Apache Spark MLLib vs Apache Mahout

Big Data Analytics Project Report

Syed Muzammil Ahmed

ERP ID: 25371

May 31st, 2022

TABLE OF CONTENT

ABSTRACT: -

In this project, we are basically setting up Apache Spark MLlib and Apache Mahout pipeline using a docker container. The goal is to compare performance of both Apache Mllib and Apache Mahout by performing different machine learning algorithms on both Mllib and mahout and checking out their performance in terms of time, speed, accuracy, and efficiency.

INTRODUCTION: -

Machine learning is the new boom in the software industry which helps in training the computer to think, organize and process data by itself. The main intent of machine learning is that the machine learns to observe data, extract important information from it and grasp on its own to predict, recommend or alter any action without any human mediation. This requires various algorithms over varied systems. For the ease of these algorithms, Apache has come up with frameworks Mahout and Spark, which in its different ways helps in implementing machine learning in a better way.

Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. It is a lightning-fast unified analytics engine for big data and machine learning. To support Python with Spark, the Apache Spark community released a tool, PySpark. Using PySpark, one can work Python programming language and that's what we are using in this Project.

Whereas The Apache Mahout distributed linear algebra framework delivers new tools and methods for performing data analysis, building machine learning data pipelines, and implementing machine learning models in production. Mahout is not an in-memory framework like spark.

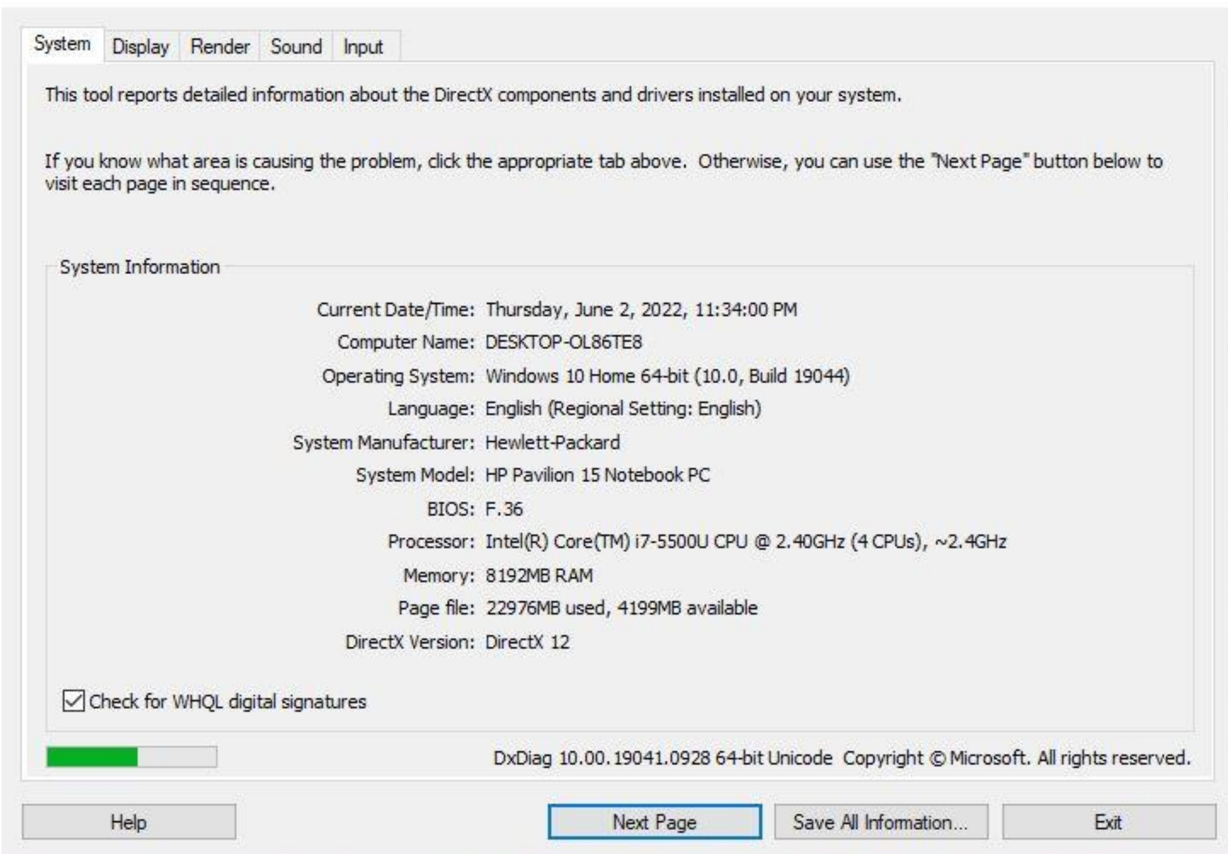
DATASET: -

Dataset we are using in this project is basically taken from famous dataset website Kaggle. Name of dataset is 1.6 million UK Accidents, this is a countrywide car accident dataset, which covers both urban and rural areas of the UK. Each accident record is described by a wide range of data attributes, including the accident location, weather, time, Date etc. This dataset is officially collected by UK Police department and contain mostly severe accidents in which police was also involved. Dataset is officially present on Government of UK website.

Size of our data is around 430mb, and it contain around **1.5million records** with **33 features**. Dataset consist of both categorical and continuous features.

