|  |  |  |
| --- | --- | --- |
| Analysis of Clinical Trial Data | Text  Description automatically generated | |
| ASIF JAVED | | @00653639 |

Contents

[Part 1 – Setup required for analysis 2](#_Toc102948900)

[Part 2 – Data Cleaning and Preparation 2](#_Toc102948901)

[Part 3 – Problem Answers 3](#_Toc102948902)

[Visualization: 16](#_Toc102948903)

[Visualization in RDD implementation: 16](#_Toc102948904)

[Visualization in DF implementation: 16](#_Toc102948905)

[Visualization in HQL implementation: 16](#_Toc102948906)

[Instructions to Run: 17](#_Toc102948907)

Analysis of clinical trial data

# Part 1 – Setup required for analysis

To analyze the clinical trial dataset, I have used Databricks to work with Pyspark and HQL using the Python & SQL notebooks to execute the code in a web based interactive interface. Different python libraries are used to extract the conclusions. I have also attached my code for each answer from Q1 to Q6. Each code segment is written with comments to explain the answers.

# Part 2 – Data Cleaning and Preparation

Dataset for clinical trial was provided in compressed csv file. To use the dataset in databricks environment I have uploaded the dataset in databricks FileStore>tables directory.

![Graphical user interface, application, email

Description automatically generated]()

After data uploading, I have uncompressed the dataset to csv format for performing the actions. To uncompressing the data I have used the unix command in python notebook and first I moved the compressed file to databricks FileStore/tmp directory and after uncompressing I again moved the file to its original directory dbfs/tables directory. For this whole process I have I have used a fileroot variable which performed the taking file path, moving and uncompressing actions.

![Graphical user interface, text, application, email

Description automatically generated]()

![Graphical user interface, text, application, Teams

Description automatically generated]()

These steps are done in PySpark both RDD & Dataframe. To abstract the data, data cleaning of the csv file was compulsory. So, I have I went through the dataset and removed/skipped the header and split the dataset using delimiters pipe.

For running the dataset throughout the file I have declared two variables one (fileroot) for file name and other (Year) for year as dataset year for relevant and accurate results.

# Part 3 – Problem Answers

1. **Problem Statement I :**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv)

Count: 387261 #Total number of distinct studies

**Code Segments Used in RDD Implementation:**

According to problem statement dataset has studies which involves the clinical trial analysis. To extract the distinct studies from the dataset I have assigned the dataset to the RDD2021 and for accurate results I have skipped the first row as it’s the header by filtering the RDD2021. Then I have taken the distinct count of the dataset studies.

RDD2021= sc.textFile('FileStore/tables/'+fileroot)

RDD2021\_New = RDD2021

RDD2021\_Header = RDD2021\_New.first() #assiging first row to RDD

RDD2021\_Count = RDD2021.filter(lambda Header: Header!=RDD2021\_Header) #skipping the header by filtering RDD

RDD2021\_Count.distinct().count() #taking distinct count from the RDD data

**Screenshot:**

![Graphical user interface, text, application, email

Description automatically generated]()

**Code Segments Used in DF Implementation:**

I have loaded the file in DF2021 and made further steps for header, schema and delimiter pipe. Then took the count for studies and results are accurate as per RDD.

ClinicalTrial2021= ('dbfs:/FileStore/tables/'+fileroot)

DF2021 = spark.read.format('csv')\

.options(header='true', inferSchema='true')\

.options(delimiter='|')\

.load(ClinicalTrial2021) #loading and ready the dataset in dataframe

DF2021.count() #taking the count of dataset

**Screenshot:**

![Graphical user interface, text, application, email

Description automatically generated]()

**Code Segments Used in HQL Implementation:**

I have made an external table clinicaltrial\_2021 by specifying the schema and inserted the data from the dataset. And choose the studies count from the table.

