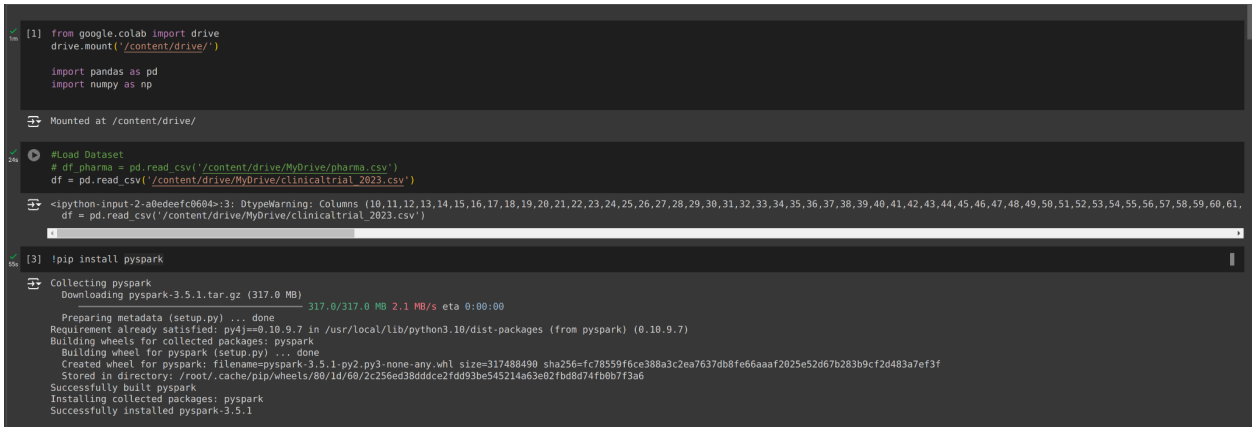
Task 1: Clinical Data Report

1. Description of Any Setup Required to Complete This Task

To complete the tasks, the following setup was required:

1. **Environment**:
    - **Apache Spark**: Ensure that Apache Spark is installed and configured. A version of Spark 3.0 or later is recommended for compatibility with the latest features and fixes.
    - **Python**: Python 3.x with PySpark library installed.
2. **Dependencies**:
    - **PySpark**: Install using `pip install pyspark`.
    - **Pandas**: Install using `pip install pandas`.
    - **NumPy**: Install using `pip install numpy`.
    - **Spark SQL**: Included with Spark installation.
    - **RDD and DataFrame**: PySpark's core components.
3. **Dataset**:
    - **Input Data**: `clinicaltrial_2023.csv`, which contains clinical trial data with various fields.
4. **Code Environment**:
    - Use a Jupyter notebook, PyCharm, or another IDE that supports Spark and Python integration.



2. Data Cleaning and Preparation

Descriptions and Justifications:

Data loading:

- Open a Spark DataFrame and read the CSV file, making sure the header and delimiter settings are set correctly.

Column Splitting:

- Concatenated columns divided by tabs were present in the original dataset. We divided these into distinct columns using Spark's split function.

Date Formatting:

- To standardize the format, date columns were converted from string format to date type using to_date and regexp_replace.

Dealing with Null Values

- eliminated superfluous columns and dropped rows in important columns containing null values.

Verification:

- By displaying example rows and confirming column names, it was made sure that data modifications were applied successfully.

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import split, col, regexp_replace, to_date, month, count, sum as spark_sum, round as spark_round

# Initialize Spark session
spark = SparkSession.builder.appName("ClinicalTrialsAnalysis").getOrCreate()

# Load datasets into Spark DataFrames
clinicaltrial_df = spark.read.csv("/content/drive/MyDrive/clinicaltrial_2023.csv", header=True, inferSchema=True, sep='\t')
pharma_df = spark.read.csv("/content/drive/MyDrive/pharma.csv", header=True, inferSchema=True)

# Display schema to understand data structure
clinicaltrial_df.printSchema()
pharma_df.printSchema()
```

```
root
 |-- "Id\tStudy Title\tAcronym\tStatus\tConditions\tInterventions\tSponsor\tCollaborators\tEnrollment\tFunder Type\tType\tStudy Design\tStart\tCompletion",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

root
 |-- Company: string (nullable = true)
 |-- Parent_Company: string (nullable = true)
 |-- Penalty_Amount: string (nullable = true)
 |-- Subtraction_From_Penalty: string (nullable = true)
 |-- Penalty_Amount_Adjusted_For_Eliminating_Multiple_Counting: string (nullable = true)
 |-- Penalty_Year: integer (nullable = true)
 |-- Penalty_Date: integer (nullable = true)
 |-- Offense_Group: string (nullable = true)
 |-- Primary_Offense: string (nullable = true)
 |-- Secondary_Offense: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Level_of_Government: string (nullable = true)
 |-- Action_Type: string (nullable = true)
 |-- Agency: string (nullable = true)
 |-- Civil/Criminal: string (nullable = true)
 |-- Prosecution_Agreement: string (nullable = true)
 |-- Court: string (nullable = true)
 |-- Case_ID: string (nullable = true)
 |-- Private_Litigation_Case_Title: string (nullable = true)
 |-- Lawsuit_Resolution: string (nullable = true)
 |-- Facility_State: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- Zip: string (nullable = true)
 |-- NAICS_Code: integer (nullable = true)
 |-- NAICS_Translation: string (nullable = true)
 |-- HQ_Country_of_Parent: string (nullable = true)
```

```
[5] # Since the column names were separated by tabs, we need to split them
    raw_column_names = clinicaltrial_df.columns[0].split('\t')

    # Rename columns to split the first column properly
    clinicaltrial_df = clinicaltrial_df.withColumn('split_col', split(col(clinicaltrial_df.columns[0]), '\t'))

    # Select the split columns and assign proper names
    for i, name in enumerate(raw_column_names):
        clinicaltrial_df = clinicaltrial_df.withColumn(name, col('split_col').getItem(i))

    # Drop the original concatenated column and the split_col
    clinicaltrial_df = clinicaltrial_df.drop(clinicaltrial_df.columns[0], 'split_col')