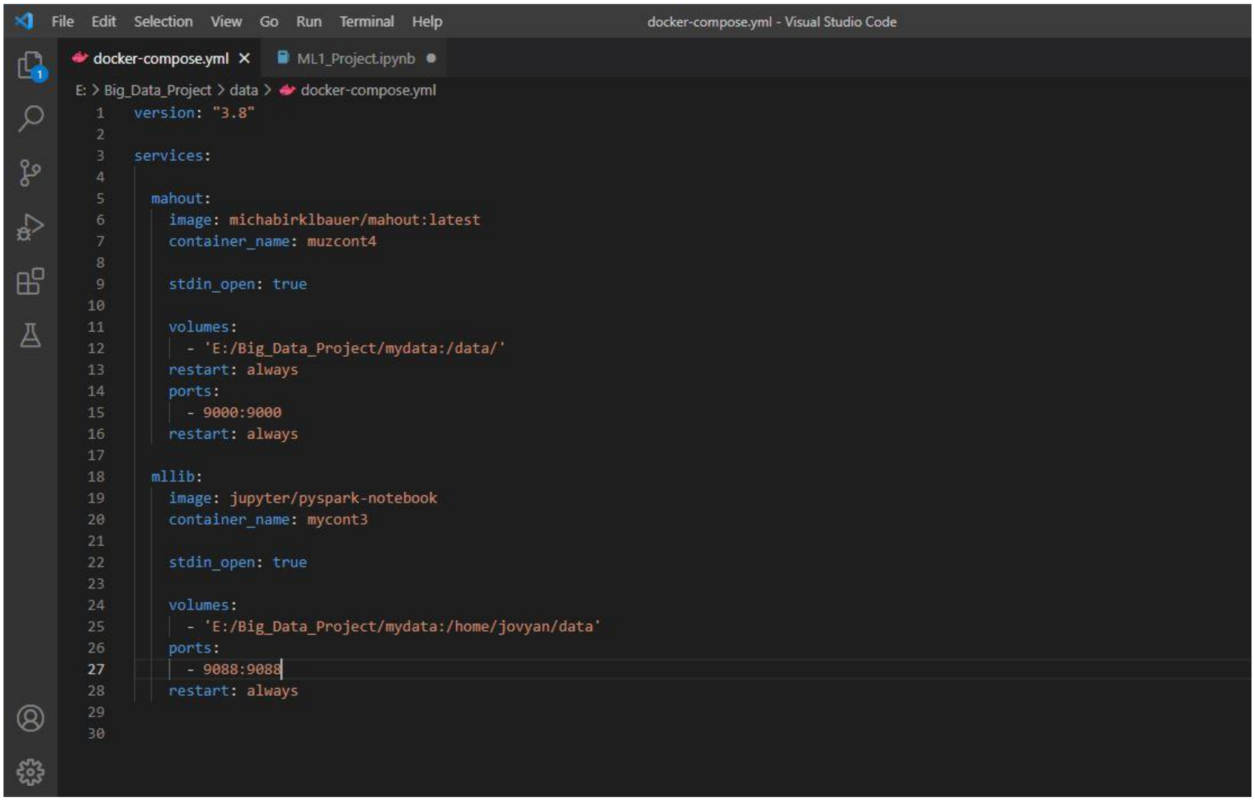
SYSTEM CONFIGURATION: -

System configuration we are using in this project are as follows:



CREATING DOCKER COMPOSE FILE: -

In this project, we are creating YAML File which will help us to access our containers. Our YAML file contain 2 containers: one for Mahout and one for MLLIB. Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down. We can access our container using YAML file with command "**Docker-compose up -d**".



Apache Spark Methodology: -

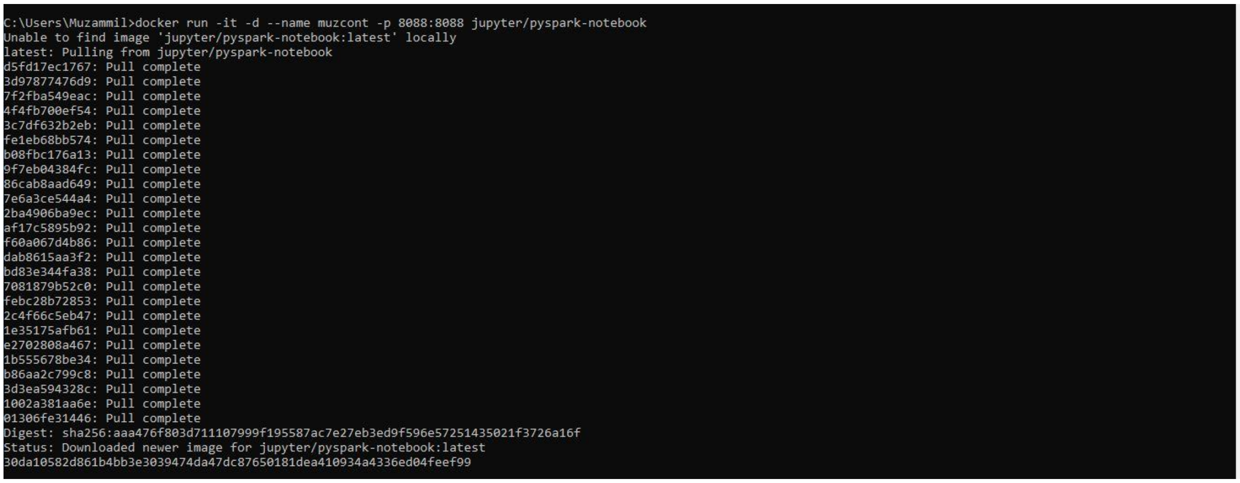
Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. It is a lightning-fast unified analytics engine for big data and machine learning

To support Python with Spark, the Apache Spark community released a tool, PySpark. Using PySpark, one can work with RDDs in Python programming language.

SETTING UP JUPYTER LAB ON DOCKER: -

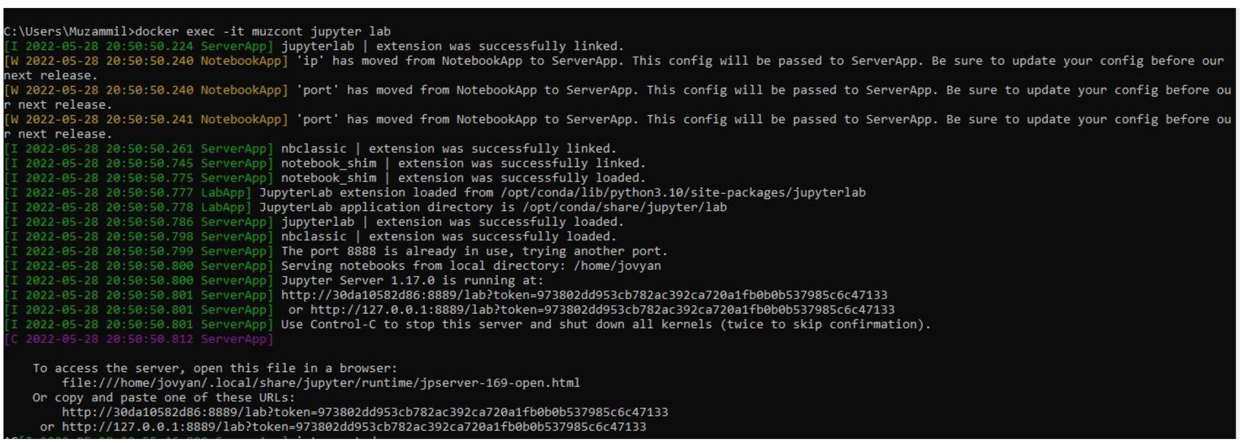
1- First thing we need to do is to create a docker container on which we we will pull docker image. We created container on docker named as “mycont2”.