CREATE EXTERNAL TABLE IF NOT EXISTS clinicaltrial\_2021 #Creating table using the clinicaltrial\_2021 datset

( Id STRING ,

Sponsor STRING ,

Status STRING ,

Start STRING ,

Completion STRING ,

Type STRING ,

Submission STRING ,

Conditions STRING ,

Interventions STRING )

USING CSV

OPTIONS (path 'dbfs:/FileStore/tables/$fileroot',

delimiter "|",

header "true");

select Distinct count(\*) from clinicaltrial\_2021; #Showing count of dataset

**Screenshot:**

![Graphical user interface, text, application, email

Description automatically generated]()

1. **Problem Statement II :**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv)

[('Interventional', **301472**),

('Observational', **77540**),

('Observational [Patient Registry]', **8180**),

('Expanded Access', **69**)]

**Code Segments Used in RDD Implementation:**

I have splitted the RDD2021\_Split by pipe ‘|’. Then I have taken the 6th index column name ‘type’ to check the types with frequencies. Then I used reduceByKey function for grouping the same keys and adding their values. As per requirement I sorted the final results in descending order for most frequent types at top.

RDD2021\_Split=RDD2021.map(lambda Word:Word.split('|')) #RDD Splitting on pipe

RDD2021\_Type=RDD2021\_Split.map(lambda index: index[5])\

.map(lambda x:(x,1)).reduceByKey(lambda v1,v2:v1+v2)\ #reducing same keys by adding their values

.sortBy(lambda v:v[1],ascending =False) #sorting RDD

RDD2021\_Type.take(4)

**Screenshot:**

**![Graphical user interface, text, application, email

Description automatically generated]()**

**Code Segments Used in DF Implementation:**

I grouped the Type column by taking their count and then ordered it in descending.

DF2021\_Count1=DF2021.groupby(['Type']).count() #grouping the same keys and taking count

DF2021\_Count=DF2021\_Count1.orderBy(['count'], ascending=False) #sorting the dataframe

DF2021\_Count.display()

**Screenshot:**

**![Graphical user interface, table

Description automatically generated]()**

**Code Segments Used in HQL Implementation:**

I made a query of clinicaltrial\_2021 table and grouped all the types from type column by taking the count and then ordered it as per desired output.

select Type, count(Type) as count from clinicaltrial\_2021 #selecting the type column and its count

group by Type

order by count

desc limit 4

**Screenshot:**

![Graphical user interface, table

Description automatically generated]()

1. **Problem Statement III :**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv)

[('Carcinoma', 13389),

('Diabetes Mellitus', 11080),

('Neoplasms', 9371),

('Breast Neoplasms', 8640),

('Syndrome', 8032)]

**Code Segments Used in RDD Implementation:**

For problem statement 3 I have mapped the each value from conditions column and splitted it against comma delimiter and skipped the nulled values by filtering the RDD. The I have mapped the RDD and assigned 1 to each key. reduceByKey function is used for grouping the same keys and adding their values and sorted the RDD on values in descending order.

RDD2021\_Disease1=RDD2021\_Split.flatMap(lambda index: index[7].split(','))\ #splitting 8th column on comma

.filter(lambda c: c != '') #skipping null values

RDD2021\_Disease=RDD2021\_Disease1.map(lambda x:(x,1))\ #assigning 1 value to each key

.reduceByKey(lambda v1,v2:v1+v2)\ #reducing keys by adding their values

.sortBy(lambda v:v[1],ascending =False) #sorting the RDD

RDD2021\_Disease.take(5)

**Screenshot:**

Graphical user interface, text, application, email

Description automatically generated

**Code Segments Used in DF Implementation:**

I splitted the column condition on comma and used the explode function to list all the items on row from the column to new column named conditions. Then I grouped the conditions and make the order descending of the conditions count.

from pyspark.sql.functions import col, split, explode

DF2021\_Split\_Conditions= DF2021.withColumn('conditions', explode(split(DF2021['conditions'],',')))

#splitting the column and exploding the values

DF2021\_Conditions\_Count1=DF2021\_Split\_Conditions.groupby(['conditions']).count()

#grouping by same type and taking count

DF2021\_Conditions\_Count=DF2021\_Conditions\_Count1.orderBy(['count'], ascending=False) #sorting/ordering

DF2021\_Conditions\_Count.display()

**Screenshot:**

![Table

Description automatically generated]()

**Code Segments Used in HQL Implementation:**

I created a view in which I splitted the column and used explode function for converting the list into row in a new column named explode\_condition. I skipped the nulled values by checking the length of each row and make a count by grouping the conditions and sorted it as per desired results.

