Evaluating Classification Models in PySpark using ML

CSP 554 | Prof. Joseph Rosen A20425171 | Sanchayni Bagade

Abstract

The aim of this project was to compare performance of various classification models in PySpark ML and get hands-on experience in building the models, understanding the parameters and fine tuning them. For this experiment I used the banking dataset 'Give me some credit' from Kaggle aiming at building a robust classification model that predict if a credit card user would default.

This project helped me understand the difference between working with sklearn in python and ML in PySpark. It was challenging to work in PySpark when the interface is a terminal as it restricts the output format. Along with using classification models in ML, I explored Pygal library to make interactive plot in the Hadoop sandbox terminal and imported the svg files in my system.

Literature Review

In this section I reviewed existing work done in the domain of my problem statement and various resources I referred which helped me in smooth completion of this project.

Hend Sayed et al., (2018)[1] in their paper on predicting banking customer churn compared performance of Spark ML and MLlib on the customer dataset. They used decision tree model across both the tools and compared accuracy and processing time. The accuracy of the model was better using Spark ML whereas the training time was high. Based on this analysis we prefer Spark ML if we want to train the model less frequently and reuse it more. The article also details on using CrossValidator function for selecting the best parameter for model selection. The authors also do necessary pre-processing steps required before building the model, but it lacked or didn't cover the data exploration section.

I reviewed an article by Wael Etaiwi et al., (2017)[2] on evaluation of different classification algorithms for banking customer behaviour. It compares two classification techniques Naive Bayes and SVM using Spark MLlib. The experiment in this article was carried out using a single Spark equipment environment with 24 GB RAM and Core i7-6700 CPU which is analogous to what I am planning. The pre-processing step on the data was well articulated and was also deemed to be important and would be great resource to refer. The performance metric used for evaluation in this paper were precision, recall and F-measure. In our problem statement we want to have a low False Negative rate as we are dealing with default rate and thus, we need a high precision. Using these metrics, it was concluded that Naive Bayes is better than SVM for the specific dataset and they also conclude that multi-class methods are more efficient than binary classification method. This insight would be helpful while picking the classification model. I will be using Naive Bayes as well as ensemble model for my problem statement, so we can check if this holds true for our dataset.

The next paper by Mehdi Assefi et al., (2017)[3] evaluates not only different software configuration but also the hardware configuration of big data analysis task. The paper reviews existing work on big data analysis/ modeling using Spark MLlib. The authors analysed efficiency of Spark MLlib on 6 different datasets with varying size and two different environments with following configuration: ENV1(number of nodes: 2, HDD: 1 TB, RAM: 8GB, CPU: 4 vCPUs) and ENV2(number of nodes: 2, HDD: 1 TB, RAM: 16GB, CPU: 8 vCPUs). It uses AUC (Area under the curve) as a metric for evaluation. Itb was observed that Apache Spark MLlib performs significantly better than Weka libraries across different environments and dataset sizes. The authors detailed on how Spark Mllib is better than Weka in in other aspects like integration. MLlib gains from several software components available in the Spark ecosystem, such as Spark GraphX, Spark SQL, and Spark Streaming which are readily accessible.

Yet another article by Diego García-Gil et al.,(2017)[4] compares the performance of batch big data processing in Apache Spark versus Apache Flink. The authors compare efficiency of two machine learning algorithms present in both the tools, Support Vector Machines (SVM) and Linear Regression (LR). As an evaluation criterion, they employed an overall runtime. For varying dataset size, the runtime for Flink was higher than Spark MLlib for SVM model. It also compared ML with MLlib and Flink, it was observed that ML is 8 times faster than Flink for LR algorithm. Spark ML performed a bit slower than MLlib for DITFS algorithm due to internal Dataframe to RDD transformation, but overall Spark ML does better than MLlib and Flink.

From the literature review, I finalized on using Spark ML for my modeling not only due to its low runtime but also because of underlying data frame structure as we would like to retain it.

For referring Spark ML code sources, I resorted to Medium and Spark documentation. I was not able to find a good documentation of using machine learning models in Spark ML but there exists good material for MLlib. In a medium article by Susan Li (Machine learning with PySpark and MLlib) she builds various classification model to predict if a client will subscribe to a term deposit or not. She imports a spark session in python notebook and codes for ML algorithms in it. She has done few pre-processing steps like one-hot encoding for categorical columns, removing correlated variables, converting numeric values to desired format and so on. The author has built classification models like Logistic Regression, Decision Tree, Random Forest and Gradient-Boosted Tree; thus, this would be a good source for referring to Spark ML codes.

I would be referring to the article 'Logistic Regression in Spark ML' by Dhiraj Rai on Medium to fine tune models hyperparameters. This article also covers handling imbalanced classes, as discussed before we have a class imbalance due to low default rate (which is good!). I will also try to use K-fold validation for evaluating my classification models.