2- After that we need to pull docker image from docker Hub into our container. Image size is around 2GB so it will take a quite time for downloading image.



```
C:\Users\Muzammil>docker run -it -d --name muzcont -p 8088:8088 jupyter/pyspark-notebook
Unable to find image 'jupyter/pyspark-notebook:latest' locally
latest: Pulling from jupyter/pyspark-notebook
d5fd17ec1767: Pull complete
3d97877476d9: Pull complete
7f2fba549eac: Pull complete
4f4fb700ef54: Pull complete
3c7df632b2eb: Pull complete
fe1eb68bb574: Pull complete
b08fbc176a13: Pull complete
9f7eb04384fc: Pull complete
86cab8aad649: Pull complete
7e6a3ce544a4: Pull complete
2ba4906ba9ec: Pull complete
af17c5895b92: Pull complete
f60a067d4b86: Pull complete
dab8615aa3f2: Pull complete
bd83e344fa38: Pull complete
7081879b52c0: Pull complete
febc28b72853: Pull complete
2c4f66c5eb47: Pull complete
1e35175afb61: Pull complete
e2702808a467: Pull complete
1b555678be34: Pull complete
b86aa2c799c8: Pull complete
3d3ea594328c: Pull complete
1002a381aa6e: Pull complete
01306fe31446: Pull complete
Digest: sha256:aaa476f803d711107999f195587ac7e27eb3ed9f596e57251435021f3726a16f
Status: Downloaded newer image for jupyter/pyspark-notebook:latest
30da10582d861b4bb3e3039474da47dc87650181dea410934a4336ed04feef99
```

3- Now, we need to setup jupyter lab on our container. For that reason, we execute jupyter lab command on container.



```
C:\Users\Muzammil>docker exec -it muzcont jupyter lab
[I 2022-05-28 20:50:50.224 ServerApp] jupyterlab | extension was successfully linked.
[W 2022-05-28 20:50:50.240 NotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our
next release.
[W 2022-05-28 20:50:50.240 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before ou
r next release.
[W 2022-05-28 20:50:50.241 NotebookApp] 'port' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before ou
r next release.
[I 2022-05-28 20:50:50.261 ServerApp] nbclassic | extension was successfully linked.
[I 2022-05-28 20:50:50.745 ServerApp] notebook_shim | extension was successfully linked.
[I 2022-05-28 20:50:50.775 ServerApp] notebook_shim | extension was successfully loaded.
[I 2022-05-28 20:50:50.777 LabApp] JupyterLab extension loaded from /opt/conda/lib/python3.10/site-packages/jupyterlab
[I 2022-05-28 20:50:50.778 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2022-05-28 20:50:50.786 ServerApp] jupyterlab | extension was successfully loaded.
[I 2022-05-28 20:50:50.798 ServerApp] nbclassic | extension was successfully loaded.
[I 2022-05-28 20:50:50.799 ServerApp] The port 8888 is already in use, trying another port.
[I 2022-05-28 20:50:50.800 ServerApp] Serving notebooks from local directory: /home/jovyan
[I 2022-05-28 20:50:50.800 ServerApp] Jupyter Server 1.17.0 is running at:
[I 2022-05-28 20:50:50.801 ServerApp] http://30da10582d86:8889/lab?token=973802dd953cb782ac392ca720a1fb0b0b537985c6c47133
[I 2022-05-28 20:50:50.801 ServerApp]  or http://127.0.0.1:8889/lab?token=973802dd953cb782ac392ca720a1fb0b0b537985c6c47133
[I 2022-05-28 20:50:50.801 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2022-05-28 20:50:50.812 ServerApp]

    To access the server, open this file in a browser:
        file:///home/jovyan/.local/share/jupyter/runtime/jpserver-169-open.html
    Or copy and paste one of these URLs:
        http://30da10582d86:8889/lab?token=973802dd953cb782ac392ca720a1fb0b0b537985c6c47133
     or http://127.0.0.1:8889/lab?token=973802dd953cb782ac392ca720a1fb0b0b537985c6c47133
```

4- Now, we need to upload dataset on the container. So, that we can perform Machine learning on it so for that we will execute Docker cp command on another command prompt.

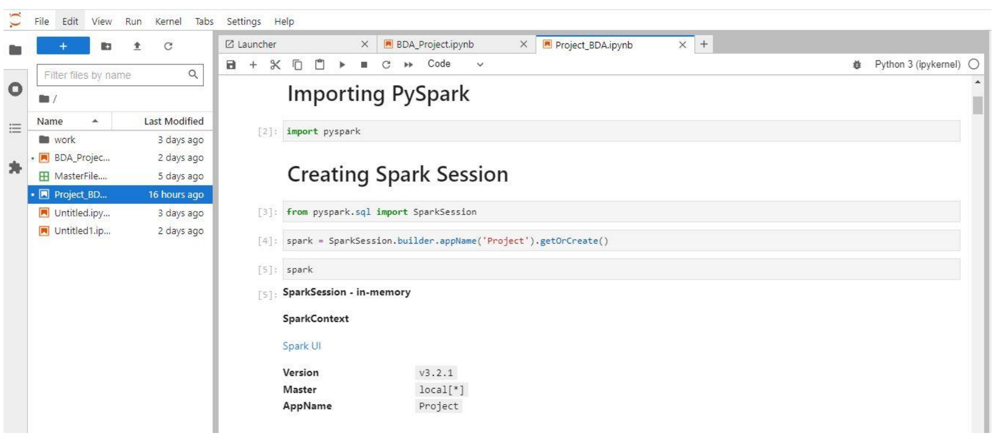
docker cp MasterFile.csv mycont2: \home\jovyan

MACHINE LEARNING USING PYSPARK: -

1) DATA LOADING AND SPARK SESSION: -

At first, we will import pyspark framework and create spark session. Spark session is a unified entry point of a spark application from Spark 2.0. It provides a way to interact with various spark's functionality with a lesser number of constructs. Instead of having a spark context, hive context, SQL context, now all of it is encapsulated in a Spark session.

