ILLINOIS INSTITUTE OF TECHNOLOGY

Machine Learning using Spark MLlib

Case study: Identifying types of Raisins

BIG DATA TECHNOLOGIES
(CSP – 554)

Submitted by:

Karan Sathiayan, ksathiayan@hawk.iit.edu, A20469856

Nandan Pandey, npandey2@hawk.iit.edu, A20493184

Niveditha Mangala Venkatesha, nmangalavenkatesha@hawk.iit.edu, A20466182

Ashwathi Sreekumar, asreekumar@hawk.iit.edu, A20477241

INTRODUCTION

Machine learning is an algorithm that performs predictions without explicitly being programmed to do so. Data is the essential input for the machine learning model. Big Data is the solution to handle large datasets. By combining Big Data and machine learning, we can handle massive data and make predictions. It can predict numerical values or categorical values using techniques called regression and classification. Before making the predictions, the input data must be clean and tidy; this process is called data cleaning. To understand more about the dataset, exploratory data analysis can be performed. Then split the input dataset into train and test datasets and finally using the training dataset, build the model and predict the target variable on fresh/ test dataset.

Apache Spark is a data processing framework that can effectively analyze large datasets and distribute data processing jobs across multiple computers, either on its own or in concert with other distributed computing technologies. Spark MLlib is used to execute machine learning in Apache Spark. MLlib is a Spark-based scalable Machine Learning toolkit that focuses on both high-quality algorithms and fast performance.

At a high level, it provides tools such as:

- **ML Algorithms:** common learning algorithms such as classification, regression, clustering, and collaborative filtering
- **Featurization:** feature extraction, transformation, dimensionality reduction, and selection
- **Pipelines:** tools for constructing, evaluating, and tuning ML Pipelines
- **Persistence:** saving and loading algorithms, models, and Pipelines
- **Utilities:** linear algebra, statistics, data handling, etc.

MLlib is a tool for analyzing big data on Hadoop, Apache Mesos, Kubernetes, standalone, or the cloud with support for Java, Python, R, and Scala. It can run on Hadoop, Apache Mesos, Kubernetes, and can be implemented in Java, Python, and R. With Spark MLlib, almost all machine learning algorithms are available 100x faster than with Map Reduce.

LITERATURE REVIEW

1. **Big Data Machine Learning using Apache Spark MLlib**

A study paper by Mehdi Assefi, Ehsun Behravesh, Guangchi Liu, and Ahmad P. Tafti was utilized as a reference for this project implementation in Spark-MLlib. To evaluate the ability of the Apache Spark MLlib 2.0 library in analyzing big data sets, the project focused on a set of supervised (classification) methods, including SVM (Support Vector Machine), Decision Tree, Naive Bayes, and Random Forest, along with a widely used unsupervised (clustering) machine learning algorithm, namely K-Means.

A total of Six various big datasets were used to analyze and compare the Apache Spark MLlib 2.0 performance. Five datasets from the UCI Machine Learning Repository, and a dataset from the US government's Bureau of Transportation Research and Innovative Technology Administration (RITA) Web sites.

2. **Towards Near-Real-Time Intrusion Detection for IoT Devices using Supervised Learning and Apache Spark**

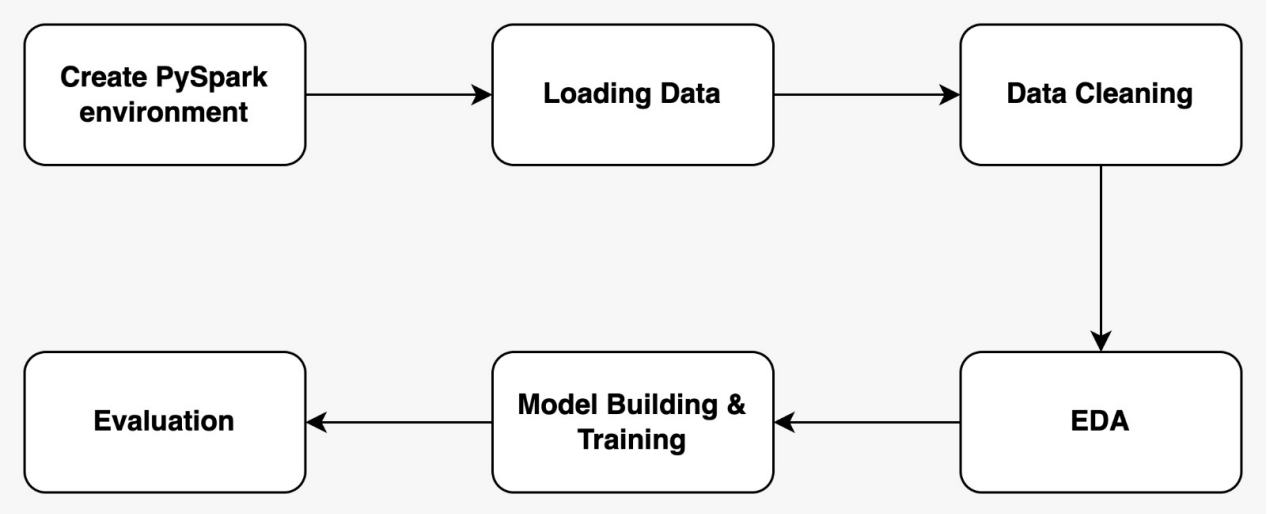
In this research journal the authors - Valerio Morfino & Salvatore Rampone worked in the field of the Internet of Things (IoT). They used supervised machine learning algorithms included in the MLlib library of Apache Spark, a fast and general engine for big data processing. The dataset named - "SYN-DoS" was obtained from the Apache Spark dataset website. They used supervised learning algorithms - 'Random Forest Algorithm' & 'Decision Tree' for the detection of SYN-DOS cyber-attacks on IoT devices. The results showed that all the Spark algorithms obtained a very good identification accuracy (>99%). Overall, the Random Forest algorithm achieved an accuracy of 1. They also reported a very short training time of 23.22 sec for Decision Tree with more than 2 million rows.

OBJECTIVE

The goal of this project is to **classify** the type of raisin (Kecimen or Besni raisin) from the seven morphological features that were extracted which include - (Area, Perimeter, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea & Extent). In this project, the binomial class is being predicted i.e., whether the given raisin is of Kecimen or Besni type. A comparative study of various supervised learning classification techniques like logistic regression, Naïve Bayes and Random Forest are performed from the Spark MLlib package.

