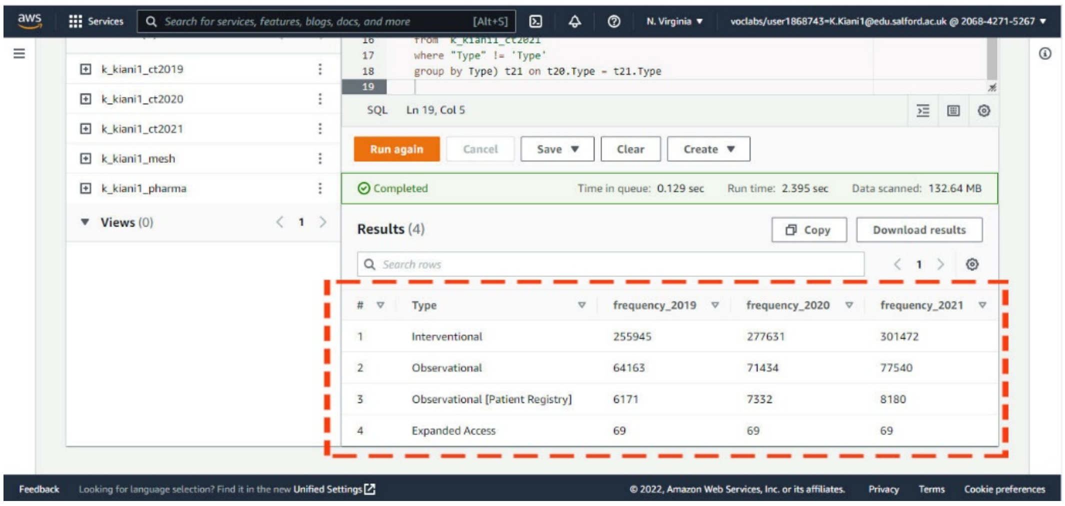
In AWS Athena query environment, SQL syntax is written as shown in figure3.2.3 to find the answer to second question. Again, in AWS implementation it is decided to run and find results for all 3 different years so, to write this query the "inner join" is used to join the 3 different queries and give the results in one single table. The final output of this SQL syntax is illustrated in figure3.2.4.



Query 9

```sql
select t19.Type, t19.frequency_2019, t20.frequency_2020, t21.frequency_2021 from
(select
    Type , count(*) as frequency_2019
    from "k_kiani1_ct2019"
    where "Type" != 'Type'
    group by Type) t19
inner join
  (select
    Type , count(*) as frequency_2020
    from "k_kiani1_ct2020"
    where "Type" != 'Type'
    group by Type) t20 on t19.Type = t20.Type
inner join
  (select
    Type , count(*) as frequency_2021
    from "k_kiani1_ct2021"
    where "Type" != 'Type'
    group by Type) t21 on t20.Type = t21.Type
```

Figure3.2.3


aws
Services
Search for services, features, blogs, docs, and more [Alt+S]
N. Virginia
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
k_kiani1_ct2019
k_kiani1_ct2020
k_kiani1_ct2021
k_kiani1_mesh
k_kiani1_pharma
Views (0)
SQL Ln 19, Col 5
Run again
Cancel
Save
Clear
Create
Copy
Download results
Search rows
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Completed Time in queue: 0.129 sec Run time: 2.395 sec Data scanned: 132.64 MB

Results (4)

#	Type	frequency_2019	frequency_2020	frequency_2021
1	Interventional	255945	277631	301472
2	Observational	64163	71434	77540
3	Observational [Patient Registry]	6171	7332	8180
4	Expanded Access	69	69	69

Figure3.2.4

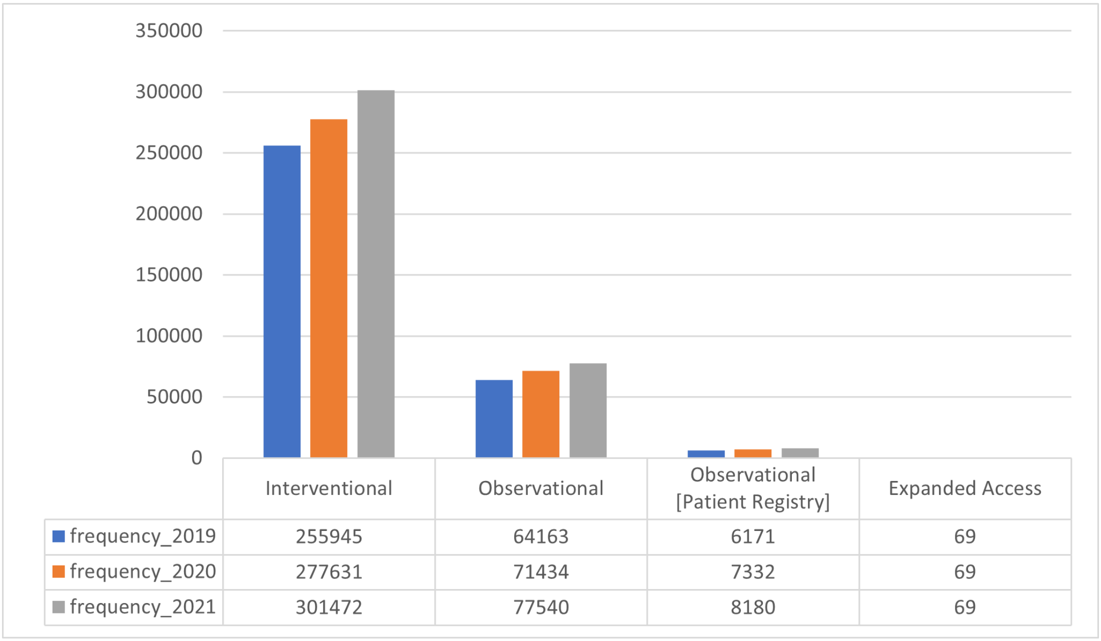
3.2.5. Results and Discussion

As it is shown in previous sections, all the 3 different ways and tools, give us the same results. These values are gathered in table3.2.1.

Type	frequency_2019	frequency_2020	frequency_2021
Interventional	**255945**	**277631**	**301472**
Observational	**64163**	**71434**	**77540**
Observational [Patient Registry]	**6171**	**7332**	**8180**
Expanded Access	**69**	**69**	**69**

Table3.2.1

Using the Microsoft Excel plotting tools, a comparison between the 4 different types in 3 years is shown in figure3.2.5. It is clear that the most frequency is "Interventional" and the least frequency is "Expanded Access". Also, in comparison between years the most frequencies of these 4 types occur in 2021. It is noticeable that, the frequency of "expanded Access" is not changed during the 3 years of our study.



	Interventional	Observational	Observational [Patient Registry]	Expanded Access
🟦 frequency_2019	255945	64163	6171	69
🟧 frequency_2020	277631	71434	7332	69
⬜ frequency_2021	301472	77540	8180	69

Figure3.2.5

3.3. Question 3

3.3.1. Assumptions

Question3: The top 5 conditions (from Conditions) with their frequencies.

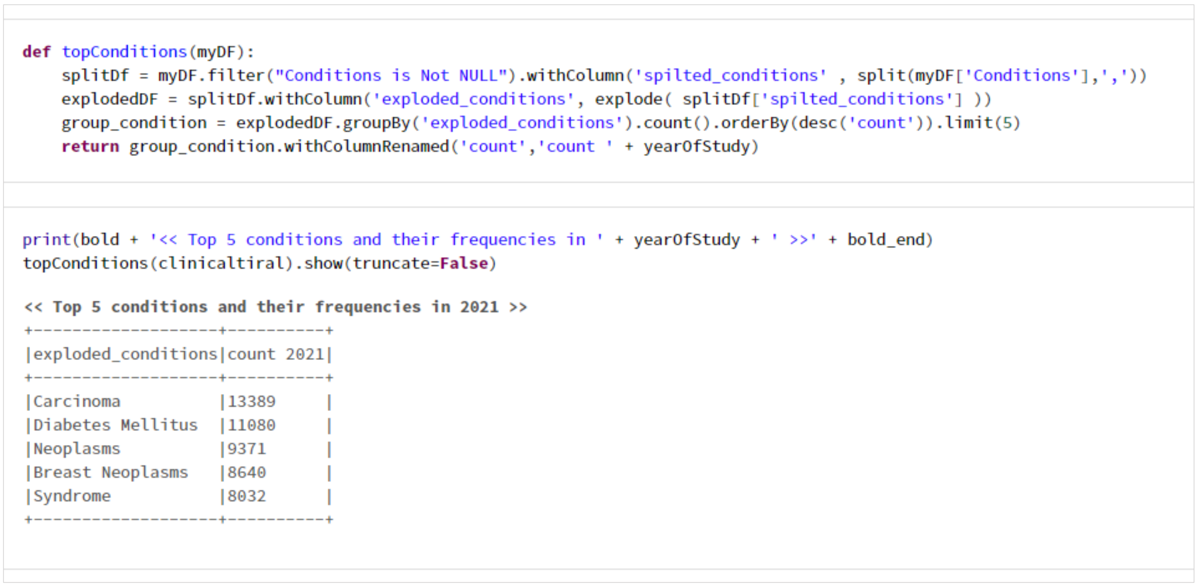
Exploring the clinical trial dataset, we will find that conditions are string values separated by ',' as the delimiter. So, we have to split these conditions as a list of values and then using the explode technique in each platform, find the frequencies of conditions.

Other techniques which will be used in this question, are grouping conditions, counting the occurrence, and ordering the results to find the top 5 conditions in the given year (2021).

3.3.2. PySpark implementation – Dataframe

In PySpark Implementation, a function named "topConditions()" is defined with input parameters of Dataframe and year. This function first splits the values of conditions which are separated by ',' as the delimiter. Then, using explode() function, it flattens the list of conditions and stores the result in a new Dataframe to the next step processes.