After that we will import our dataset using pyspark commands and check the type of our data fram e. Data frame created by pyspark is different than pandas they are pyspark.sql. dataframe.DataFra me .

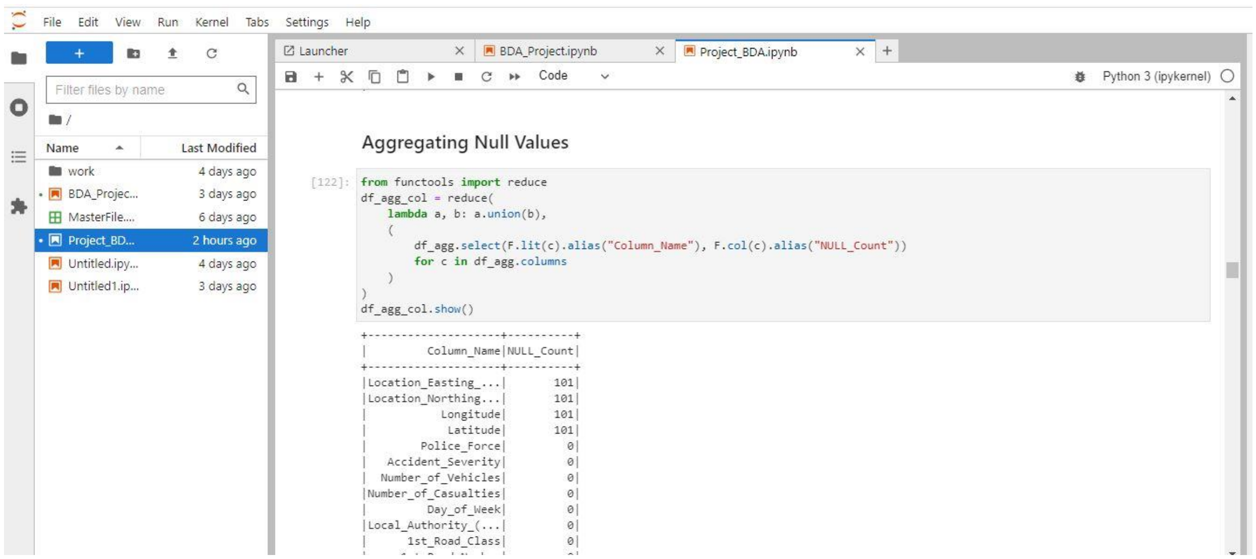


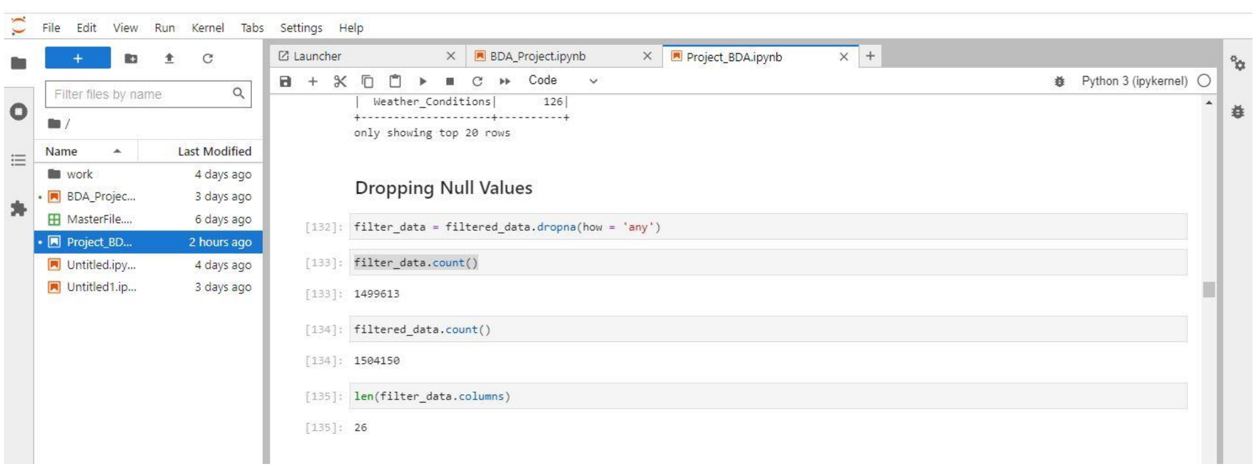
2- DATA PRE – PROCESSING: -

Now, we are performing Data pre-processing using pysparks code. We are performing different operations such as cleaning Null values, dropping null values, dropping irrelevant features, che cking different data ttypes of features etc.