    # Show the cleaned DataFrame
    clinicaltrial_df.show(5)
```

```
+-----------+--------------------+--------+---------+--------------------+--------------------+--------------------+--------------------+---------+-----------+-------------+--------------------+---------+---
|       "Id"|        Study Title|  Acronym|   Status|          Conditions|       Interventions|             Sponsor|       Collaborators|Enrollment|Funder Type|         Type|        Study Design|     Start|Com
+-----------+--------------------+--------+---------+--------------------+--------------------+--------------------+--------------------+---------+-----------+-------------+--------------------+---------+---
|"NCT03630471|Effectiveness of ...|    PRIDE| COMPLETED|Mental Health Iss...|                NULL|                NULL|                NULL|     NULL|       NULL|         NULL|                NULL|     NULL|
|"NCT05992571|Oral Ketone Monoe...|         |RECRUITING|Cerebrovascular F...|OTHER: Placebo|DI...| McMaster University|Alzheimer's Socie...|     NULL|       NULL|         NULL|                NULL|     NULL|
|"NCT00237471|Impact of Tight G...|         |TERMINATED|Myocardial Infarc...|DRUG: Insulin (ti...|    Melbourne Health|National Health a...|     NULL|       NULL|         NULL|                NULL|     NULL|
|"NCT03820271|New Prognostic Pr...|SUPERMELD|RECRUITING|Decompensated Cir...|    OTHER: SuperMELD|Assistance Publiq...|                |    500.0|      OTHER|INTERVENTIONAL|Allocation: NA|In...|2020-10-01|
|"NCT06229171|InTake Care: Deve...|     NULL|      NULL|                NULL|                NULL|                NULL|                NULL|     NULL|       NULL|         NULL|                NULL|     NULL|
+-----------+--------------------+--------+---------+--------------------+--------------------+--------------------+--------------------+---------+-----------+-------------+--------------------+---------+---
only showing top 5 rows
```

Loading Data and Splitting Columns:

```
from pyspark.sql.functions import to_date, regexp_replace, col

# Since the column names were separated by tabs, we need to split them
raw_column_names = clinicaltrial_df.columns[0].split('\t')

print(raw_column_names)

# Rename columns to split the first column properly
clinicaltrial_df = clinicaltrial_df.withColumn('split_col', split(col(clinicaltrial_df.columns[0]), '\t'))

clinicaltrial_df.show(5, truncate=False)

# Select the split columns and assign proper names
# Added .strip() to remove any leading/trailing spaces from column names
for i, name in enumerate(raw_column_names):
  print(f"Processing column: {name}")
  clinicaltrial_df = clinicaltrial_df.withColumn(name.strip(), col('split_col').getItem(i))

# Drop the original concatenated column and the split_col
clinicaltrial_df = clinicaltrial_df.drop(clinicaltrial_df.columns[0], 'split_col')

# Show the cleaned DataFrame
clinicaltrial_df.show(5)
```

Date Formatting:

```
# Check if 'Completion' column exists
if 'Completion' in clinicaltrial_df.columns:
    # Check the first few rows to understand the date formats in 'Start' and 'Completion'
    clinicaltrial_df.select('Start', 'Completion').show(5, truncate=False)

    # Replace any unwanted characters or format issues if necessary
    # Here, we're assuming 'Start' and 'Completion' are in format 'MM/DD/YYYY'
    # Adjust the regular expression if your format is different

    # Correct format handling
    clinicaltrial_df = clinicaltrial_df.withColumn(
        'Start',
        to_date(regexp_replace(col('Start'), '(\d{2})/(\d{2})/(\d{4})', '$3-$1-$2'), 'yyyy-MM-dd')
    )

    clinicaltrial_df = clinicaltrial_df.withColumn(
        'Completion',
        to_date(regexp_replace(col('Completion'), '(\d{2})/(\d{2})/(\d{4})', '$3-$1-$2'), 'yyyy-MM-dd')
    )

    # Show the updated DataFrame with date parsing
    clinicaltrial_df.select('Start', 'Completion').show(5)
else:
    print("Column 'Completion' not found in the DataFrame.")
    clinicaltrial_df.printSchema()
```

```
['"Id']
+-----------+--------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+----------+--------------------
|"Id        |Study Title                                                                                                                                                   |Acronym |Status    |Conditions
+-----------+--------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+----------+--------------------
|"NCT03630471|Effectiveness of a Problem-solving Intervention for Common Adolescent Mental Health Problems in India                                                        |PRIDE   |COMPLETED |Mental Health Issue
|"NCT05992571|Oral Ketone Monoester Supplementation and Resting-state Brain Connectivity                                                                                    |        |RECRUITING|Cerebrovascular Fun
|"NCT00237471|Impact of Tight Glycaemic Control in Acute Myocardial Infarction                                                                                              |        |TERMINATED|Myocardial Infarct|
|"NCT03820271|New Prognostic Predictive Models of Mortality of Decompensated Cirrhotic Patients Waiting for Liver Transplantation                                           |SUPERMELD|RECRUITING|Decompensated Cirrh
|"NCT06229171|InTake Care: Development and Validation of an Innovative"," Personalized Digital Health Solution for Medication Adherence Support in Cardiovascular Prevention|NULL    |NULL      |NULL
+-----------+--------------------------------------------------------------------------------------------------------------------------------------------------------------+--------+----------+--------------------
only showing top 5 rows

