Big Data & Machine Learning

**Credit-Score Class Prediction**

horizontal line

# 

## 

## 

## 

## 

## 

## 

## 

## Introduction

Machine-learning-based credit scoring is the most relevant and promising one in today’s world. Simply put, credit scoring is an evaluation of how well the bank’s customers can pay and are willing to pay off their debt. Only half of the 3 billion people who use banks regularly are eligible for lending, so this decision must be made carefully. Machine-learning-based credit scoring resolutions are based on a lot of data, such as total pay, credit history, transaction reports, work experience, and even Google Analytics. In essence, scoring shows a mathematical model established by mathematical methods and accounting for a large number of facts. As a result, credit scoring utilizing machine intelligence specifies more sensitive, distinguished credit score appraisals based on an array of additional actual-occasion determinants, giving an approach to finance to a wider audience with profit potential. Therefore, the goal of this project is to develop an intelligent system that will classify people according to their credit scores in order to reduce manual labor using data from the bank about credit scores.

## Table of Contents

[Introduction](#_arolcxe0i15c) 1

[Table of Contents](#_x9ooeogj8ti3) 2

[About Dataset](#_la5jp5tnimjw) 2

[Loading the data](#_dnswp4fwxyyn) 4

[Infer the schema of data](#_c0k7pf9qimr5) 4

[Data Preprocessing](#_sz0rr9spmqi3) 5

[Removing column with unique values](#_mox10ryu0ia0) 5

[Finding unclean data using a function](#_ny8xtbqvmx0t) 5

[Exploratory Data Analysis(EDA)](#_ce6usdnfni59) 8

[Outlier Detection and Handling Outliers](#_ljnynpob7689) 13

[Tableau Dashboard](#_ant8rn55z4e1) 14

[Feature Engineering and advanced preprocessing](#_11p2q5lz380x) 15

[Model training and evaluation](#_li490mi39vuw) 17

[Handling the problem of data imbalance](#_8ljkzuqogtn9) 20

[Conclusion](#_xd9sgay70l77) 22

[Appendix](#_f5omsns1539w) 23

## About Dataset

This dataset was obtained from a global finance company. Over the years, the company has collected basic bank details and gathered a lot of credit-related information. The management wants to build an intelligent system to segregate people into credit score brackets to reduce manual efforts.

This dataset contains 100,000 rows and 27 columns. The attributes of the dataset are shown below:

* **ID**: Represents a unique identification of an entry
* **Customer\_ID**: Represents a unique identification of a person
* **Name**: Represents the name of a person
* **Month**: Represents the month of the year
* **Age**: Represents the age of the person
* **SSN**: Represents the social security number of a person
* **Occupation**: Represents the occupation of the person
* **Annual\_Income**: Represents the annual income of the person
* **Monthly\_Base\_Salary**: Represents the monthly base salary of a person
* **Num\_Bank\_Accounts**: Represents the number of bank accounts a person holds
* **Num\_Credit\_Card**: Represents the number of other credit cards held by a person
* **Interest\_Rate**: Represents the interest rate on credit card
* **Num\_of\_Loan**: Represents the number of loans taken from the bank
* **Type\_of\_Loan**: Represents the types of loan taken by a person
* **Delay\_from\_due\_date**: Represents the average number of days delayed from the payment date
* **Num\_of\_delayed\_Payment**: Represents the average number of payments delayed by a person
* **Changed\_Credit\_Limit**: Represents the percentage change in credit card limit
* **Num\_Credit\_Inquiries**: Represents the number of credit card inquiries
* **Credit\_Mix**: Represents the classification of the mix of credits
* **Outstanding\_Debt**: Represents the remaining debt to be paid (in USD)
* **Credit\_Utilization\_Ratio**: Represents the utilization ratio of credit card
* **Credit\_History\_Age**: Represents the age of credit history of the person
* **Payment\_of\_Min\_Amount**: Represents whether only the minimum amount was paid by the person
* **Total\_EMI\_per\_month**: Represents the monthly EMI payments (in USD)
* **Amount\_invested\_monthly**: Represents the monthly amount invested by the customer (in USD)
* **Payment\_Behaviour**: Represents the payment behavior of the customer (in USD)
* **Monthly\_Balance**: Represents the monthly balance amount of the customer (in USD)
* **Credit\_Score**: Represents the bracket of credit score (Poor, Standard, Good)

## Loading the data

When dealing with big data, Spark RDD and SQL modules are frequently used. In addition, Spark SQL is a framework for analyzing structured data, and in contrast to Spark RDD, its APIs provide additional insight concerning the implementation and processes. Even though the same processing engine is used to do operations, this is not the case ("Spark SQL & Data Frames, Spark 3.1.2 Documentation," 2021). The Spark SQL module along with pandas is used for most of the analysis in this work, and Tableau, Matplotlib, Seaborn is used to making visualizations and Scikit learn framework is loaded to create Machine learning classification models.

**from** **pyspark.context** **import** SparkContext

**from** **pyspark.sql.session** **import** SparkSession

sc = SparkContext.getOrCreate()

spark = SparkSession(sc)

train\_path="/content/drive/MyDrive/Credit score prediction/train.csv"

test\_path="/content/drive/MyDrive/Credit score prediction/test.csv"

# lOADING THE DATASET

infer\_schema = "true"

first\_row\_is\_header = "true"

delimiter = ","

df = spark.read.load(train\_path,format='csv',header='true', inferSchema='true')

type(df)

### Infer the schema of data

df.printSchema()

This shows the schema of the data. It shows the datatype of each column and also ensures the presence of null values in any column. This would appear similarly as,

root

|-- ID: string (nullable = true)

This dataframe can also be converted into a pandas dataframe using toPandas()

function.

This dataframe has about **100000** rows and **27** columns including the target column **Credit\_Score**.

## Data Preprocessing

Here, we would be starting with the first order of data preprocessing or some of the basic data preprocessing steps that can be done before moving into EDA.

### Removing column with unique values

The **ID** column in the dataset has 50,000 unique values and this should be removed from the dataset as it would make the model overfit on the data by training the model on each unique value. So this column is dropped from the dataframe using the **drop()** function.