1) Data Cleaning: -

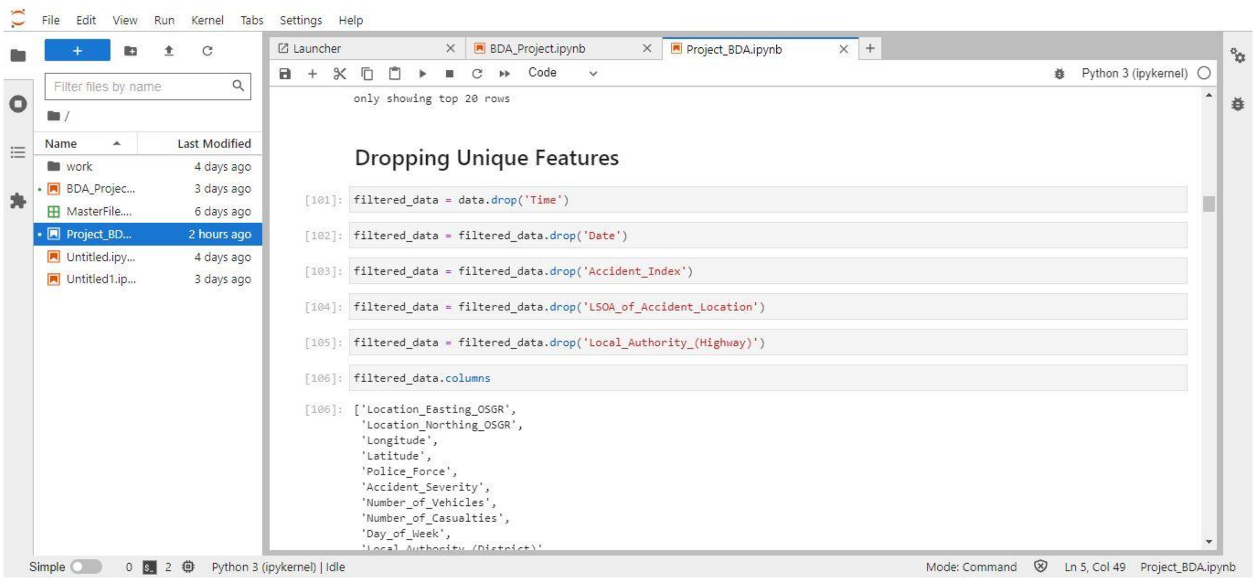
Data cleaning is the process of eliminating or changing data that is inaccurate, incomplete, irrelevant, redundant, or incorrectly formatted to prepare it for analysis. When it comes to data analysis, this data is usually not necessary or beneficial because it can slow down the process or produce false results. Faulty data will badly impact our result, so we must remove it. So now after importing our dataset, we now check for null values using python pyspark's library and find that one of our column junction details is almost null. So, if we drop null values before dropping this column then it will make all our dataset empty. So, we are dropping this column junction detail. Now, we again check for null values and drop all the null values using pandas "dropna(how = "any")" function.





2) Data Transformation: -

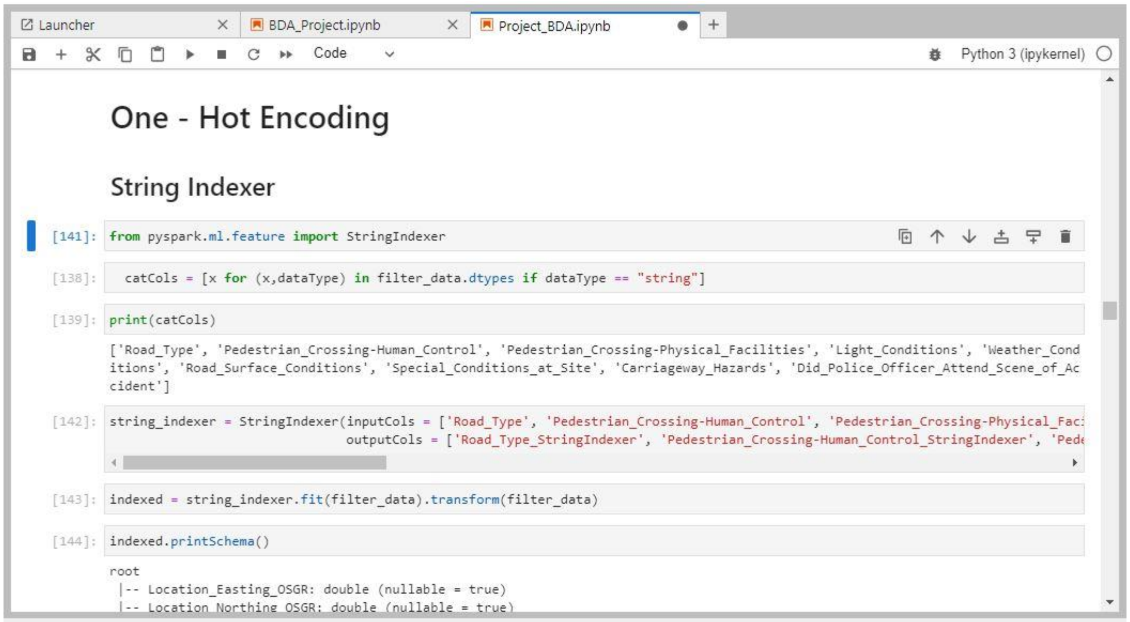
Now we are checking our columns and dropping those columns which have all its values u nique such as Date, time etc. Because in next step we must perform One-hot encoding. So, if also encode unique item columns, then it will generate too many columns which we cann ot manage. So, we are dropping all the columns which has totally unique values. First, we a re checking uniqueness of column using pandas. unique () function and check out different columns. The columns which we dropped here are Time, Date, Accident Index, LSOA Acc ident location, LSOA Highway.