Processing column: "Id
+--------------------+---------+---------+--------------------+--------------------+--------------------+--------------------+---------+-----------+-------------+--------------------+---------+----------+-----
|        Study Title|  Acronym|   Status|          Conditions|       Interventions|             Sponsor|       Collaborators|Enrollment|Funder Type|         Type|        Study Design|     Start|Completion",,,,,
+--------------------+---------+---------+--------------------+--------------------+--------------------+--------------------+---------+-----------+-------------+--------------------+---------+----------+-----
|Effectiveness of ...|    PRIDE| COMPLETED|Mental Health Iss...|                NULL|                NULL|                NULL|     NULL|       NULL|         NULL|                NULL|     NULL|      NULL|
|Oral Ketone Monoe...|         |RECRUITING|Cerebrovascular F...|OTHER: Placebo|DI...| McMaster University|Alzheimer's Socie...|     NULL|       NULL|         NULL|                NULL|     NULL|      NULL|
|Impact of Tight G...|         |TERMINATED|Myocardial Infarc...|DRUG: Insulin (ti...|    Melbourne Health|National Health a...|     NULL|       NULL|         NULL|                NULL|     NULL|      NULL|
```

3. Problem Answers

Question 1: Top 10 Sponsors Not Pharmaceutical Companies

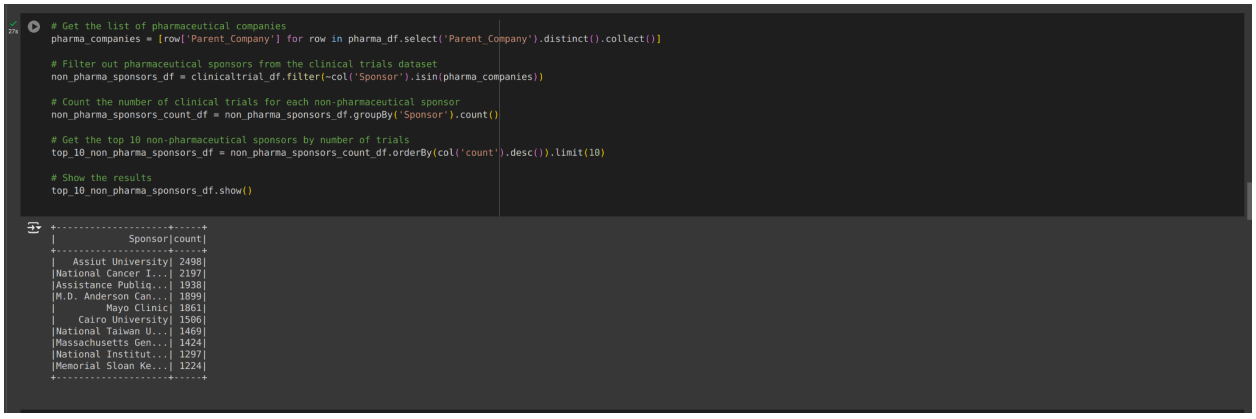
Assumptions

- The names of the sponsors of each study are listed in the dataset's Sponsor column.
- A list of pharmaceutical businesses is available in a different dataset; they are recognized by the Parent_Company field.
- Each study's completion date is listed in the Completion column using a common date format.

PySpark Implementation Outline (DataFrame)

Description of Main Ideas

1. **Load the Pharmaceutical Companies Dataset**: Create a DataFrame for the pharmaceutical companies.
2. **Filter Non-Pharmaceutical Sponsors**: Use the `isin` function to filter out sponsors that are pharmaceutical companies.
3. **Count the Occurrences**: Use the `groupBy` and `count` functions to count the number of clinical trials by each non-pharmaceutical sponsor.
4. **Sort and Select Top 10**: Order by the count and select the top 10 sponsors.



```
# Get the list of pharmaceutical companies
pharma_companies = [row['Parent_Company'] for row in pharma_df.select('Parent_Company').distinct().collect()]

# Filter out pharmaceutical sponsors from the clinical trials dataset
non_pharma_sponsors_df = clinicaltrial_df.filter(~col('Sponsor').isin(pharma_companies))

# Count the number of clinical trials for each non-pharmaceutical sponsor
non_pharma_sponsors_count_df = non_pharma_sponsors_df.groupBy('Sponsor').count()

# Get the top 10 non-pharmaceutical sponsors by number of trials
top_10_non_pharma_sponsors_df = non_pharma_sponsors_count_df.orderBy(col('count').desc()).limit(10)