### Finding unclean data using a function

**def** **unclean\_data\_finding**(df):

dirty = []

columns = df.columns

**for** col **in** columns:

dtype = df[col].dtypes

nunique = df[col].nunique()

null = df[col].isnull().sum()

duplicates = df[col].duplicated().sum()

dirty.append([col,dtype,nunique,null,duplicates])

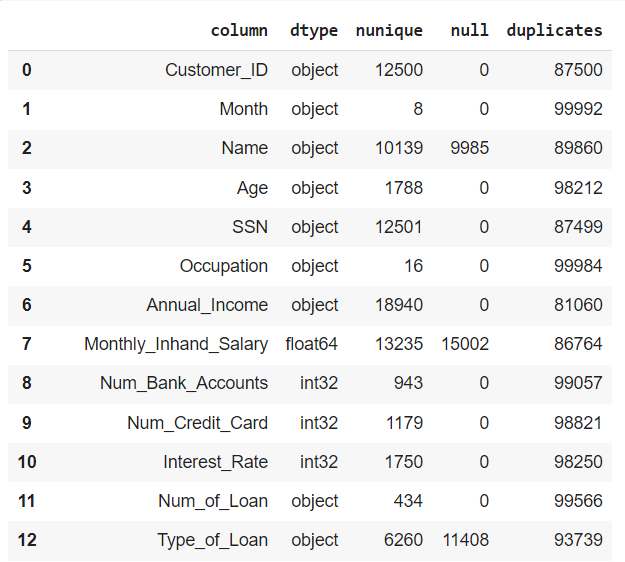
df\_dirty\_data\_finding = pd.DataFrame(dirty)

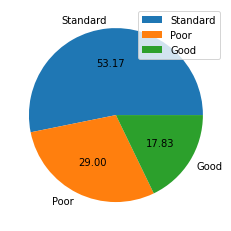
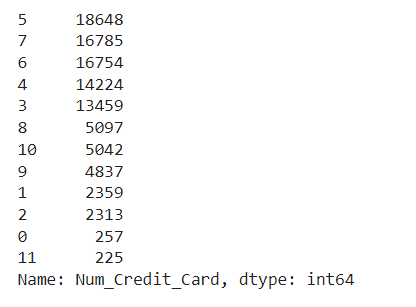
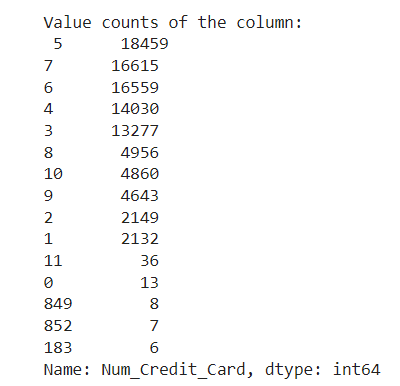
df\_dirty\_data\_finding.columns = ['column','dtype','nunique','null','duplicates']

