STA-CN-7031 – BIG DATA ANALYTICS

# 

# Project

**Big Data Analytics Report using Hadoop and Spark**

**By**

**OLANREWAJU OSAHON UZAMERE**

**SILICON TECH ANALYTICS**

# The Table of Content

**The Table of Content** ………………………………………………………………… 1

[**Tasks and Question Paper**](#_1c06hkfmngak) …………………………………………………………… 2

[**Task 2: Big Data Query & Analysis by Apache Hive**](#_544308vgb9uc) ………………………………… 4

[**Task 3: Advanced Analytics using PySpark**](#_njo8pg6bmsch) ………………………………………….. 13

[Analyze and Interpret Big Data](#_njo8pg6bmsch) ……………………………………………………….. 13

[Design and Build a Classifier](#_4yub74hb1tvg) …………………………………………………………. 20

[**Task 4: Individual Assessment**](#_59hgcq94uo8z) ………………………………………………………… 26

[Alternative Technologies for Tasks 2 and 3 and How they Differ](#_59hgcq94uo8z) …………………….. 26

[The New Thinking that Evoked and/or Neglected](#_gh3wy6oaqfau) …………………………………….. 27

[**References**](#_zgpt55ksb5np)……………………………………………………………………………….. 28

**Tasks and Question Paper:**

### (1) Understanding Dataset: UNSW-NB15

The raw network packets of the UNSW-NB15 dataset were created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviours. Tcpdump tool is used to capture 100 GB of the raw traffic (e.g., Pcap files). This data set has nine types of attacks, namely, *Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms*. The Argus and Bro-IDS tools are used and twelve algorithms are developed to generate a total of 49 features with the class label.

a) The features are described here.

b) The number of attacks and their sub-categories are described here.

c) In this Project, we use the total number of 10-million records that were stored in the CSV file (download). The total size is about 600MB, which is big enough to employ big data methodologies for analytics. As big data specialists, firstly, we would like to read and understand its features and then apply modelling techniques. If you want to see a few records of this dataset, you can import it into Hadoop HDFS, then make a Hive query for printing the first 5-10 records for your understanding.

### (2) Big Data Query & Analysis by Apache Hive

This task is using Apache Hive for converting big raw data into useful information for the end-users. To do so, firstly understand the dataset carefully. Then, **make at least 4 Hive queries (refer to the marking scheme)**. Apply appropriate visualization tools to present your findings numerically and graphically. Interpret shortly your findings.

Finally, take a screenshot of your outcomes (e.g., tables and plots) together with the scripts/queries into the report.

**Tip:** This section depends on the level of your HIVE queries’ complexities, for instance using the simple select query is not supposed for the full mark.

1 source: <https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/>

#### 

### (3) Advanced Analytics using PySpark

In this section, you will conduct advanced analytics using PySpark.

#### 3.1. Analyze and Interpret Big Data

We need to learn and understand the data through **at least 4 analytical methods** (descriptive statistics, correlation, hypothesis testing, density estimation, etc.). You need to present your work numerically and graphically. Apply tooltip text, legend, title, X-Y labels etc. accordingly to help end-users for getting insights.

#### 3.2. Design and Build a Classifier

a) Design and build a binary classifier over the dataset. Explain your algorithm and its configuration. Explain your findings in both numerical and graphical representations. Evaluate the performance of the model and verify the accuracy and the effectiveness of your model. *[15 marks]*

b) Apply a multi-class classifier to classify data into ten classes (categories): one normal and nine attacks (e.g., *Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms*). Briefly explain your model with supportive statements on its parameters, accuracy and effectiveness. *[20 marks]*

**Tip:** you can use this link (<https://spark.apache.org/docs/2.2.0/ml-classification-regression.html>) for more information on modelling.

### (4) Individual Assessment

Discuss (1) what other alternative technologies are available for tasks 2 and 3 and how they differ (use academic references), and (2) what was surprisingly new thinking evoked and/or neglected at your end?

Tip: add an individual assessment of each member in the same report.

### 

### (5) Documentation

Document all your work. Your final report must follow 5 sections detailed in the **“format of final submission”** section (refer to the next page). Your work must demonstrate an appropriate understanding of academic writing and integrity.

Task 2: Big Data Query & Analysis by Apache Hive

# Hive Queries

### Hive Query 1: Let's show the outcome of the number of connections from each source IP address (SRCIP).

SELECT unswnb15.srcip AS SourceIP,

COUNT(unswnb15.label) AS Numberofconnections

FROM UNSWNB15

GROUP BY unswnb15.srcip

ORDER BY Numberofconnections DESC;

In the above **hive query**, I queried the dataset to SELECT the SRCIP column (unswnb15.srcip) and COUNT the LABEL(unswnb15.label) per source IP connection *(for each count represents a single connection)*, named the result columns (AS) SourceIP and Numberofconnections respectively, FROM the UNSWNB15 dataset. I then grouped the result by (GROUP BY) the SRCIP (unswnb15.srcip) and sorted it using (ORDER BY)the Numberofconnections in descending order (DESC).

This Hive query is designed to analyze network connections in the dataset named "UNSWNB15" and provide a count of connections for each unique source IP address (srcip). Here's a breakdown of the query:

SELECT Clause:

* unswnb15.srcip AS SourceIP: It selects the srcip column from the UNSWNB15 table and aliases it as SourceIP.
* COUNT(unswnb15.label) AS Numberofconnections: It calculates the count of non-null values in the label column for each group of distinct source IP addresses. This count is aliased as Numberofconnections.

FROM Clause:

* FROM UNSWNB15: Specifies the source table for the data, which is assumed to be a dataset named "UNSWNB15."

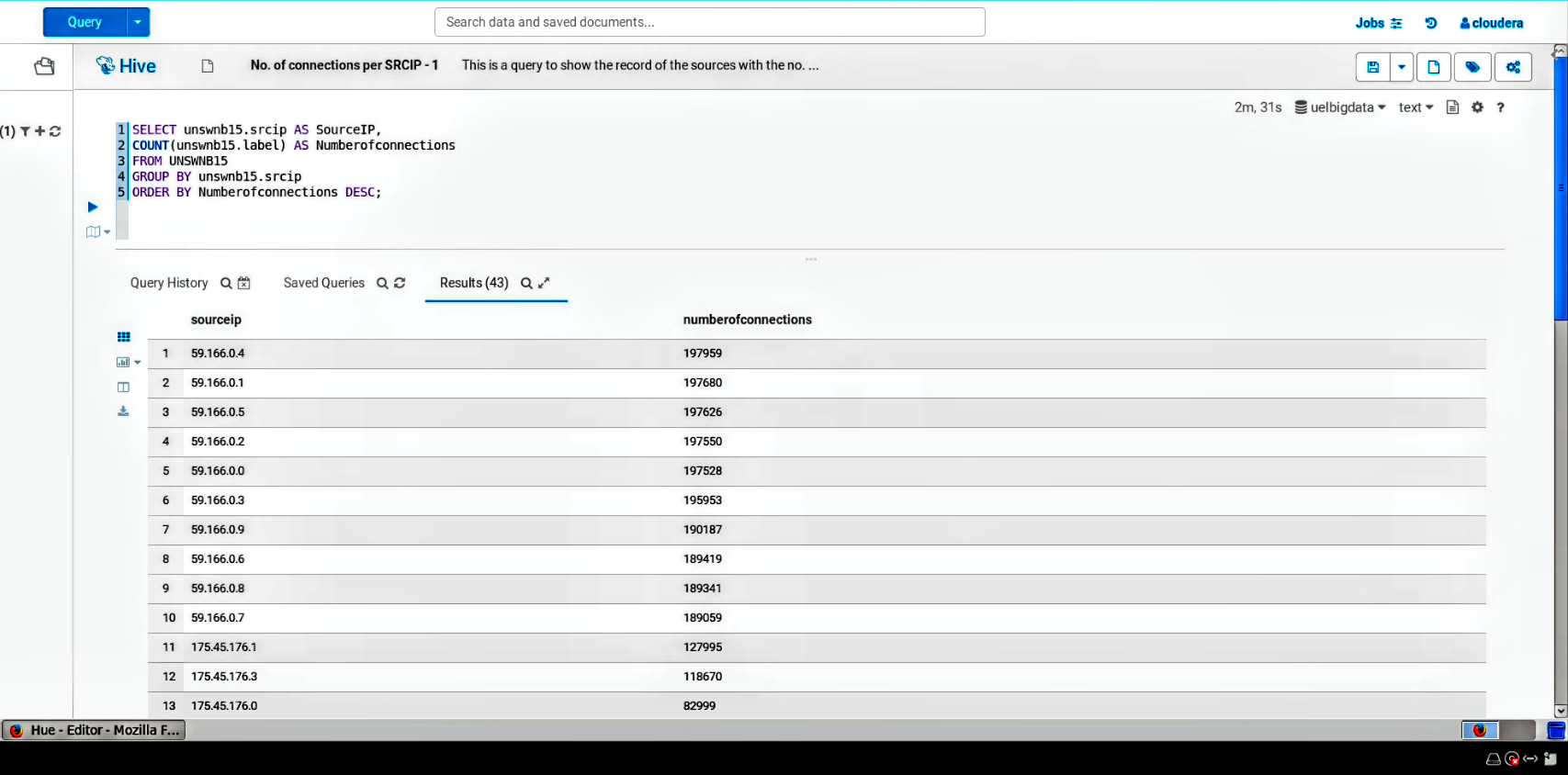
GROUP BY Clause:

* GROUP BY unswnb15.srcip: Groups the data by the srcip column, so the subsequent aggregation functions (like COUNT) will be applied to each unique source IP.