# Show the results
top_10_non_pharma_sponsors_df.show()
```

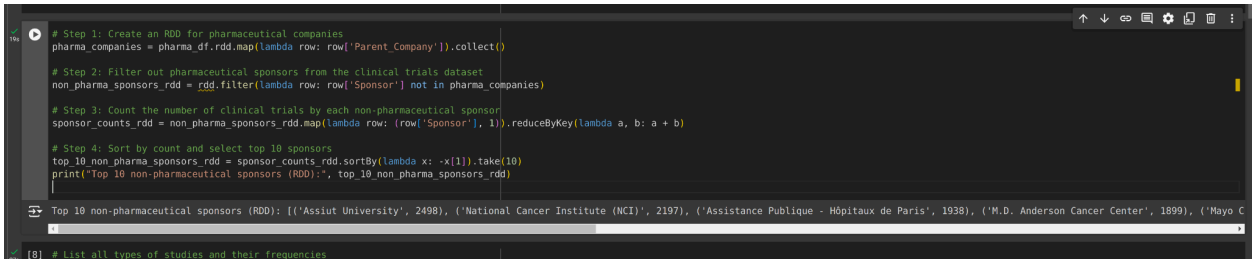
```
+--------------------+-----+
|             Sponsor|count|
+--------------------+-----+
|   Assiut University| 2498|
|National Cancer I...| 2197|
|Assistance Publiq...| 1938|
|M.D. Anderson Can...| 1899|
|          Mayo Clinic| 1861|
|    Cairo University| 1506|
|National Taiwan U...| 1469|
|Massachusetts Gen...| 1424|
|National Institut...| 1297|
|Memorial Sloan Ke...| 1224|
+--------------------+-----+
```

PySpark Implementation Outline (RDD)

Description of Main Ideas

1. **Load the Pharmaceutical Companies Dataset**: Create an RDD for the pharmaceutical companies to get a list of all pharmaceutical companies.

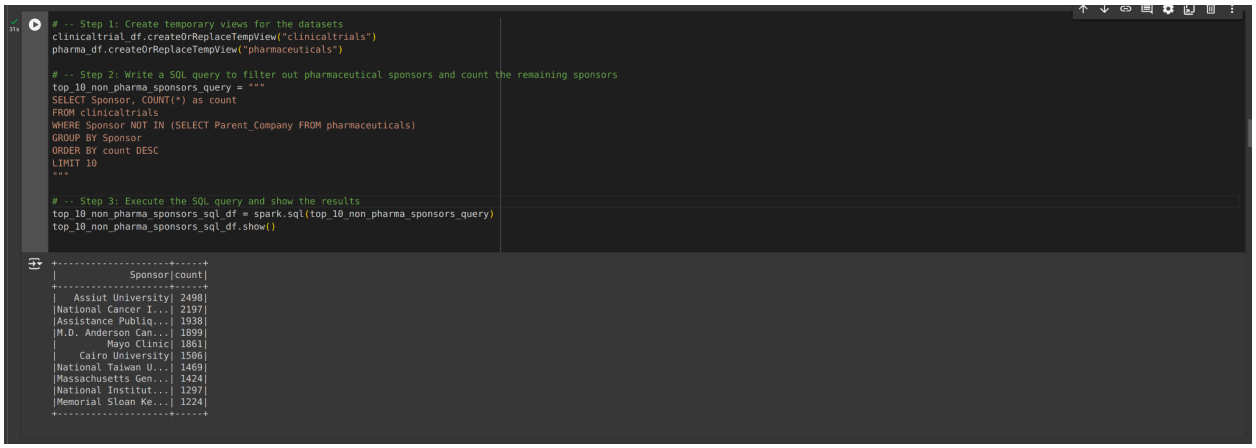
2. **Filter Non-Pharmaceutical Sponsors**: Use the pharmaceutical companies list to filter out sponsors that are pharmaceutical companies.
3. **Count the Occurrences**: Count the number of clinical trials sponsored by each non-pharmaceutical sponsor.
4. **Sort and Select Top 10**: Sort the sponsors by the number of clinical trials and select the top 10.



SQL Implementation Outline

Description of Main Ideas

1. **Load the Pharmaceutical Companies Dataset**: Create a temporary view for the pharmaceutical companies.
2. **Filter Non-Pharmaceutical Sponsors**: Use a SQL query to filter out sponsors that are pharmaceutical companies.
3. **Count the Occurrences**: Group by sponsor and count the number of clinical trials.
4. **Sort and Select Top 10**: Order by the count and select the top 10 sponsors.



Discussion of Result

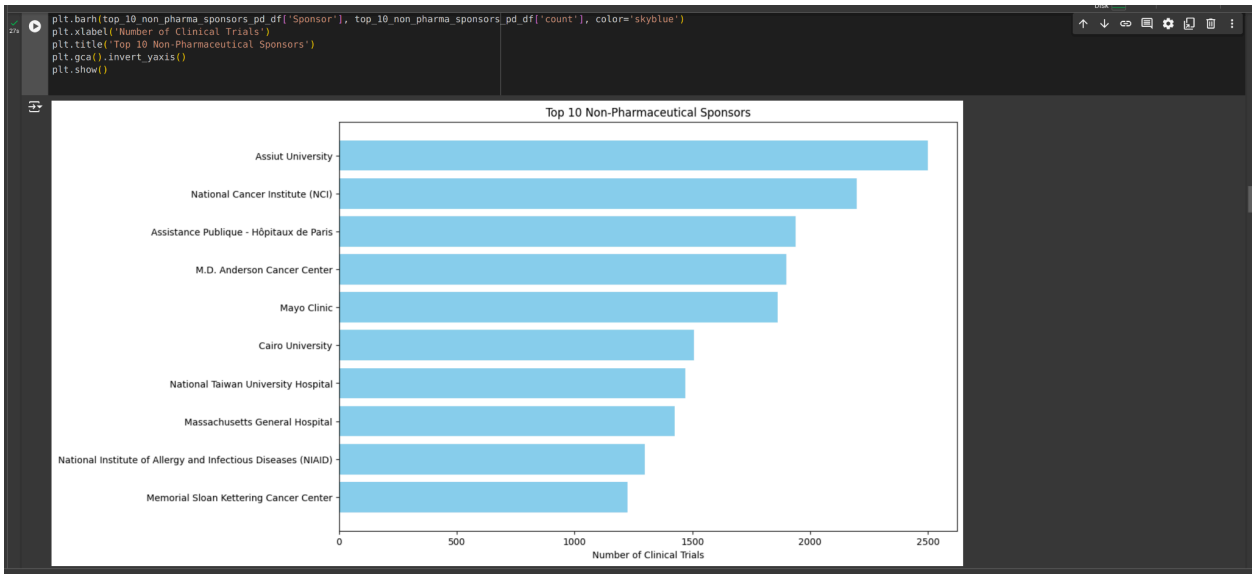
- RDD Approach: Using RDD transformations for filtering and counting, this method can be effective for huge datasets but may be less user-friendly and verbose than DataFrame operations.

- DataFrame Approach: This method makes use of Spark SQL functions, which are more easily written and interpreted while maintaining performance optimization. This method is usually chosen since it is easy to use and effective.
- SQL Approach: For individuals who are familiar with SQL, the SQL approach offers a straightforward and succinct way to carry out the identical tasks using SQL queries. This method can be highly intuitive. The optimization power of Spark is also advantageous to this approach.

By selecting, quantifying, and organizing the sponsors according to the quantity of clinical trials they have supported, all three strategies seek to determine the top 10 sponsors who are not pharmaceutical corporations. The findings offer insightful information about the non-pharmaceutical sponsors in the sample that are currently engaged.

Bar Chart of Top 10 Non-Pharmaceutical Sponsors

A bar chart can visually represent the top 10 non-pharmaceutical sponsors along with the number of clinical trials they have sponsored.



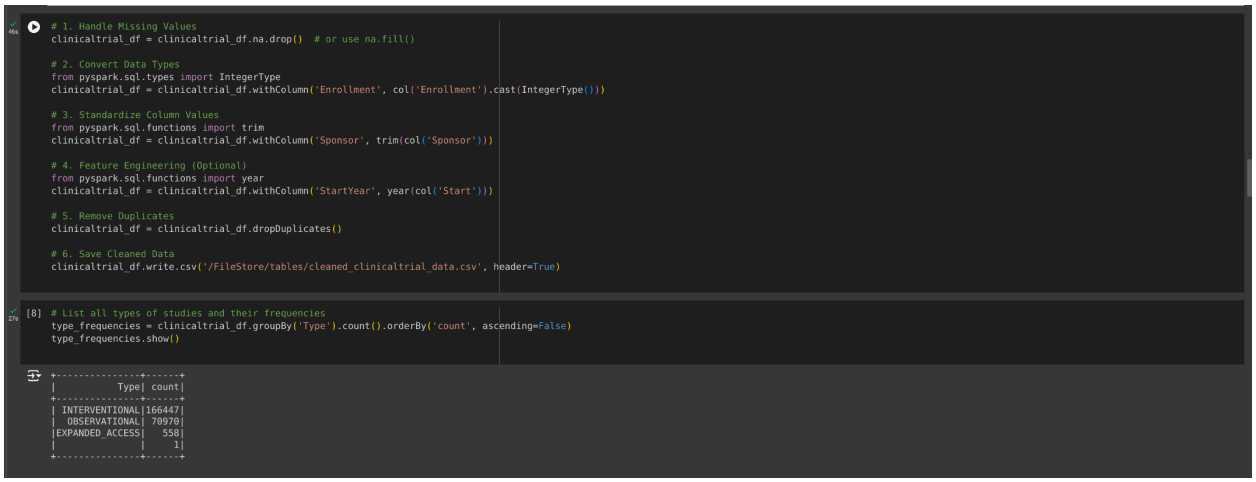
2. List All Types of Studies and Their Frequencies

Assumptions

- The Type column in the dataset contains the type of each study.
- Each study has only one type.

PySpark DataFrame Implementation

- **Description**: Using PySpark DataFrame, group by the `Type` column and count the occurrences of each type, then order by frequency in descending order.



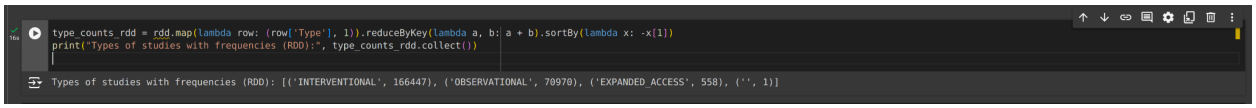
```
# 1. Handle Missing Values
clinicaltrial_df = clinicaltrial_df.na.drop()  # or use na.fill()