**return** df\_dirty\_data\_finding

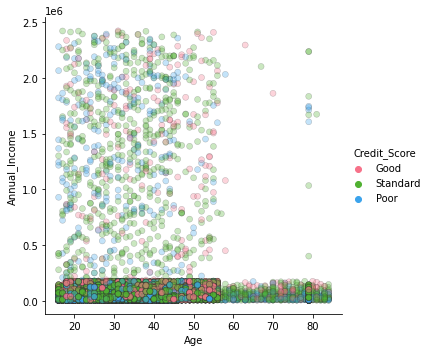
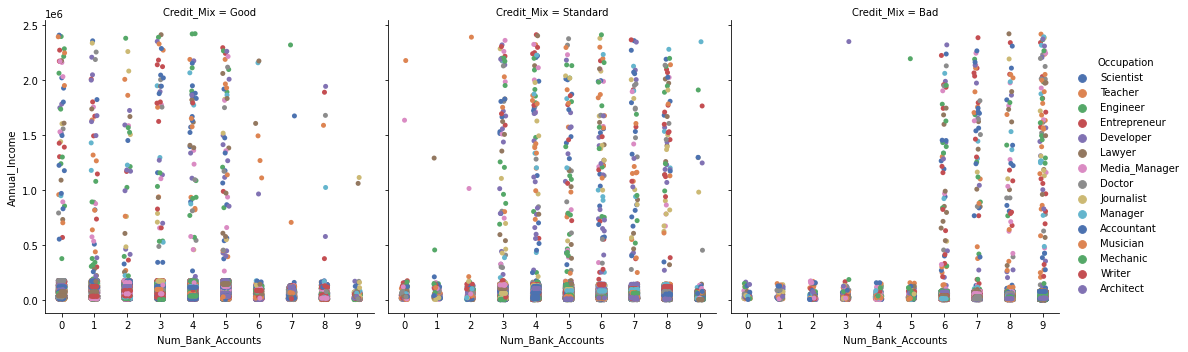
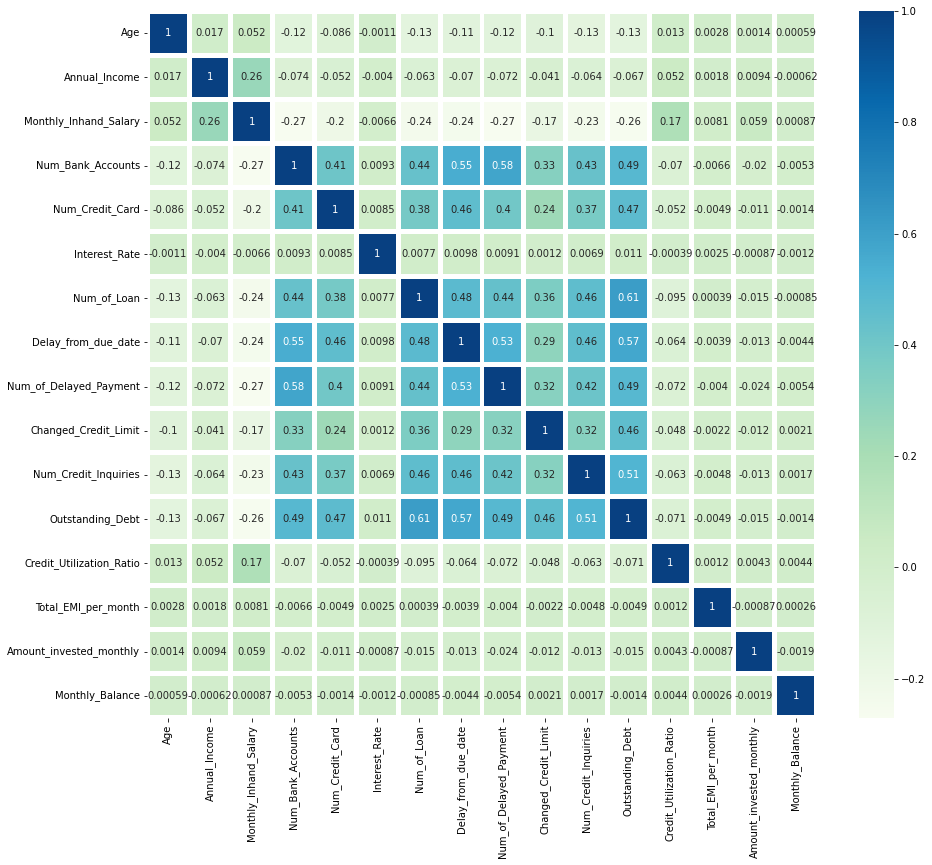
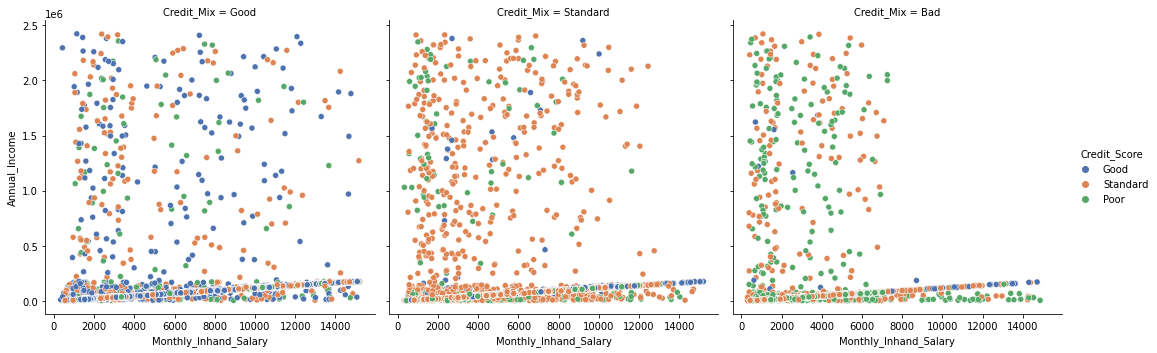
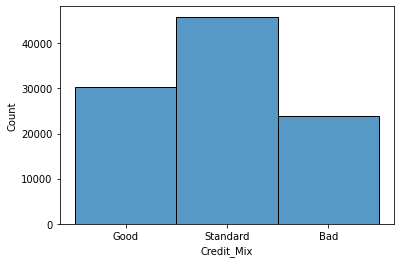
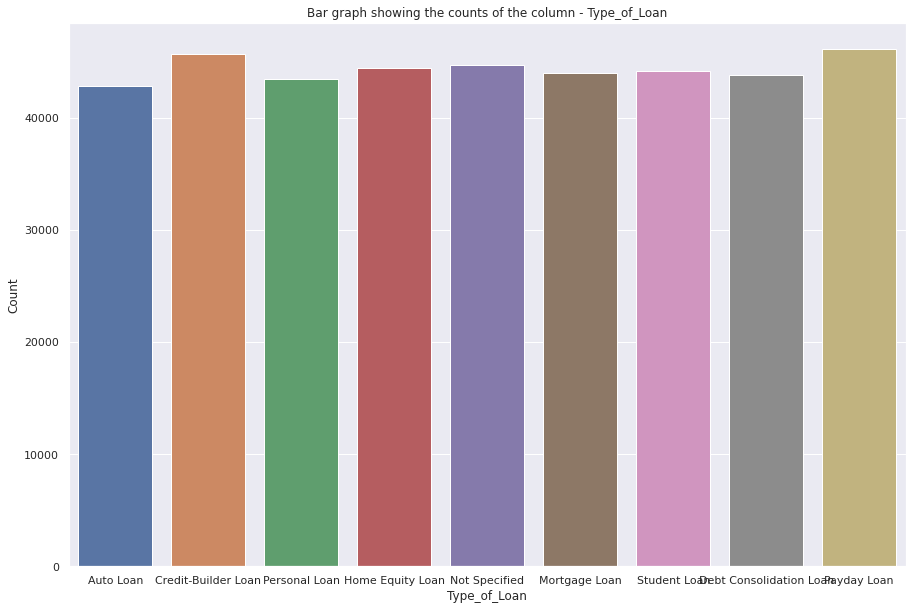
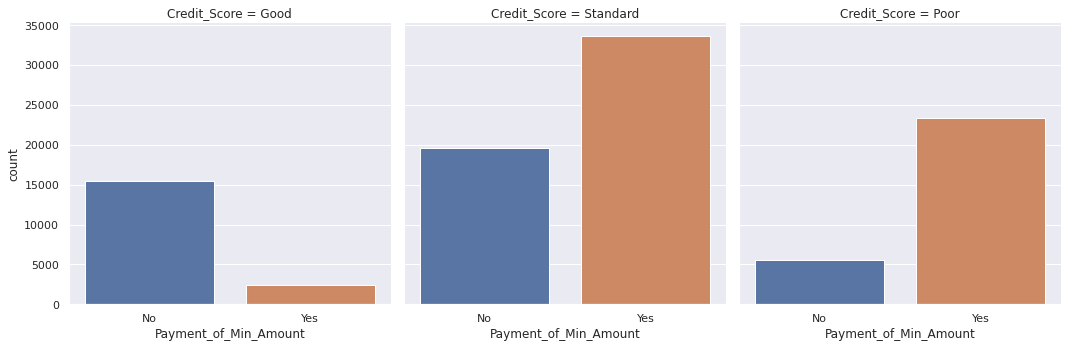
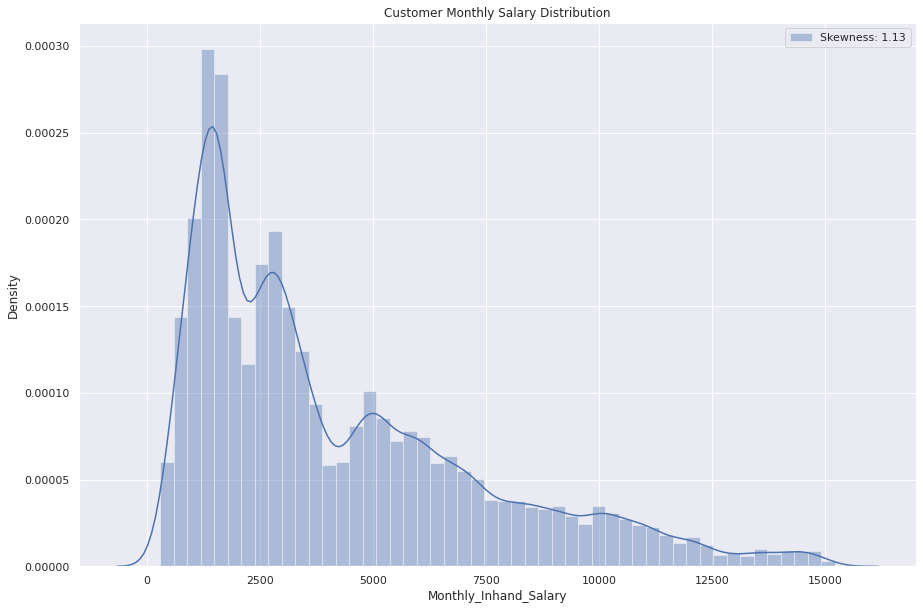
unclean\_data\_finding(data\_train)