ORDER BY Clause:

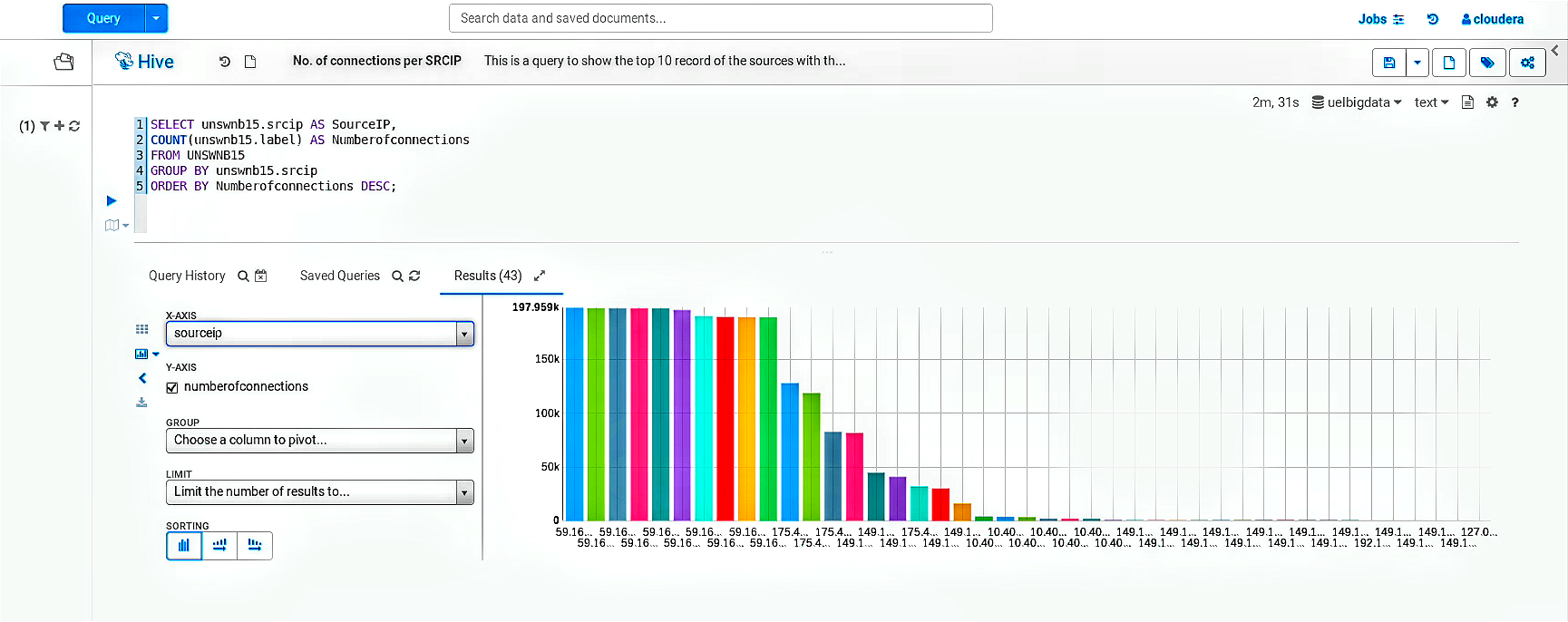
* ORDER BY Numberofconnections DESC: Orders the result set in descending order based on the count of connections (Numberofconnections). The source IP addresses with the highest number of connections will appear first.

The query returned **43 records** with the Source IP and counted the number of labels (0 and 1) for each source IP as Numberofconnections in descending order. The number of labels (0 and 1) per source IP here is used to determine the number of times the source IP (SRCIP) is connected to the network by counting the number of labels for each source IP. Screenshots of the hive query and the output are shown below.



*Fig 1A: Hive Query 1 - Number of Connections per Source IP*

Below is a bar chart presentation of the above report:



*Fig 1B: Hive Query 1 - Visual Presentation Using Bar Chart*

### Hive Query 2: Now, let's determine the count of connections per source IP (SRCIP) using the HTTPS service and TCP protocol.

SELECT unswnb15.srcip AS SOURCEIP,

SUM(unswnb15.dur) AS TOTALDURATION,

SUM(unswnb15.sbytes) AS SOURCEBYTES,

SUM(unswnb15.dbytes) AS DESTINATIONBYTES,

COUNT(unswnb15.service) AS NUMBEROFHTTPCONNECTION

FROM UNSWNB15

WHERE unswnb15.service = 'http' AND unswnb15.proto = 'tcp'

GROUP BY unswnb15.srcip

ORDER BY NUMBEROFHTTPCONNECTION DESC;

This Hive query is designed to analyze network connections in the dataset named "UNSWNB15" and provide aggregated statistics for HTTP connections.

Here's a breakdown of the query:

SELECT Clause:

* unswnb15.srcip AS SOURCEIP: It selects the srcip column from the UNSWNB15 table and aliases it as SOURCEIP.
* SUM(unswnb15.dur) AS TOTALDURATION: It calculates the total duration of connections for each unique source IP address.
* SUM(unswnb15.sbytes) AS SOURCEBYTES: It calculates the total number of source bytes transferred for each unique source IP address.
* SUM(unswnb15.dbytes) AS DESTINATIONBYTES: It calculates the total number of destination bytes transferred for each unique source IP address.
* COUNT(unswnb15.service) AS NUMBEROFHTTPCONNECTION: It counts the number of HTTP connections for each unique source IP address.

FROM Clause:

* FROM UNSWNB15: Specifies the source table for the data, which is assumed to be a dataset named "UNSWNB15."

WHERE Clause:

* WHERE unswnb15.service = 'http' AND unswnb15.proto = 'tcp': Filters the data to include only rows where the service is 'http' and the protocol is 'tcp'. This ensures that only HTTP connections are considered.

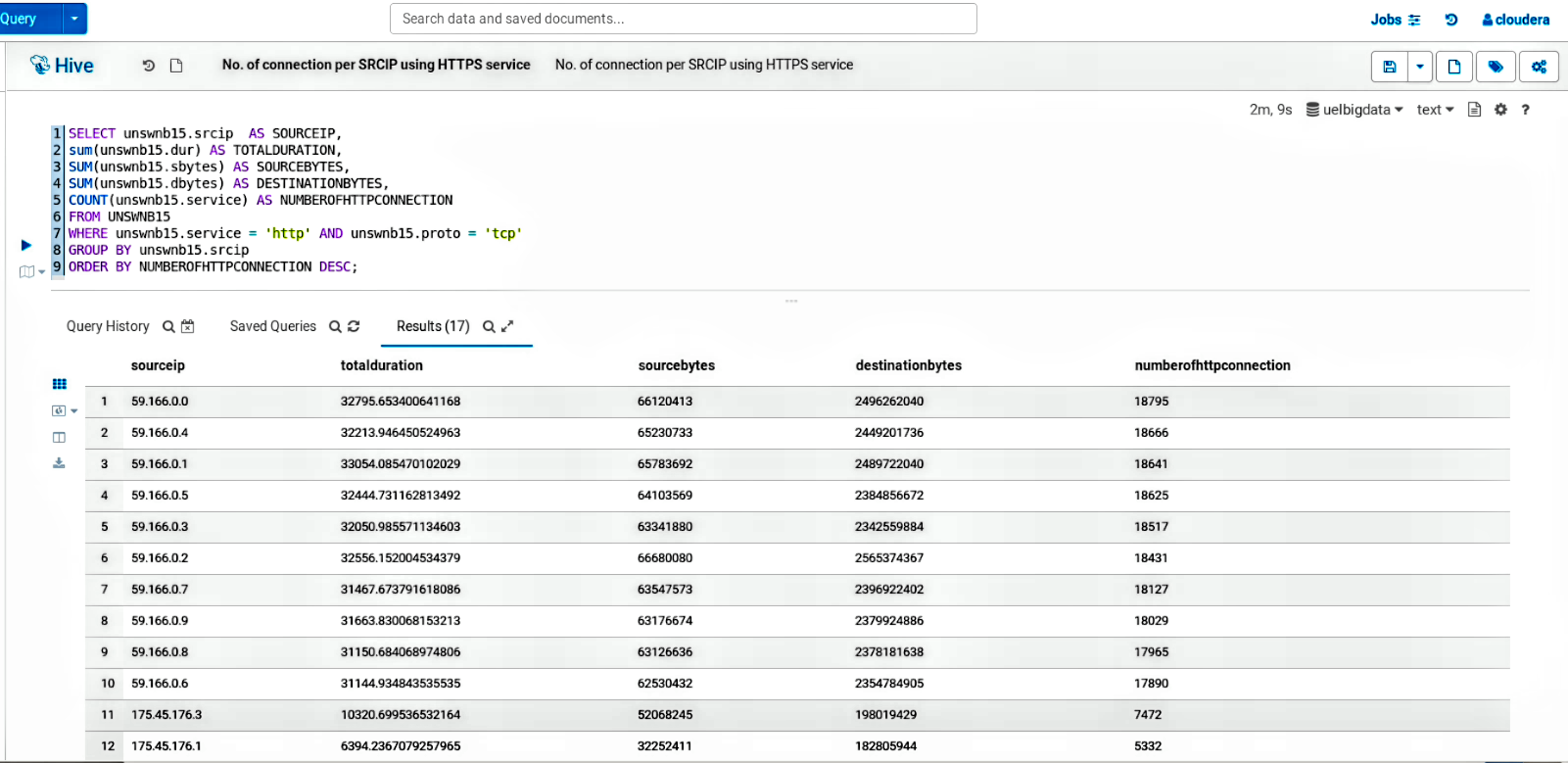
GROUP BY Clause:

* GROUP BY unswnb15.srcip: Groups the data by the srcip column, so the subsequent aggregation functions (like SUM and COUNT) will be applied to each unique source IP.

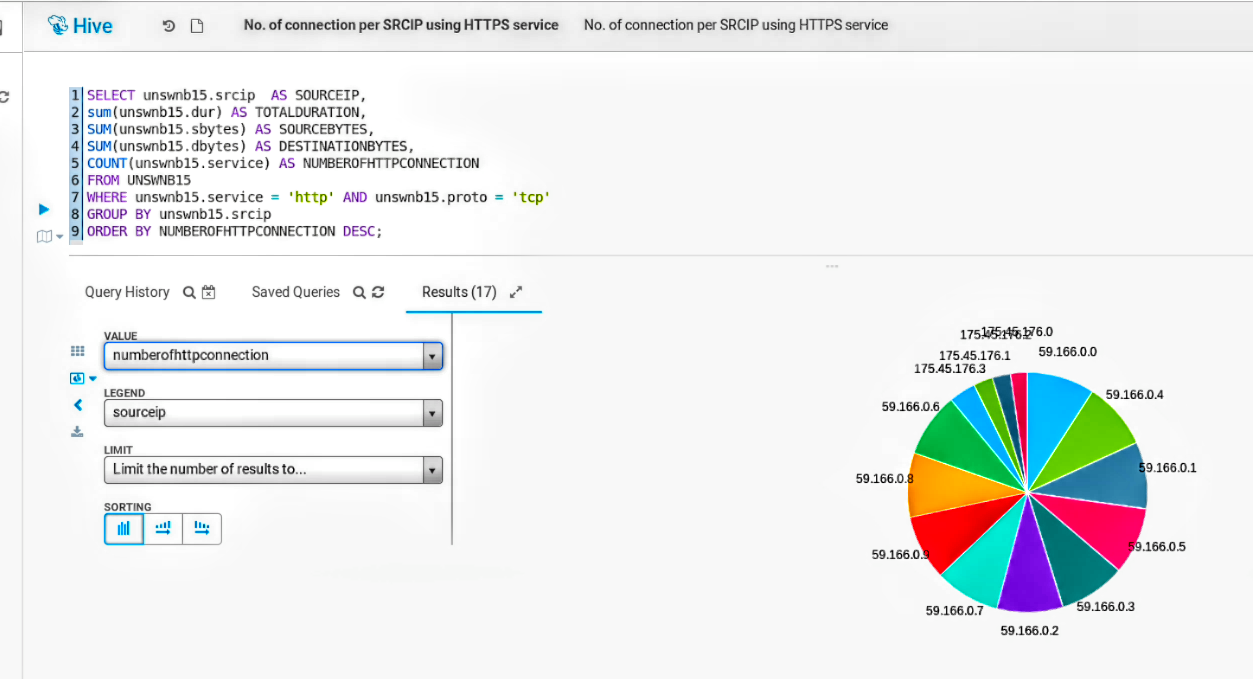
ORDER BY Clause:

* ORDER BY NUMBEROFHTTPCONNECTION DESC: Orders the result set in descending order based on the count of HTTP connections (NUMBEROFHTTPCONNECTION). The source IP addresses with the highest number of HTTP connections will appear first.

In the given query, the dataset is filtered to retrieve information about network connections. The Source IP (unswnb15.srcip), the total duration of each source IP connection (unswnb15.dur), the sum of source bytes transmitted by each source IP (unswnb15.sbytes), the sum of destination bytes transmitted by each source IP (unswnb15.dbytes), and the count of connections by each IP through a specific service (unswnb15.service) are selected. The resulting columns are renamed as SOURCEIP, TOTALDURATION, SOURCEBYTES, DESTINATIONBYTES, and NUMBEROFHTTPCONNECTION, respectively. The data is extracted from the UNSWNB15 dataset, focusing on cases where the service type is HTTP and the protocol type is TCP. The results are then grouped by the Source IP (unswnb15.srcip) and sorted in descending order based on the NUMBEROFHTTPCONNECTION using the ORDER BY clause.

The query produced 17 entries, displaying details such as Source IP (sourceip), the total Source Bytes (sourcebytes), the total Destination Bytes (destinationbytes), and the count of HTTP connections (numberofhttpconnections) for each Source IP that connected to the network via the HTTP service and TCP protocol. A snapshot of the Hive query and its outcomes is provided in the following image.

*Fig 2A: Hive Query 2 - Number of connections per source IP (SRCIP) using the HTTPS service and TCP protocol.*

Here is a pie chart representation corresponding to the report mentioned above:

*Fig 2B: Hive Query 2 - Visual Presentation Using Pie Chart*

### Hive Query 3: Now, let's enumerate the highest amount of source bytes transmitted by individual source IP addresses within a single connection.

SELECT unswnb15.srcip AS `SOURCE IP`,

MAX(unswnb15.sbytes) AS `MAXIMUM SOURCE BYTES`

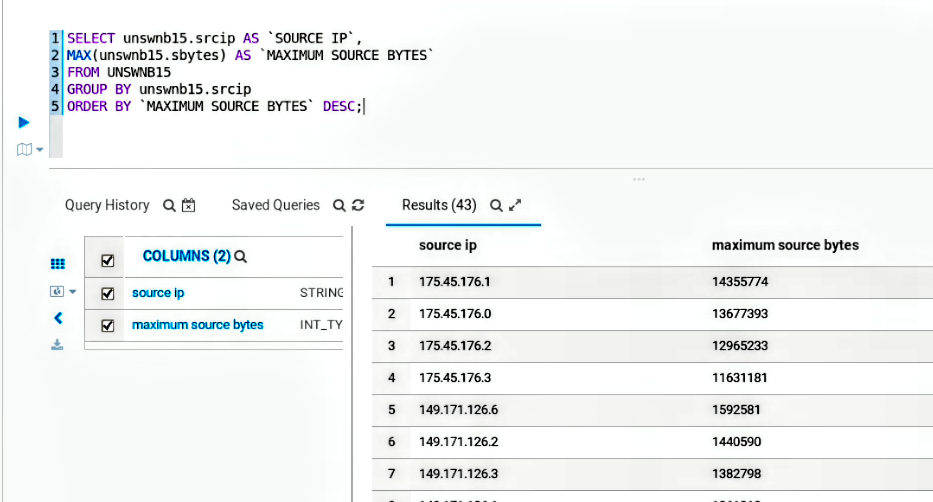
FROM UNSWNB15

GROUP BY unswnb15.srcip

ORDER BY `MAXIMUM SOURCE BYTES` DESC;