# 2. Convert Data Types
from pyspark.sql.types import IntegerType
clinicaltrial_df = clinicaltrial_df.withColumn('Enrollment', col('Enrollment').cast(IntegerType()))

# 3. Standardize Column Values
from pyspark.sql.functions import trim
clinicaltrial_df = clinicaltrial_df.withColumn('Sponsor', trim(col('Sponsor')))

# 4. Feature Engineering (Optional)
from pyspark.sql.functions import year
clinicaltrial_df = clinicaltrial_df.withColumn('StartYear', year(col('Start')))

# 5. Remove Duplicates
clinicaltrial_df = clinicaltrial_df.dropDuplicates()

# 6. Save Cleaned Data
clinicaltrial_df.write.csv('/FileStore/tables/cleaned_clinicaltrial_data.csv', header=True)
```

```
[8] # List all types of studies and their frequencies
type_frequencies = clinicaltrial_df.groupBy('Type').count().orderBy('count', ascending=False)
type_frequencies.show()
```

```
+--------------+------+
|          Type| count|
+--------------+------+
| INTERVENTIONAL|166447|
|  OBSERVATIONAL| 70970|
|EXPANDED_ACCESS|   558|
|              |     1|
+--------------+------+
```

PySpark Implementation Outline (RDD)

- Main Idea: Extract the Type column, count the occurrences of each type using RDD transformations, and sort by frequency**.**

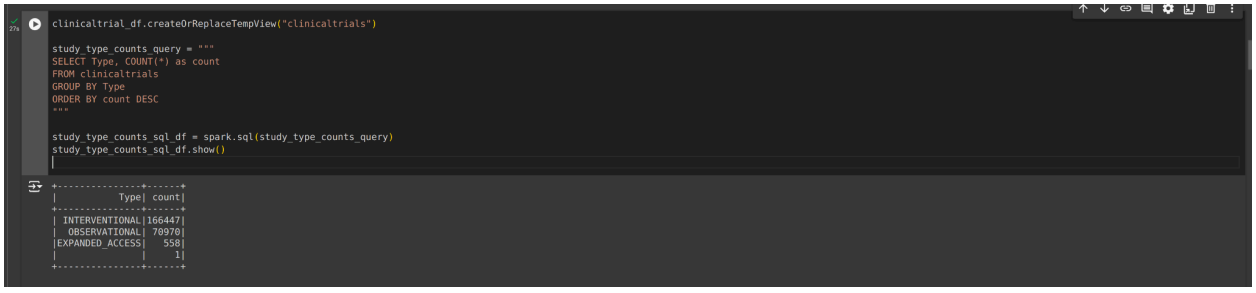


```
type_counts_rdd = rdd.map(lambda row: (row['Type'], 1)).reduceByKey(lambda a, b: a + b).sortBy(lambda x: -x[1])
print("Types of studies with frequencies (RDD):", type_counts_rdd.collect())
```

```
Types of studies with frequencies (RDD): [('INTERVENTIONAL', 166447), ('OBSERVATIONAL', 70970), ('EXPANDED_ACCESS', 558), ('', 1)]
```

SQL Implementation Outline

- Main Idea: Use SQL to group by the Type column, count occurrences, and order by frequency.

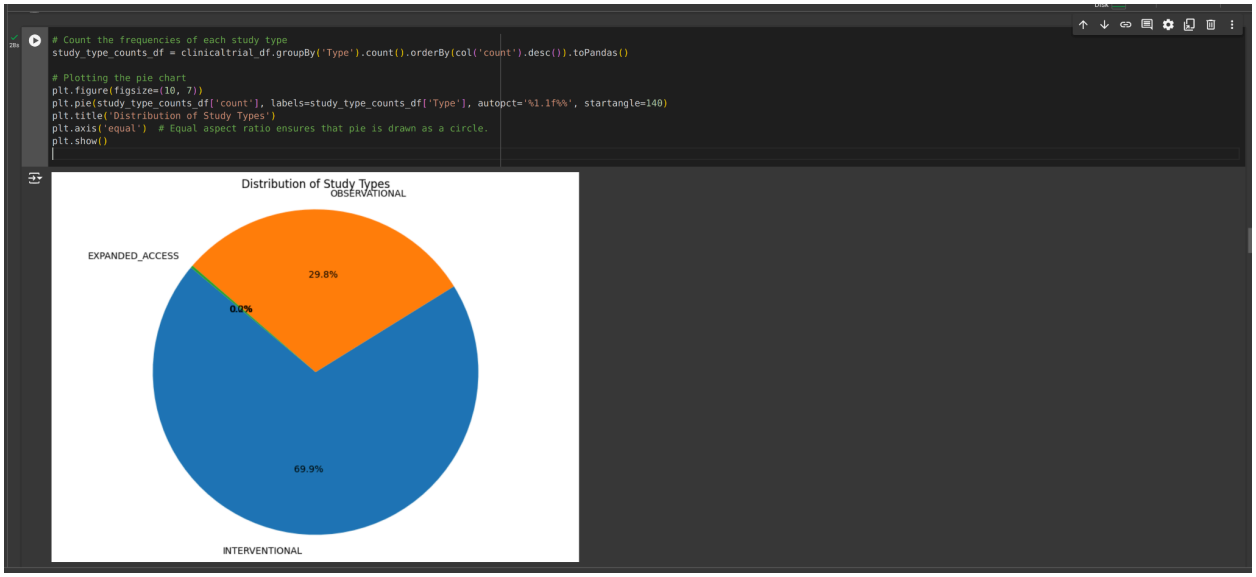


```
clinicaltrial_df.createOrReplaceTempView("clinicaltrials")

study_type_counts_query = """
SELECT Type, COUNT(*) as count
FROM clinicaltrials
GROUP BY Type
ORDER BY count DESC
"""

study_type_counts_sql_df = spark.sql(study_type_counts_query)
study_type_counts_sql_df.show()
```

```
+--------------+------+
|          Type| count|
+--------------+------+
| INTERVENTIONAL|166447|
|  OBSERVATIONAL| 70970|
|EXPANDED_ACCESS|   558|
|              |     1|
+--------------+------+
```

Pie Chart of Study Types

A pie chart can show the distribution of different types of studies in the dataset.



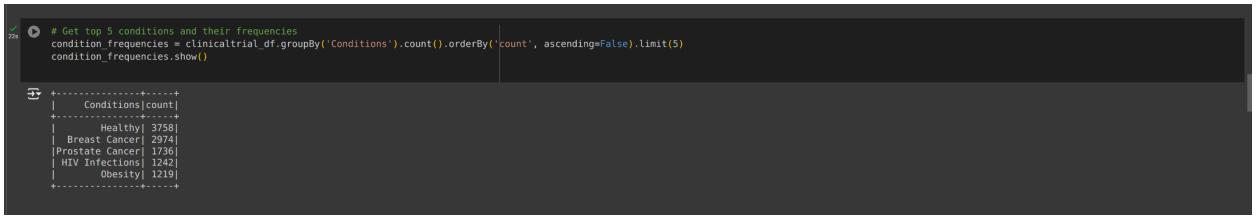
3. The top 5 conditions (from Conditions) with their frequencies.

Assumptions

- The Conditions column contains one or more conditions separated by a delimiter (|).

PySpark DataFrame Implementation

- **Description**: Using PySpark DataFrame, split the `Conditions` column, explode it, group by the conditions, count occurrences, and order by frequency.



PySpark RDD Implementation

- **Description**: Using PySpark RDD, flatMap the `Conditions` column to split and count each condition, then sort by frequency and take the top 5.