This function returns a dataframe showing column name, datatype, unique values, number of null values and number of duplicates. A partial section of it can be seen in the below image and the whole dataframe can be seen in the code notebook.

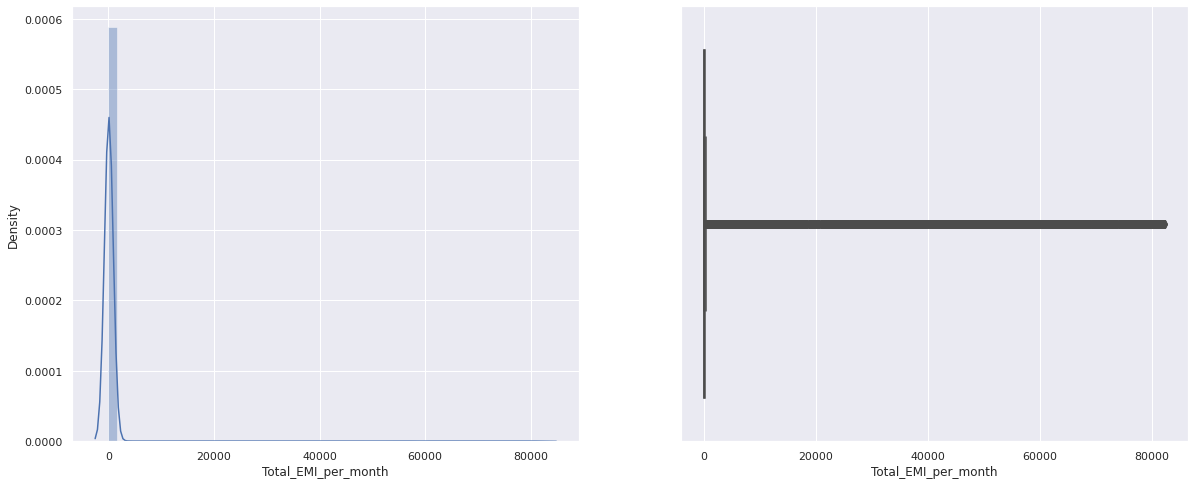


* Also, when we try to plot the piechart to observe the count of values for each category in the target column, we can observe that the data is highly imbalanced which can be visualized using the below plot. So we need to keep this as an important pointer to handle the imbalance in data while training the models.  
  
* We can also observe that in some columns the null values are represented using '', 'nan', '!@9#%8', '#F%$D@\*&8' so these values need to be replaced using NaN values from numpy such that these can be imputed while imputing null values. This can be done using the below code  
  data\_train = data\_train.applymap(**lambda** x: x **if** x **is** np.NaN **or** **not** isinstance(x, str) **else** str(x).strip('\_ ,"')).replace(['', 'nan', '!@9#%8', '#F%$D@\*&8'], np.NaN)
* We can also observe that there are some datatypes which are incorrectly provided, for example age is given as string but it should be a numeric value, similarly there are other columns like annual income, number of loans, number of delayed payments, changed credit card limit, outstanding debt, amount invested monthly and monthly balance which are also given as string objects but they need to be converted either into int or float.
* As we have converted the special characters into NaN, now we need to impute these null values, Imputation of null values can be done in different ways like mean value imputation or median for numeric and mode for categorical, and here we use a different method that is **ffill,** this fills the null values with the values in the previous row. Thus we fill the null values using this method.
* Now we try to inspect other categorical columns to check the number of occurrences of each category in a column so that we can try to replace the least occurring categories of each column. For example, In the Num Credit Card column, we can observe that there are some categories which have occurred least number of times, so these are to be replaced with some most occurring elements. Similarly, this is done for other columns like Number of Loans, Number of delayed payments, number of people who made credit enquiries etc.  
  
* Thus we have successfully performed the first stage of preprocessing by handling null values, and removing unrequired columns at the first instance and by preprocessing some of the columns such that they contain suitable values.