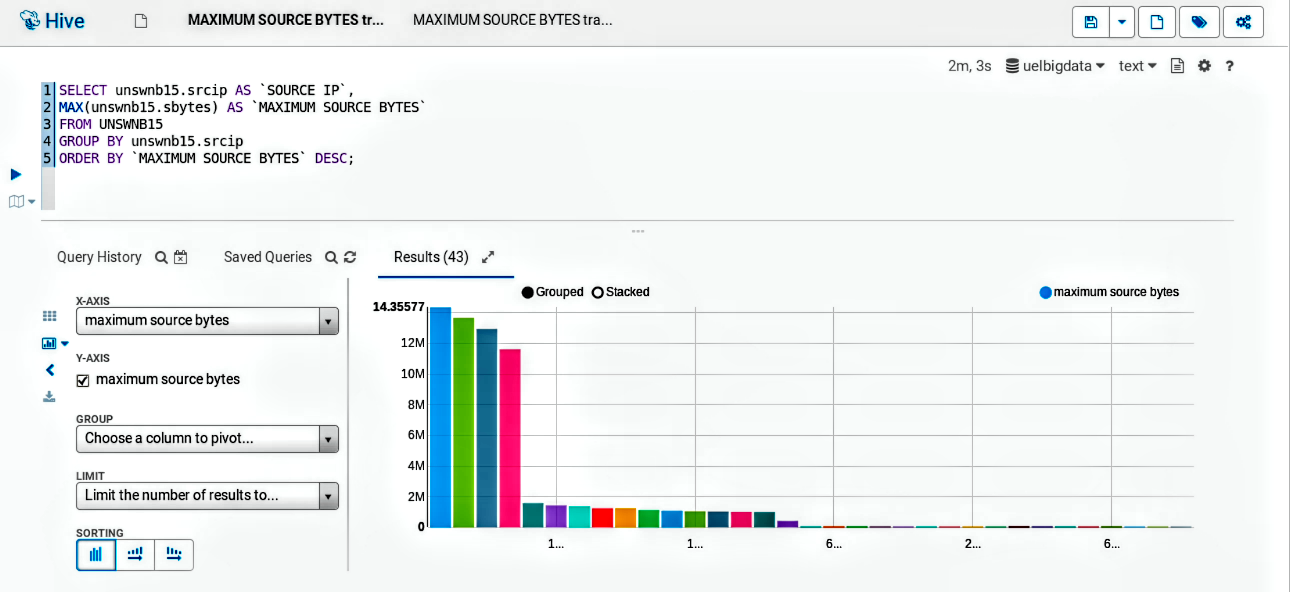
To compile the list of the highest source bytes transmitted by each source IP in a single connection, the dataset was queried. It involved selecting the Source IPs (unswnb15.srcip) and finding the maximum source bytes within each group of source IPs (MAX(unswnb15.sbytes)) from the UNSWNB15 dataset. The columns in the result were named SOURCE IP and MAXIMUM SOURCE BYTES. Subsequently, the results were grouped by source IP (GROUP BY unswnb15.srcip) and sorted in descending order based on the MAXIMUM SOURCE BYTES (ORDER BY MAXIMUM SOURCE BYTES DESC).

The query produced 43 results, displaying each unique Source IP as "source IP" and indicating the maximum Source Bytes transmitted in a single connection by each Source IP as "maximum source bytes." The results are presented in descending order. Visual representations of both the Hive query and its outcomes are provided in the accompanying screenshots..



*Fig 3A: Hive Query 3 - Maximum Source Bytes Transmitted per Source IP*

Here is a graphical representation in the form of a bar chart for the previously mentioned report.



*Fig 3B: Hive Query 3 - Visual Presentation Using Bar Chart*

### 

### Hive Query 4: Let’s find out the number of normal connections and attacks on the network.

SELECT unswnb15.label AS `CONNECTION TYPE`,

COUNT(unswnb15.label) AS `Total Number of Connections’

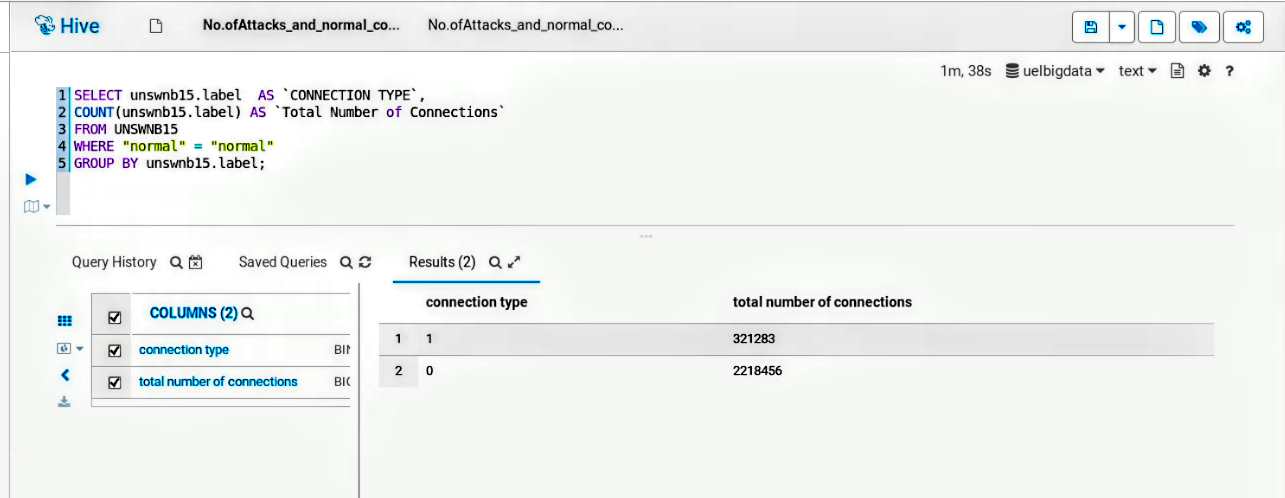
FROM UNSWNB15

WHERE "normal" = "normal"

GROUP BY unswnb15.label;

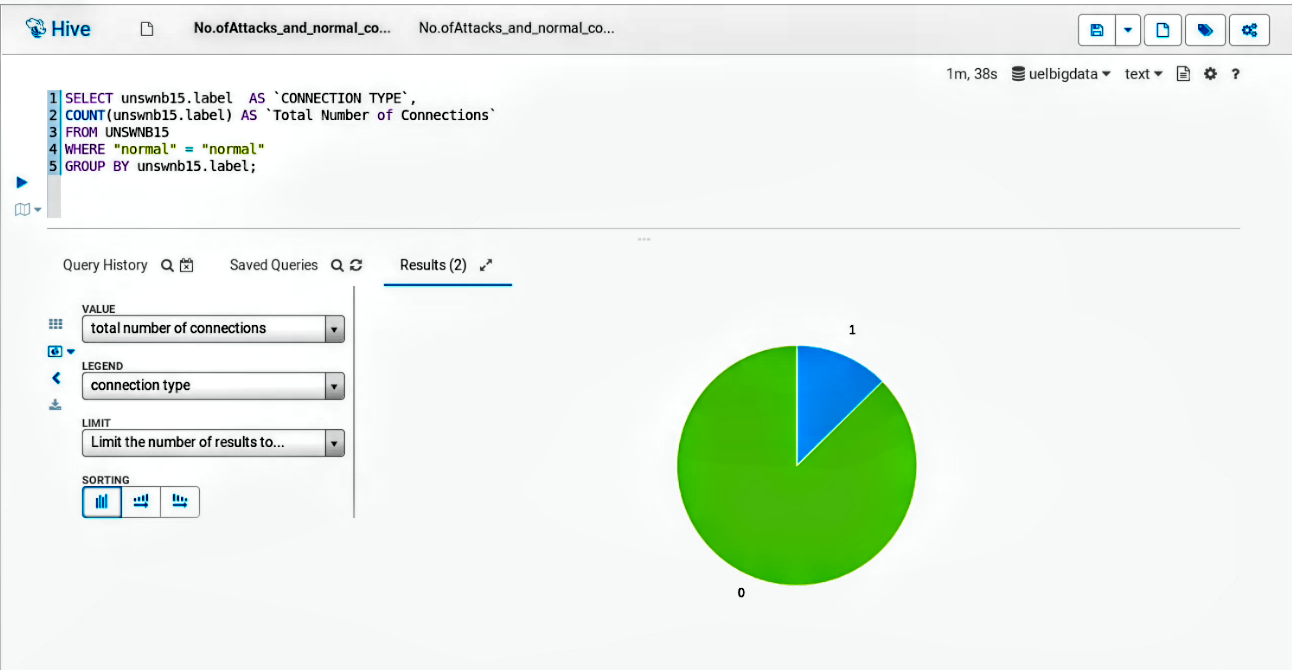
Here, I queried the dataset to get the total number of normal connections and the total number of attacks on the network by selecting the Label column (SELECT unswnb15.label) AS CONNECTION TYPE, COUNT the number of times each label appeared on the dataset AS Total Number of Connections *(for each count represents a single connection)*, FROM the UNSWNB15 dataset, WHERE the binary normal is normal then GROUP BY label (unswnb15.label).

The query produced **2 results** displaying the connection types labeled as 0 and 1, along with the respective counts of connections for each label within the total connection count. The labels 0 and 1 are binary values denoting a normal connection and an attack in the dataset. The outcome indicates the presence of 2,218,456 normal connections and 321,283 attacks across the network. Accompanying this explanation are screenshots of both the Hive query and its resultant data.