Demo/ Experiment Description
I will be evaluating performance of classification models like Logistic Regression, Naïve Bayes, Decision Tree, Random Forest and Gradient Boosting Trees based on metrics like accuracy, recall, precision and Area under the curve (AUC). I will be building a classification model in Spark ML to predict if someone would default and experience a financial distress. I used the 'Give Me Some Credit' Kaggle competition dataset for my analysis and modeling. The dataset originally consists of two sections training and test dataset with 150K and 102K rows respectively with 11 columns each. Due to restriction on task size at node to 100KB, I have reduced the dataset size to around 450 rows which is around 30 KB.

I also performed Exploratory Data Analysis (EDA) on our dataset to get a thorough understanding of distribution and variance of variables, their interdependency and correlation with each other and checking for missing values. I will be showcasing various analysis plots which are plotted using *Pygal* library in python. I generated the plots in pyspark, saved them as svg files and then imported them on my system. I have performed some pre-processing steps like Imputation and Standardization on the dataset to prepare it for modeling. We will be dividing our dataset into train and test and as our dataset has an imbalance target variable, we used stratified sampling for partitioning our dataset. We will build our model on train dataset and evaluate it on the test, as we are dealing with credit scoring, we would emphasize on precision rather than accuracy.

The dataset has two demographic variables: Age and Number of Dependents, it also has monetary variables: Monthly Income, Dept Ratio, Number of open credit lines and loans and

Number of Real Estate Loans or Lines. We have 4 credit related variables: Revolving Utilization of unsecured lines, Number of times 30-59 days past due, Number of times 60-89 days past due and Number of times 90 days late. We would expect the credit related variable to have relatively high importance in predicting the probability to default.

Exploratory Data Analysis
I analysed various univariate and bivariate plots on dependent and independent variables for getting a deeper insight into our dataset.

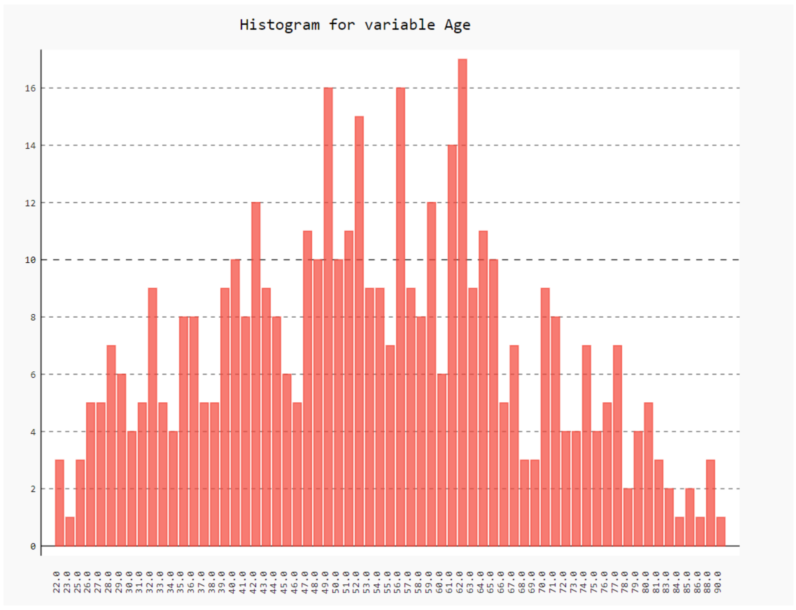


Figure 1. Histogram of Age

As expected, variable Age follows a normal distribution. Lot of credit card user's age is between 49 to 62 years, with average of the whole distribution being around 56 years old. This is not very intuitive as one would expect a lot of people in the age group of 30 - 40 years. This might be due to random sampling of the data from the original dataset

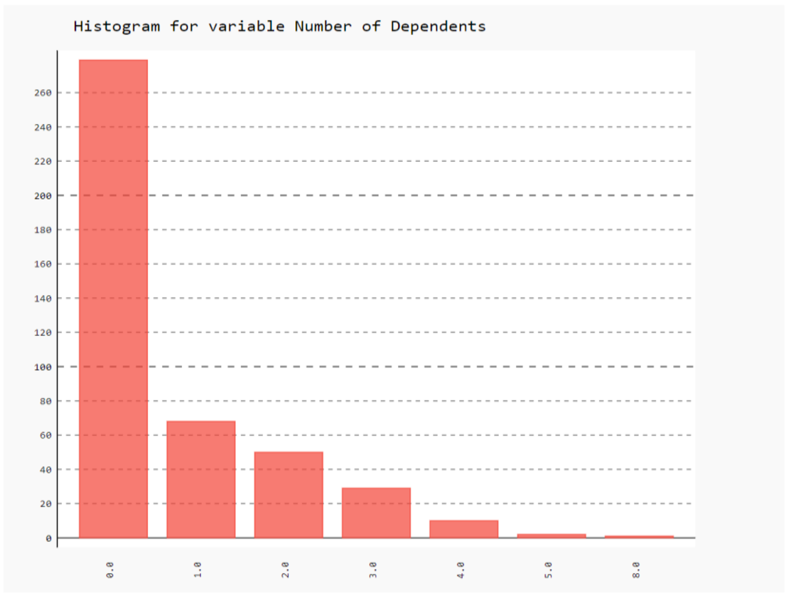


Figure 2. Histogram of Number of Dependents

Given the age range we have and a lot of users age lying between 40-60 years, I expected the Number of Dependents to be at least 1. But as we can see the mode is 0.