By grouping, counting, and ordering the exploded conditions we will find the answer to the question. Figure3.3.1 shows the implementation of this function, calling the function and final output.



```
def topConditions(myDF):
    splitDf = myDF.filter("Conditions is Not NULL").withColumn('spilted_conditions' , split(myDF['Conditions'],','))
    explodedDF = splitDf.withColumn('exploded_conditions', explode( splitDf['spilted_conditions'] ))
    group_condition = explodedDF.groupBy('exploded_conditions').count().orderBy(desc('count')).limit(5)
    return group_condition.withColumnRenamed('count','count ' + yearOfStudy)
```

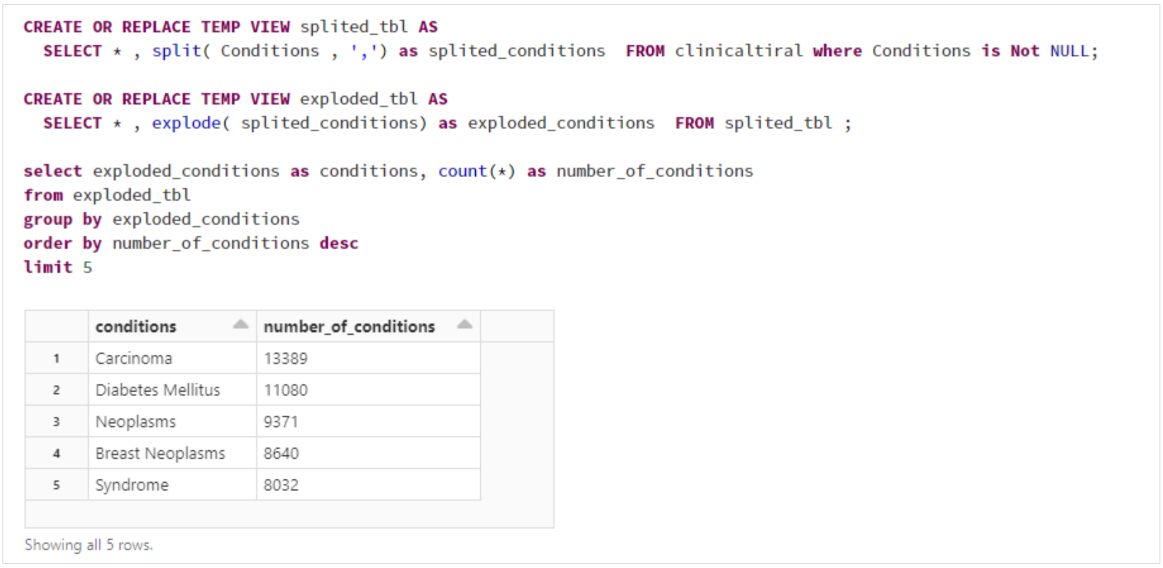
```
print(bold + '<< Top 5 conditions and their frequencies in ' + yearOfStudy + ' >>' + bold_end)
topConditions(clinicaltiral).show(truncate=False)

<< Top 5 conditions and their frequencies in 2021 >>
+-------------------+----------+
|exploded_conditions|count 2021|
+-------------------+----------+
|Carcinoma          |13389     |
|Diabetes Mellitus  |11080     |
|Neoplasms          |9371      |
|Breast Neoplasms   |8640      |
|Syndrome           |8032      |
+-------------------+----------+
```

Figure3.3.1

3.3.3. HiveQL implementation

In HiveQL implementation, we use split() function to separate the conditions. As mentioned before the delimiter is ',' and after splitting the conditions results are stored in a temp view to use in next steps.

Then, the explode() function is used to flatten the split list of conditions and the resulting view is stored in another temp view named "exploded_tbl". In the final step, grouping, counting, and ordering syntax are used to count the frequencies of conditions. Figures 3.3.2 is the implementations and output.



```
CREATE OR REPLACE TEMP VIEW splited_tbl AS
  SELECT * , split( Conditions , ',') as splited_conditions  FROM clinicaltiral where Conditions is Not NULL;

CREATE OR REPLACE TEMP VIEW exploded_tbl AS
  SELECT * , explode( splited_conditions) as exploded_conditions  FROM splited_tbl ;

select exploded_conditions as conditions, count(*) as number_of_conditions
from exploded_tbl
group by exploded_conditions
order by number_of_conditions desc
limit 5
```

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

Showing all 5 rows.

Figure3.3.2

3.3.4. AWS implementation

In the AWS Athena query environment, we create a new query and run the following sequence of codes to get the final results:

First step, creating a view using the split function and ',' as the delimiter of conditions.

```
CREATE OR REPLACE VIEW splited_c_19 AS
SELECT *,
    split(Conditions, ',') as splited_conditions
FROM "k_kiani1_ct2019"
where Conditions is Not NULL
and Conditions != 'Conditions'
```

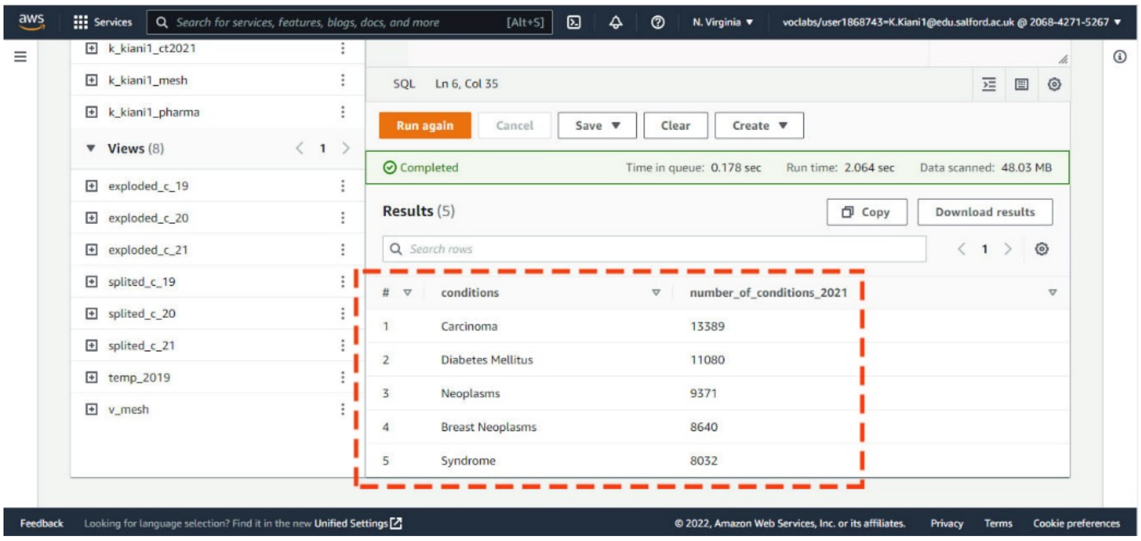
Second step, creating a view using "Cross Join Unnest" based on the splited view created in the previous step. This syntax flattens the splitted view and stores the result as exploded_c_XX view.

```
CREATE OR REPLACE VIEW exploded_c_19 AS
SELECT *
FROM splited_c_19
    CROSS JOIN UNNEST(splited_conditions) as t(exploded_conditions)
```

Third step, is the main syntax where counts the number of conditins by grouping the exploded conditions and then selecting the top 5 conditions using order by command.

```
select exploded_conditions as conditions,
    count(*) as number_of_conditions_2019
from exploded_c_19
where exploded_conditions != ''
group by exploded_conditions
order by number_of_conditions_2019 desc
limit 5
```

Figure3.3.3 is the outputs for the given year 2021.



Completed   Time in queue: 0.178 sec   Run time: 2.064 sec   Data scanned: 48.03 MB

#	conditions	number_of_conditions_2021
1	Carcinoma	13389
2	Diabetes Mellitus	11080
3	Neoplasms	9371
4	Breast Neoplasms	8640
5	Syndrome	8032

Figure3.3.3

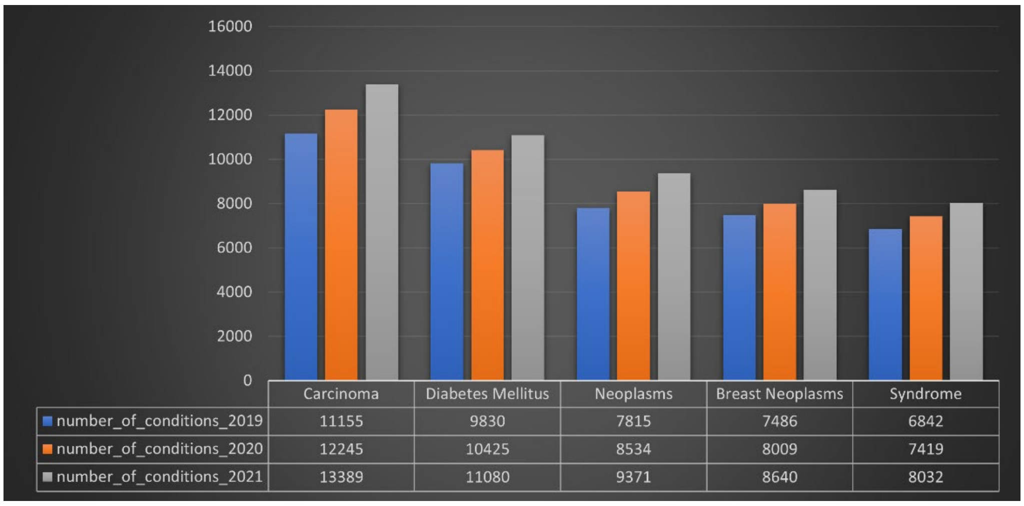
3.3.5. Results and Discussion

After running all the implementations, the final results are shown in table3.3.1.

conditions	number_of_conditions_2019	number_of_conditions_2020	number_of_conditions_2021
Carcinoma	**11155**	**12245**	**13389**
Diabetes Mellitus	**9830**	**10425**	**11080**
Neoplasms	**7815**	**8534**	**9371**
Breast Neoplasms	**7486**	**8009**	**8640**
Syndrome	**6842**	**7419**	**8032**

Table3.3.1

Figure3.3.4, indicates that the most conditions are "Carcinoma" and the least conditions belongs to "Syndrome". If we want to compare the values according to study years, it is clear that there is a linear growth in each condition number from 2019 to 2021.