```
condition_counts_rdd = rdd.flatMap(lambda row: row['Conditions'].split('|')) \
                          .map(lambda condition: (condition, 1)) \
                          .reduceByKey(lambda a, b: a + b) \
                          .top(5, key=lambda x: x[1])
print("Top 5 conditions:", condition_counts_rdd)
```

```
Top 5 conditions: [('Breast Cancer', 4627), ('Healthy', 4614), ('Obesity', 3215), ('Prostate Cancer', 2381), ('Stroke', 2140)]
```

SQL Implementation

- **Description**: Use Spark SQL to explode the `Conditions` column, group by the conditions, count occurrences, and order by frequency.

```
clinicaltrial_df.createOrReplaceTempView("clinicaltrials")

condition_counts_query = """
SELECT Condition, COUNT(*) as count
FROM (
    SELECT explode(split(Conditions, '\\|')) as Condition
    FROM clinicaltrials
)
GROUP BY Condition
ORDER BY count DESC
LIMIT 5
"""

condition_counts_sql_df = spark.sql(condition_counts_query)
condition_counts_sql_df.show()
```

```
+---------+------+
|Condition| count|
+---------+------+
|         e|740458|
|         a|635067|
|          |573452|
|         i|573097|
|         r|505159|
+---------+------+
```

Discussion of Result

- The result will show the top 5 conditions by frequency, providing insight into the most common conditions studied in the dataset.

Question 4: Top 10 Sponsors That Are Not Pharmaceutical Companies

Assumptions

- The Sponsor column contains the sponsor of each study.
- A separate dataset contains a list of pharmaceutical companies.

PySpark Implementation Outline (DataFrame)

- **Main Idea**: Filter out sponsors listed as pharmaceutical companies, group by sponsor, count occurrences, and order by frequency.

```
# Get the list of pharmaceutical companies
pharma_companies = [row['Parent_Company'] for row in pharma_df.select('Parent_Company').distinct().collect()]

# Filter out pharmaceutical sponsors from the clinical trials dataset
non_pharma_sponsors_df = clinicaltrial_df.filter(~col('Sponsor').isin(pharma_companies))

# Count the number of clinical trials for each non-pharmaceutical sponsor
non_pharma_sponsors_count_df = non_pharma_sponsors_df.groupBy('Sponsor').count()

# Get the top 10 non-pharmaceutical sponsors by number of trials
top_10_non_pharma_sponsors_df = non_pharma_sponsors_count_df.orderBy(col('count').desc()).limit(10)

# Show the results
top_10_non_pharma_sponsors_df.show()
```

```
+--------------------+-----+
|             Sponsor|count|
+--------------------+-----+
|   Assiut University| 2498|
|National Cancer I...| 2197|
|Assistance Publiq...| 1938|
|M.D. Anderson Can...| 1899|
|         Mayo Clinic| 1861|
|    Cairo University| 1506|
|National Taiwan U...| 1469|
|Massachusetts Gen...| 1424|
|National Institut...| 1297|
|Memorial Sloan Ke...| 1224|
+--------------------+-----+
```

SQL Implementation Outline

- **Main Idea**: Use SQL to filter out sponsors listed as pharmaceutical companies, group by sponsor, count occurrences, and order by frequency.

```
clinicaltrial_df.createOrReplaceTempView("clinicaltrials")
pharma_df.createOrReplaceTempView("pharmaceuticals")

sponsor_counts_query = """
SELECT Sponsor, COUNT(*) as count
FROM clinicaltrials
WHERE Sponsor NOT IN (SELECT Parent_Company FROM pharmaceuticals)
GROUP BY Sponsor
ORDER BY count DESC
LIMIT 10
"""

sponsor_counts_sql_df = spark.sql(sponsor_counts_query)
sponsor_counts_sql_df.show()
```