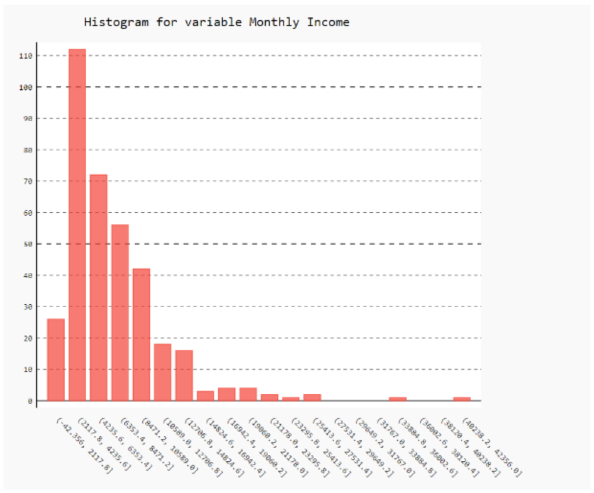


Figure 3. Histogram of Monthly Income

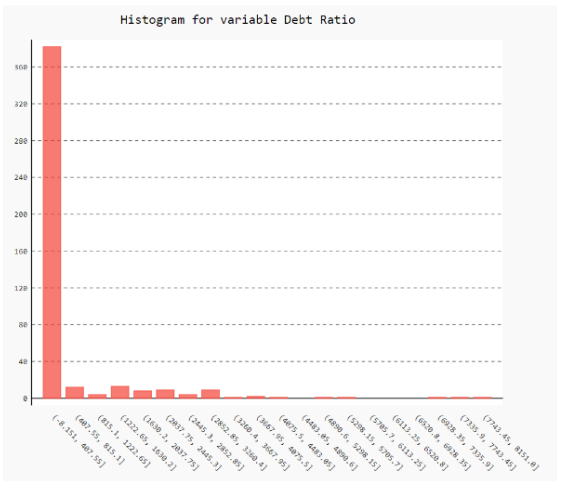
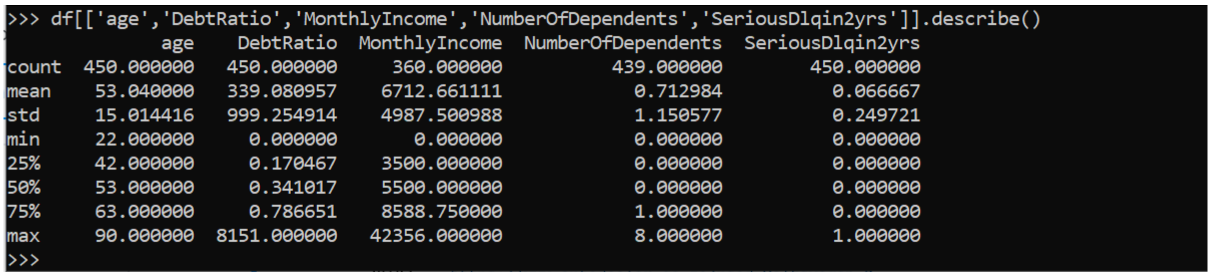


Figure 4. Histogram of Dept Ratio

The average Income is around $7,000 but as we can see the value is highly influenced by outliers. These high-income users can be an error, but we can't say for sure. A lot of users have monthly income between $2,000 to $4,000. In figure 3 you can see that the first bucket has a negative value, indicating that few users have negative values. This is not true; the minimum Monthly income value is zero. We reason this happens is due to the way cut function is defined in pandas, in future rather than using the function I would try to define my own function for creating these buckets.