WORK-FLOW

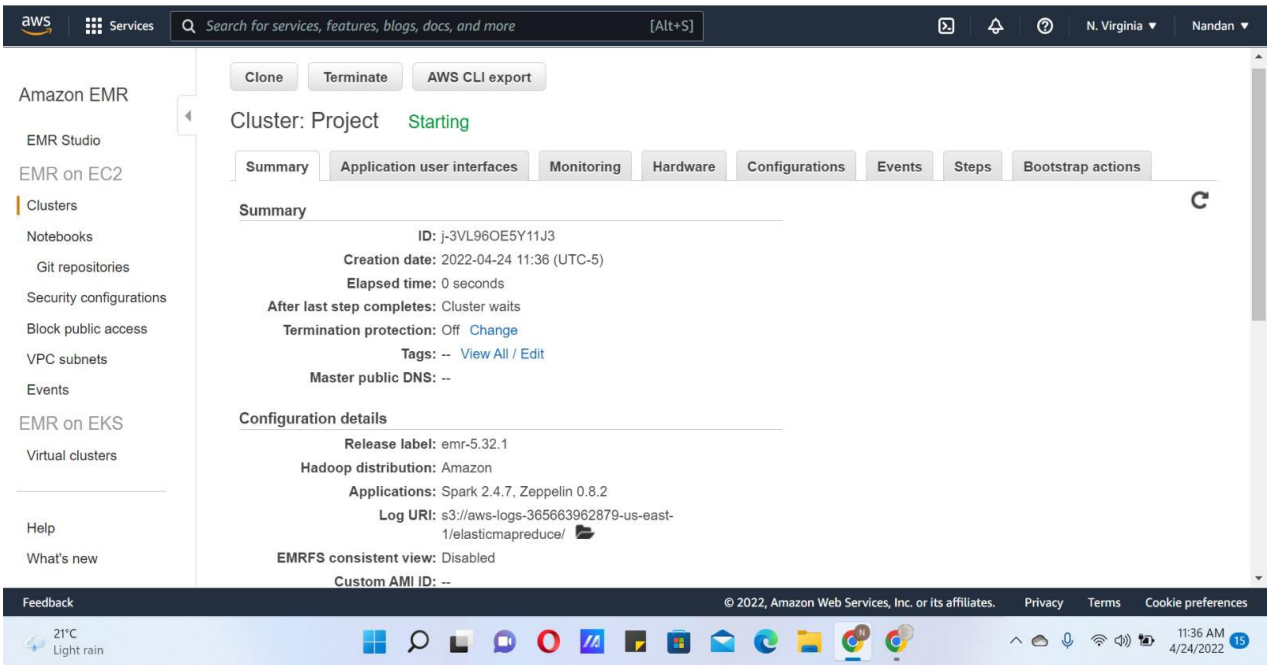
The steps implemented in this project are explained in the flowchart diagram below. The first and foremost task is to set up the PySpark environment. In this project, the PySpark environment is set up using Hadoop EMR cluster. Once the environment is set up, the data can be loaded in csv format. Then the data pre-processing steps which involve Data cleaning, Exploratory Data Analysis and finally Model building and prediction. Once the predictions are made, it is essential to evaluate the model and decide which models can be deployed in real-time for a particular use case.

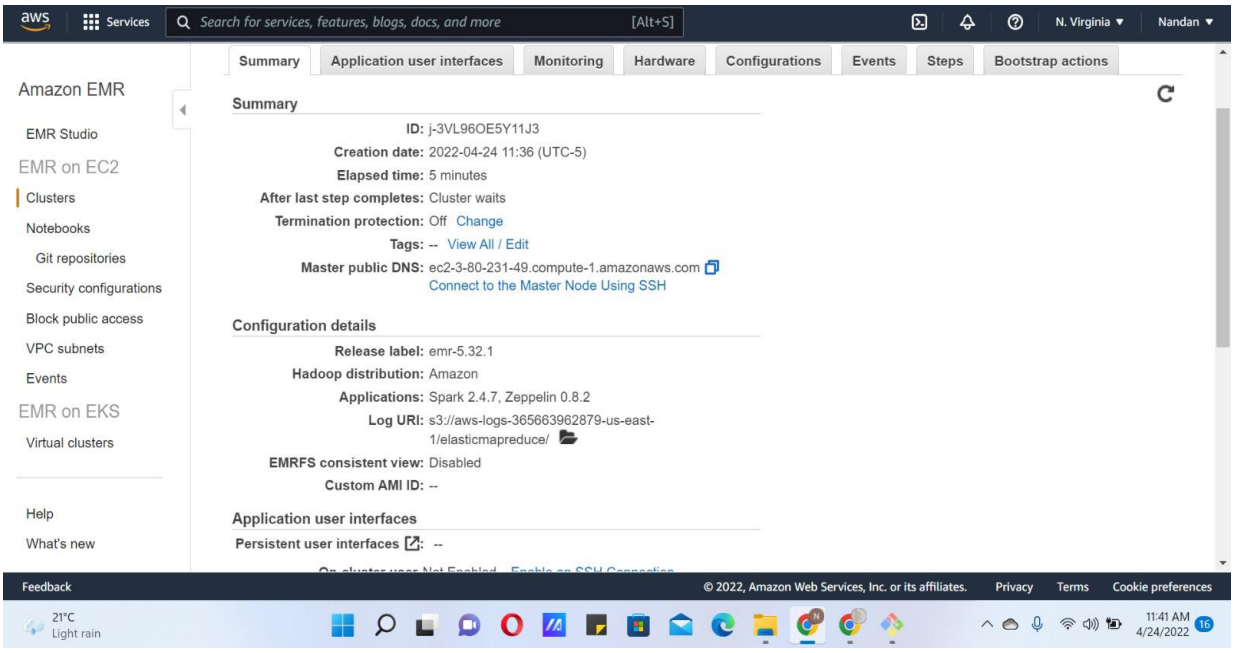


1. CREATE PYSPARK ENVIRONMENT

To use Spark MLlib, we have used an AWS EMR instance. Let us see how to create an EMR instance.

Step 1: Create a cluster by selecting EMR service and select Spark as software configuration as below.