```
+--------------------+-----+
|             Sponsor|count|
+--------------------+-----+
|   Assiut University| 2498|
|National Cancer I...| 2197|
|Assistance Publiq...| 1938|
|M.D. Anderson Can...| 1899|
|         Mayo Clinic| 1861|
|    Cairo University| 1506|
|National Taiwan U...| 1469|
|Massachusetts Gen...| 1424|
|National Institut...| 1297|
|Memorial Sloan Ke...| 1224|
+--------------------+-----+
```

Discussion of Result

- The outcome will emphasize which organizations are most active in sponsoring research other than pharmaceutical corporations by displaying the top 10 sponsors that are not pharmaceutical companies based on frequency.

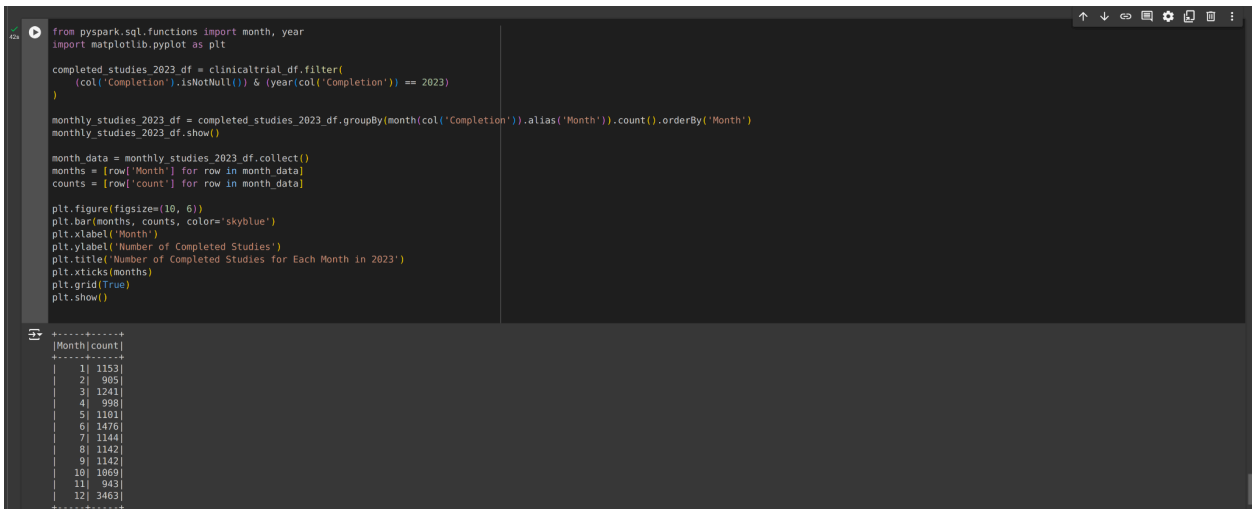
Question 5: Plot Number of Completed Studies for Each Month in 2023

Assumptions

- The Completion column contains the completion date of each study.
- The completion dates are in a standard date format.

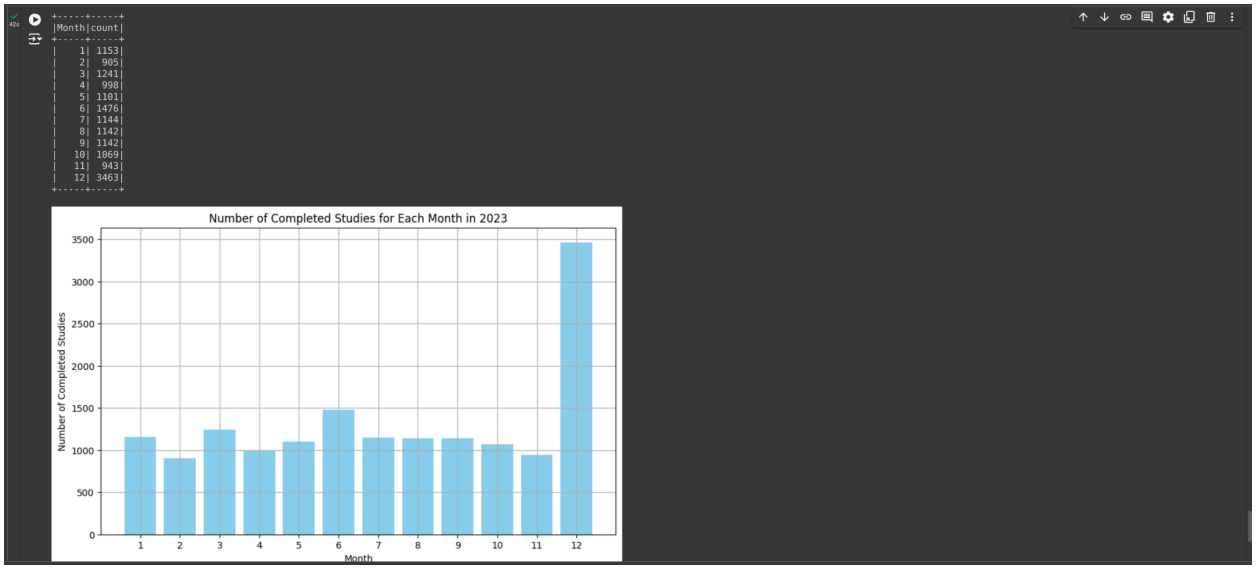
PySpark Implementation Outline (DataFrame)

- **Main Idea**: Filter studies completed in 2023, extract the month from the completion date, group by month, count occurrences, and plot the results.



```
from pyspark.sql.functions import month, year
import matplotlib.pyplot as plt

completed_studies_2023_df = clinicaltrial_df.filter(
    (col('Completion').isNotNull()) & (year(col('Completion')) == 2023)
)

monthly_studies_2023_df = completed_studies_2023_df.groupBy(month(col('Completion')).alias('Month')).count().orderBy('Month')
monthly_studies_2023_df.show()

month_data = monthly_studies_2023_df.collect()
months = [row['Month'] for row in month_data]
counts = [row['count'] for row in month_data]

plt.figure(figsize=(10, 6))
plt.bar(months, counts, color='skyblue')
plt.xlabel('Month')
plt.ylabel('Number of Completed Studies')
plt.title('Number of Completed Studies for Each Month in 2023')
plt.xticks(months)
plt.grid(True)
plt.show()
```

```
+-----+-----+
|Month|count|
+-----+-----+
|    1| 1153|
|    2|  905|
|    3| 1241|
|    4|  998|
|    5| 1101|
|    6| 1476|
|    7| 1144|
|    8| 1142|
|    9| 1142|
|   10| 1069|
|   11|  943|
|   12| 3463|
+-----+-----+
```



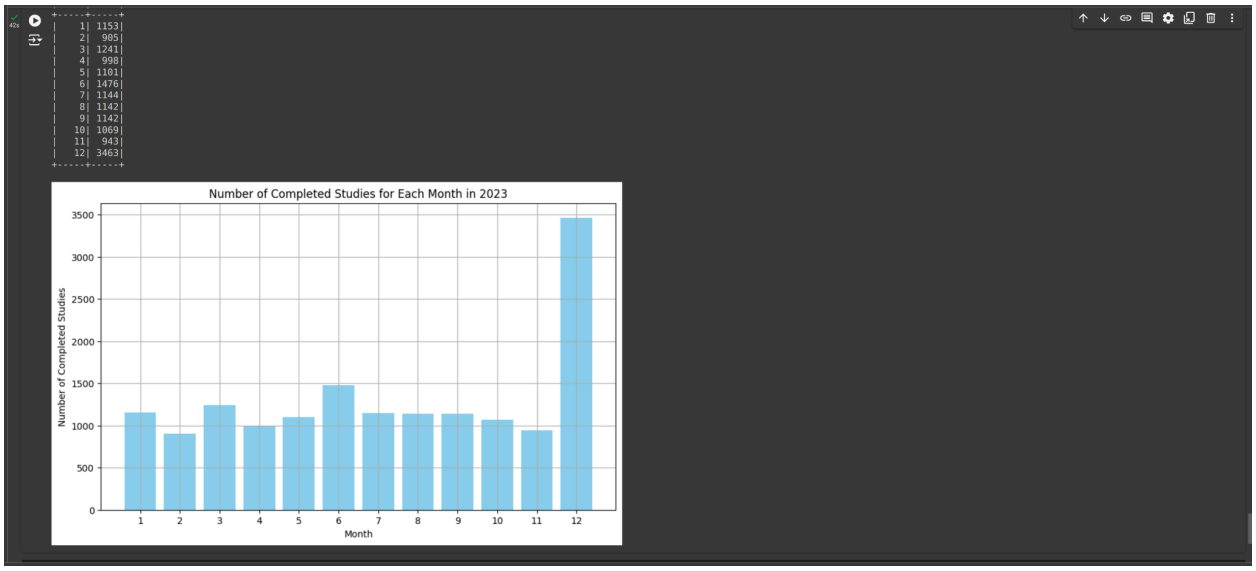