The maximum value for Dept ratio of 8000% took me by surprise. Dept ratio as per the description from Kaggle is monthly debt payments, alimony, living costs divided by monthly gross income. These values must be outliers spiked due to misreporting of low income or high dept/ alimony cost. If we get a summary on all these variables, we get a better representation of Dept Ratio, with the 75th quantile to be 0.786 which is still less than 1.



```
>>> df[['age','DebtRatio','MonthlyIncome','NumberOfDependents','SeriousDlqin2yrs']].describe()
              age    DebtRatio  MonthlyIncome  NumberOfDependents  SeriousDlqin2yrs
count  450.000000   450.000000     360.000000          439.000000        450.000000
mean    53.040000   339.080957    6712.661111            0.712984          0.066667
std     15.014416   999.254914    4987.500988            1.150577          0.249721
min     22.000000     0.000000       0.000000            0.000000          0.000000
25%     42.000000     0.170467    3500.000000            0.000000          0.000000
50%     53.000000     0.341017    5500.000000            0.000000          0.000000
75%     63.000000     0.786651    8588.750000            1.000000          0.000000
max     90.000000  8151.000000   42356.000000            8.000000          1.000000
>>>
```

Figure 5. Summary on Age, Dept Ratio, Monthly Income, Number of Dependents and Serious Delinquency

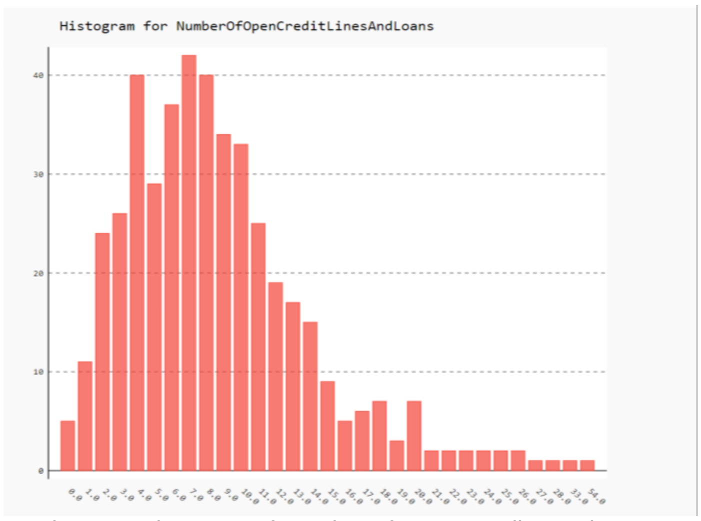
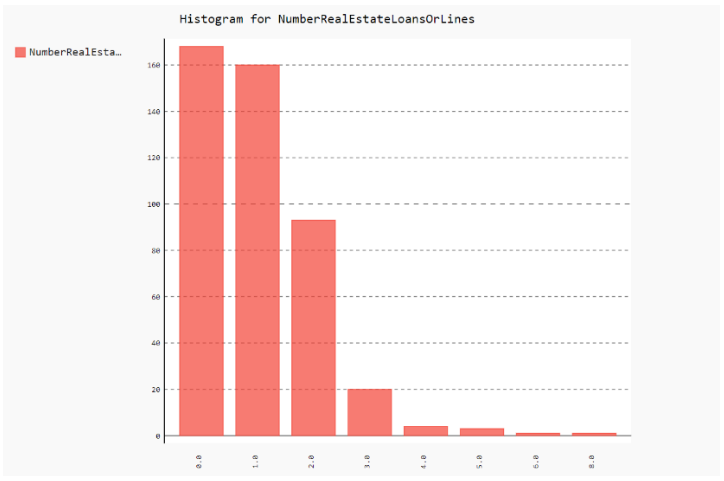


Figure 6. Histogram of Number of Open Credits or Lines

For our dataset the average number of Open loans and Lines of credit is around 8, which is reasonable. A person might have 3-4 credit cards and 3-4 open loans like car, house etc. The maximum value for this variable is 54, which is likely to be red flagged. This again depends on how much % of 54 is credit cards or otherwise. A higher value of this variable can signify either high credit cards or loans. I would have preferred a separate variable for both these factors.



This variable is defined as Number of mortgage and real estate loans like HELOC. I would expect this variable to have high importance as well in predicting a defaulter, as it an indicator for secondary loan.