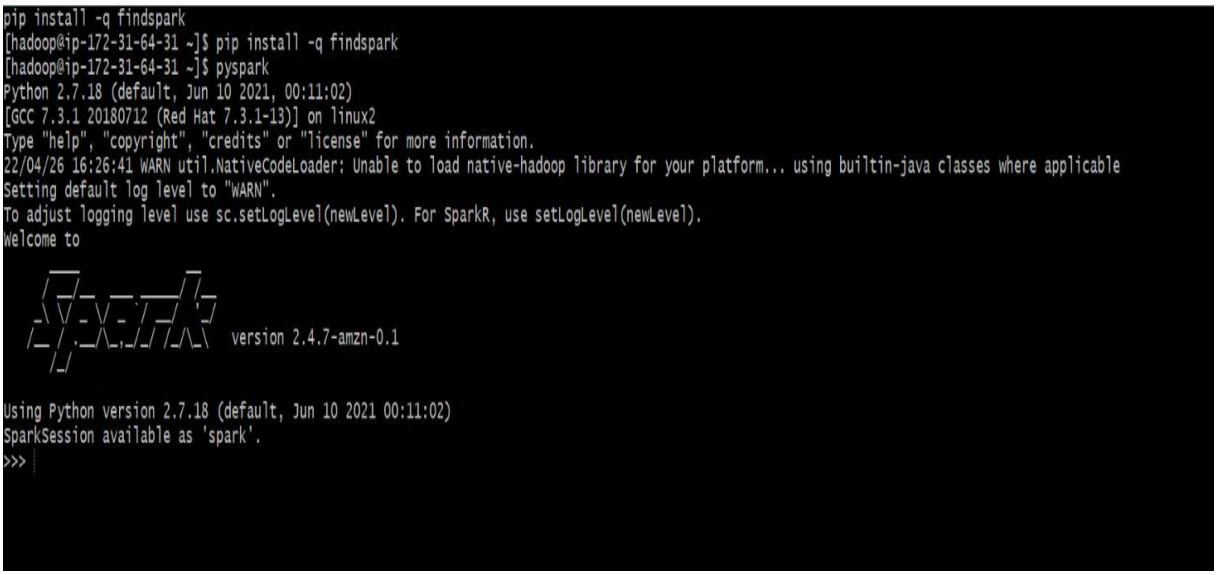




Step 2: Once the cluster is ready, open terminal and ssh hadoop@<DNS Name> -I EMR-key-pair.pem. You should see the below screen.

ssh -i C:/emr-key-pair.pem hadoop@ec2-3-80-231-49.compute-1.amazonaws.com

Step 3: Enter the pyspark command, which will redirect to the pyspark shell, where you can execute the .py files you have written for machine learning models. The pyspark shell looks something like this.

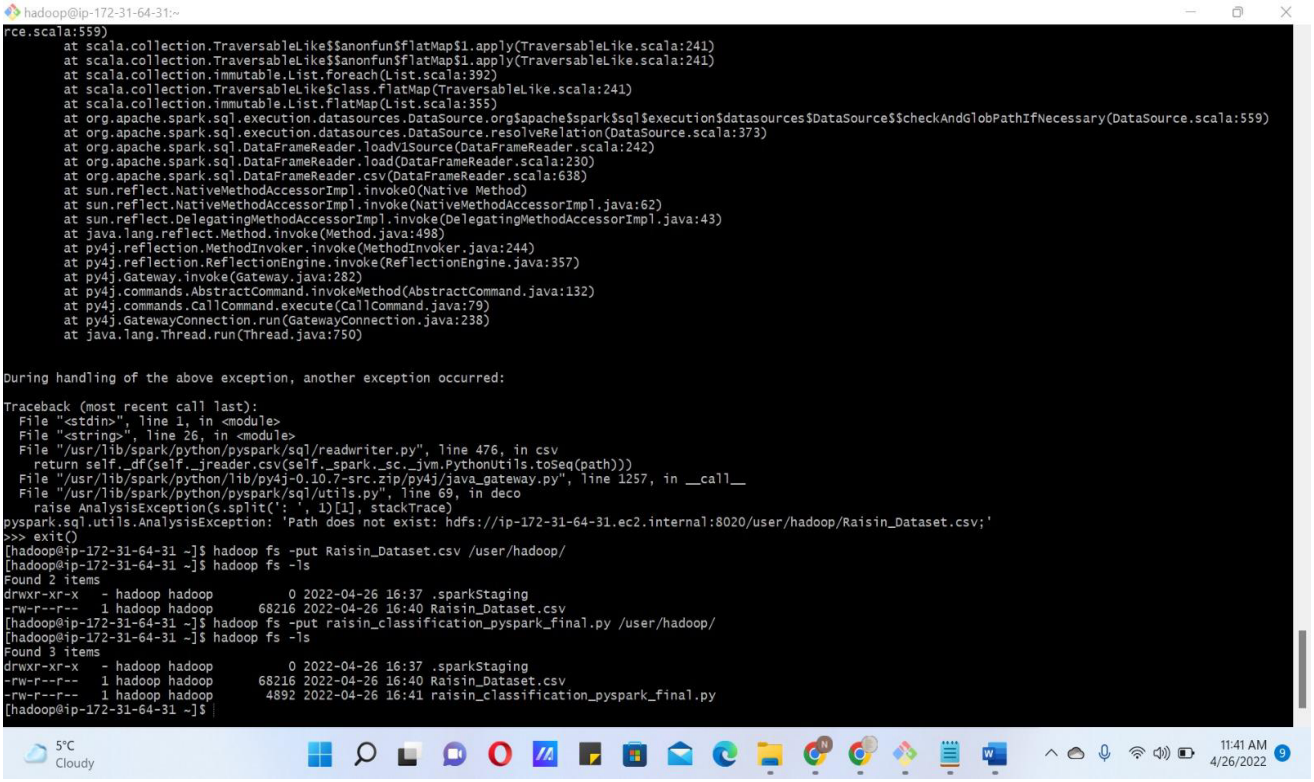


Step 4: Before running the code files, we need to send the local files to the Hadoop cluster. This can be done by running the below command in your Windows/Mac machine terminal.

In this way, you need to send creditcard.py, creditcard.csv, and Requirements.txt files available in the classification folder.

After this, send the CSV file to /user/Hadoop using the below command:

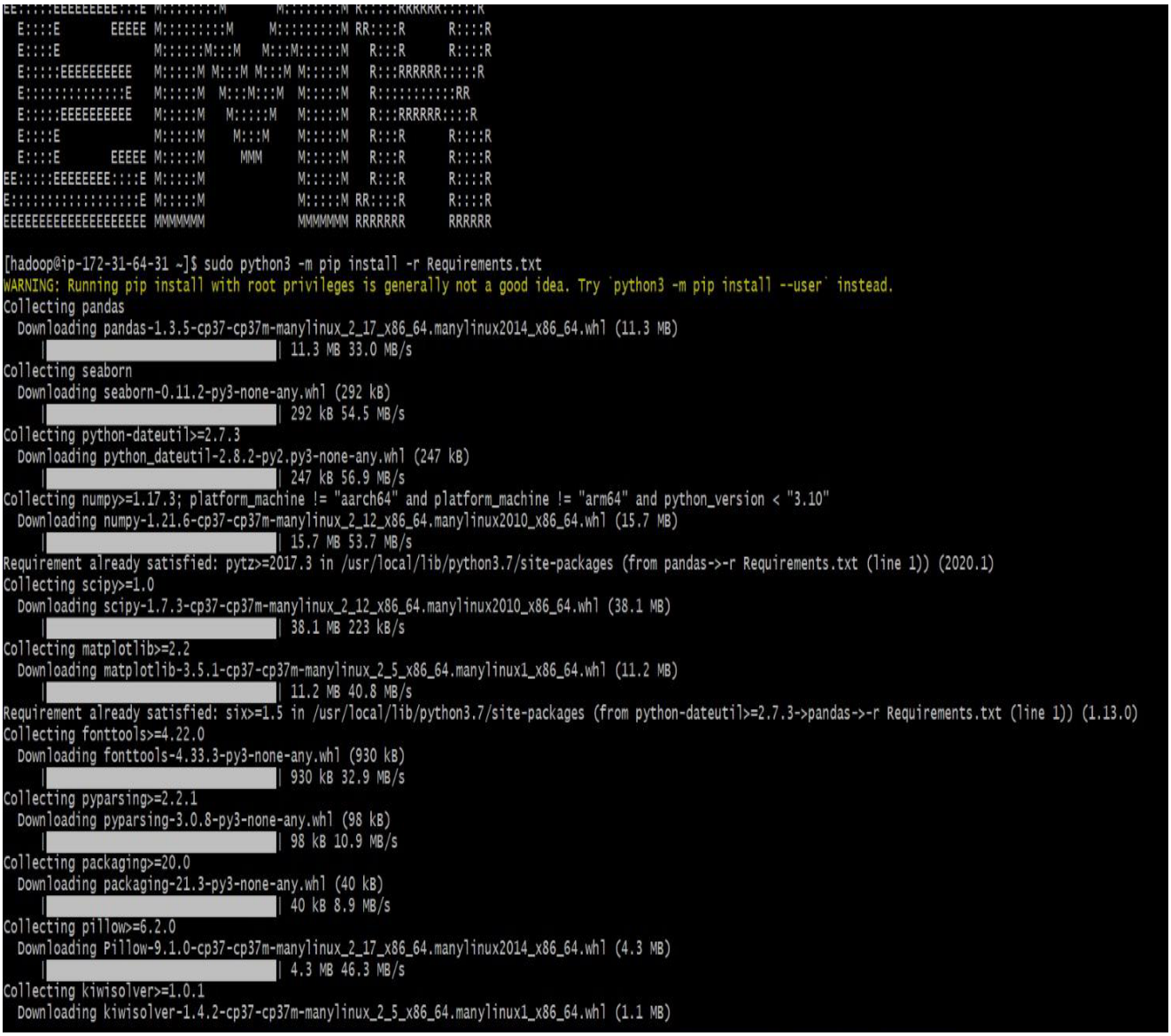
hadoop fs -copyFromLocal /home/hadoop/Raisin_Dataset.csv /user/hadoop/Raisin_Dataset.csv



Step 5: Once the above command is successful, we can access the files in the Hadoop cluster. You need to install the required libraries to run the creditcard.py program. This can be done using the Requirements.txt file.

sudo python3 -m pip install -r Requirements.txt

This command installs the required libraries. Now, we are all ready to run the program file to train the machine learning model.



2. LOAD DATA

Dataset Information

Data consist of images of Kecimen and Besni raisin varieties grown in Turkey were obtained with CVS. A total of 900 raisin grains were used, including 450 pieces from both varieties. These images were subjected to various stages of pre-processing and 7 morphological features were extracted. These features have been classified using three different artificial intelligence techniques.

Attribute Information:

i. Area: Gives the number of pixels within the boundaries of the raisin.
ii. Perimeter: It measures the environment by calculating the distance between the boundaries of the raisin and the pixels around it.
iii. MajorAxisLength: Gives the length of the main axis, which is the longest line that can be drawn on the raisin.
iv. MinorAxisLength: Gives the length of the small axis, which is the shortest line that can be drawn on the raisin.
v. Eccentricity: It gives a measure of the eccentricity of the ellipse, which has the same moments as raisins.
vi. ConvexArea: Gives the number of pixels of the smallest convex shell of the region formed by the raisin.
vii. Extent: Gives the ratio of the region formed by the raisin to the total pixels in the bounding box.
viii. Class: Kecimen and Besni raisin.

A spark data frame is created for reading the Raisin_Dataset.csv file and it is done as shown below.

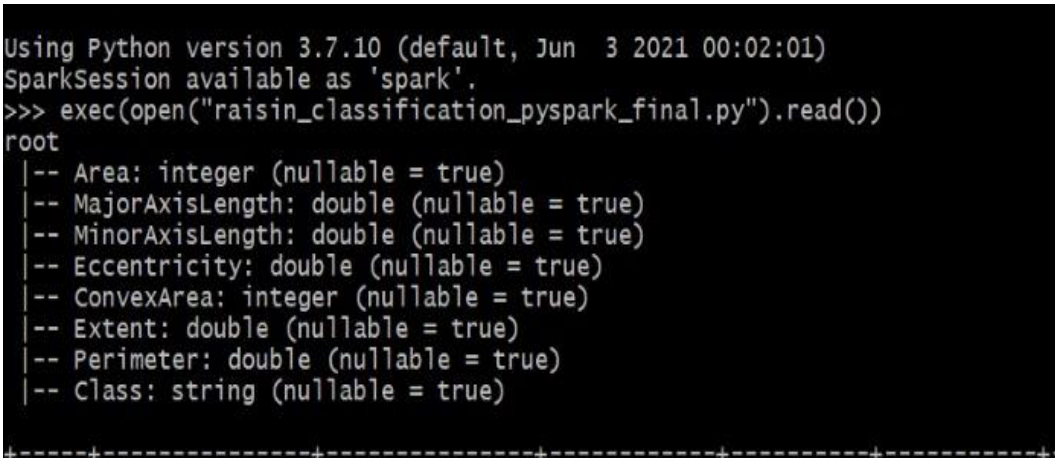
```
# Loading the csv file
df = spark.read.csv('Raisin_Dataset.csv',inferSchema=True, header =True)

# Printing the dataframe Schema, data types
df.printSchema()

# Printing the top 10 rows of the dataFrame
df.show(10)
```

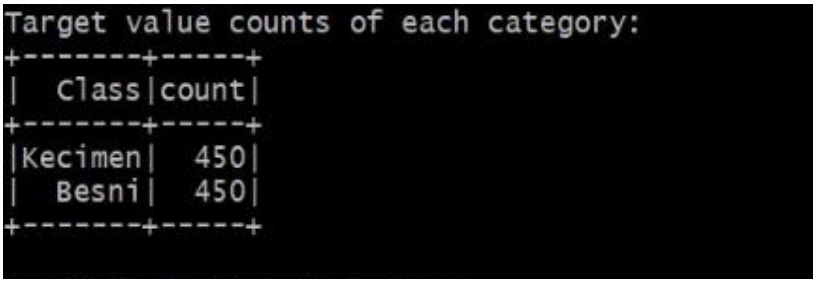
Dataframe Schema

Prints the schema of the Spark data frame and the columns available. There are 7 input features, and one target variable called Class.