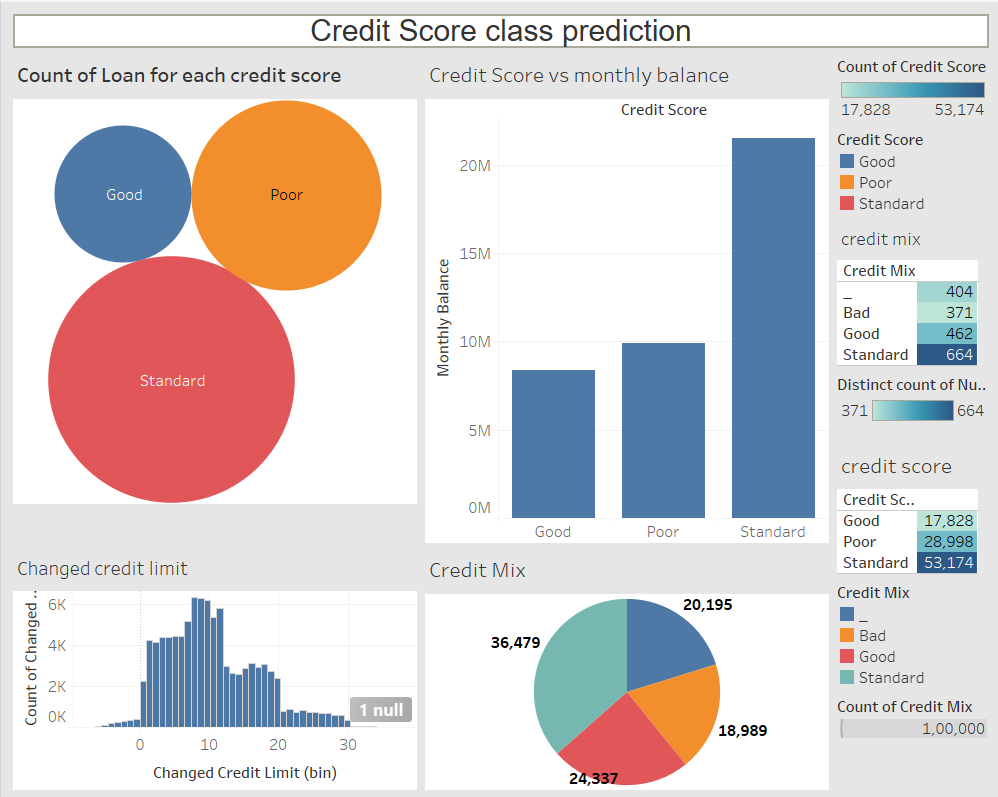
## Exploratory Data Analysis(EDA)

* The first plot that we would examine is a **relplot** that shows a figure-level interface for relational plots, This helps us in getting access to several axes-level functions. So the first plot is done between the Age and Annual\_Income columns and credit score is set as hue.  
  
  + Code can be observed as:  
    sns.relplot(x = 'Age', y = 'Annual\_Income' ,hue = 'Credit\_Score', data = data\_train,alpha=.**3**, edgecolor=".2", linewidth=.**5**,palette = 'husl')
  + We can observe that mostly the credit score can be seen for the age <60 and the credit score is well maintained for the age>30 and age<60 and the credit score for some people seems not to be well maintained especially for the younger ones.
* The second plot that we would observe is a catplot that even gives us access to multiple axes. This plot is done between Number of bank accounts, and annual income and occupation is used as a hue with the column values representing credit\_mix.  
    
  
  + Code can be observed as:  
    sns.catplot(data=data\_train, x="Num\_Bank\_Accounts", y="Annual\_Income", hue="Occupation",col = 'Credit\_Mix', palette="deep")
  + We can also observe that the credit\_mix value is especially good for people having num of bank accounts<=5 and the credit\_mix value is bad for people having num of bank accounts>5
* We have also tried to take some good observations from the correlation plot.
  + We can observe that the outstanding debt and the number of loans taken are highly correlated and this would surely be as the people taking many loans would have larger debts as compared to people who took less loans
  + We can also observe that the number of delayed payments is highly correlated with the number of bank accounts that a person has.
  + The number of credit cards used by people are also strongly correlated to outstanding debt.
  + The increase in the number of crest cards directly implies the increase in different bank accounts.
  + We can also observe that there is a strong positive correlation between the people having high outstanding debt and the people having their loan payments delayed so usually people taking on more debts delay the payment and this leads to decrease in the credit\_score.
  + Thus we can finally observe that there are totally 8 different features which are mutually dependent on each other.  
    
* Also when we try to plot a relplot between credit mix and monthly inhand salary, we can observe that people with less monthly inhand salary have poor credit score.  
  
* When we just try to plot a univariate plot on the credit\_mix column, we can also observe that the credit\_mix values for standard and good are high as compared to bad.  
  
* When we try to check for the the most taken loans we observe that the payday loans and the credit builder loans are the most taken  
  
* When we also try to plot a factor plot for the minimum\_payment done vs credit score, we observe that the people with good credit score have not paid the minimum amount whereas people with poor credit score have paid the minimum amount.
  + Code for factor plot:  
    sns.factorplot('Payment\_of\_Min\_Amount', col = 'Credit\_Score', data = data\_train, kind = 'count', col\_wrap = **3**)
* We also observed from a distplot that the monthly inhand salary feature is right skewed and also the age column is highly skewed. So log transformation and power transformation  
  

### Outlier Detection and Handling Outliers

* We observed the presence of outliers in two columns : Total EMI per month and Annual income
  + Here we tried to use the InterQuartile Range to get the upper limit and lower limit and then remove the outliers using these limits.  
      
    

## Tableau Dashboard



From this report we can observe:

* The count of each type of loan taken by people by taking their credit score class into consideration. People with standard credit scores have many loans as compared to people with poor or good credit scores.
* The plots in the dashboard also show the count of credit score and credit\_mix features, and the count of credit\_score directly helps us in understanding that the data has a problem of class imbalance.
* We can also observe that the number of delayed payments are even high for people with a standard credit mix and then it is followed by people with a bad credit score mix.

## Feature Engineering and advanced preprocessing

* Now we remove the next unwanted columns that we observed from EDA. **customer\_ID**, **Name** and **SSN** columns are removed from the dataframe.
* Ordinal encoding is used to encode the categorical columns such that they can be represented in numerical form and these encoded columns can be given as input to the model.
  + Code for ordinal encoding:  
    **from** **sklearn.preprocessing** **import** OrdinalEncoder

ordinal\_encoder = OrdinalEncoder()

**for** column **in** data\_train.columns:

**if** data\_train[column].dtypes == 'object':

data\_train[column] = ordinal\_encoder.fit\_transform(data\_train[[column]])

* Now we split the data into X and y, where X stores the independent features and y stores the dependent features then we obtain the important features using the mutual\_info\_regression. This method works completely depending on the entropy of the features in the data. Here it uses mutual information(MI), which is a non-negative value measuring the dependency between two variables. The Higher value of MI denotes higher dependency and it is zero for independent variables.

Here we write a function that calculates the mutual information scores and we further plot these scores to check for the best features.