Figure 7. Histogram of Number of Real Estate Loans or Lines

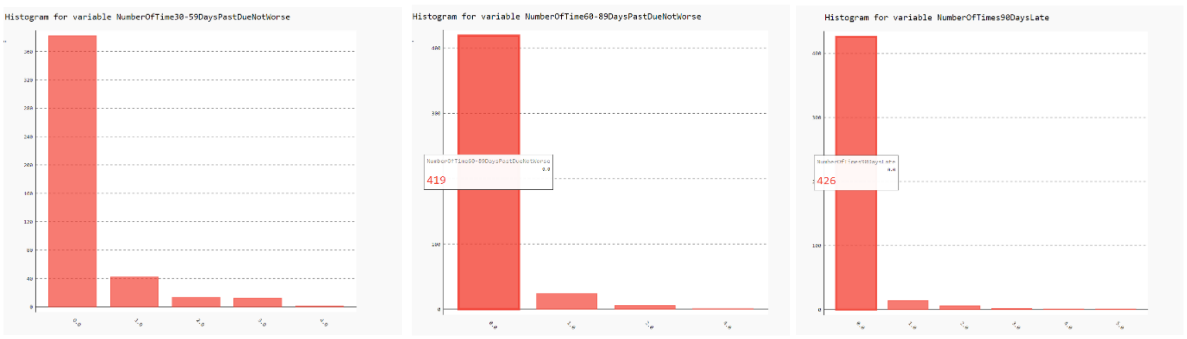
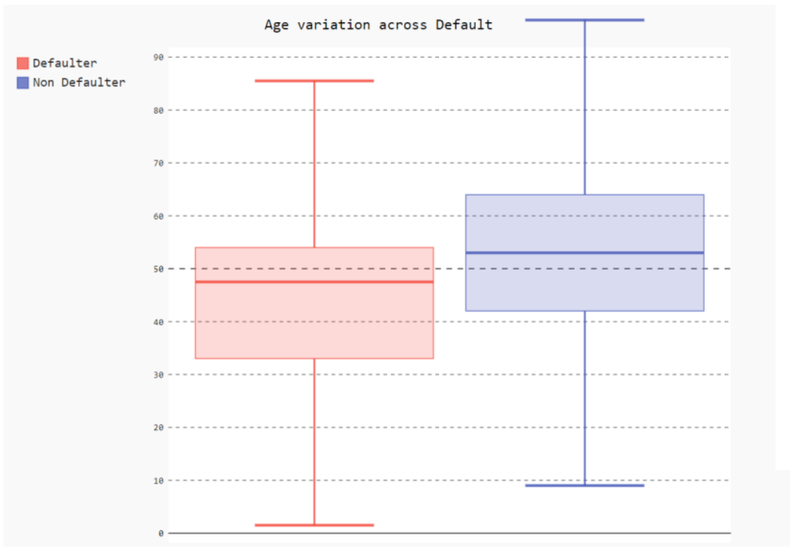


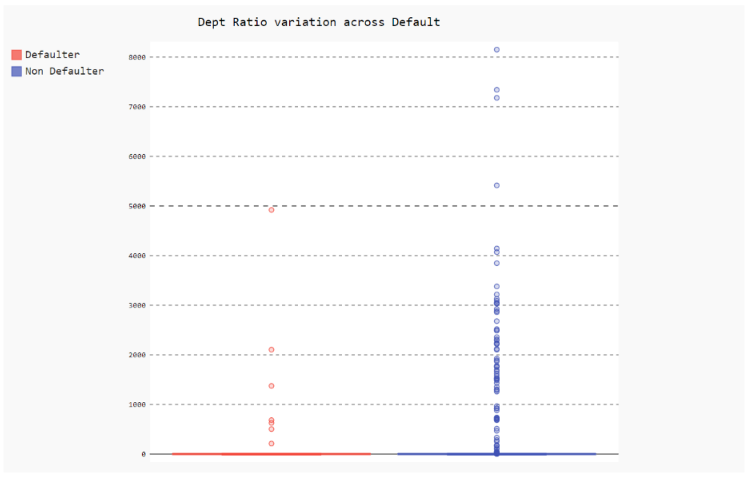
Figure 8. Frequency plot for (a) NumberOfTime30-59DaysPastDueNotWorse; (b) NumberOfTime60-89DaysPastDueNotWorse; (c) NumberOfTimes90DaysLate

From Figure 8 we can conclude that very few users pay the amount due after 30 days of billing date. Around 15% of the total users are past 30- 59 days past due date, out of which around 50% pay and then 6.88% of the users are still left and go in 60-89 days past due cycle. In the last cycle of 90 days past due we have 5.33% of the users. As these values are aggregated at user level, we can't use these numbers directly as an indicators of % of people from one past due bucket to another. Users who frequency fall under these cycles are likely to be red flagged as potential defaulters.



The average age of defaulters is a bit less than non-defaulters. There isn't a significant difference, but it can be said that young credit card users are likely to default.

Figure 9. Boxplot of Age versus Delinquency



This boxplot show variance of Dept ratio across defaulters and non-defaulters. This plot was counter intuitive as users with relatively high Dept ratio were non-defaulters.