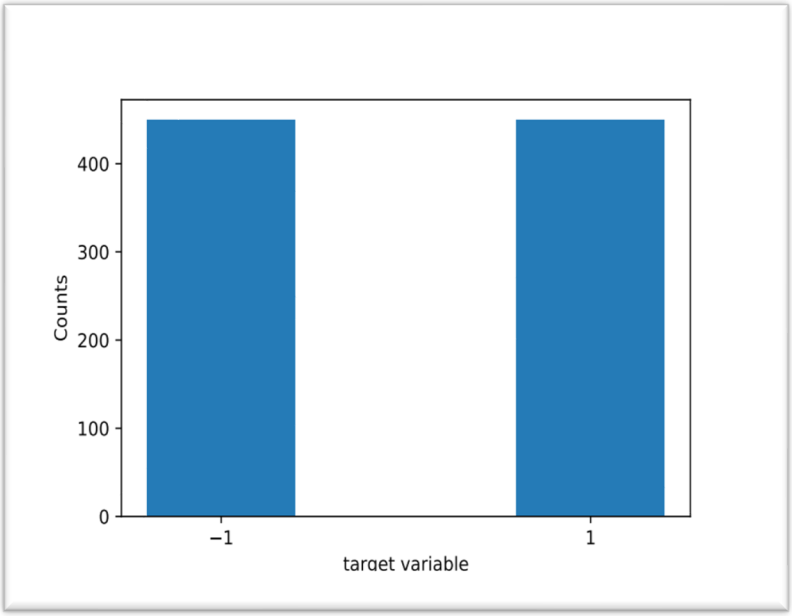


Target Value Insights

Target value counts for each category.



Let us also see the bar plot for this.



Observation Values

Top 10 rows of the data frame for the dataset is shown below

Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
87524	442.2460114	253.291155	0.819738392	90546	0.758650579	1184.04	Kecimen
75166	406.690687	243.0324363	0.801805234	78789	0.68412957	1121.786	Kecimen
90856	442.2670483	266.3283177	0.798353619	93717	0.637612812	1208.575	Kecimen
45928	286.5405586	208.7600423	0.684989217	47336	0.699599385	844.162	Kecimen
79408	352.1907699	290.8275329	0.56401133	81463	0.792771926	1073.251	Kecimen
49242	318.125407	200.12212	0.777351277	51368	0.658456354	881.836	Kecimen
42492	310.1460715	176.1314494	0.823098681	43904	0.665893562	823.796	Kecimen
60952	332.4554716	235.429835	0.706057518	62329	0.74359819	933.366	Kecimen
42256	323.1896072	172.5759261	0.845498789	44743	0.698030924	849.728	Kecimen
64380	366.9648423	227.7716147	0.784055626	66125	0.664375716	981.544	Kecimen

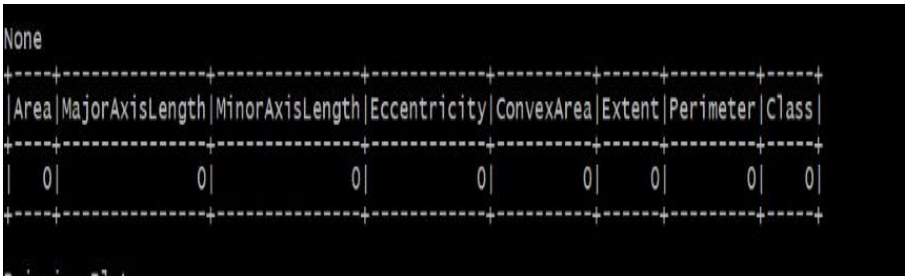
only showing top 10 rows

3. DATA CLEANING

Data cleaning involves the removal of null values or replacing null values with the mean/median/mode of the column. This is an important step as the existence of null values tamper with the accuracy of the model.

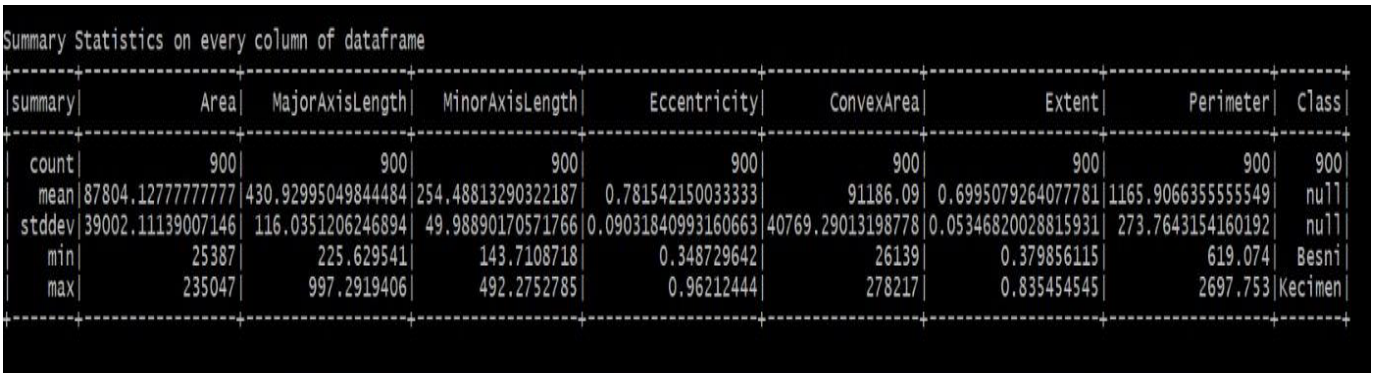
Null Values for each column -

There are no null values in the dataset as seen below.