*Fig 4A: Hive Query 4 - Total Number of Attacks and Normal Connections*

Below is a comparative pie chart presentation of the above report:



*Fig 4B: Hive Query 4 - Visual Presentation Using Pie Chart*

Task 3: Advanced Analytics using PySpark

# 3.1. Analyze and Interpret Big Data

**Loading Data as PySpark DataFrame:** Here, we will load data into a PySpark DataFrame without using column names in the header. Our code is written in PySpark and is designed to load a CSV file into a PySpark DataFrame. Let's break down the code:

i. Setting the Data CSV Path:

data\_csv\_path = "/content/drive/MyDrive/UNSW-NB15.csv"

* Above, data\_csv\_path is a variable that holds the file path to the CSV file containing the dataset. I replaced this path with the actual path to my "UNSW-NB15.csv" file.

ii. Loading the CSV Data into a PySpark DataFrame:

data=spark.read.csv(data\_csv\_path, header=False, inferSchema=True)

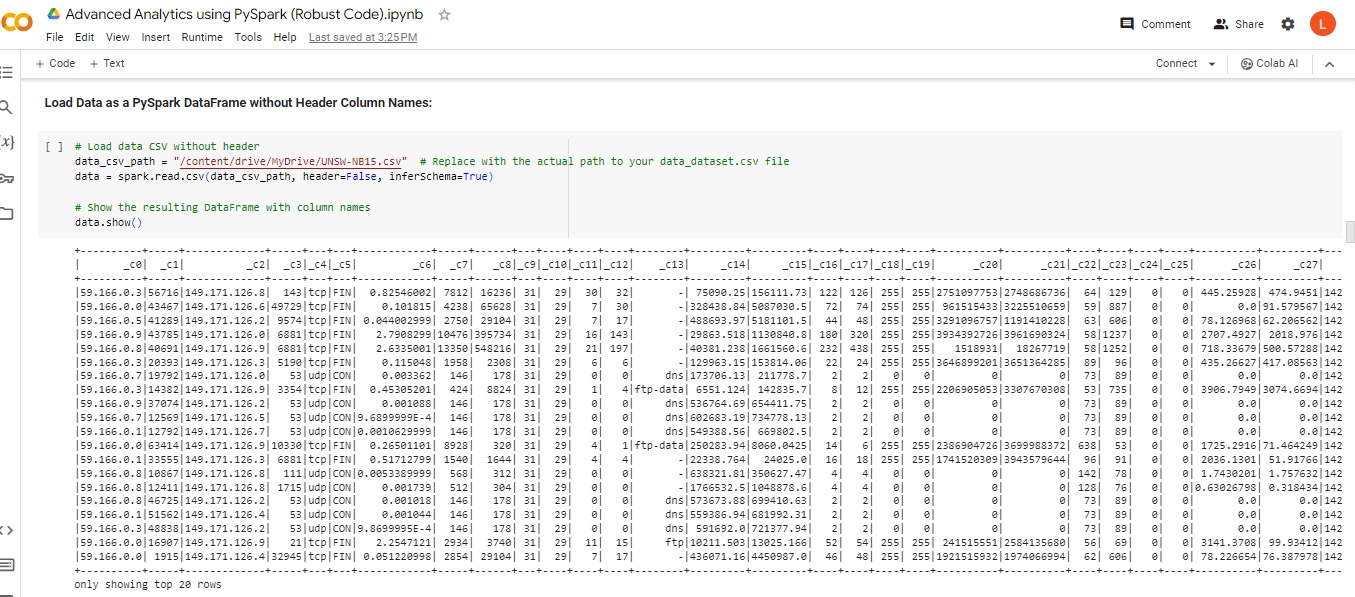
* spark.read.csv(): This is a method provided by SparkSession to read CSV data.
* data\_csv\_path: The path to the CSV file.
* header=False: Specifies that the CSV file doesn't have a header row. If set to True, the first row of the CSV file would be considered as a header, naming the columns.
* inferSchema=True: Enables automatic inference of the schema (data types) of columns in the DataFrame. Spark will attempt to identify the appropriate data types for each column based on the data in the CSV file.
* The result of this operation is a PySpark DataFrame named data.

iii. Displaying the DataFrame:

codedata.show()

* data.show(): This method is used to display the content of the DataFrame. It shows the first few rows of the DataFrame along with column names.

**So, the purpose of this code is to load the "UNSW-NB15.csv" dataset into a PySpark DataFrame named data, assuming that the CSV file doesn't have a header row, and Spark should infer the schema for each column. Finally, it displays the content of the DataFrame**



*Fig 5: Loading data into a PySpark DataFrame without using column names*

**Descriptive statistics:** Here, we will utilize the PySpark .describe() function to present descriptive statistics for the UNSW-NB15 dataset. This function calculates summary statistics for the dataset within the PySpark environment on Google Colab..

Using it with our Dataset: UNSWNB15.describe().show()

The syntax above generates overall descriptive statistics for the PySpark DataFrame named UNSWNB15, assuming that UNSWNB15 has been previously defined.Count – Count/ total number of values of each column

* Mean – The average value for each column.
* Stddev – The measure of how spread out the values are, indicating the degree of deviation from the mean for each column.
* Min – The smallest value present in each column
* Max – The largest value present in each column.

Below is a screenshot of the python input codes and the output.



*Fig 6: Describe/Summary Statistics of UNSWNB15 Dataframe*

The descriptive statistics for four columns, including total count, mean, standard deviation, minimum, and maximum values, were examined using UNSWNB15.describe('sbytes', 'dbytes', 'Spkts', 'Dpkts').show(). The results are presented in the diagram, showcasing the analyzed metrics.

* The dataset has a total of 2,539,739 values in each of the sbytes, dbytes, Spots, and Dpkts columns.
* The average or mean values for the sbytes, dbytes, Spots, and Dpkts columns are 4340.072263330996, 36432.01132478574, 33.2925678583508, and 42.731821655689814, respectively.
* The standard deviation of the sbytes, dbytes, Spots, and Dpkts columns is 56409.39812286232, 161105.30400824756, 76.28775585678378, and 121.50842004704197, respectively.
* The smallest or least or minimum value in the sbytes, dbytes, Spots, and Dpkts columns is 0.
* The largest or maximum values in the sbytes, dbytes, Spots, and Dpkts columns are 14,355,774, 14,657,531, 10,646, and 11,018, respectively.

View ***Descriptive Statistics*** in my Google Colaboratory (Colab) Notebook -

<https://colab.research.google.com/drive/1rqJmX3Uq_j2tMxvXAXJmmMMw2LgOteaW#scrollTo=pGg_Aw2hEvsJ>

**Correlation Matrix:** We are going toperform correlation analysis on selected numeric features of a PySpark DataFrame (UNSWNB15). Let's break it down:

Calculating Correlation Matrix:

correlation\_matrix = UNSWNB15.select(['dur', 'Sload', 'Dload', 'Spkts', 'Dpkts']).toPandas().corr()

* UNSWNB15.select(['dur', 'Sload', 'Dload', 'Spkts', 'Dpkts']): Selects a subset of numeric features ('dur', 'Sload', 'Dload', 'Spkts', 'Dpkts') from the original DataFrame (UNSWNB15).
* .toPandas(): Converts the selected PySpark DataFrame to a Pandas DataFrame. This is done to use Pandas functions for calculating the correlation matrix.
* .corr(): Computes the pairwise correlation coefficients between the selected numeric features.
* The resulting correlation\_matrix is a Pandas DataFrame containing correlation values between the specified numeric features.

Displaying Correlation Matrix as a Heatmap:

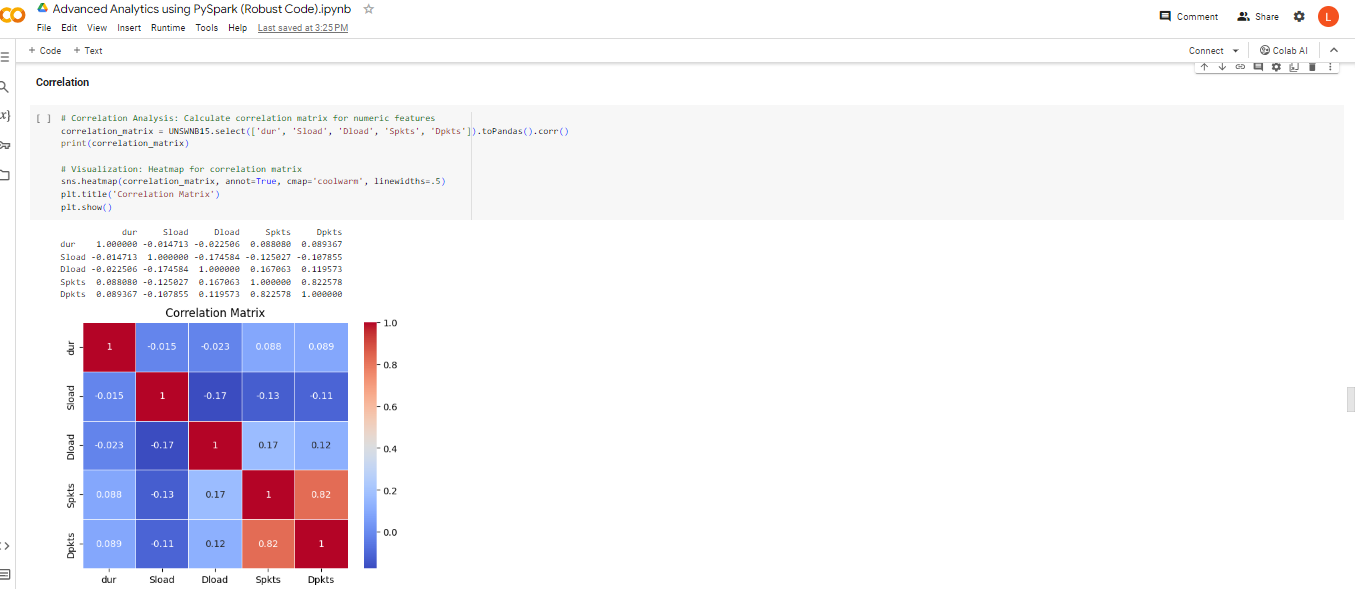
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', linewidths=.5)

plt.title('Correlation Matrix')