CREATE OR REPLACE VIEW ConditionSplit\_view

AS

SELECT id, explode\_codition

FROM clinicaltrial\_2021

lateral VIEW explode(split(Conditions,",")) condition\_view AS explode\_codition

where length(explode\_codition) > 0 ;

select explode\_codition,count(explode\_codition) as count

from ConditionSplit\_view

group by explode\_codition

order by count desc limit 6

**Screenshot:**

![Graphical user interface, table

Description automatically generated]()

1. **Problem Statement IV :**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv and Mesh.csv)

[('C04', 143994),

('C23', 136079),

('C01', 106674),

('C14', 94523),

('C10', 92310),

('C06', 85646),

('C08', 70720),

('C13', 42599),

('C18', 41276),

('C12', 40161)]

**Code Segments Used in RDD Implementation:**

Each condition has its own hierarchy code. To get access with it I have used mesh file and cleaned it by skipping the header and splitting it on comma and full stop to get the desired part of the code related to condition. Then I joined the clinicaltrial\_20211’s conditions with cleaned mesh file to join conditions and codes. Then I used reducebykey to group the similar keys and added their values. And sorted it in descending order to show the most frequent at top.

RDDmesh\_Header = RDDmesh.first()

RDDmesh\_Header\_Remove = RDDmesh.filter(lambda Header: Header!=RDDmesh\_Header)

RDDmesh\_Split = RDDmesh\_Header\_Remove.map(lambda x:(x.split(",")[0], x.split(",")[1].split(".")[0]))

RDDmesh\_Split.take(2)

RDD2021\_Recount= RDD2021\_Disease1.map( lambda x:[x,1] )

RDD2021\_DiseaseCode = RDD2021\_Recount.join(RDDmesh\_Split).map(lambda y: (y[1][1], 1))\

.reduceByKey(lambda v1, v2: v1+ v2).sortBy( lambda x: -x[1] )

RDD2021\_DiseaseCode.take(10)

**Screenshot:**

**![Graphical user interface, text, application, email

Description automatically generated]()**

**Code Segments Used in DF Implementation:**

I have loaded the mesh file into DFmesh by defining the schema and delimiter. Then to take code part from mesh file I have used substr function by taking first three letters from the code with a new column name NewCode. Then I compared the condition file and mesh file on the basis of conditions and joined both dataframes. Now each condition has its own code and then I have group them and made the count and sorted it in descending order.

mesh=('dbfs:/FileStore/tables/mesh.csv')

DFmesh = spark.read.format('csv')\

.options(header='true', inferSchema='true')\

.options(delimiter=',')\

.load(mesh)

display(DFmesh)

DFmesh\_Code= DFmesh.withColumn('NewCode', DFmesh.tree.substr(1,3))

DFmesh\_Join1= DF2021\_Split\_Conditions.conditions == DFmesh\_Code.term

DFmesh\_Join= DF2021\_Split\_Conditions.join(DFmesh\_Code, on = DFmesh\_Join1, how='inner')

#joining 2 DFs to get code

DFmesh\_Disease\_Code=DFmesh\_Join.select('NewCode').groupby(['NewCode']).count().orderBy(['count'],

ascending= False)

DFmesh\_Disease\_Code.display()

**Screenshot:**

**![Graphical user interface, application

Description automatically generated]()**

**Code Segments Used in HQL Implementation:**

I have created table using the mesh file. Then I created a view from code by using substring function . then I joined the condition column and disease code where both conditions are same. Then I made a count, grouped by same keys and sorted it according to requirement.

CREATE External TABLE IF NOT EXISTS mesh ( #creating table from mesh file

term STRING ,

tree STRING )

USING CSV

OPTIONS ( path "dbfs:/FileStore/tables/mesh.csv",

delimiter ",",

header "true" );

CREATE OR REPLACE VIEW disease\_code As

select term,tree,substr( tree,1,3 ) as disease\_code from mesh #using substring function to get first 3 letters of code

CREATE OR REPLACE VIEW joinDiseasecodeView As

SELECT c.id,c.explode\_codition,d.disease\_code

FROM ConditionSplit\_view as c

JOIN disease\_code as d ON c.explode\_codition=d.term

select disease\_code, count(disease\_code) as count

from joinDiseasecodeView

group by disease\_code

order by count desc limit 10

**Screenshot:**

**![Table

Description automatically generated]()**

1. **Problem Statement V**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv and Pharma.csv)