Summary statistics of each column -

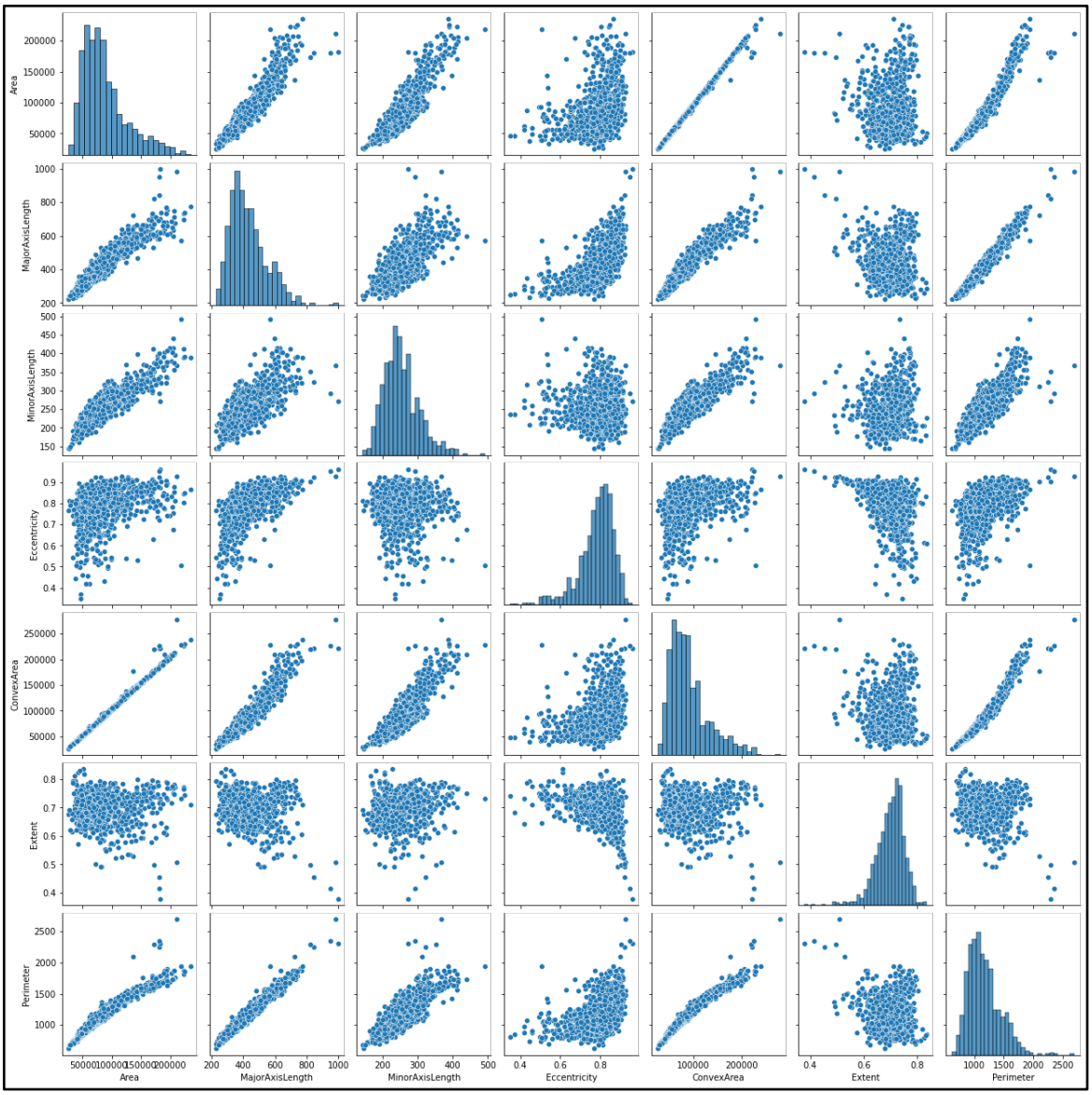
We have calculated summary statistics for each column, which tells the count, mean, standard deviation, minimum and maximum of each feature.



PairWise Plot -

A pairs plot shows the distribution of single variables as well as the relationships between them. Pair plots are an excellent way to spot tendencies for further investigation.

This function creates a grid of Axes by default, with each numeric variable in data shared over the y-axes in a single row and the x-axes in a single column. A univariate distribution plot is constructed to show the marginal distribution of the data in each column for the diagonal plots.



In the above graph, it is the matrix of scatterplots that lets us understand the pairwise relationship between different variables in the given dataset. The variables of the dataset are shown along with the diagonal boxes as plots for each. For example, the box in the top right corner of the matrix displays a scatterplot of values for area and perimeter.

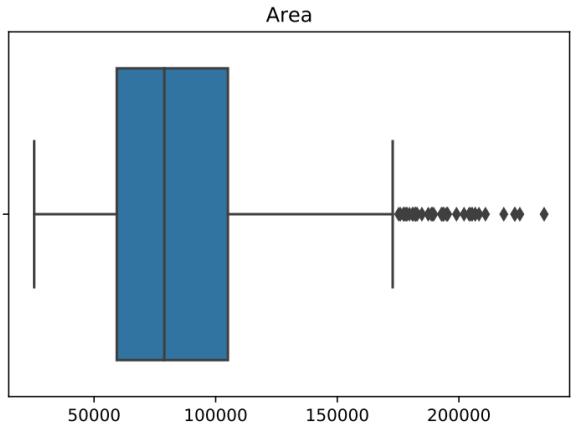
4. EXPLORATORY DATA ANALYSIS

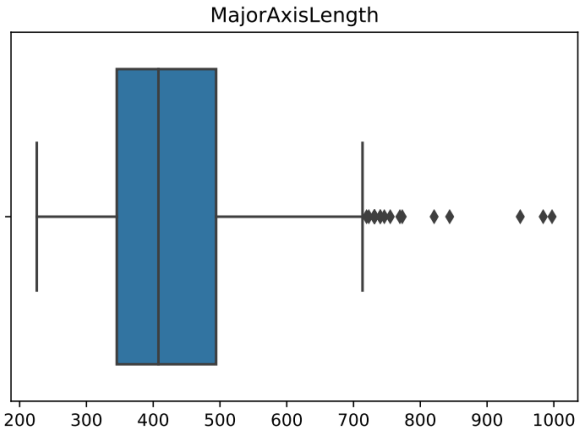
Box Plot:

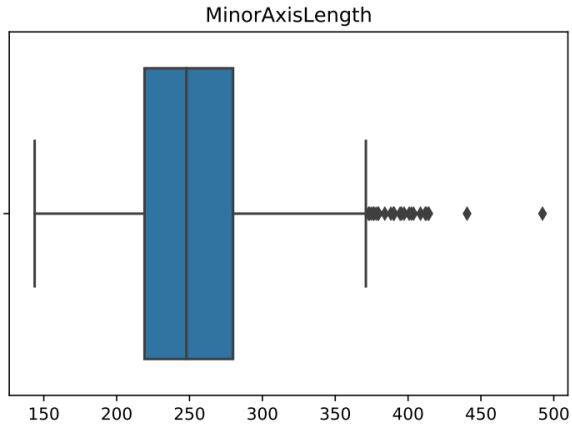
A boxplot is a standardized method of depicting data distributions using a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It gives

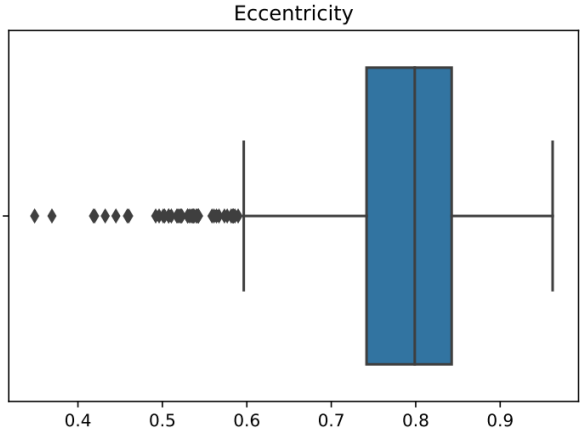
information about the outliers and their values. It can also determine whether the data is symmetrical, how tightly the data is packed, and whether the data is skewed.