	Carcinoma	Diabetes Mellitus	Neoplasms	Breast Neoplasms	Syndrome
🟦 number_of_conditions_2019	11155	9830	7815	7486	6842
🟧 number_of_conditions_2020	12245	10425	8534	8009	7419
⬜ number_of_conditions_2021	13389	11080	9371	8640	8032

Figure3.3.10

3.4. Question 4

3.4.1. Assumptions

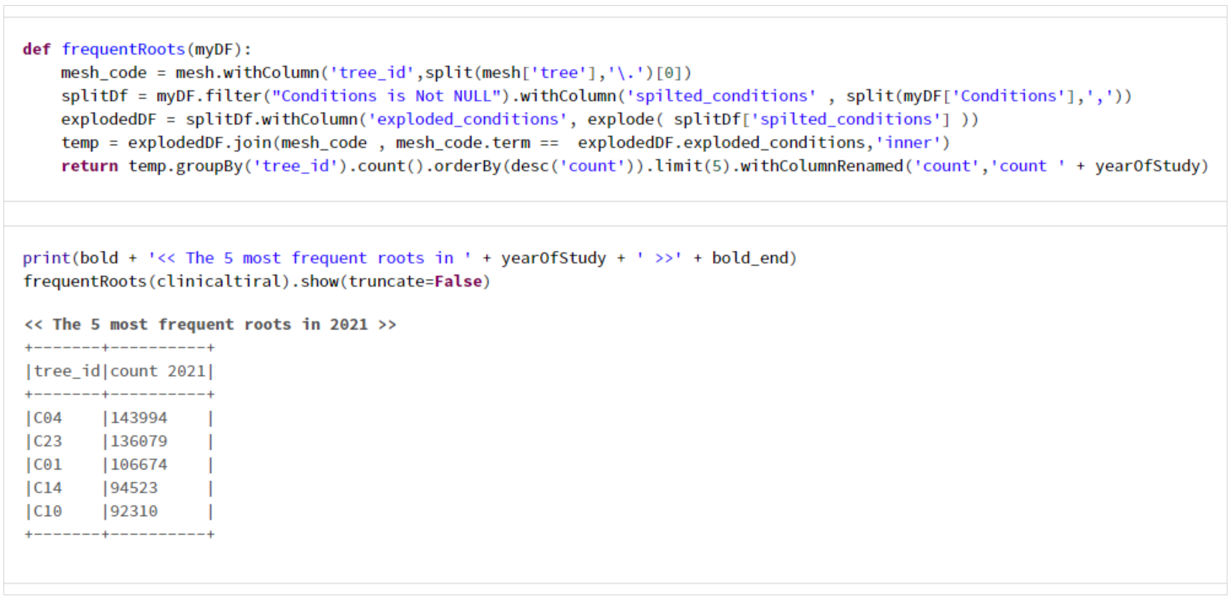
Question4: Each condition can be mapped to one or more hierarchy codes. The client wishes to know the 5 most frequent roots after this is done.

To find the answer for this question we need to extract a new field called "tree_id" from the "mesh" data set. To do this, we have to split the "tree" field based on the "." delimiter and retrieve the first part of the string value as the "tree_id".

By joining this dataset with clinical trials dataset, we can find the 5 most frequent roots. In next sections, 3 different implementations will be used based on this idea to solve the problem.

3.4.2. PySpark implementation – Dataframe

As explained in section3.4.1, first we have to use the split() function in PySpark to retrieve the "tree_id" as a part of the "tree" string value. The output is saved in a new dataframe "mesh_code". Then, this new dataframe is joined with clinical trials dataset, by calling the "frequentRoots()" function to count the number of each "tree_id" and find the 5 most frequent roots. Figure3.4.1 is the implementation in PySpark and we can see that splitting, exploding, joining, grouping, counting, and sorting techniques are used in the " frequentRoots ()" function to find the answer to this question.



```
def frequentRoots(myDF):
    mesh_code = mesh.withColumn('tree_id',split(mesh['tree'],'\.')[0])
    splitDf = myDF.filter("Conditions is Not NULL").withColumn('spilted_conditions' , split(myDF['Conditions'],','))
    explodedDF = splitDf.withColumn('exploded_conditions', explode( splitDf['spilted_conditions'] ))
    temp = explodedDF.join(mesh_code , mesh_code.term ==  explodedDF.exploded_conditions,'inner')
    return temp.groupBy('tree_id').count().orderBy(desc('count')).limit(5).withColumnRenamed('count','count ' + yearOfStudy)
```

```
print(bold + '<< The 5 most frequent roots in ' + yearOfStudy + ' >>' + bold_end)
frequentRoots(clinicaltiral).show(truncate=False)

<< The 5 most frequent roots in 2021 >>
+-------+----------+
|tree_id|count 2021|
+-------+----------+
|C04    |143994    |
|C23    |136079    |
|C01    |106674    |
|C14    |94523     |
|C10    |92310     |
+-------+----------+
```

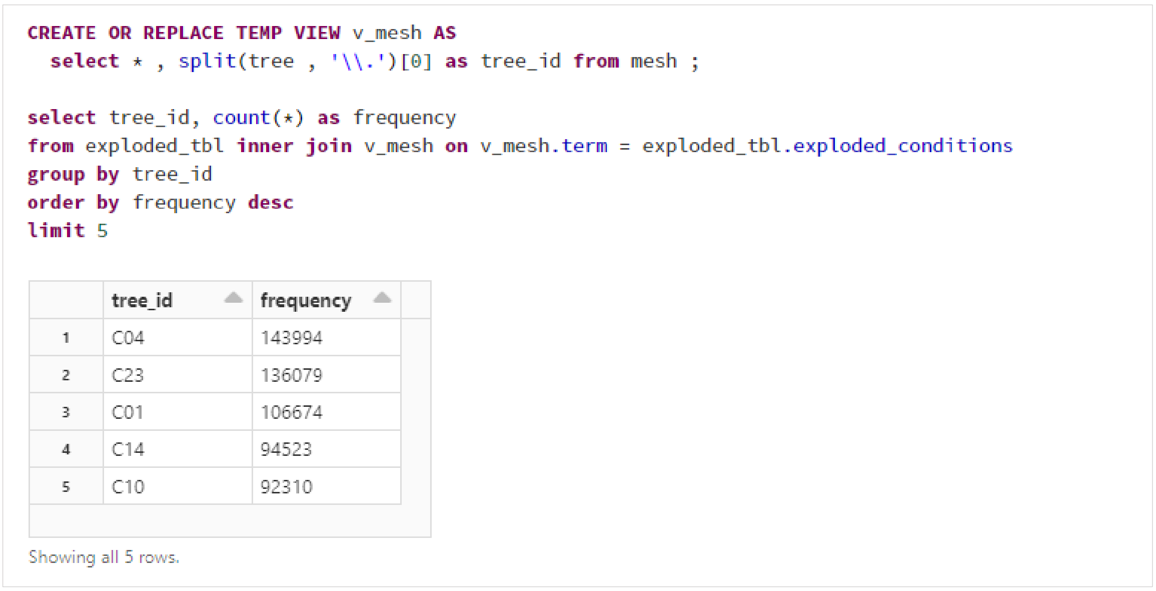
Figure3.4.1

3.4.3. HiveQL implementation

HiveQL implementation for question 4 will use the same logic explained in the first section of this part. So, first a new temp view "v_mesh" is created by spliting the "tree" values in "mesh" as it is in the following code:

```
CREATE OR REPLACE TEMP VIEW v_mesh AS
  select * , split(tree , '\\.')[0] as tree_id from tbl_mesh
```

Then, as it is shown in figure3.4.2, "v_mesh" view is joined with exploded clinical trial view to count the number of "tree_id" in the given year.



```
CREATE OR REPLACE TEMP VIEW v_mesh AS
  select * , split(tree , '\\.')[0] as tree_id from mesh ;

select tree_id, count(*) as frequency
from exploded_tbl inner join v_mesh on v_mesh.term = exploded_tbl.exploded_conditions
group by tree_id
order by frequency desc
limit 5
```

	tree_id	frequency
1	C04	143994
2	C23	136079
3	C01	106674
4	C14	94523
5	C10	92310

Showing all 5 rows.

Figure3.4.2

3.4.4. AWS implementation

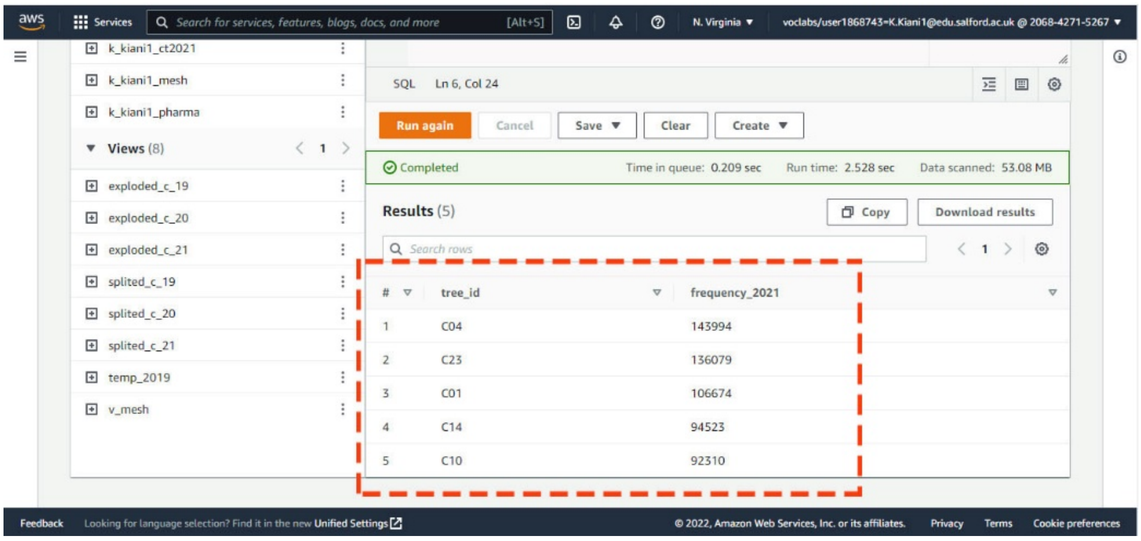
The following code in AWS Athena query environment, is used to create the "v_mesh" view which holds the splited values of "tree" as "tree_id".

```sql
CREATE OR REPLACE VIEW v_mesh AS
select *,
    split(tree, '.')[1] as tree_id
from "k_kiani1_mesh"
where "term" != 'term'
```

Then, the "v_mesh" view is joined with exploded clinical trials views to find the most frequent roots seperated by years.

```sql
select tree_id,
    count(*) as frequency_2019
from exploded_c_19
    inner join v_mesh on v_mesh.term = exploded_c_19.exploded_conditions
group by tree_id
order by frequency_2019 desc
limit 5
```

Figure 3.4.3 is the output values for the year 2021, in AWS Athena view.