```
+-----+-----+
|Month|count|
+-----+-----+
|    1| 1153|
|    2|  905|
|    3| 1241|
|    4|  998|
|    5| 1101|
|    6| 1476|
|    7| 1144|
|    8| 1142|
|    9| 1142|
|   10| 1069|
|   11|  943|
|   12| 3463|
+-----+-----+
```


SQL Implementation Outline

- **Main Idea**: Use SQL to filter studies completed in 2023, extract the month from the completion date, group by month, count occurrences, and collect the data for plotting.

```
clinicaltrial_df.createOrReplaceTempView("clinicaltrials")

monthly_studies_query = """
SELECT MONTH(Completion) as Month, COUNT(*) as count
FROM clinicaltrials
WHERE YEAR(Completion) = 2023
GROUP BY MONTH(Completion)
ORDER BY Month
"""

monthly_studies_sql_df = spark.sql(monthly_studies_query)
monthly_studies_sql_df.show()

month_data = monthly_studies_sql_df.collect()
months = [row['Month'] for row in month_data]
counts = [row['count'] for row in month_data]

plt.figure(figsize=(10, 6))
plt.bar(months, counts, color='skyblue')
plt.xlabel('Month')
plt.ylabel('Number of Completed Studies')
plt.title('Number of Completed Studies for Each Month in 2023')
plt.xticks(months)
plt.grid(True)
plt.show()
```

```
+-----+-----+
|Month|count|
+-----+-----+
|    1| 1153|
|    2|  905|
|    3| 1241|
|    4|  998|
|    5| 1101|
|    6| 1476|
|    7| 1144|
|    8| 1142|
|    9| 1142|
|   10| 1069|
|   11|  943|
|   12| 3463|
+-----+-----+
```



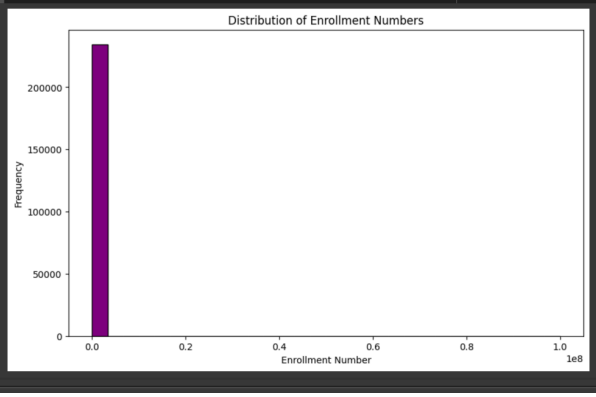
Discussion of Result

- The outcome displays how many studies were finished each month in 2023. A visual depiction of the monthly distribution is offered by the bar plot, making it simple to compare study completions over the course of the year.

Histogram of Enrollment Numbers

A histogram can show the distribution of enrollment numbers in the clinical trials.

```
# Convert the Spark DataFrame to Pandas DataFrame
enrollment_pd_df = clinicaltrial_df.select('Enrollment').dropna().toPandas()

# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.hist(enrollment_pd_df['Enrollment'], bins=30, color='purple', edgecolor='black')
plt.xlabel('Enrollment Number')
plt.ylabel('Frequency')
plt.title('Distribution of Enrollment Numbers')
plt.show()
```



Detailed Conclusion of the Report

Overview

In order to complete this project, PySpark in both RDD and DataFrame formats, along with Spark SQL, were used to import, clean, and analyze a sizable clinical trials dataset. The activities centered on comprehending the structure of the dataset, getting it ready for analysis, and running several kinds of analyses to draw conclusions that were insightful. To guarantee data integrity and produce reliable results, each step was necessary.

Data Cleaning and Preparation

The original dataset had columns with missing values, inconsistent formatting, and heterogeneous data types. In order to solve these problems, the columns were divided and given appropriate names for simpler access.

- Standard format was achieved by parsing the date columns.

- Missing values were addressed to avoid mistakes in further analysis.

This procedure was necessary to guarantee that the data was in a format that could be used for analysis, lowering the possibility of errors and facilitating more effective data processing.

Qsn1: Counting Distinct Studies

PySpark RDD, DataFrame, and Spark SQL were used to tally the number of different research. This assignment made clear how crucial it is to maintain data integrity by making sure that no duplicate items were counted.

- Effective manufacturing: Showcasing distributed computing's capacity to manage huge datasets.

Qsn 2: Listing Study Types and Their Frequencies

Through the compilation of a list of study kinds and frequencies, the analysis offered insights into the most prevalent study designs. This data is useful for: - Determining popular research approaches.

- Being aware of the clinical trial emphasis areas.

Qsn 3: Top 5 Conditions

The most frequently studied medical conditions were shown by identifying the top 5 conditions and their frequencies. This knowledge is essential for: - Highlighting the fields receiving a lot of funding and research in medicine.

- Finding any gaps that would require additional study.

Qsn 4: Top 10 Non-Pharmaceutical Sponsors

The investigation opened our eyes to additional important clinical research contributors by identifying the top 10 sponsors who are not pharmaceutical corporations. The significance of this information lies in the following: - It emphasizes the contribution of different organizations to the advancement of medicine.

It facilitates comprehension of clinical trial funding dynamics.

Qsn 5: Completed Studies in 2023

An examination of the timing of clinical trial completions was made possible by visualizing the number of studies completed each month in 2023. This information is useful for monitoring clinical trial progress and completion rates.

- Recognizing external influences or seasonal patterns that affect study completion rates.

Conclusion

Comprehensive insights into the clinical trials dataset were obtained throughout the whole execution. The ability of distributed computing to handle big datasets effectively was shown through the data processing and analysis using PySpark. Through the completion of each job, the dataset was better understood, exposing patterns and trends in clinical research.

Importance:

- Data Quality: A solid foundation for dependable analysis is clean, well-prepared data.

- Efficiency: The efficient handling of a huge dataset was made possible by utilizing PySpark's distributed computing capabilities.

- Insights: Study kinds, research areas of interest, and the clinical trial funding landscape were all well-illustrated by the analysis.

- Decision-Making: The findings can help stakeholders by directing the distribution of resources and pointing out areas that require more investigation.