Function to calculate MI:  
**def** **make\_mi\_scores**(X, y, discrete\_features):

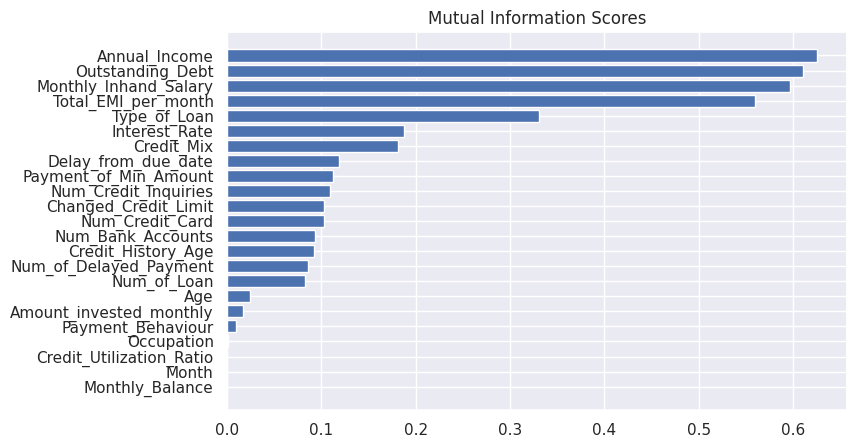
mi\_scores = mutual\_info\_regression(X, y, discrete\_features=discrete\_features)

mi\_scores = pd.Series(mi\_scores, name="MI Scores", index=X.columns)

mi\_scores = mi\_scores.sort\_values(ascending=False)

**return** mi\_scores

mi\_scores = make\_mi\_scores(X, y, discrete\_features)



* Now we remove all the unnecessary columns which have less feature importance like **'Month', 'Monthly\_Balance', 'Occupation', 'Credit\_Utilization\_Ratio', 'Payment\_Behaviour'**
* We then normalize the features using mean and std values obtained from each column. For example, the Annual income column is normalized using the below code and this is similarly done for other columns.  
  data\_train.Annual\_Income = (data\_train.Annual\_Income - data\_train.Annual\_Income.mean())/data\_train.Annual\_Income.std()
* So now the feature engineering and preprocessing of data is done and the pandas dataframe is again converted into spark dataframe using **spark.createDataframe()**
* Now a feature transformer like Vector Assembler is used to merge all the columns into a single vector so that a spark ML model can be trained on the data.

Code for vector assembler:  
**from** **pyspark.ml.linalg** **import** Vectors

**from** **pyspark.ml.feature** **import** VectorAssembler

assembler=VectorAssembler(inputCols=[

'Age',

'Annual\_Income',

'Monthly\_Inhand\_Salary',

'Num\_Bank\_Accounts',

'Num\_Credit\_Card',

'Interest\_Rate',

'Num\_of\_Loan',

'Type\_of\_Loan',

'Delay\_from\_due\_date',

'Num\_of\_Delayed\_Payment',

'Changed\_Credit\_Limit',

'Num\_Credit\_Inquiries',

'Credit\_Mix',

'Outstanding\_Debt',

'Credit\_History\_Age',

'Payment\_of\_Min\_Amount',

'Total\_EMI\_per\_month',

'Amount\_invested\_monthly'],outputCol="features")

Now the data is split into train and test.  
train\_data,test\_data=final\_data.randomSplit([**0.8**,**0.2**])

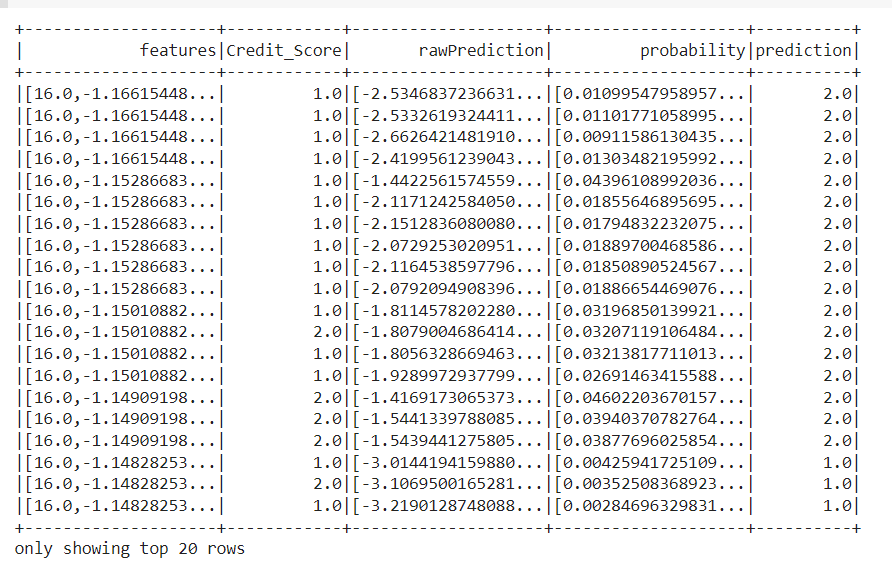
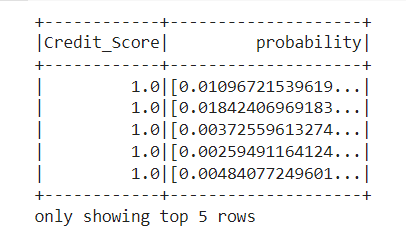
The schema of the spark dataframe train\_data can be observed as:  
root

|-- features: vector (nullable = true)

|-- Credit\_Score: double (nullable = true)

## Model training and evaluation

We have used several machine learning models on train\_data and evaluated it on the test data.

* The first model that we used is logistic regression, we have built the model on the data obtained from the vector Assembler. The code used to train the logistic regression model can be seen below:  
  lr = LogisticRegression(featuresCol = 'features', labelCol = 'Credit\_Score', maxIter=**30**)  
  lrModel = lr.fit(train\_data)
* log summary is obtained from this trained model and the predictions obtained from this can be seen as:  
  
* Now this model is transformed over the test data and the credit score and probability on test data is obtained.
  + code   
    predictions = lrModel.transform(test\_data)  
    predictions.select('Credit\_Score','probability').show(**5**)
* Also the model is evaluated using several other metrics and the output can be seen as:  
  Accuracy: **0.6366033464023414**

FPR: **0.296749878034027**

TPR: **0.6366033464023414**

F-measure: **0.6300590542933062**

Precision: **0.634931641879445**

Recall: **0.6366033464023414**

Similarly, the random forest model is trained on the data and the output is obtained and it is observed that random forest has given better performance over logistic regression.