Completed   Time in queue: 0.209 sec   Run time: 2.528 sec   Data scanned: 53.08 MB

Results (5)

#	tree_id	frequency_2021
1	C04	143994
2	C23	136079
3	C01	106674
4	C14	94523
5	C10	92310

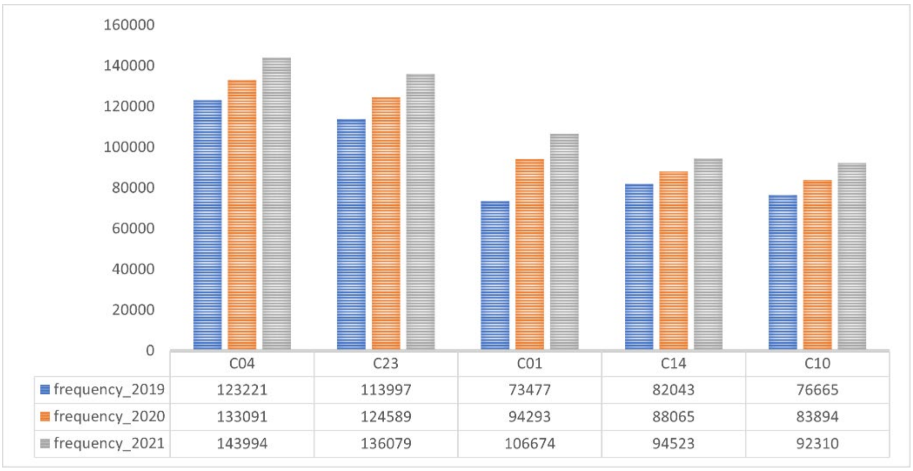
Figure3.4.3

3.4.5. Results and Discussion

Table3.4.1, summarizes all the results for each year, retrieved by 3 different implementations. To have a clear view and a comparative sight, in figure3.4.4 a bar chart is plotted using Microsoft Excel tools. We can find that the "C04" is the most frequent in each year between 2019 and 2021 following by "C23" as the second most frequent root. Also, "C01" in 2019 has the least frequent between all the roots.

tree_id	frequency_2019	tree_id	frequency_2020	tree_id	frequency_2021
C04	123221	C04	133091	C04	143994
C23	113997	C23	124589	C23	136079
C14	82043	C01	94293	C01	106674
C10	76665	C14	88065	C14	94523
C01	73477	C10	83894	C10	92310

Table3.4.1



	C04	C23	C01	C14	C10
frequency_2019	123221	113997	73477	82043	76665
frequency_2020	133091	124589	94293	88065	83894
frequency_2021	143994	136079	106674	94523	92310

Figure3.4.4

3.5. Question 5

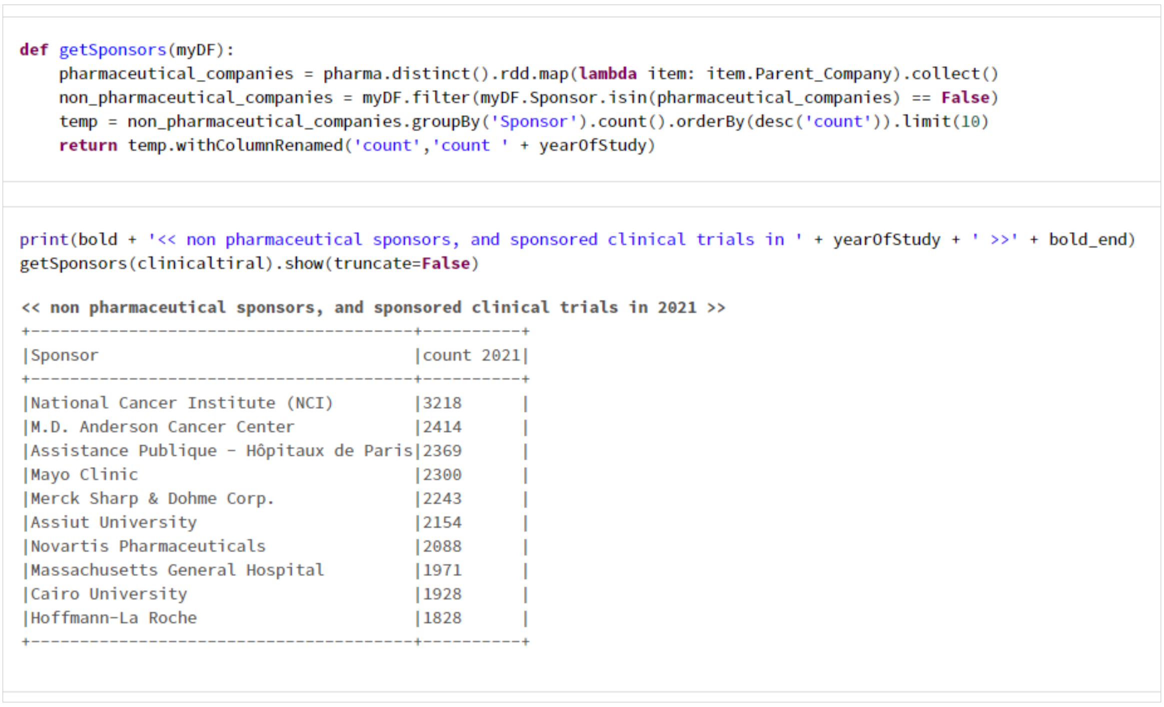
3.5.1. Assumptions

Question5: Find the 10 most common sponsors that are not pharmaceutical companies, along with the number of clinical trials they have sponsored.

To answer this question, we need no refer to pharma dataset and find the "pharmaceutical companies list". In the next step we should filter the sponsors in clinical trials dataset who are not in the "pharmaceutical companies list". In next sections 3 different implementation are used based on this logic.

3.5.2. PySpark implementation – Dataframe

To find the answer using PySpark, first we define a function named "getSponsors()" which filters the sponsor names who are not in the "pharmaceutical_companies" dataframe. After filtering the sponsor names, by grouping and counting the number of clinical trials, we find the 10 most common sponsors. Figure3.5.1 is the implementation of this function and the output after running the code which shows the 10 most sponsors in year 2021.



```
def getSponsors(myDF):
    pharmaceutical_companies = pharma.distinct().rdd.map(lambda item: item.Parent_Company).collect()
    non_pharmaceutical_companies = myDF.filter(myDF.Sponsor.isin(pharmaceutical_companies) == False)
    temp = non_pharmaceutical_companies.groupBy('Sponsor').count().orderBy(desc('count')).limit(10)
    return temp.withColumnRenamed('count','count ' + yearOfStudy)
```

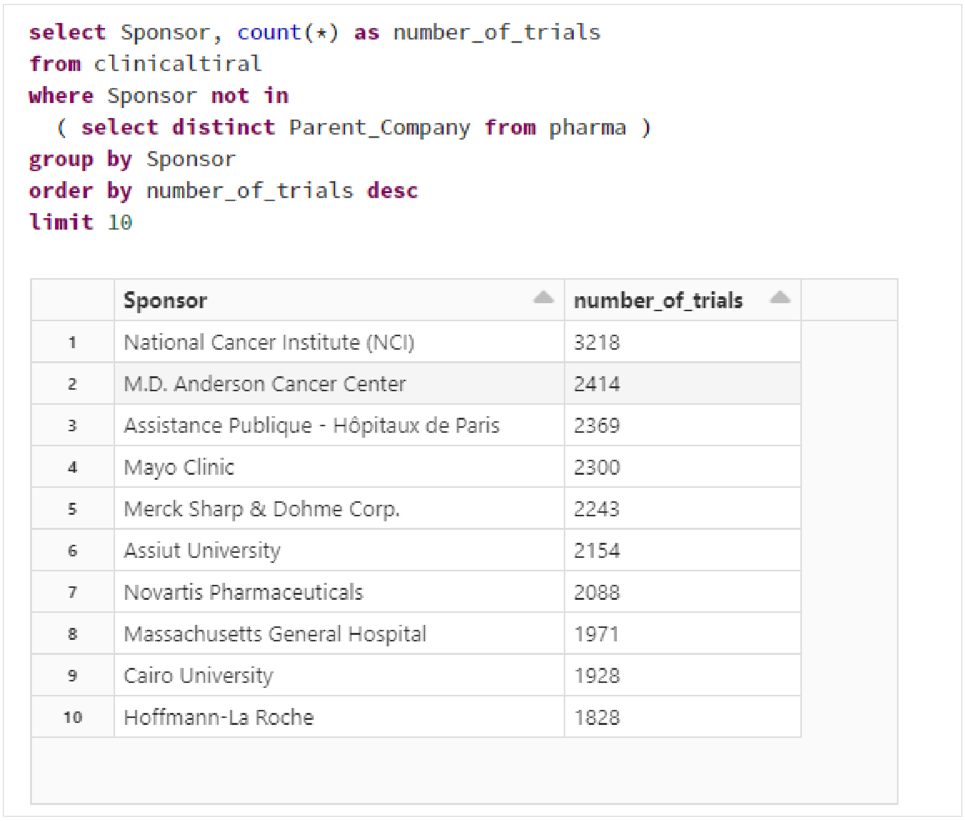
```
print(bold + '<< non pharmaceutical sponsors, and sponsored clinical trials in ' + yearOfStudy + ' >>' + bold_end)
getSponsors(clinicaltiral).show(truncate=False)

<< non pharmaceutical sponsors, and sponsored clinical trials in 2021 >>
+---------------------------------------+----------+
|Sponsor                                |count 2021|
+---------------------------------------+----------+
|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      |
+---------------------------------------+----------+
```

Figure3.5.1

3.5.3. HiveQL implementation

The logic behind the HiveQL implementation for question number 5, is the same as explained in first section of this part. But to find the answer with HiveQL, we use a nested select syntax and "in" operator. The inner select finds the companies from "pharma" that are pharmaceutical companies, and in the outer SQL code we mention that we need sponsors that are not in the inner list.