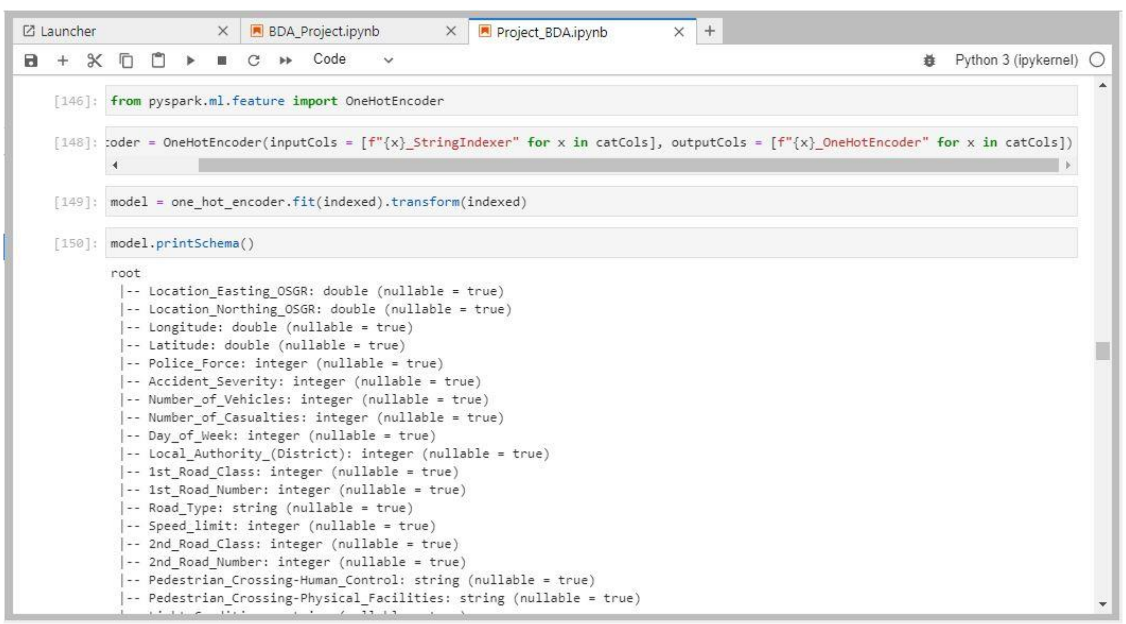


3) One-Hot Encoding: -

One hot encoding is a process of converting categorical data variables so they can be provi ded to machine learning algorithms to improve predictions. One hot encoding is a crucial p art of feature engineering for machine learning. our machine learning algorithms only unde rstand numbers, so we must provide them numbers by converting categorical variables into continuous variables. It Generate different columns and assign them binary values. One hot encoding is essential before running machine learning algorithm on data set. Some algorith ms can understand categorical data directly such as decision tree but most of the supervised learning algorithm cannot operate on categorical data, they require all input variables to be numeric and also generate output in numeric value. This technique of transforming column s into binary variables data set is quite famous in Supervised learning algorithm. Thes e binary variables are also known as dummy variables in statistics

Now, we must convert our categorical variable into continuous variables because Machine l earni ng only understand numbers. So, in pyspark, applying one hot encoding is not as si mple as pandas because in this first you must implement string indexer. it is necessary to us e StringIndexer before OneHotEncoder, because OneHotEncoder needs a column of catego ry indices as input. After string indexing, we apply One-Hot encoder.





3- Vector Assembler: -

After One-Hot encoding, we use vector assembler to combine all features into one feature. The idea here is to assemble everything into a vector. This is reasonable since after one-hot encoding and stuff, you end up with a mishmash of integers, floats, sparse vectors, and maybe dense vectors. And what we do next is bundle them altogether and call it features.

```
[151]: from pyspark.ml.feature import VectorAssembler

[152]: vector_assembler = VectorAssembler(
           inputCols = ["Location_Easting_OSGR","Location_Northing_OSGR","Longitude","Latitude","Police_Force","Number_of_Vehicles","Nu
       )

[153]: output = vector_assembler.transform(model)

[154]: model_filter_data = output.select("features","Accident_Severity")
```

4- Train – Test Split: -

Next, we are doing is splitting our data into train and test dataset. Our train dataset contains 1051002 and test data contain 448611.

Train - Test Split

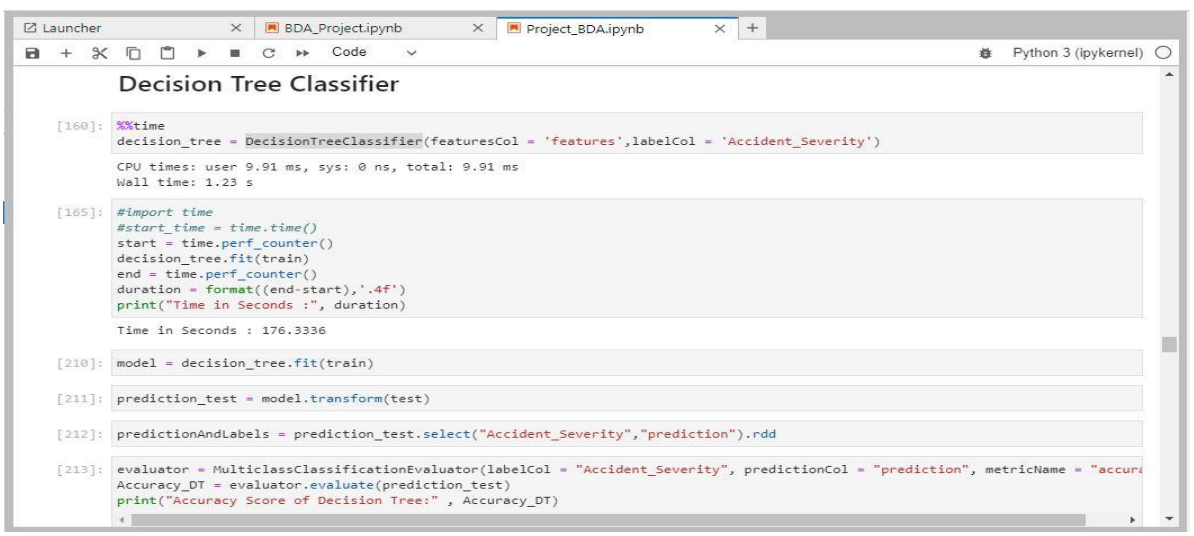
```
[166]: train , test = model_filter_data.randomSplit([0.7,0.3])
       print("Training Dataset Count: " + str(train.count()))
       print("Test Dataset Count: " + str(test.count()))

       Training Dataset Count: 1051002
       Test Dataset Count: 448611
```

Machine Learning Models: -

Now we are implementing our models and checking out their time and accuracy in MLlib. The m odels we are using are Decision Tree Classifier, Random Forest Classifier and Logistic Regressio n. For measurement of accuracy, we are using Multiclassification Evaluator module of Pyspark M l. So, below are the results we got after implementing models. All the models are implemented on basic parameters. Accuracy of each model is calculated using Multiclass evaluator module of a pyspark ml framework.