[('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)]

**Code Segments Used in RDD Implementation:**

In this implementation Pharma file has all the pharmaceutical companies list but in clinical trial file there are some sponsors which were not pharmaceutical company. So I used the sponsor column from clinicaltrial file and skipped the nulled fields by filtering the RDD. I used leftouterjoin function to pick the unmatched sponsor companies from clinicaltrial file which were not in Pharma file. Then I again filtered the resulted RDD to skipped the null fields. Reducebykey function and count function used to find the frequency of each sponsor for trial study.

RDD2021\_Sponsor = RDD2021\_Split.map(lambda s: s[1]) \

.filter(lambda c: c != '') \

.map(lambda x: (x, 1))

RDDpharma\_Split=RDDpharma.map(lambda x: x.replace('"','').split(',')[1]) \ #splitting the rdd’s 2nd index on comma

.map(lambda v: (v,1))

RDD2021\_Joined = RDD2021\_Sponsor.leftOuterJoin(RDDpharma\_Split).filter(lambda x : x[1][1] == None) \

.map(lambda y: (y[0], 1)) \ #joining 2 RDDs by left out join function to get values of 1st Rdd which are not in 2nd

.reduceByKey(lambda v1, v2: v1+ v2).sortBy(lambda x: -x[1]) #reducing keys by adding its values and sorting

RDD2021\_Joined.take(10)

**Screenshot:**

**![Graphical user interface, text, application, email

Description automatically generated]()**

**Code Segments Used in DF Implementation:**

I loaded the pharma file and read it as csv file by defining the schema and delimiter.Then compared and joined the clinicaltrial’s sponsor column and pharma file’s parent company column. By using the where statement I checked the values where sponsor is not matched with parents company column. Then I grouped by sponsor and order it to extract the desired results.

pharma=('dbfs:/FileStore/tables/pharma.csv')

DFpharma=spark.read.format('csv') \

.options(header='true', inferSchema='true') \

.options(delimiter=',') \

.load(pharma)

DFpharma.display()

DFpharma\_Join1=DF2021.Sponsor == DFpharma.Parent\_Company

DFpharma\_Join= DF2021.join(DFpharma, on = DFpharma\_Join1, how='left')

DFpharma\_Join.display()

DFpharma\_Sponsor1= DFpharma\_Join.select('Sponsor', 'Parent\_Company').where('Parent\_Company is null')

DFpharma\_Sponsor=DFpharma\_Sponsor1.groupby(['Sponsor']).count().orderBy(['count'], ascending=False)

DFpharma\_Sponsor.display()

**Screenshot:**

**![Graphical user interface, text, application, email

Description automatically generated]()**

**Code Segments Used in HQL Implementation:**

I have created table using the pharma file. Then I created to view to check the different values by using the leftouterjoin function which were not in pharma file but in clinicaltrial file’s sponsor column. By grouping the sponsor and making the count I got the desired output as per client’s requirement.

CREATE External TABLE IF NOT EXISTS pharma ( #Creating the table

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 (path "dbfs:/FileStore/tables/pharma.csv",

delimiter ",",

header "true" ) ;

CREATE OR REPLACE VIEW joinPharmaCompany #creating view

AS

select c.\*,p.Parent\_Company

from clinicaltrial\_2021 as c LEFT OUTER JOIN pharma p

#by using leftouterjoin function taking first table’s column values which are not in 2nd table’s column

ON (c.Sponsor=p.Parent\_Company) #condition for joining

select

Sponsor,count(Sponsor)as count

from joinPharmaCompany

where

Parent\_Company is null group by sponsor order by count(Sponsor) desc Limit 10

**Screenshot:**

![Table

Description automatically generated]()

1. **Problem Statement VI :**

**Answer**:

(dataset used to produce answer: clinicaltrial\_2021.csv, Mesh.csv and Pharma.csv)

[('Jan', 1131),

('Feb', 934),

('Mar', 1227),

('Apr', 967),

('May', 984),

('Jun', 1094),

('Jul', 819),

('Aug', 700),

('Sep', 528),

('Oct', 187)]

**Code Segments Used in RDD Implementation:**

For this problem statement in RDD implementation I have used the filtered RDD without header and splitted the completion column to separate the month and year. Also I skipped the nulled values. Then I filtered the resulted RDD to take the data where study status was completed and year were 2021. The used the reducebykey function to take the completed study count.