Again, by grouping and counting techniques, we can find the 10 most common sponsors. Figure3.5.2 shows the implementations and output of year 2021.



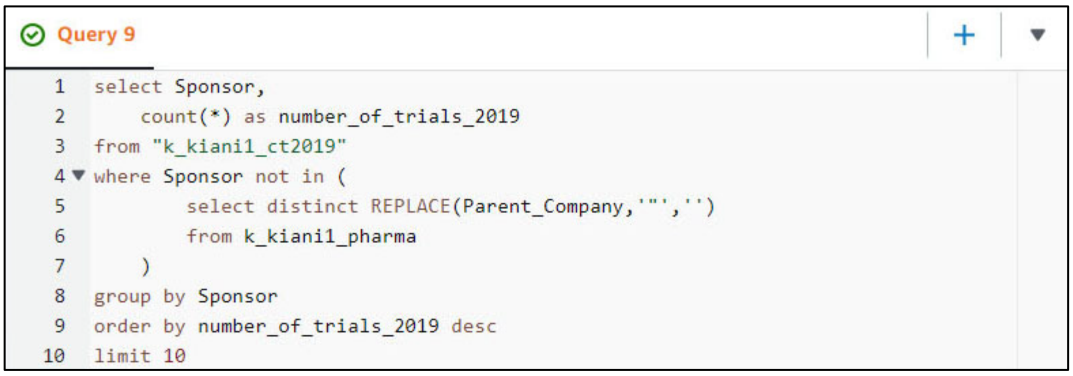
```
select Sponsor, count(*) as number_of_trials
from clinicaltiral
where Sponsor not in
  ( select distinct Parent_Company from pharma )
group by Sponsor
order by number_of_trials desc
limit 10
```

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

Figure3.5.2

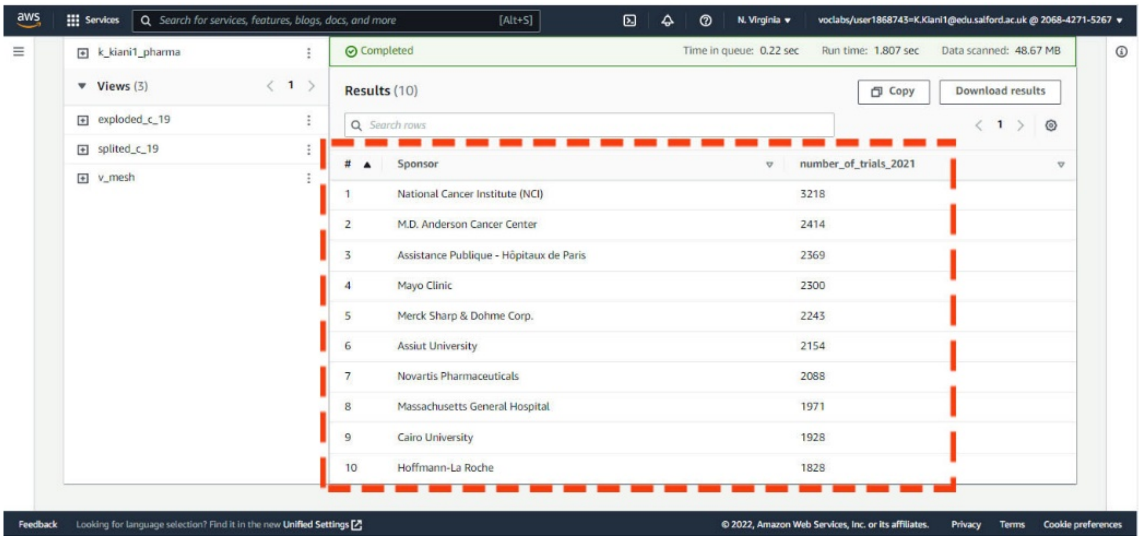
3.5.4. AWS implementation

In AWS Athena we have used the same syntax and logic explained in HiveQL implementation. Figure3.5.3 shows the sample syntax that selects the non-pharmaceutical companies and count the number of clinical trials for each of them. The output is clear in figure3.5.4 for the given year 2021.



```
select Sponsor,
    count(*) as number_of_trials_2019
from "k_kiani1_ct2019"
where Sponsor not in (
        select distinct REPLACE(Parent_Company,'"','')
        from k_kiani1_pharma
    )
group by Sponsor
order by number_of_trials_2019 desc
limit 10
```

Figure3.5.3


aws
Services
Search for services, features, blogs, docs, and more [Alt+S]
N. Virginia
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
k_kiani1_pharma
Views (3)
exploded_c_19
splited_c_19
v_mesh
Copy
Download results
Search rows
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Completed Time in queue: 0.22 sec Run time: 1.807 sec Data scanned: 48.67 MB

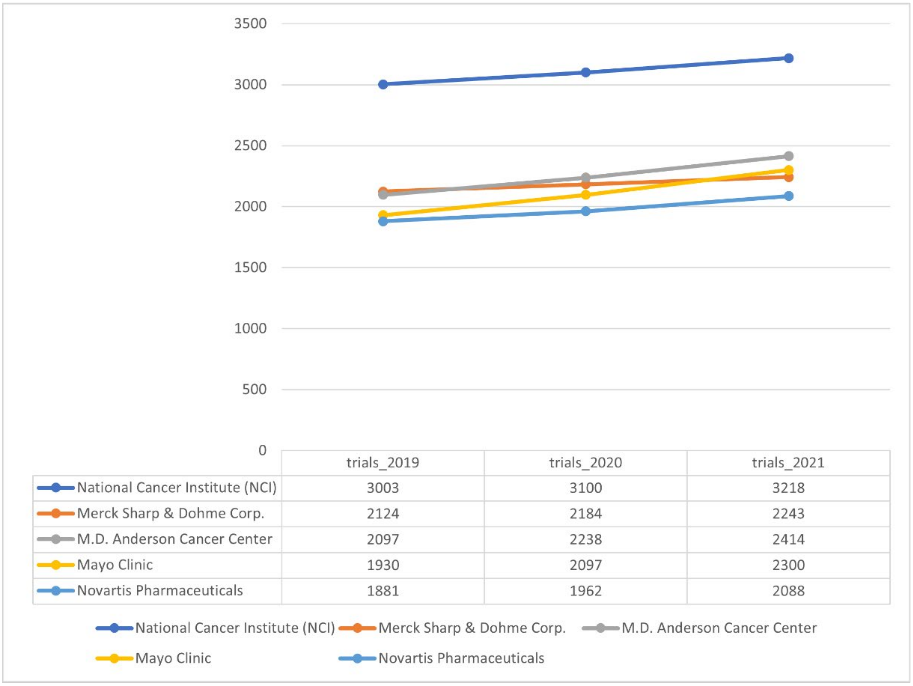
Results (10)

#	Sponsor	number_of_trials_2021
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

Figure3.5.3

3.5.5. Results and Discussion

In this section, a diagram is plotted for 5 common sponsors to see the trend in 3 years between 2019 and 2021. As it is visible in this diagram (figure3.5.4), 4 of the companies except "Merck sharp & dohme corp" have a linier growth which is not sharp in 3 years. The "Merck sharp & dohme corp" company has a steady growth and the number of sponsored studies is not dramatically increasing during the 3 years.



	trials_2019	trials_2020	trials_2021
National Cancer Institute (NCI)	3003	3100	3218
Merck Sharp & Dohme Corp.	2124	2184	2243
M.D. Anderson Cancer Center	2097	2238	2414
Mayo Clinic	1930	2097	2300
Novartis Pharmaceuticals	1881	1962	2088