Figure 10. Boxplot of Dept Ratio versus Delinquency

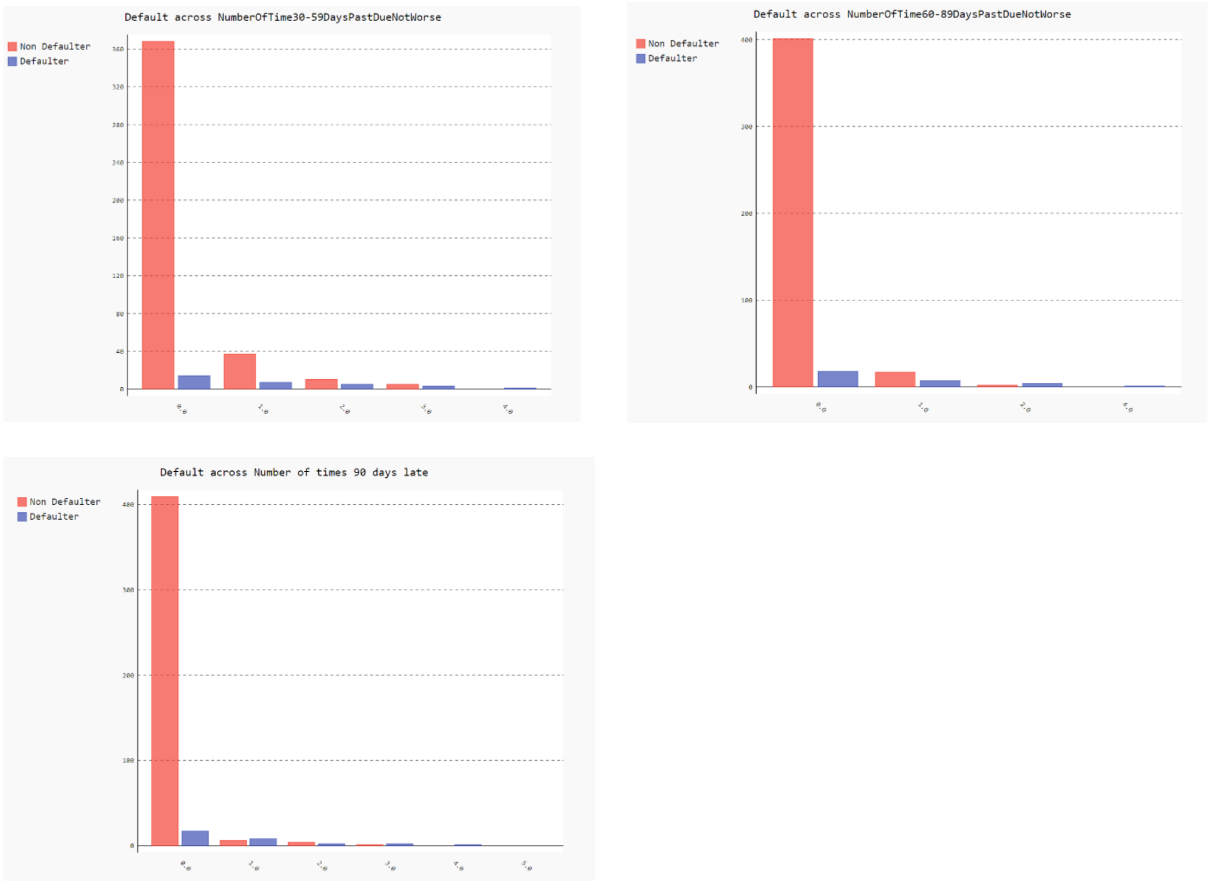
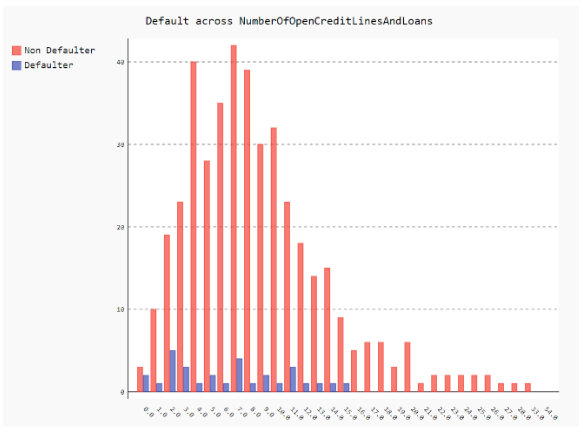


Figure 8. Boxplot plot of Delinquency across (a) NumberOfTime30-59DaysPastDueNotWorse; (b) NumberOfTime60-89DaysPastDueNotWorse; (c) NumberOfTimes90DaysLate

The result of these plots is very intuitive. Users who haven't missed a payment are less likely to default in future, even though we see few users who have zero late payments in past defaulting on payment. Even though the percentage is very low, it still exists. The higher the frequency of missing a further late cycle higher is the separation between defaulters and non-defaulters. Users missing more payment past 60-89 days are more likely to be red flagged than users missing more payments past 30-59 days.