Models	Speed/Time	Accuracy
Decision Tree	176.33 sec	85.15%
Random Forest	347.45 sec	85.15%
Logistic Regression	107.33 sec	85.14%



Random Forest Classifier

```
[167]: from pyspark.ml.classification import RandomForestClassifier

       rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'Accident_Severity')
```

```
[168]: start = time.perf_counter()
       rf.fit(train)
       end = time.perf_counter()
       duration = format((end-start),'.4f')
       print("Random Forest Classifier Time in Seconds :", duration)
```

```
Random Forest Classifier Time in Seconds : 347.4570
```

```
[205]: model = rf.fit(train)
```

```
[206]: prediction_test = model.transform(test)
```

```
[208]: predictionAndLabels = prediction_test.select("Accident_Severity","prediction").rdd
```

```
[214]: evaluator = MulticlassClassificationEvaluator(labelCol = "Accident_Severity", predictionCol = "prediction", metricName = "accur
       Accuracy_RF = evaluator.evaluate(prediction_test)
       print("Accuracy Score of Random Forest:" , Accuracy_RF)
```

```
Accuracy Score of Random Forest: 0.8515105514577217
```

Logistic Regression

```
[171]: from pyspark.ml.classification import LogisticRegression

       log_reg = LogisticRegression(featuresCol='features',labelCol='Accident_Severity')
```

```
[181]: start = time.perf_counter()
       log_reg.fit(train)
       end = time.perf_counter()
       duration = format((end-start),'.4f')
       print("Logistic Regression Time in Seconds :", duration)
```

```
Logistic Regression Time in Seconds : 107.3300
```

```
[189]: log_reg2 = LogisticRegression(featuresCol='features',labelCol='Accident_Severity', maxIter = 20)
       model = log_reg2.fit(train)
```

```
[191]: prediction_test = model.transform(test)
```

```
[192]: prediction_test.show()
```

```
+--------------------+-----------------+--------------------+--------------------+----------+
|            features|Accident_Severity|       rawPrediction|         probability|prediction|
+--------------------+-----------------+--------------------+--------------------+----------+
|(57,[0,1,2,3,4,5,...|                3|[-8.7757203119566...|[3.38519414336258...|       3.0|
```

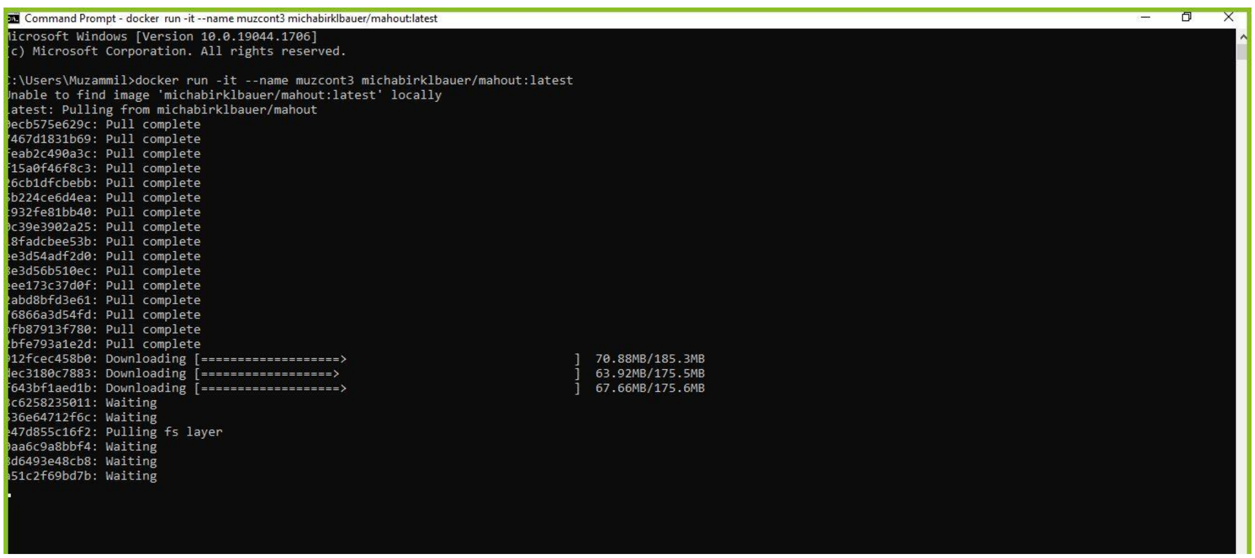
APACHE MAHOUT METHODOLOGY

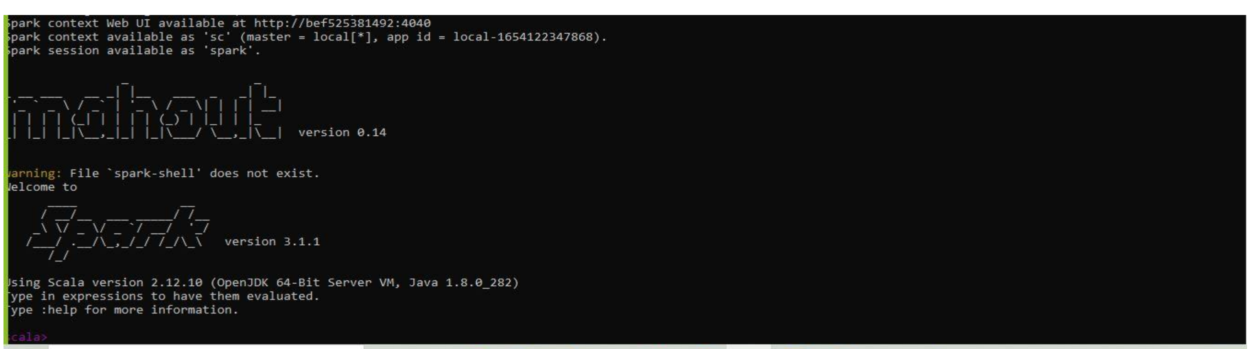
Now, after getting results from apache spark, we must move forward to apply ML models on Apache M ahout, which is also framework of Hadoop, specially designed to perform ML models on Big Data sets. Mahout run over HDFS, so its speed is known to be quite slow as compared to apache spark which is i n-memory framework. Mahout is rarely used, many of models are outdated and very less resource is avail able on internet on this framework.