Figure3.5.4

3.6. Question 6

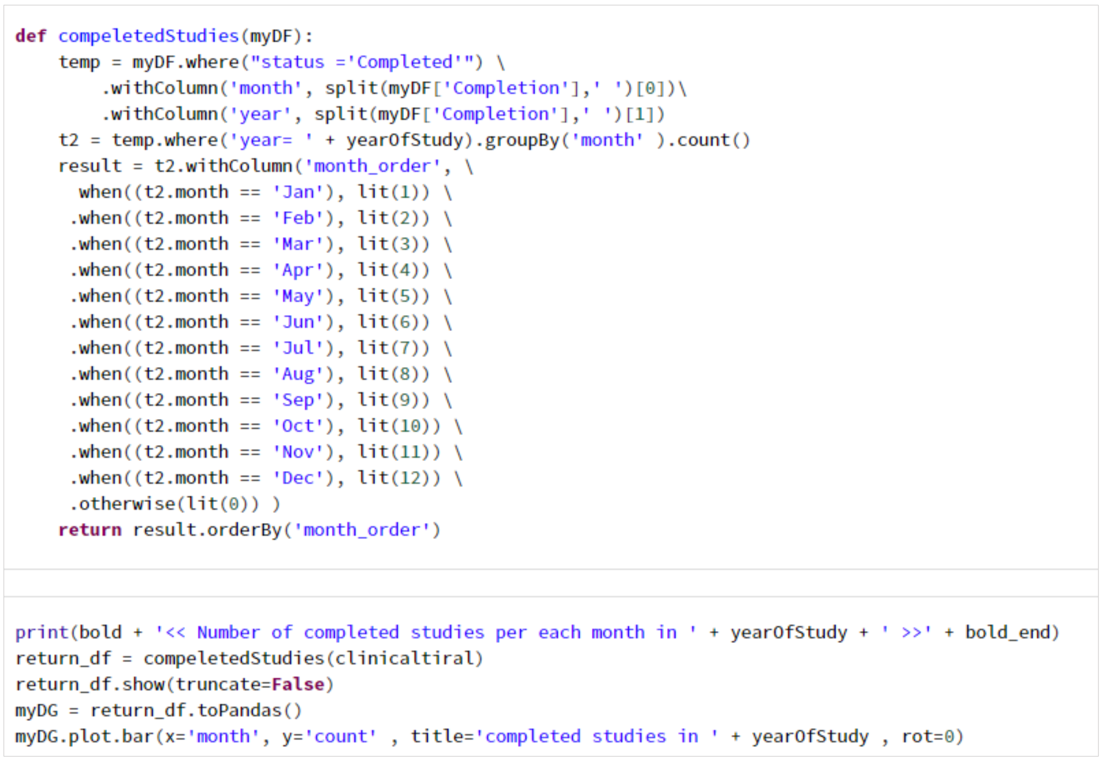
3.6.1. Assumptions

Question6: Plot number of completed studies each month in a given year for the submission dataset, the year is 2021.

To find the completed studies, we must refer to clinical trials datasets, and based on the year and month values, count the number of records.

3.6.2. PySpark implementation – Dataframe

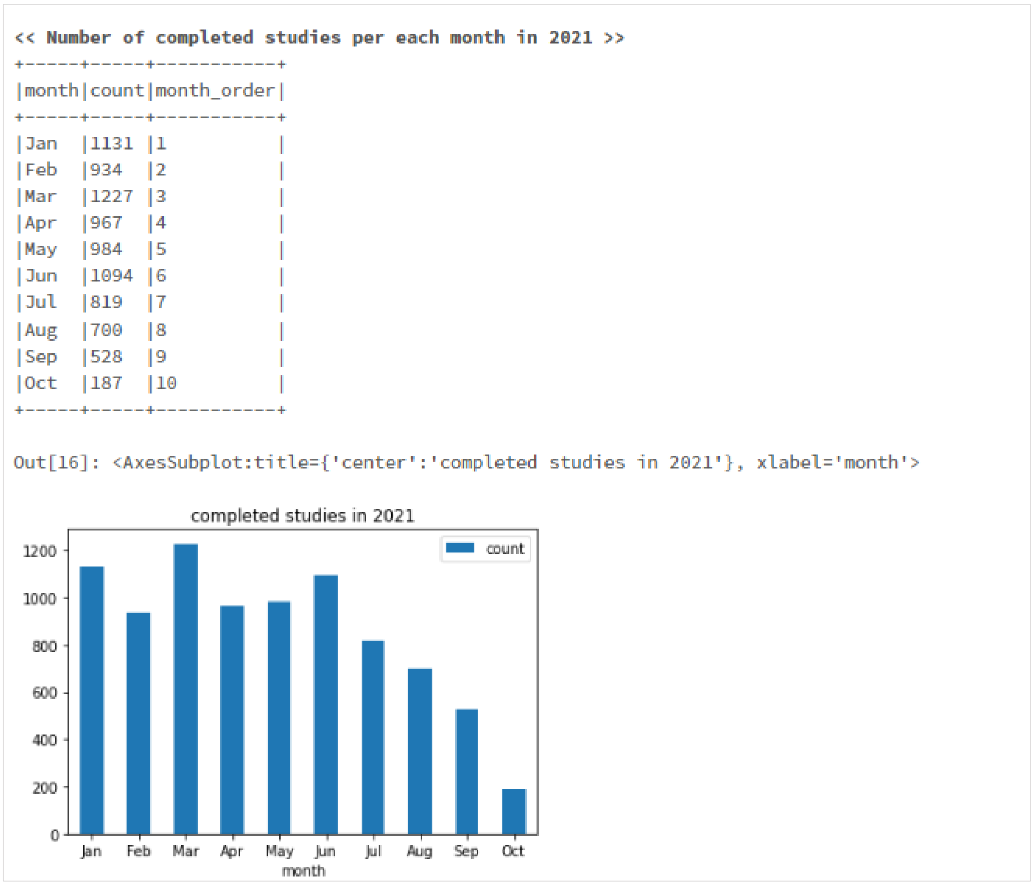
In PySpark implementation, first we define a function to split the "completion" value into "year" and "month". Then, by filtering the year and grouping the month and counting the records, we can find the result dataframe and return it to the calling code as it is shown in figure3.6.1. The resulting values and output bar plot is indicated in figure3.6.2.



```
def compeletedStudies(myDF):
    temp = myDF.where("status ='Completed'") \
        .withColumn('month', split(myDF['Completion'],' ')[0])\
        .withColumn('year', split(myDF['Completion'],' ')[1])
    t2 = temp.where('year= ' + yearOfStudy).groupBy('month' ).count()
    result = t2.withColumn('month_order', \
      when((t2.month == 'Jan'), lit(1)) \
     .when((t2.month == 'Feb'), lit(2)) \
     .when((t2.month == 'Mar'), lit(3)) \
     .when((t2.month == 'Apr'), lit(4)) \
     .when((t2.month == 'May'), lit(5)) \
     .when((t2.month == 'Jun'), lit(6)) \
     .when((t2.month == 'Jul'), lit(7)) \
     .when((t2.month == 'Aug'), lit(8)) \
     .when((t2.month == 'Sep'), lit(9)) \
     .when((t2.month == 'Oct'), lit(10)) \
     .when((t2.month == 'Nov'), lit(11)) \
     .when((t2.month == 'Dec'), lit(12)) \
     .otherwise(lit(0)) )
    return result.orderBy('month_order')
```

```
print(bold + '<< Number of completed studies per each month in ' + yearOfStudy + ' >>' + bold_end)
return_df = compeletedStudies(clinicaltiral)
return_df.show(truncate=False)
myDG = return_df.toPandas()
myDG.plot.bar(x='month', y='count' , title='completed studies in ' + yearOfStudy , rot=0)
```

Figure3.6.1

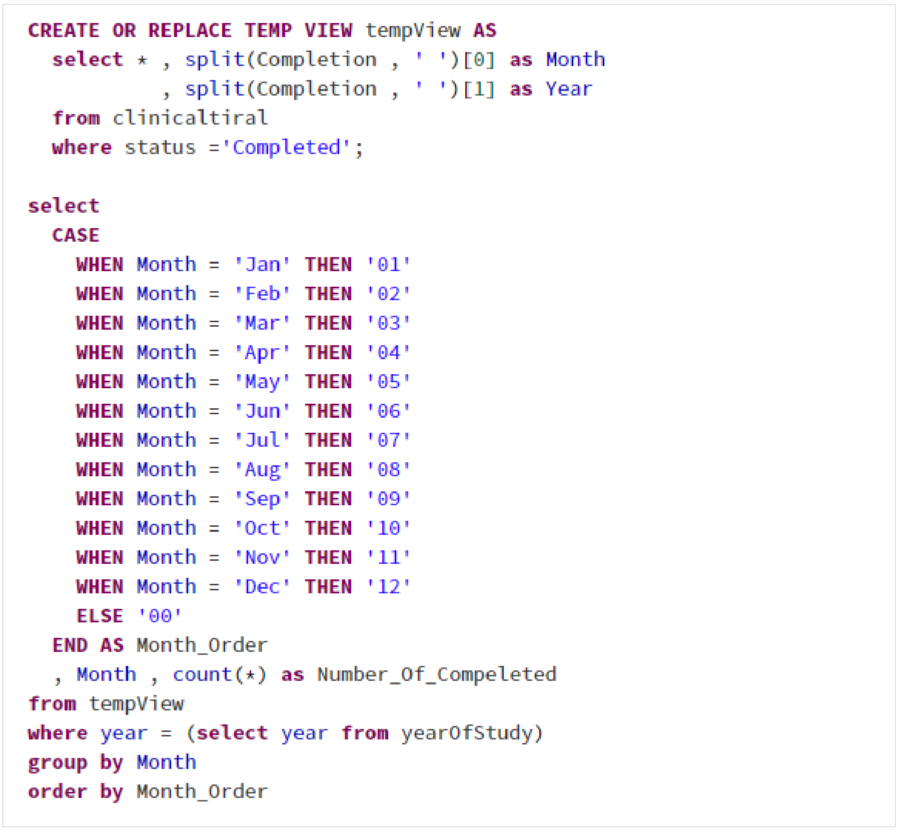


```
<< Number of completed studies per each month in 2021 >>
+-----+-----+-----------+
|month|count|month_order|
+-----+-----+-----------+
|Jan  |1131 |1          |
|Feb  |934  |2          |
|Mar  |1227 |3          |
|Apr  |967  |4          |
|May  |984  |5          |
|Jun  |1094 |6          |
|Jul  |819  |7          |
|Aug  |700  |8          |
|Sep  |528  |9          |
|Oct  |187  |10         |
+-----+-----+-----------+

Out[16]: <AxesSubplot:title={'center':'completed studies in 2021'}, xlabel='month'>
```