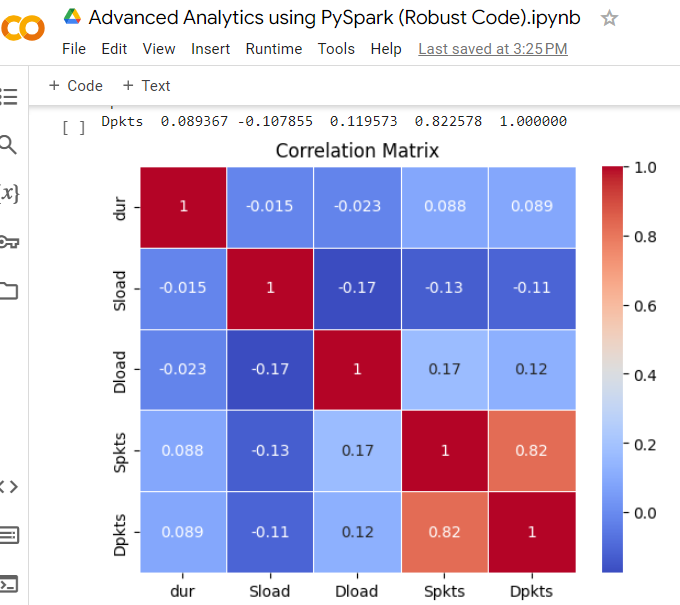
plt.show()

Here’s a breakdown of the code below:

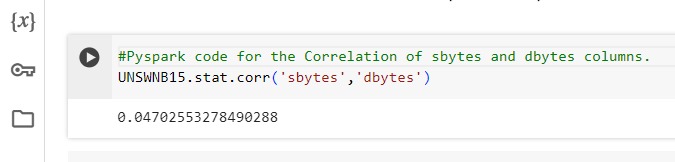
* sns.heatmap(): Creates a heatmap using Seaborn's heatmap function.
* correlation\_matrix: The Pandas DataFrame containing correlation values.
* annot=True: Displays the correlation values in each cell of the heatmap.
* cmap='coolwarm': Specifies the color map for the heatmap.
* linewidths=.5: Sets the width of the lines between cells in the heatmap.
* plt.title('Correlation Matrix'): Adds a title to the heatmap.
* plt.show(): Displays the heatmap.



*Fig 7: Correlation Matrix*

 *Fig 8: Correlation Matrix (Close View)*

**Correlation:** The correlation between two columns can be computed with the stat.corr() function in PySpark. By applying UNSWNB15.stat.corr('sbytes', 'dbytes'), I examined the correlation between the 'sbytes' and 'dbytes' columns, revealing a correlation coefficient of 0.04702553278490288, as illustrated in Figure 9.



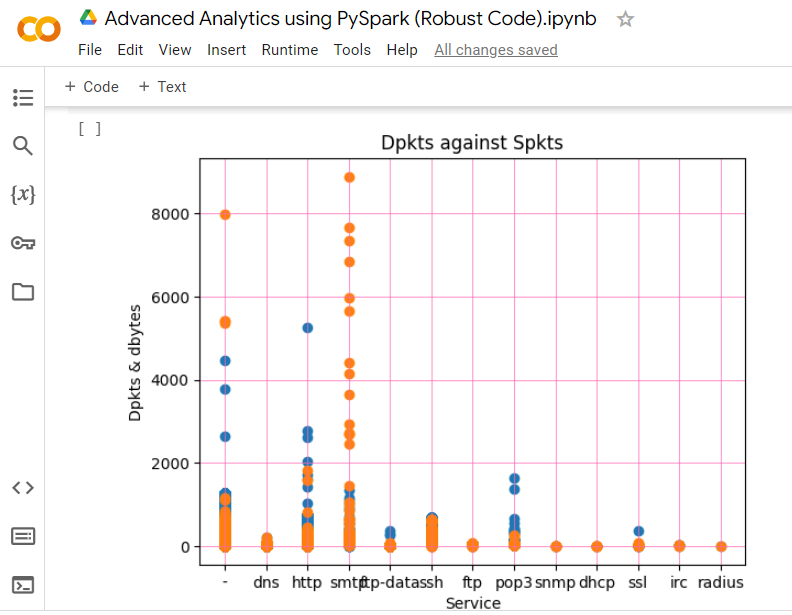
*Fig 9: Correlation between sbytes and dbytes columns*

The lack of correlation between sbytes and dbytes, as depicted in Figure 9, suggests that there is no association in the variations of these variables. In other words, the bytes transmitted from the source to the destination do not exhibit a proportional relationship with the bytes transmitted from the destination to the source.

See ***Correlation*** in my Google Colab Notebook - <https://colab.research.google.com/drive/1rqJmX3Uq_j2tMxvXAXJmmMMw2LgOteaW#scrollTo=bMhKngDkcqAK>

**Dpkts against Spkts:**

The code displayed(as visualized in the colab image in Fig 10) shows the relationship between the "Service" feature and two other features, "Dpkts" (destination packets) and "Spkts" (source packets), from the UNSW-NB15 dataset. It creates a scatter plot where the x-axis represents the "Service" values, and for every tenth data point, it shows corresponding values of "Dpkts" and "Spkts" on the y-axis. The title of the plot is "Dpkts against Spkts," and the axes are labeled accordingly. The plot is displayed with Matplotlib, with data points represented by scattered markers, and a grid is added for clarity.



*Fig 10: Correlation between sbytes and dbytes columns*

*For content summarization; We would stop here and continue to the next phase.*

*For detailed viewing pls check the google colab link:* <https://colab.research.google.com/drive/1rqJmX3Uq_j2tMxvXAXJmmMMw2LgOteaW#scrollTo=bMhKngDkcqAK>

# 3.2 Design and Build a Classifier

## A). Design and build a binary classifier over the dataset:

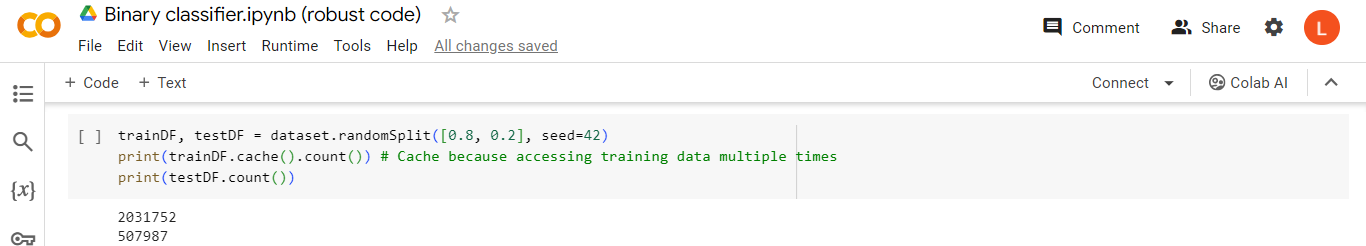
### First loading and then Renaming UNSW-NB15 Dataset

When I loaded the UNSWNB-15 dataset to the PySpark environment on Google Colab using data\_csv\_path = "/content/drive/MyDrive/UNSW-NB15.csv" # Replace with the actual path to your data\_dataset.csv file

data = spark.read.csv(data\_csv\_path, header=False, inferSchema=True)), It dawned on me that the data frame does not include column names. I observed that the attack\_cat column had null values for all instances of the 'Normal' attack category, and there were inaccuracies in some attack\_cat values (e.g., 'Backdoor' instead of 'Backdoors', 'Reconnaissance ' instead of 'Reconnaissance', 'Fuzzers 'instead of 'Fuzzers', and 'Shellcode ' instead of 'Shellcode'). To address this, I utilized the .withColumnRenamed() method to rename the column names and created a new DataFrame named "dataset." I then updated the null values in the attack\_cat column to 'Normal' within the "dataset" DataFrame. Subsequently, I used the .withColumn() method to modify the values in the attack\_cat column, resulting in the creation of a new DataFrame named "updated\_dataset."