SETTING UP MAHOUT CONTAINER: -

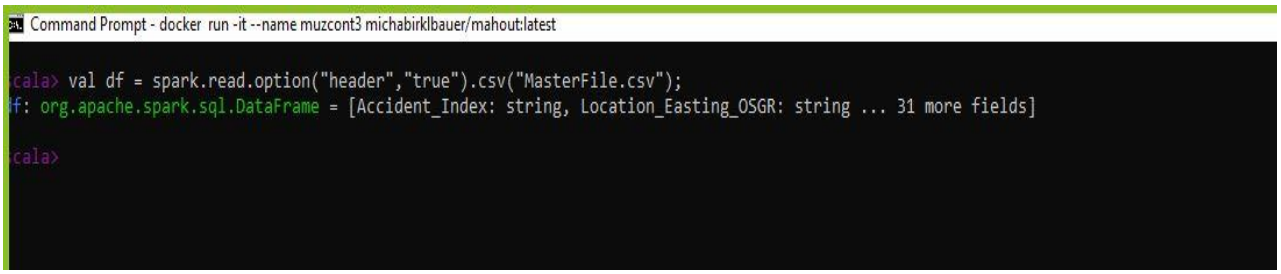
1- So, we need to create separate container for apche mahout, on which we we will pull docker i mage. We created container on docker named as "mycont3". Apache Mahout only support mo dels in Java or Scala language so python is not supported in Mahout. So here we are also pulling i mage from Docker Hub.

2- After that we need to pull docker image from docker Hub into our container. Image size is around 3.9 GB so it will take a quite time for downloading image.



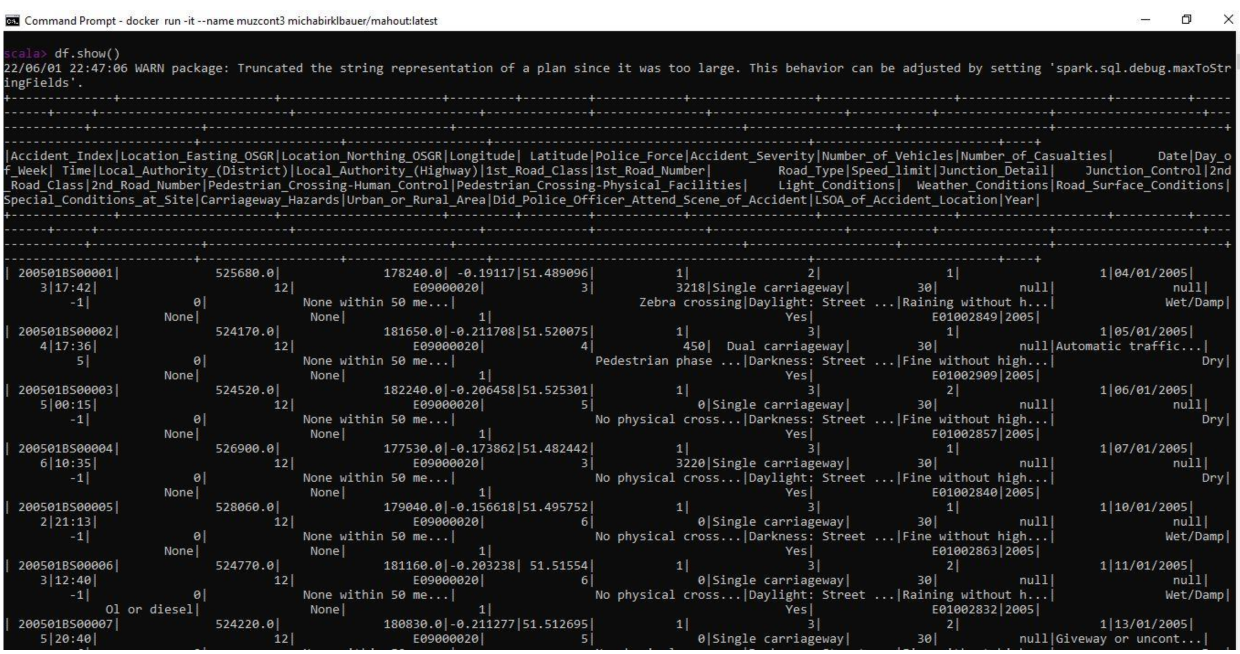


3- Now, we need to upload dataset on the container. So, that we can perform Machine learning on it so for that we will execute Docker cp command on another command prompt. And the import dataset on Scala using spark.read command.



DATA EXPLORATION ON SCALA: -

Now we are performing some data exploration on Scala. We will check out rows and data structures to check and explore more about the data.



```
scala> df.counts
<console>:46: error: value counts is not a member of org.apache.spark.sql.DataFrame
       df.counts
          ^

scala> df.counts()
<console>:46: error: value counts is not a member of org.apache.spark.sql.DataFrame
       df.counts()
          ^

scala> df.count
res4: Long = 1504150
```

Importing some importing features

```
Command Prompt - docker exec -it muzcont4 /bin/bash

scala> import org.apache.mahout.math._
import org.apache.mahout.math._

scala> import org.apache.mahout.math._
import org.apache.mahout.math._

scala> import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.scalabindings._

scala> import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm._

scala> import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.scalabindings.RLikeOps._

scala> import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.math.drm.RLikeDrmOps._

scala> import org.apache.mahout.sparkbindings._
import org.apache.mahout.sparkbindings._

scala>

scala> implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)
sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = org.apache.mahout.sparkbindings.SparkDistributedContext@7c8c5798
```

We can’t process more and not able to run any model on Apache Mahout because of less resources available on Apache mahout on internet. I am not able to find anything useful enough to train my model on Apache mahout. Even Mahout official documentation is not providing any relevant information about mahout algorithms. So, it is very difficult for us at a moment to run any model for comparison. Only one algorithm of Least linear regression is available, but our dataset is classification, and no information is available on that,so that’s why we are leaving it here.

DIFFICULTIES DURING PROJECT:

These are some difficulties I faced during this project.

1- Naïve Bayes algorithm on pyspark, not working on Jupyter lab and giving Java Error.
2- Gradient Boosting on Pyspark not supporting multi-class problem.
3- No proper resources of mahout available on internet.
4- Mahout algorithms and resources not even available on official sites.
5- Mahout does not support Python.

CONCLUSION: -

So, I am only able to complete pyspark modelling in this project and not able to perform modelling on mahout because of almost no resource available on modelling on mahout. I have tried my best to find some useful resource but not able to succeed. So that's why we are not able to compare performance of both Mllib and Mahout. I believe Apache Mahout is outdated now as people are using spark because of its high speed and efficiency. Apache mahout algorithms resources are also deleted from official page of apache mahout.