Figure3.6.2

3.6.3. HiveQL implementation

The same logic as explained before, is used to implement this question with HiveQL. Figure3.6.3 is the SQL code, and the following figure indicate the output bar plot in the given year 2021.



```sql
CREATE OR REPLACE TEMP VIEW tempView AS
  select * , split(Completion , ' ')[0] as Month
           , split(Completion , ' ')[1] as Year
  from clinicaltiral
  where status ='Completed';

select
  CASE
    WHEN Month = 'Jan' THEN '01'
    WHEN Month = 'Feb' THEN '02'
    WHEN Month = 'Mar' THEN '03'
    WHEN Month = 'Apr' THEN '04'
    WHEN Month = 'May' THEN '05'
    WHEN Month = 'Jun' THEN '06'
    WHEN Month = 'Jul' THEN '07'
    WHEN Month = 'Aug' THEN '08'
    WHEN Month = 'Sep' THEN '09'
    WHEN Month = 'Oct' THEN '10'
    WHEN Month = 'Nov' THEN '11'
    WHEN Month = 'Dec' THEN '12'
    ELSE '00'
  END AS Month_Order
  , Month , count(*) as Number_Of_Compeleted
from tempView
where year = (select year from yearOfStudy)
group by Month
order by Month_Order
```

Figure3.6.3

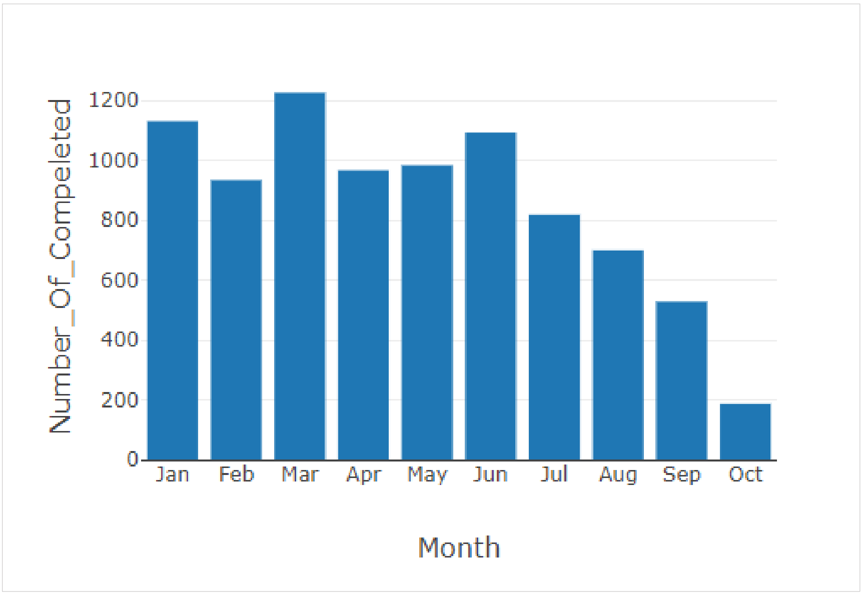
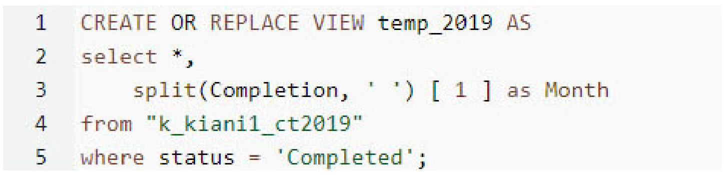


Figure3.6.4

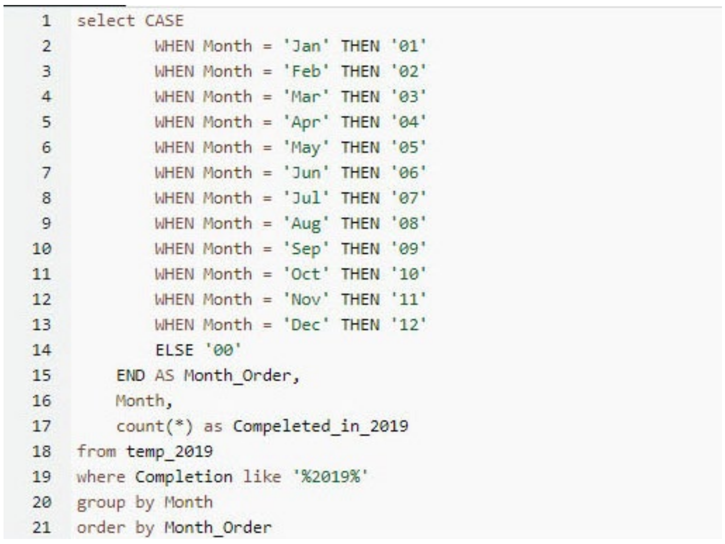
3.6.4. AWS implementation

In AWS Athena, the following code is written to create a view with splited value of "month". Then, the SQL query shown in figure3.6.6 is used to find the results.



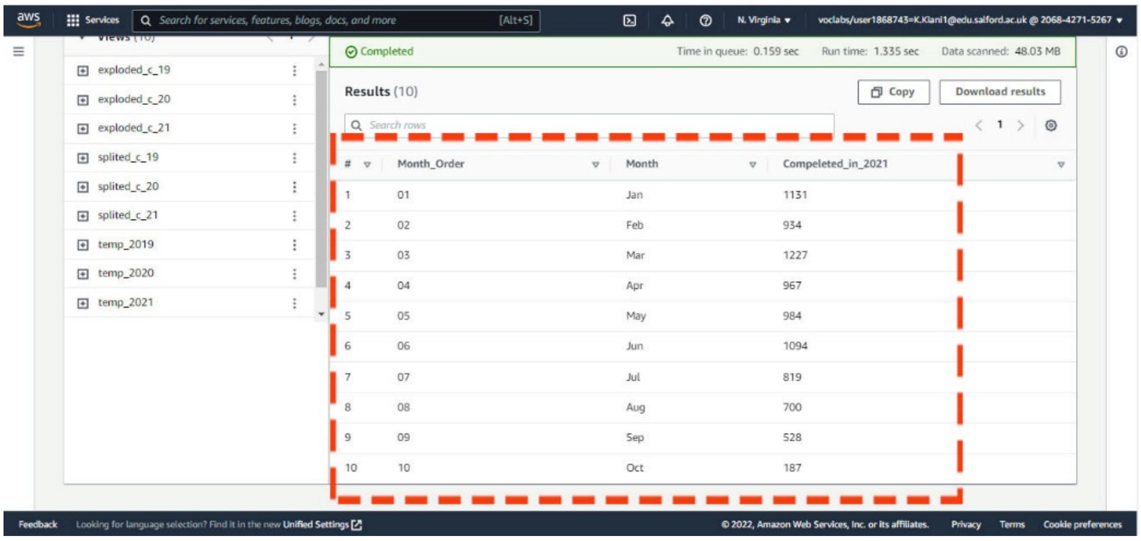
```sql
CREATE OR REPLACE VIEW temp_2019 AS
select *,
    split(Completion, ' ') [ 1 ] as Month
from "k_kiani1_ct2019"
where status = 'Completed';
```

Figure3.6.5



```sql
select CASE
        WHEN Month = 'Jan' THEN '01'
        WHEN Month = 'Feb' THEN '02'
        WHEN Month = 'Mar' THEN '03'
        WHEN Month = 'Apr' THEN '04'
        WHEN Month = 'May' THEN '05'
        WHEN Month = 'Jun' THEN '06'
        WHEN Month = 'Jul' THEN '07'
        WHEN Month = 'Aug' THEN '08'
        WHEN Month = 'Sep' THEN '09'
        WHEN Month = 'Oct' THEN '10'
        WHEN Month = 'Nov' THEN '11'
        WHEN Month = 'Dec' THEN '12'
        ELSE '00'
    END AS Month_Order,
    Month,
    count(*) as Compeleted_in_2019
from temp_2019
where Completion like '%2019%'
group by Month
order by Month_Order
```

Figure3.6.6


aws
Services
Search for services, features, blogs, docs, and more
[Alt+S]
N. Virginia
voclabs/user1868743=K.Kiani1@edu.salford.ac.uk @ 2068-4271-5267
Views (10)
exploded_c_19
exploded_c_20
exploded_c_21
splited_c_19
splited_c_20
splited_c_21
temp_2019
temp_2020
temp_2021
Copy
Download results
Search rows
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Completed Time in queue: 0.159 sec Run time: 1.335 sec Data scanned: 48.03 MB

Results (10)

#	Month_Order	Month	Compeleted_in_2021
1	01	Jan	1131
2	02	Feb	934
3	03	Mar	1227
4	04	Apr	967
5	05	May	984
6	06	Jun	1094
7	07	Jul	819
8	08	Aug	700
9	09	Sep	528
10	10	Oct	187

Figure3.6.7

3.6.5. Results and Discussion

Table3.6.1 illustrates the resulting values for this question. To have a better sight and a comparison view for these values, Microsoft Excel is used to draw 2 diagrams show in figures 3.6.8, 3.6.9. The trend shows us, the most completed studies have been done after April 2019. The 2020 year has the least values in each month between these 3 years.