* As the random forest has performed well, so hyperparameter tuning is performed using prambuilder() that builds the required parameters for tuning and 5- fold cross validation is used to assure the results obtained.
  + Code for hyperparameter tuning can be seen as:  
    **from** **pyspark.ml.tuning** **import** ParamGridBuilder, CrossValidator

paramGrid = (ParamGridBuilder()

#.addGrid(rf.maxDepth, [2, 5, 10, 20, 30])

.addGrid(rf.maxDepth, [**2**, **5**, **10**])

#.addGrid(rf.maxBins, [10, 20, 40, 80, 100])

.addGrid(rf.maxBins, [**5**, **10**, **20**])

#.addGrid(rf.numTrees, [5, 20, 50, 100, 500])

.addGrid(rf.numTrees, [**5**, **20**, **50**])

.build())

cv = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=**5**)

# Running cross validations. This can take some time since it is training over 20 trees!

cvModel = cv.fit(train\_data)

predictions = cvModel.transform(test\_data)

evaluator.evaluate(predictions)

* + Now the best parameters for the random forest model are obtained and these are used to evaluate the model as shown below:  
    train\_Summ = cvModel.bestModel.summary

accuracy\_acc = train\_Summ.accuracy

falsePositiveRate\_fpr = train\_Summ.weightedFalsePositiveRate

truePositiveRate\_tpr = train\_Summ.weightedTruePositiveRate

fMeasure\_fm = train\_Summ.weightedFMeasure()

precision\_pr = train\_Summ.weightedPrecision

recall\_re = train\_Summ.weightedRecall

**print**("Accuracy: %s**\n**FPR: %s**\n**TPR: %s**\n**F-measure: %s**\n**Precision: %s**\n**Recall: %s"

% (accuracy\_acc, falsePositiveRate\_fpr, truePositiveRate\_tpr, fMeasure\_fm, precision\_pr, recall\_re))

* Now we observe that the performance has increased for the random forest model, but it is important to observe how are we evaluating it, as this is an imbalanced data, accuracy cannot be a deciding factor and it is not the best practice to get into a conclusion by observing a model trained on imbalanced data.

### Handling the problem of data imbalance

Data imbalance is a major issue in today’s world but the harsh truth is that, most of the data that we have and the data we obtain is imbalanced in the real world, for example credit card frauds, data anomalies, cancer detection etc are also relevant to imbalanced data.

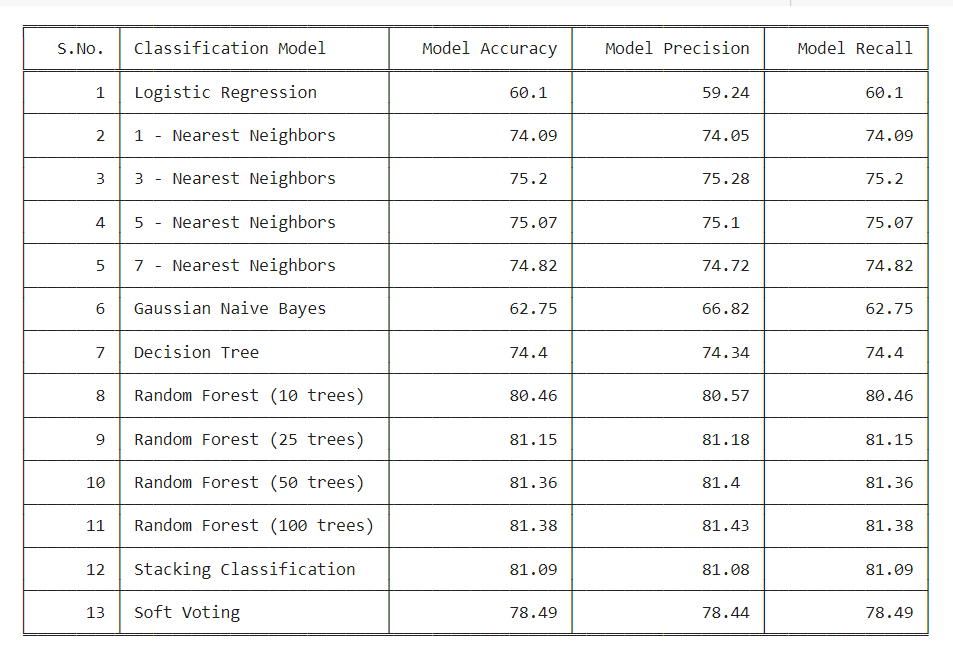
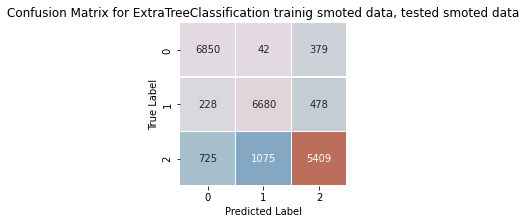
* We have already observed this using a pie chart, but we can obtain exact values of it using the value\_counts() function.  
   data\_train['Credit\_Score'].value\_counts()  
  This function gives the count of each class as output, as we have already encoded the target variables so the output is obtained for each encoded category.  
  **2.0** **48589**

**1.0** **26753**

**0.0** **15151**

Name: Credit\_Score, dtype: int64

Now we handle this data imbalance using SMOTE(Synthetic Minority Oversampling Techniques) using the below code and this balances the data by oversampling the minority class such that now each category would have 48589 data points. SMOTE can be implemented using the below code:  
 SMT = SMOTE()  
 x\_SMT,y\_SMT= SMT.fit\_resample(data\_train.drop('Credit\_Score',axis=**1**), data\_train['Credit\_Score'])

* This oversampled data is again used to train machine learning models, now we have used this data to train several models like Logistic regression, KNN with 1,3,5 and 7 nearest neighbors, Gaussian Naive Bayes, Decision Trees, Random Forest with 10, 25, 50 and 100 trees, stacking classifier and soft voting classifier.
* Now the best model obtained is the random forest model with 100 trees and the performance of each model on the test data can be seen below.  
  