**Data Splitting**

The data were randomly split into training and test sets and set seeds for reproducibility.



*Fig 11: Splitting of the Data*

**Processing Features and converting categorical variables into numerical representations.**

To construct a model for forecasting the attack category (attack\_cat) based on the dataset features (srcip, dstip, proto, state), the initial stage involves manipulating or preprocessing these features to comply with the format required by MLlib. As the algorithms, such as logistic regression, necessitate numeric features, I utilized StringIndexer and OneHotEncoder to transform categorical variables into a set of numeric variables with values restricted to 0 and 1.

### The .fit() function was invoked to obtain a StringIndexerModel, which was subsequently utilized to apply transformations to the dataset. Following this, the .transform() method of StringIndexerModel generated a new DataFrame with additional columns. Finally, I consolidated all feature columns into a unified feature vector using the VectorAssembler tool from MLlib.

### Definition of the Model

Also, the chosen model was defined using Logistic Regression (LR)as follows;

### Import Logistic Regression

### from pyspark.ml.classification import LogisticRegression

### Define the model using Logistic Regression

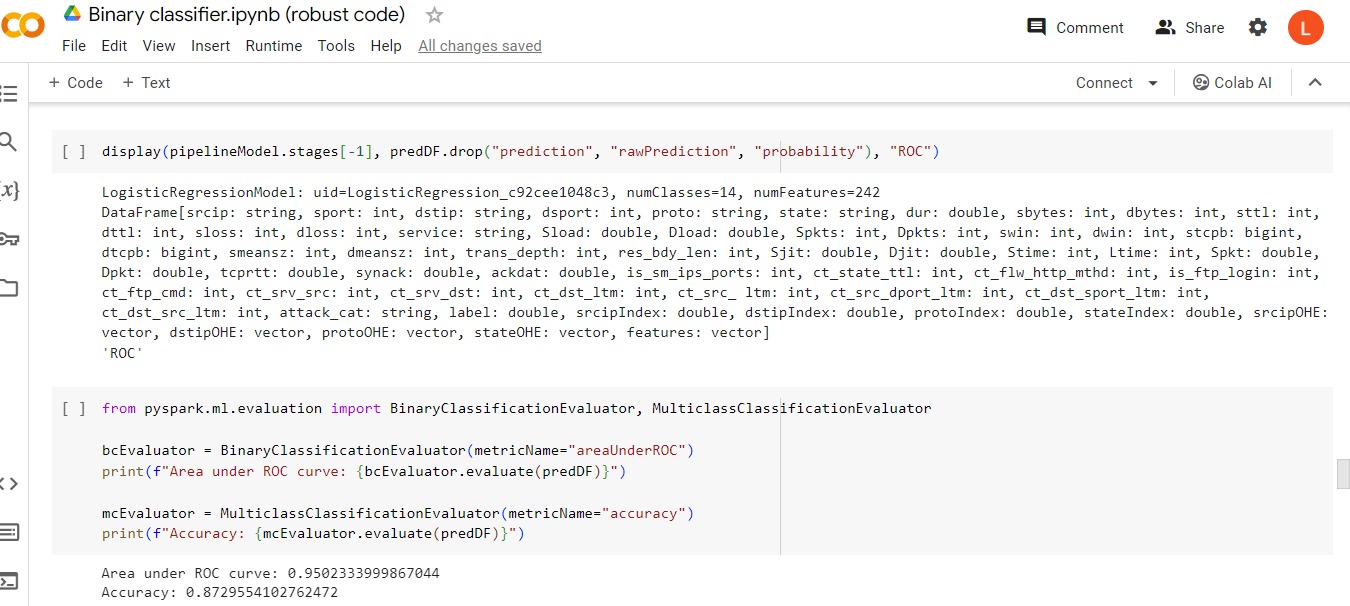
### logistic\_regression = LogisticRegression(featuresCol="features", labelCol="label", regParam=1.0)

### Building the Pipeline

### The automation of the pipeline ensures the consistent application of transformations to a dataset. This is achieved by defining the pipeline and subsequently applying it to the test dataset. Similar to the approach taken with StringIndexer, the Pipeline functions as an estimator. I invoked it using the pipeline.fit() method to obtain a PipelineModel, which acts as a transformer.

### Evaluation of the Model

To assess the model's performance, the BinaryClassificationEvaluator was employed to assess the ROC curve's area under, and the MulticlassClassificationEvaluator was used to validate accuracy. The ROC curve's area under is 0.9502324167040523, and the accuracy is 0.8729554102762472, as demonstrated in the results in Fig 12.



*Fig 12: Model Evaluation*

See code from my Google Colab Notebook - <https://colab.research.google.com/drive/1mFHP5QNatsemyByY25YDhyRH-uUYj6K5#scrollTo=b36stAsqwZPb>

## B). Multi-class classifier to classify data into ten classes (categories):

### Dropping missing values

After loading the UNSWNB-15 dataset to the PySpark environment on Google colab, This code is defining a new dataset named dataset by replacing null (missing) values in the "attack\_cat" column of the original UNSWNB15 dataset with the string value "normal". The na.fill method is used to fill null values, and it takes the specified value ("normal") and applies it to the specified column ("attack\_cat") to replace any null entries in that column. The resulting dataset is then displayed using the show() method.

**Next, Feature engineering and Pipeline**

### Subsequently, I then derived diverse attributes from the UNSW-NB15 dataset, serving as inputs to the machine, for utilization in prediction tasks.

### Pipeline Stage

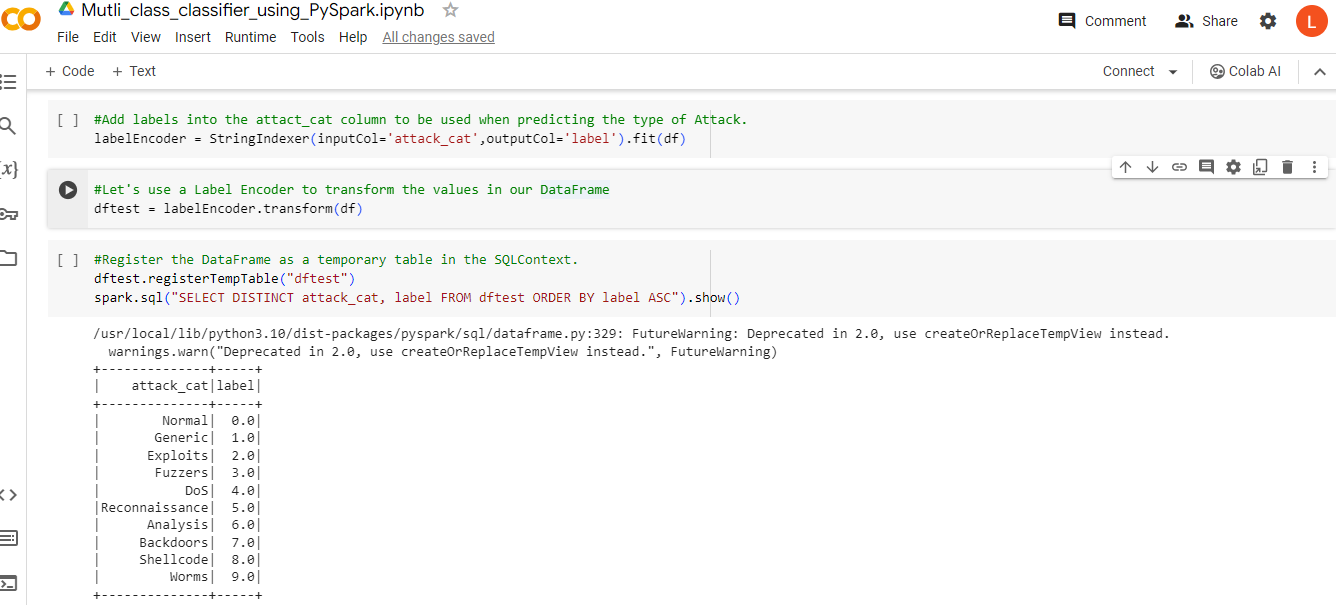
## There are a total of 5 pipeline stages, divided into Transformers and Estimators. Within the Transformers category, there are 4 stages (namely tokenizer, stopwords\_remover, vectorizer, and idf - Inverse Document Frequency) responsible for taking and fitting data into the features. The last stage, LogisticRegression, falls into the Estimators category and is utilized for constructing the model. LogisticRegression is a statistical analysis method used for predicting an output based on pattern recognition and analysis (section.io, 2021). The pipeline stages are organized sequentially, with the initial stage outputting a column named srcip transformed into mytokens. The subsequent transformations continue until reaching the vectorizedFeatures after the completion of the four initial pipeline stages. The VectorizedFeatures are then used as input for the final pipeline stage, LogisticRegression, where the model is built. This process follows a sequential order from the tokenizer stage to the idf stage as displayed in the figure below:.