Month	2019	2020	2021
Jan	1368	1544	1131
Feb	1149	1286	934
Mar	1470	1740	1227
Apr	1368	1080	967
May	1342	1176	984
Jun	1647	1424	1094
Jul	1547	1237	819
Aug	1406	1126	700
Sep	1421	1167	528
Oct	1310	1176	187
Nov	1223	1078	-
Dec	2690	2084	-

Table3.6.1

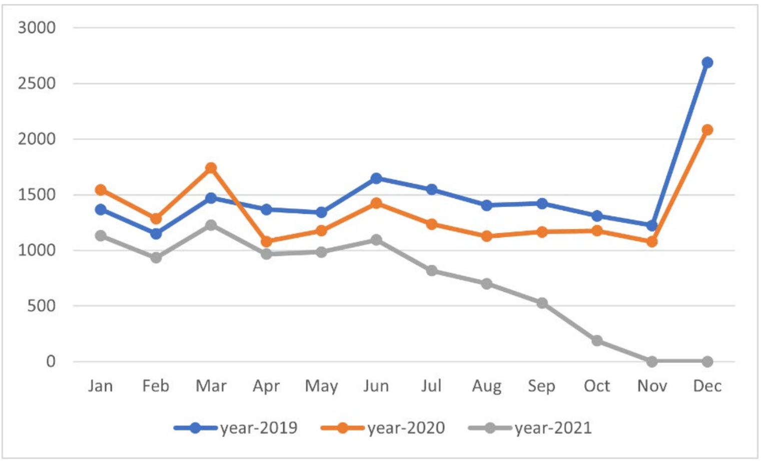


Figure3.6.8

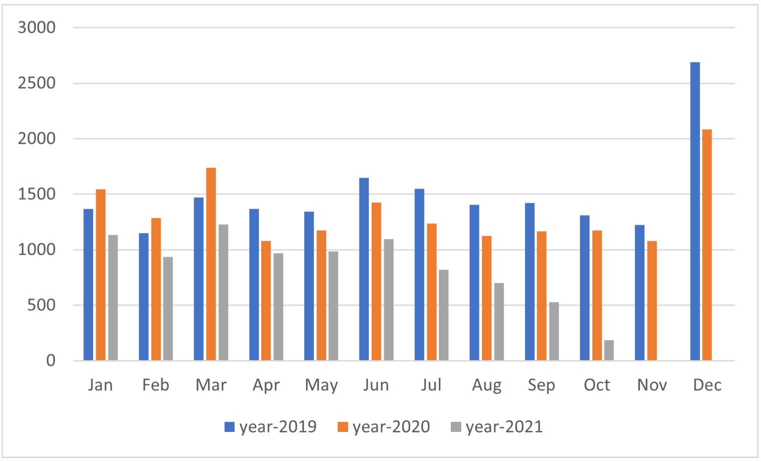


Figure3.6.9

3.7. Question 7 - further analysis

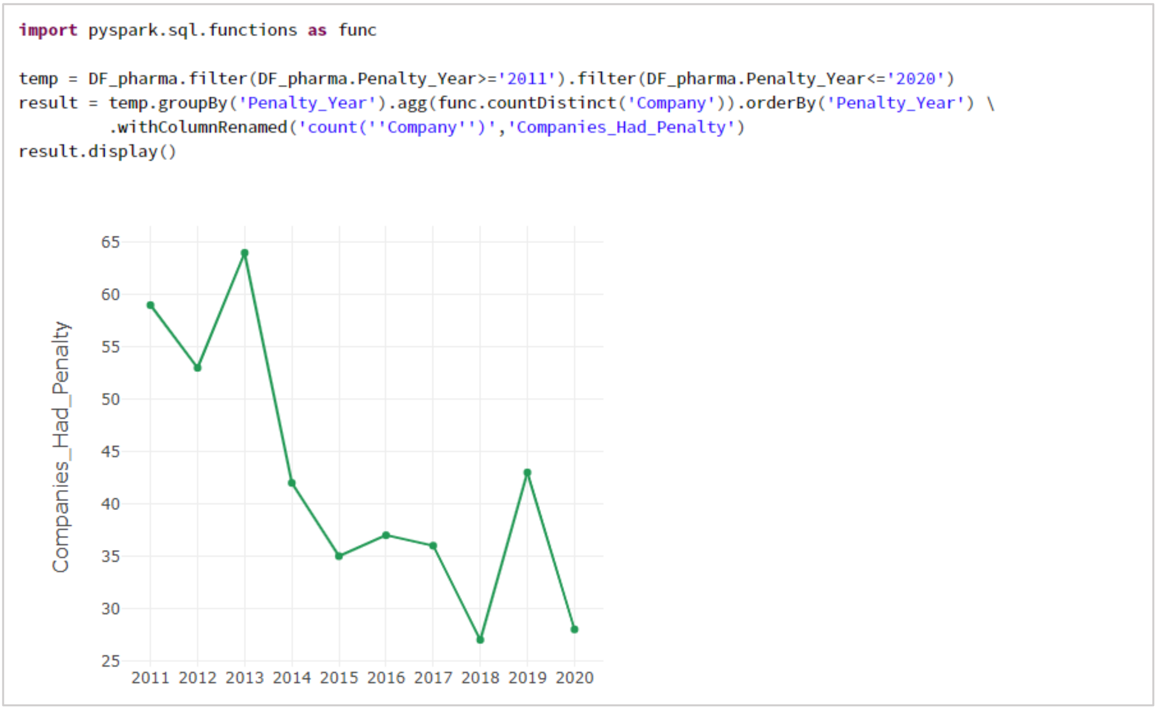
3.7.1. Assumptions

Question 7 (Extra): The number of different companies which they had penalty and plotting the trend in a 10 year period from 2011 to 2020.

The aim of this further question is to draw the 10-year trend for number of companies who paid any penalty. To achieve this goal, we have to refer to pharma dataset and filter the records that occur between 2011 and 2020. Then, by grouping and counting techniques we will find the required values.

3.7.2. PySpark implementation – Dataframe

Figure3.7.1 is the implementation and output trend digram using PySpark. First the dataframe is filtered for records between 2011 and 2020 according to the "penalty_year". Then, we count the distinct companies based on the grouped "penalty_year". Finally, the trend is shown between years 2011 and 2020.

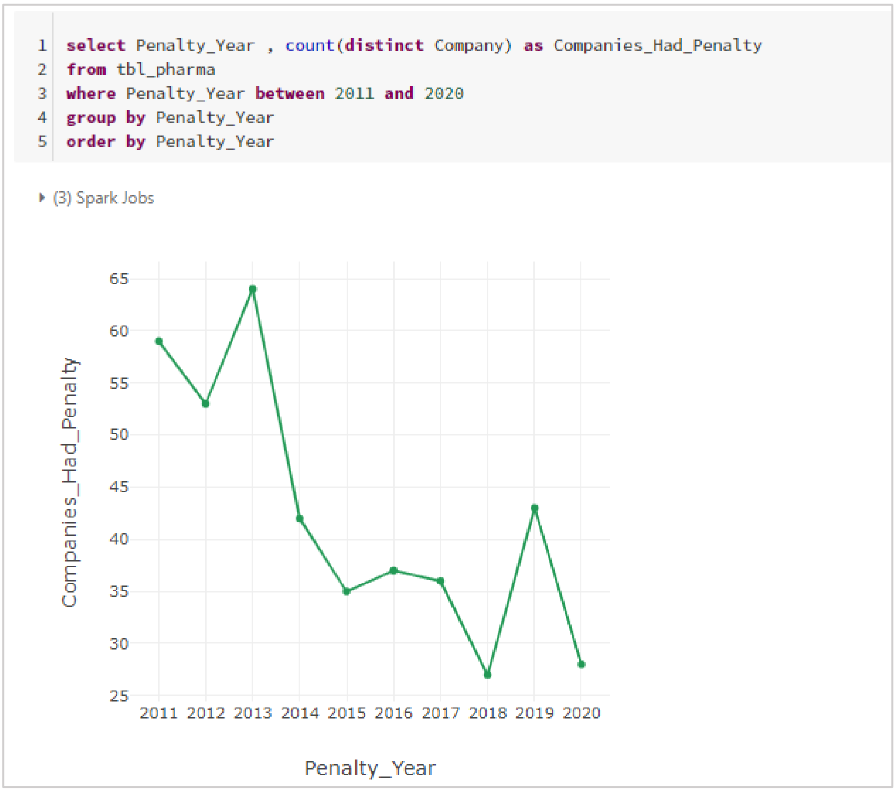


```
import pyspark.sql.functions as func

temp = DF_pharma.filter(DF_pharma.Penalty_Year>='2011').filter(DF_pharma.Penalty_Year<='2020')
result = temp.groupBy('Penalty_Year').agg(func.countDistinct('Company')).orderBy('Penalty_Year') \
         .withColumnRenamed('count(''Company'')','Companies_Had_Penalty')
result.display()
```

Figure3.7.1

3.7.3. HiveQL implementation

The HiveQL implementation for this question is indicated in figure3.7.2 and the resulting diagram is also visible in this figure. The same logic as explained in previous section is used with HiveQL to draw the trend diagram.



```
select Penalty_Year , count(distinct Company) as Companies_Had_Penalty
from tbl_pharma
where Penalty_Year between 2011 and 2020
group by Penalty_Year
order by Penalty_Year
```

Figure3.7.2

3.7.4. Results and Discussion

Figure3.7.3 indicates the trend for number of companies who had penalty in a 10-year period. It is clear in this diagram that, the most penalty number was in 2013 with 64 penalties, and after this year a dramatic drop in penalty numbers occurrs by 27 until 2018 which is the least number of penalties in this year.

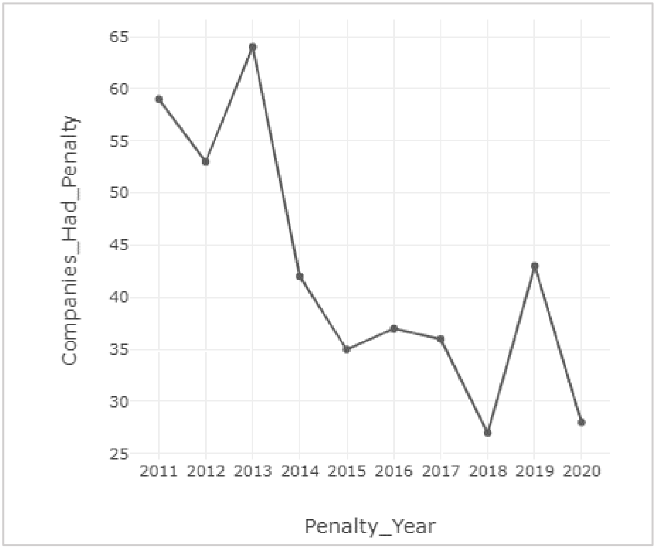


Figure3.7.3