* Let us observe the values and scores for some models and examine it, Firstly when looking over the logistic regression model, the model is able to generalize well but it is not able to learn from the whole data due to the large data size and complexity in the categorical columns and it is one of the fastest model to execute as compared to the other models.
* The K-Nearest Neighbor and decision tree are also good models but the execution or run time of KNN is exponentially high. The decision tree model even works well with 74% accuracy as pruning is even applied to reduce the problem of overfitting so let’s try to check if the bagging implementation of decision tree could improve the performance of the model.
* The Random Forest model is trained using 10 trees,25 trees, 50 trees and finally 100 trees. And it is observed that the Random forest with 100 trees has given the best performance.
* Also a stacking classifier and voting classifier is also used to ensure if stacking of different models would improve the performance and we can observe that the voting classifier has given lesser accuracy as compared to random forest models and also the stacking classifier has similar accuracy when compared to random forest with 100 trees.
* But the stacking classifier takes more execution time as compared to random forest as it is stacked with different ML models and also the interpretation of this model is less as compared to random forest so we have chosen random forest as the best model and tuned it using hyperparameter tuning. And we observed that the performance has slightly increased to 83%.
* Thus the Random forest model is the best model and the confusion matrix of this model can be visualized as:  
  
  + We can observe that most of the data points are labeled correctly as most of the points lie over the diagonal.
  + We can also observe that there is some confusion in between the standard and poor classes while classifying the data points; this may happen due to some correlation or even the further tuning of parameters could help the model.

## Conclusion

* We have solved a critical problem in the banking industry, banks usually look for customers with good credit scores so that they can give them promotional offers to lend more money to them in terms of loans, insurance, schemes etc.
* We have preprocessed the whole dataset, removed the outliers, imputed the null values, removed unnecessary columns , performed feature importance to extract the best features that can be given to our model.
* We have trained several machine learning models on this preprocessed data but we even observed that there was an issue of class imbalance in the data due to which accuracy cannot be taken as a suitable metric for this problem and the predictions cannot be interpreted easily so there was a need to handle this class imbalance.
* There are several methods for undersampling and oversampling the data but we chose one of the best methods that is SMOTE to oversample the data and get a balance around all the classes.
* Now we have trained all the machine learning models like logistic regression, K-nearest neighbor, random forest with different number of trees, stacking classifier and soft voting classifier and we have received random forest with 100 trees as the best model that has given best performance as:  
   The accuracy of this model **is** **81.38** %.

The precision of this model **is** **81.43** %.

The recall of this model **is** **81.38** %.

Thus we have successfully performed the credit score class classification into the 3 respective classes with an accuracy of 81%, precision of 81% and recall of 81% on the balanced data.

## Appendix

**Appendix 1: Tableau Public Link:**

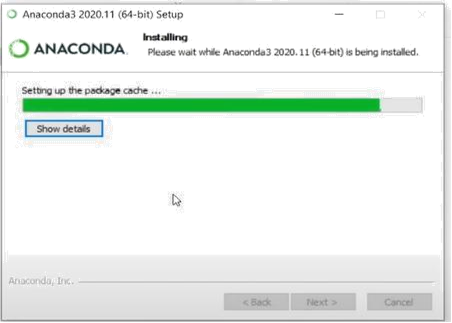
**Appendix 2: GitHub Link:**

**Appendix 3: Installation screenshots**

In this project, the PySpark setup is created in an anaconda environment and a Jupyter notebook is used as an IDE tool, which helps to navigate the code and text easily and also eases the installation of libraries that are used for machine learning models.

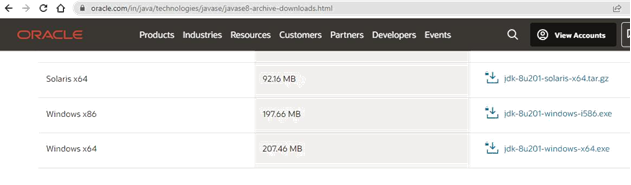
**Step 1-Anaconda Environment:** The first step is to create the anaconda framework by using the below-mentioned link. It helps to write Pyspark code with the help of a Jupyter notebook

<https://www.anaconda.com/products/distribution>

****

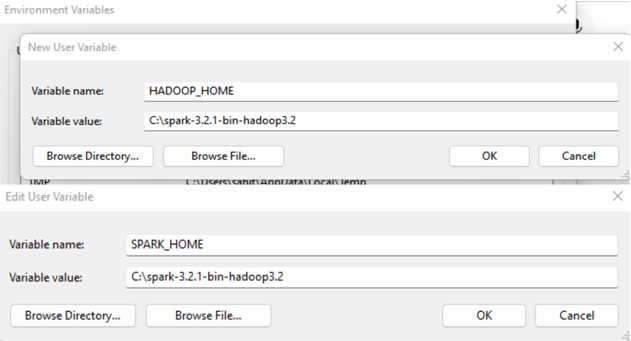
**Step2:** Installing the latest Java version from Oracle with the help of the below link

(<https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html> )

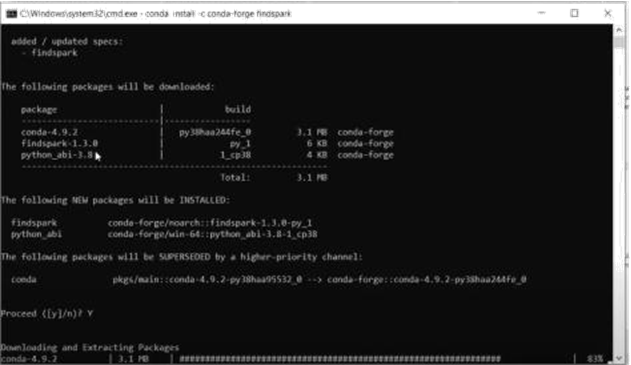
****

**Step 3**: Get the latest versions of Spark (https://spark.apache.org/downloads.html) and the Winutils Hadoop framework [(https://github.com/cdarlint/winutils)](https://github.com/cdarlint/winutils).

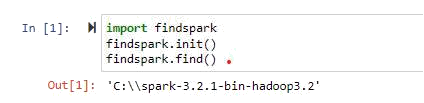
**Step 4:** Create and set up the environment variables for Spark, Hadoop, and Java as shown in the below figure (Figure 3). This step is mandatory to run the Spark environment without any issues



**Step 5:** Install all required libraries like Findspark, Pandas, Matplotlib, seaborn, and sci-kit-learn for data analysis and machine learning modeling



**Appendix 4: Code screenshots**



The files in the set directory are then examined with the ls function, which displays the downloaded spark file and the SPARK HOME configuration. To set up a sparking home, it was necessary to import a find spark and call the find spark.init() function to prevent errors. The required libraries for running the PySpark session were imported, and it was tested to ensure its functionality.