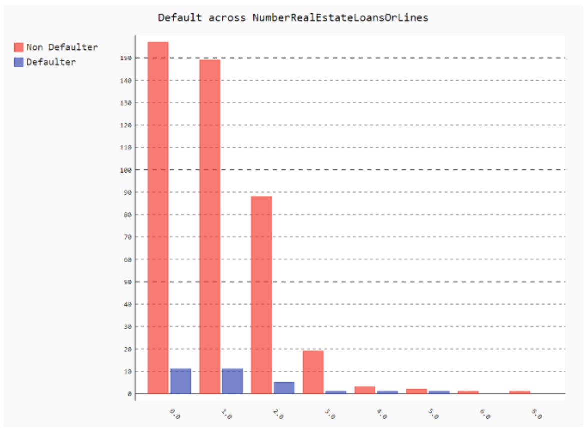
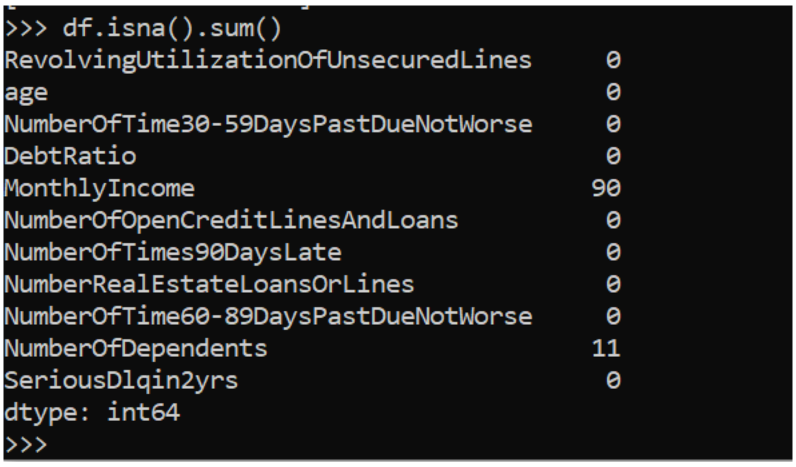


Figure 11. Frequency plot of Number of Open Credits or Lines and Delinquency

Figure 12. Frequency plot of Number of Real Estate Loans or Lines and Delinquency

As I mentioned earlier a higher value of Number of Open Credits or Lines might be defaulters but as the Figure 11 indicated it's the other way around. The higher values of this variables might be due to more credit cards than open loans, which can be an indicator of higher income or wealth. We would be able to better analyse this if we had separate variables for open credit lines and loan lines.
The plot for Number of Real Estate Loans or Lines and Delinquency was counterintuitive for me, as I expected the percent of defaulter increasing with increase in number of lines. As the plot shows the percent of defaulters do increase relatively but it's still less than non-defaulters.

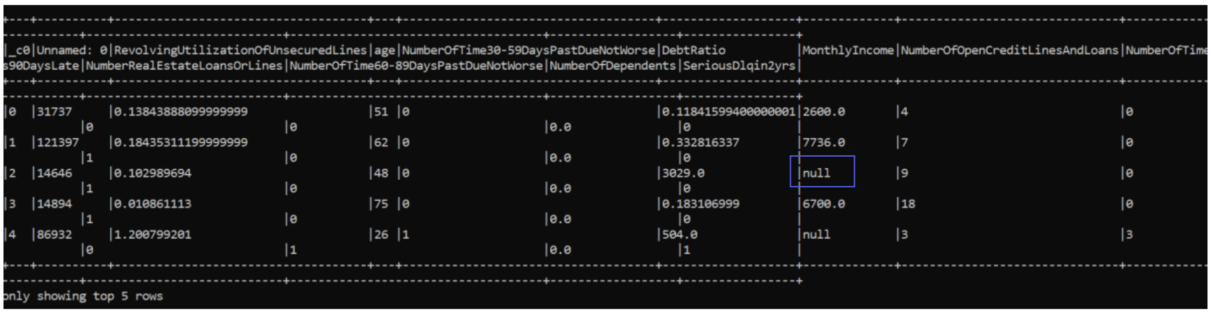
Pre-processing
Reading the data is very convenient with spark.read.csv(). It also has a parameter to specify if you want pyspark to understand the underlying schema of the dataset i.e., inferSchema. We had NAs present in two columns of our dataset, Monthly Income and Number of Dependents. Rather than removing those columns or rows, as I suspect these variables would be important in prediction, I used the Imputer function in ML.



```
>>> df.isna().sum()
RevolvingUtilizationOfUnsecuredLines      0
age                                       0
NumberOfTime30-59DaysPastDueNotWorse      0
DebtRatio                                 0
MonthlyIncome                            90
NumberOfOpenCreditLinesAndLoans           0
NumberOfTimes90DaysLate                   0
NumberRealEstateLoansOrLines              0
NumberOfTime60-89DaysPastDueNotWorse      0
NumberOfDependents                       11
SeriousDlqin2yrs                          0
dtype: int64
>>>
```