I have used the pythons library calendar to sort the months from January to December.

RDD1 = RDD2021\_Split.first()

RDD2021\_Count = RDD2021\_Split.filter(lambda Header: Header!=RDD1) #skipping header

RDD2021\_Study\_Completion=RDD2021\_Count.map(lambda x: (x[4].split(),x[2]))\

.filter(lambda x: len(x[0])!=0)\ #skipping null values/index

.filter(lambda y:(y[1]=='Completed'))\

.filter(lambda z: (z[0][1]==Year))\

.map(lambda a: (a[0][0],1))\

.reduceByKey(lambda v1, v2: v1+ v2)

RDD2021\_Study\_Completion.take(12)

import calendar #python library calender to sort the months/keys sorting

Sorting = {a:z for z,a in enumerate(calendar.month\_abbr[1:],1)}

RDD2021\_Study\_Completion\_Sorting=RDD2021\_Study\_Completion.sortBy(lambda g: Sorting.get(g[0]))

RDD2021\_Study\_Completion\_Sorting.collect()

**Screenshot:**

![Graphical user interface, text, application, email

Description automatically generated]()

**Code Segments Used in DF Implementation:**

I have splitted the column completion to get the month and year and took the data from status column where study was completed. Then I used groupby function to take the count of completed study count month wise in 2021.

To sort the month from January to December I used the sql function for custom sorting to get the desired output.

DF2021\_Study1=DF2021.withColumn('Month', split(DF2021['Completion'], ' ').getItem(0))\

.withColumn('year', split(DF2021['Completion'], ' ').getItem(1))

DF2021\_Study1.show()

DF2021\_Study\_Filter=DF2021\_Study1.filter((DF2021\_Study1.Status=='Completed') &

(DF2021\_Study1.year==Year)).groupby('Month').count()

DF2021\_Study\_Filter.show()

#importing sql function to sort months from Jan to Dec

![Graphical user interface, text, application, email

Description automatically generated]()

**Screenshot:**

Graphical user interface, text, application, Word

Description automatically generated

**Code Segments Used in HQL Implementation:**

I have created a view where I get the month and year from a column completion by splitting the values. Again I created a view to get the status completed and I used a global variable to get the year. Then I grouped the keys to get the count of completed studies.

Case function was used to sort the months.

CREATE OR REPLACE VIEW month\_year

AS

select \*, split(completion,' ')[0] as Month, split(completion,' ')[1] as Year from clinicaltrial\_2021 where completion is not null

CREATE OR REPLACE VIEW counting\_view

AS

select month,count(month)as count from month\_year where Status=='Completed' and Year=='$Year' group by month

#creating view and using CASE function to sort months

![Table

Description automatically generated]()

select month,count from final\_month\_view

**Screenshot:**

![Table

Description automatically generated]()

# Visualization:

# Visualization in RDD implementation:

I have used the matplotlib library of python. To show months and study count on axis I have mapped the final RDD and collected its each value. Multiple graphs for visualization is used to show the analysis.

|  |  |  |
| --- | --- | --- |
| Chart, bar chart  Description automatically generated | Chart  Description automatically generated | Chart, line chart  Description automatically generated |

# Visualization in DF implementation:

I used matplot library of python here also. And different settings and configuration is used to visualize the final dataframe.

|  |  |  |
| --- | --- | --- |
| Chart, scatter chart  Description automatically generated | Chart, histogram  Description automatically generated | Chart, bar chart  Description automatically generated |

# Visualization in HQL implementation:

For visualization in Hive SQL I have used the built available plotting options. Just selecting the multiple plot layouts I represent the data.

|  |  |
| --- | --- |
| Chart, bar chart  Description automatically generated | Chart, line chart  Description automatically generated |

# Instructions to Run:

Use web based cloud platform Databricks for running the codes. This has the python and sql notebook interface. Upload/import the .ipynb file into databricks workspace and execute it.