## 

*Fig 13: Model Pipeline Stage*

Subsequently, I then proceeded to assign labels to the attack\_cat column utilizing the StringIndexer approach. These labels play a crucial role during prediction, aiding the model in understanding the intended prediction.

Following this, I divided the dataset into two segments: the primary set and the test set. These subsets will serve as inputs in the ultimate pipeline stage, facilitating the construction of estimators that process the data, train on it, and generate the model for subsequent predictions.



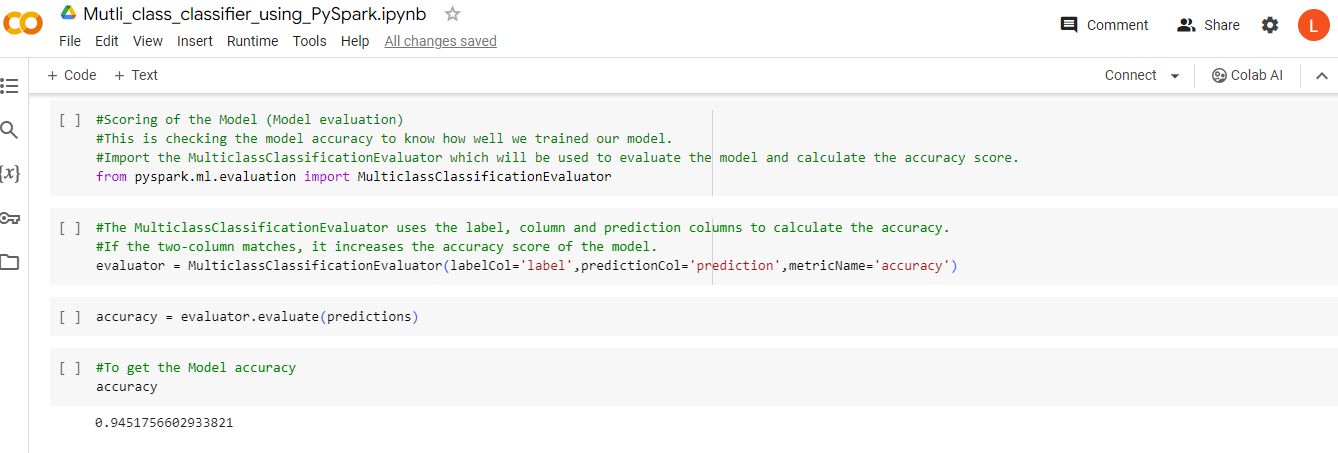
*Fig 14: Prediction Labels*

### Building the Trained Model

### Following the construction of the data pipeline, the five initialized stages were incorporated into the Pipeline() method using pipeline=Pipeline(stages=[tokenizer,stopwords\_remover, vectorizer,idf,lr]. Subsequently, the model was constructed by fitting it to the primary dataset through the fit() method, with the mainDataFrame as the parameter.

### Using the Accuracy Metrics/Testing for Accuracy

To find out how well the model trained and hence assess the model's accuracy, I incorporated the MulticlassClassificationEvaluator, a tool designed to evaluate the model and compute the accuracy score. This evaluator utilizes the label, column, and prediction columns to assess accuracy. When the two columns align, it enhances the accuracy score of the model. The calculated accuracy for the model is 0.9451756602933821, as depicted in Fig 15.



*Fig 15: Model Accuracy Testing/Score*

See code from my Google Colab Notebook - <https://colab.research.google.com/drive/1UByRvXVxye4Kz1h7KOKatl6e0X8s_dkI#scrollTo=X-NPeoguRR9K>

Task 4: Individual Assessment

# 4.1 Alternative Technologies for Tasks 2 and 3 and How they Differ

**Introduction**

The sheer size and the expansive nature of the UNSW-NB15 dataset presents unique challenges, surpassing the capabilities of traditional relational database management systems. This vast dataset poses hurdles in terms of effective querying, analysis, and the generation of insightful findings. Additionally, the sheer volume makes it intricate to employ visualization tools for presenting both numerical and graphical insights. To overcome these challenges, the adoption of big data technologies such as Apache Hive and PySpark becomes imperative for streamlined querying and analysis. Alternatively, various other big data tools, including Hadoop, HBase, Apache Impala, Pig, Presto, and more, offer viable alternatives for comprehensive analytics and visualization tasks. In the context of tasks 2 and 3, we'll delve into HBase and Apache Impala as potential alternative technologies.

HBase

### What is Apache HBase?

The Hadoop database is a big data store that is distributed and scalable. Apache HBase is an open-source, distributed, versioned, column-oriented database similar to Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. HBase provides Bigtable-like functionalities on top of Apache Hadoop, just as Bigtable does with the distributed data storage provided by the Google File System (StackShare).

### How it differs

Apache HBase stands out as an open-source, distributed, versioned, and column-oriented database designed to provide functionalities similar to Google's Bigtable. It operates on top of Apache Hadoop, offering distributed data storage akin to the Google File System.

Impala

### What is Apache Impala?

Described as "Real-time Query for Hadoop," Apache Impala stands as an MPP SQL query engine for Apache Hadoop, providing real-time querying capabilities for data stored in HDFS or Apache HBase.).

### How it differs

Impala lacks support for complex types, unlike Hive, and is more focused on short queries, being less fault-tolerant. Conversely, Spark accommodates both short and long-running queries and can recover from mid-query faults.

## Presto

### What is Presto?

### As per information from StackShare, Presto is a distributed SQL query engine that operates as open-source software, enabling the execution of interactive analytic queries on data sources across a broad range of sizes, spanning from gigabytes to petabytes.

### How it differs

### Now, Presto prioritizes achieving low latency, while Hive is designed for tasks demanding high query throughput and substantial memory resources. Additionally, Spark is known for its versatility and widespread use in data transformation and machine learning tasks (Alla, 2018). In contrast, Presto has built-in support for querying data in object stores like S3 and boasts an extensive array of connectors.

### 4.2 Surprising new thinking that evoked and/or neglected at my end

Everything related to big data and its analysis was entirely new territory for me, but my limited technical and database knowledge proved to be valuable. Numerous questions arose, such as how to manage this vast volume of data and how to effectively analyze big data. The added complexity was that the data involved network-related aspects, which I wasn't familiar with. Utilizing Hive query language on Cloudera allowed me to address some of these questions, yet Spark posed challenges, particularly in Task 3 where considerable time was spent.

**During my exploration of the UNSW-NB15 dataset, several unexpected observations emerged:**

1.The absence of column names in the UNSW-NB15 dataset was surprising.

2.All values in the 'attack\_cat' column where 'attack\_cat' is labeled as 'Normal' turned out to be null.

3.Beyond the nine attack categories, there were four misspelled values: 'Backdoors,' 'Reconnaissance ' instead of 'Reconnaissance,' 'Fuzzers ' instead of 'Fuzzers,' and 'Shellcode ' instead of 'Shellcode.'

**References**

Ahmed, H. (2020, November 26). A Comprehensive Analysis: Apache Spark vs. Impala. Naukri.com. [Online Article]. Available at: <https://www.naukri.com/learning/articles/spark-vs-impala>

Alla, S. (2018). Unveiling Real-Time Analytics with Apache Spark. In "Big Data Analytics with Hadoop 3: Constructing Highly Effective Analytics Solutions for In-Depth Insights into Your Big Data." [Book].

Apache Hive vs HBase: Unraveling the Variances. StackShare. [Online Comparison]. Available at: <https://stackshare.io/stackups/apache-hive-vs-hbase>

Meher, E. (2021, June 9). A Thorough Examination: Hive vs HBase - Unveiling 7 Critical Differences. Learn | Hevo. [Online Article]. Available at: <https://hevodata.com/learn/hive-hbase-comparison/>

Section.io. (2021, September 12). Mastering Multi-Class Text Classification with PySpark. Engineering Education (EngEd) Program | Section. [Online Guide]. Available at: <https://www.section.io/engineering-education/multiclass-text-classification-with-pyspark/#building-model>

Zoghi, Z., & Serpen, G. (2020). In-Depth Analysis of UNSW-NB15 Computer Security Dataset through Visualization. [Online Research Paper]. Available at: <https://arxiv.org/pdf/2101.05067.pdf>

Sincere, J. (2019, March 15). Navigating the Big Data Landscape: Apache Hadoop vs. Apache Hive. Tech Insights. [Online Article]. Available at: <https://www.techinsights.io/blog/hadoop-vs-hive/>

Apache Impala Explained: A Deep Dive into Real-time Querying. DataFlair. [Online Resource]. Available at: <https://data-flair.training/blogs/apache-impala/>

PrestoDB vs Hive: Unraveling the Differences. DataCamp. [Online Comparison]. Available at: <https://www.datacamp.com/community/tutorials/prestodb-vs-hive>