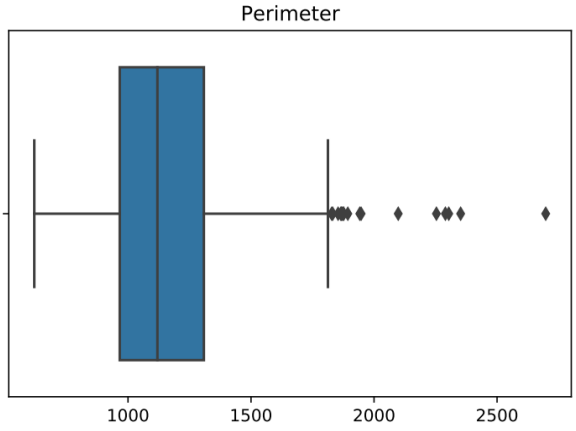
Let us see some of the box plots for the numeric features:

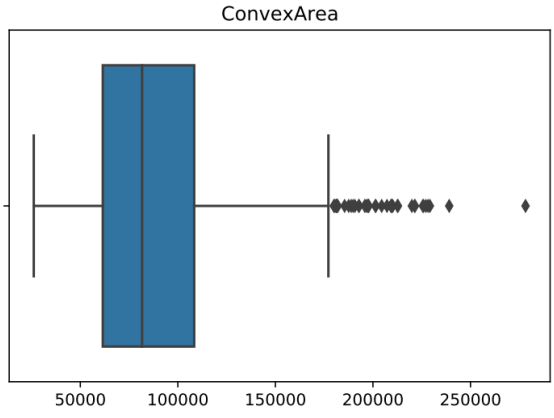


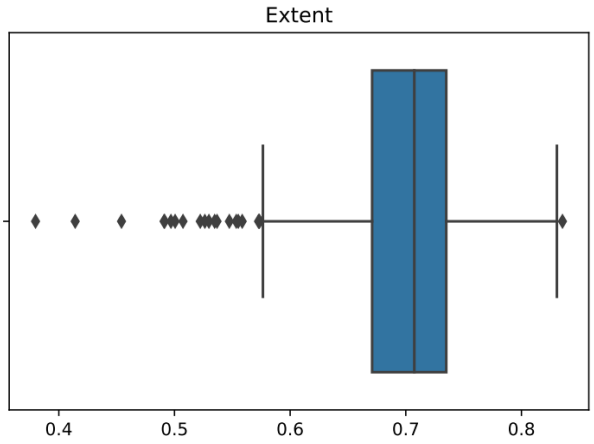








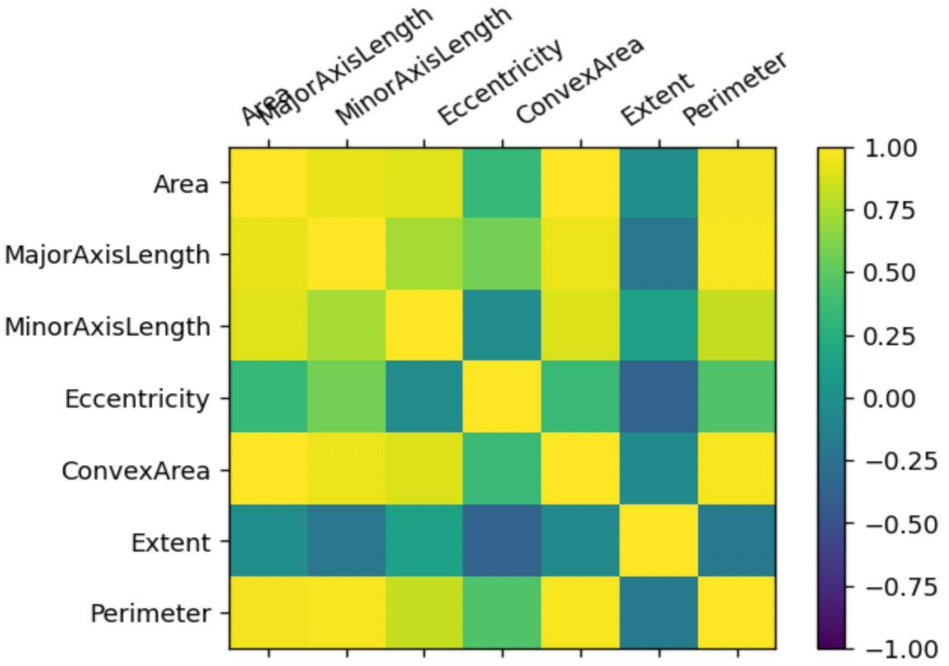




In the boxplots above, the presence of outliers can be detected. These outliers can be treated by replacing the data values with the mean/median or mode of the data corresponding to each column.

Correlation Matrix

A correlation matrix is essentially a table that shows the relationship between two variables. It works well with variables that have a linear relationship with one another. The correlation between all possible pairings of values in a table is represented by the matrix.

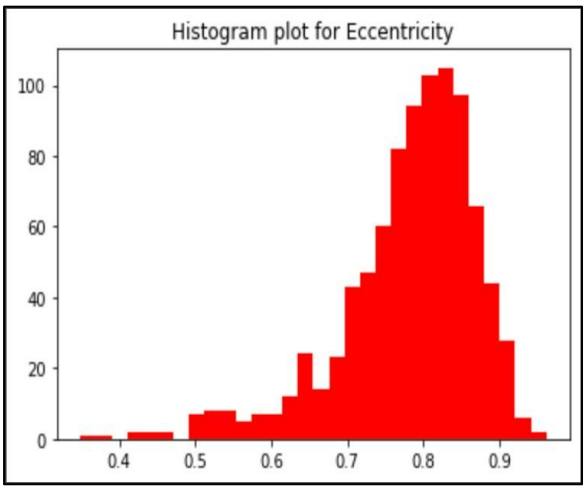


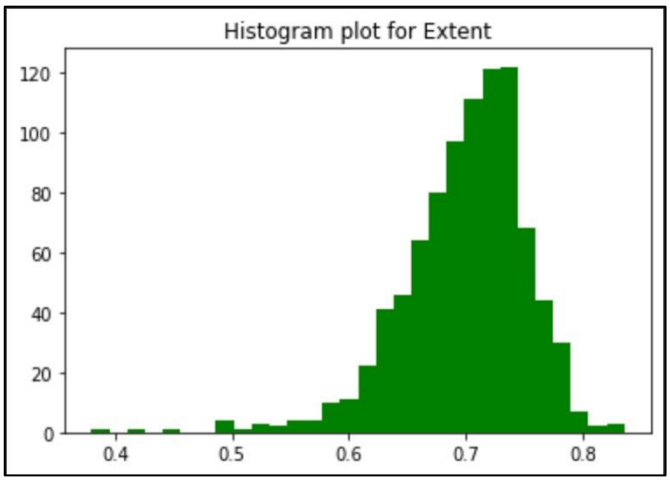
And in the above graph we see that yellow color shows the highest positive correlation value while dark green shows the least correlation value and dark blue shoes highest negative correlation. From the correlation matrix, setting a threshold of 0.85, we are deleting the columns which all have a high correlation value. In the end, only the columns ‘Area, Eccentricity, and Extent’ are used for further calculations.

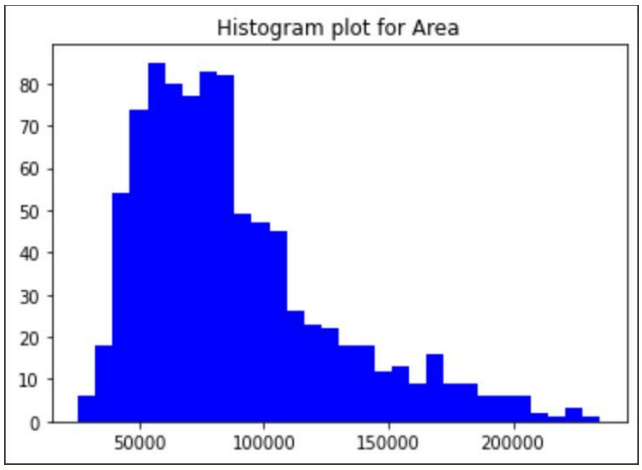
Histogram

A histogram is a graphical representation that divides a set of data points into ranges defined by the user. The histogram, which resembles a bar graph in appearance, condenses a data series into an easily interpreted visual by grouping many data points into logical ranges.

As a part of the exploratory data analytics, we plot the histogram for the finally selected columns for our predictions.







5. MODEL BUILDING AND TRAINING:

Test Train Split:

Before building the model, we must split the data into train and test data so that we can validate the model and compute the accuracy. To perform this, we use the below code.

```
#splitting data into train and test sets
train_df, test_df = df.randomSplit([.7,.3])

label = 'Class'
numerical_cols = ['Area', 'Eccentricity','Extent']

# Indexer for classification label:
label_indexer = StringIndexer(inputCol=label, outputCol=label+'_indexed')

#assemble all features as vector to be used as input for Spark MLLib
assembler = VectorAssembler(inputCols= numerical_cols, outputCol='features')

# Creating data processing pipeline
pipeline = Pipeline(stages= [label_indexer, assembler])
```

After splitting the data into a training dataset and a test dataset, an assembler is used to aggregate all the input columns as a vector which is used as the input for the training model. In this project, 3 different models are used to find the accuracy of classifying the type of raisin [6]. They are:

I. Logistic Regression:

The method of modeling the probability of a discrete result given an input variable is known as logistic regression. By estimating probabilities using a logistic regression equation, it is employed in statistical software to comprehend the relationship between the dependent variable and one or more independent variables. This form of analysis can assist in predicting the chances of an occurrence or a decision occurring.

II. Random Forest:

Random forests, also known as random choice forests, are an ensemble learning method for classification, regression, and other tasks that works by building many decision trees during training. For classification tasks, the random forest's output is the class chosen by most trees.

In this project, we are only working with a subset of features in this model, hence it is faster to train than other regression models. We can easily work with hundreds of features.]

III. Naïve Bayes:

Naive Bayes uses a similar approach to forecast the likelihood of various classes based on various attributes. This approach is typically used for text classification and problems with several classes. The Bayes Theorem is used to create a Naive Bayes classifier. It calculates membership probabilities for each class, such as the likelihood that a certain record or data point belongs to that class. The most likely class is defined as the one having the highest probability.

Using all these models we are classifying the raisins into ‘kecimen’ and ‘besni’ with utmost accuracy with each model. Below shows the code to create a pipeline and build the model classify the raisins.

```
lr = LogisticRegression(featuresCol='features', labelCol=label+'_indexed')
rfc = RandomForestClassifier(featuresCol='features', labelCol=label+'_indexed', numTrees=100)
nb = NaiveBayes(featuresCol='features', labelCol=label+'_indexed')

# creating pipelines with machine learning models
pipeline_lr = Pipeline(stages=[pipeline, lr])
pipeline_rfc = Pipeline(stages=[pipeline, rfc])
pipeline_nb = Pipeline(stages=[pipeline, nb])

#fitting models with train subset
lr_fit = pipeline_lr.fit(train_df)
rfc_fit = pipeline_rfc.fit(train_df)
nb_fit = pipeline_nb.fit(train_df)

# predictions for test subset
pred_lr = lr_fit.transform(test_df)
pred_rfc = rfc_fit.transform(test_df)
pred_nb = nb_fit.transform(test_df)
```

6. EVALUATION

After the model has been built with both the training dataset and testing dataset, we have an accuracy of the range from 80% to 85%. The area under the ROC curve for logistic regression, random forest, and Naive Bayes is 0.8494, 0.8494, and 0.8118 respectively. For the logistic regression and random forest, we get an accuracy of 84.94% and an AUC of 0.85, and for Naive Bayes, we get an accuracy of 80.33% and 0.81 AUC. Below are the results of the prediction.

```
The Area Under the ROC Curve for Logistic Regression, Random Forest
0.8489414255469302 0.8489414255469302 0.8118207480592802
Logistic Regression AUC:  0.85
Random Forest AUC:  0.85
Naive Bayes AUC:  0.81
Logistic Regression accuracy: 84.94%
Random Forest accuracy: 84.94%
Naive Bayes accuracy: 80.33%
>>>
```

Comparing results with scikit

The same has been implemented in scikit learn also; each and every step explained above can be replicable through sci-kit – learn package from python in Jupyter notebook. The implementation can be found in raisin_classification_scikit.py. The below image is the small snippet doing the transformation and performing the cross-validation random forest classifier.

The accuracy for the random forest classifier is 83.83%.

```
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(X_train, Y_train)

# performing predictions on the test dataset
y_pred = clf.predict(X_test)

# metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("Accuracy of the Random forest classifier model: ", metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy of the Random forest classifier model:  0.8383838383838383
```

CONCLUSION

In this project, with the help of PySpark on AWS EMR, various ML models such as Logistic Regression, Naive Bayes, Random Forest have been implemented. It is observed that “Random Forest” and “Logistic Regression” were giving best accuracy among the models that were applied. Both the models had accuracy of 85%

To do a comparative study, the ML models have been implemented using Scikit-learn as well. We observed that Random Forest of PySpark was giving better accuracy of 85% as compared to accuracy (84%) of Random Forest implemented using Scikit-learn.

Thus, we can conclude that the PySpark when implemented, was able to learn more features and improve the accuracy of the model when compared with scikit learn. For larger datasets, there will be a drastic change in the accuracies with PySpark having an upper hand for the same ML model.

REFERENCES

[1] https://archive.ics.uci.edu/ml/datasets/Raisin+Dataset (Dataset)

[2] https://spark.apache.org/docs/latest/ml-classification-regression.html

[3] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8258338

[4] https://www.mdpi.com/2079-9292/9/3/444

[5] 3.2.4.1.5. sklearn.linear_model.LogisticRegressionCV — scikit-learn 0.23.2 documentation (scikit-learn.org) – logistic regression cv scikit learn

[6] https://www.tgaspar.com/subpages/heart-disease-classification-pyspark