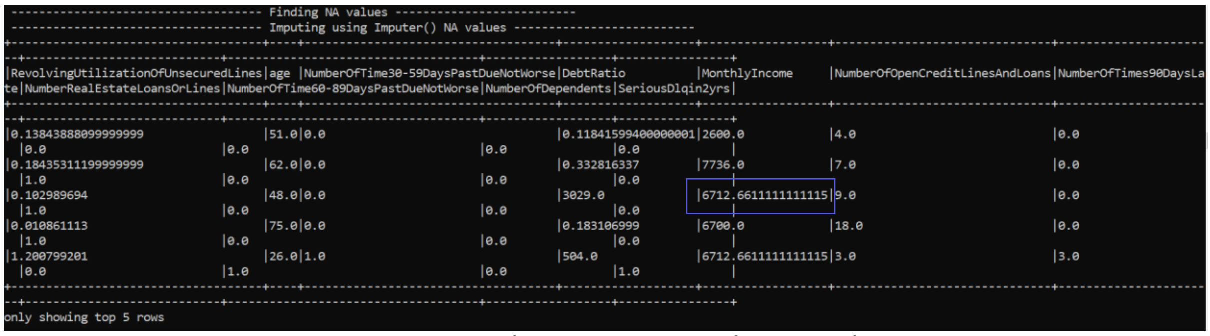
Screenshot 01. Number of NAs in our dataset

I didn't require any further cleaning as the data was significantly clean. As you can see in the screenshots below, the 3rd and 5th entry for Monthly Income is imputed with value $6712.66.



```
|_c0|Unnamed: 0|RevolvingUtilizationOfUnsecuredLines|age|NumberOfTime30-59DaysPastDueNotWorse|DebtRatio          |MonthlyIncome|NumberOfOpenCreditLinesAndLoans|NumberOfTime
s90DaysLate|NumberRealEstateLoansOrLines|NumberOfTime60-89DaysPastDueNotWorse|NumberOfDependents|SeriousDlqin2yrs|

|0  |31737     |0.13843888099999999                 |51 |0                                   |0.11841599400000001|2600.0       |4                              |0
           |0                          |0                                   |0.0               |0               |
|1  |121397    |0.18435311199999999                 |62 |0                                   |0.332816337        |7736.0       |7                              |0
           |1                          |0                                   |0.0               |0               |
|2  |14646     |0.102989694                         |48 |0                                   |3029.0             |null         |9                              |0
           |1                          |0                                   |0.0               |0               |
|3  |14894     |0.010861113                         |75 |0                                   |0.183106999        |6700.0       |18                             |0
           |1                          |0                                   |0.0               |0               |
|4  |86932     |1.200799201                         |26 |1                                   |504.0              |null         |3                              |3
           |0                          |1                                   |0.0               |1               |

only showing top 5 rows
```

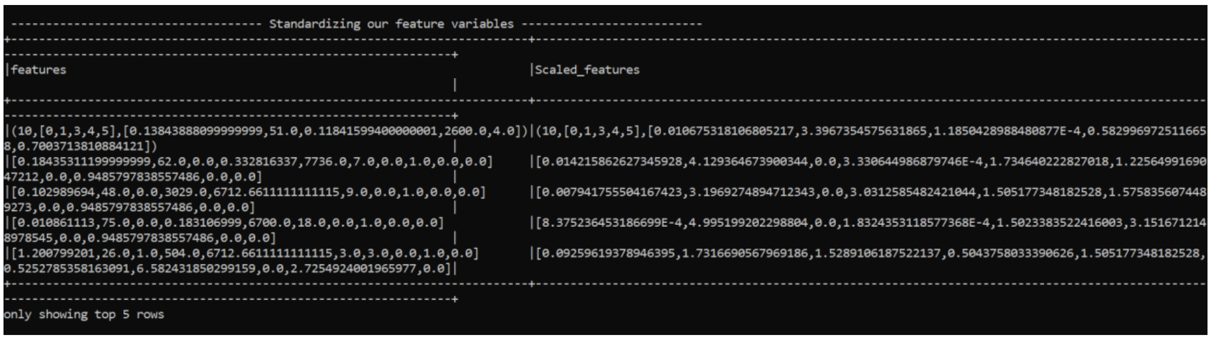
Screenshot 02. Top 5 rows before Imputation



```
 ---------------------------------- Finding NA values -------------------------
 ---------------------------------- Imputing using Imputer() NA values -------------------------

|RevolvingUtilizationOfUnsecuredLines|age |NumberOfTime30-59DaysPastDueNotWorse|DebtRatio          |MonthlyIncome     |NumberOfOpenCreditLinesAndLoans|NumberOfTimes90DaysLa
te|NumberRealEstateLoansOrLines|NumberOfTime60-89DaysPastDueNotWorse|NumberOfDependents|SeriousDlqin2yrs|

|0.13843888099999999                 |51.0|0.0                                 |0.11841599400000001|2600.0            |4.0                            |0.0
  |0.0                         |0.0                                 |0.0               |
|0.18435311199999999                 |62.0|0.0                                 |0.332816337        |7736.0            |7.0                            |0.0
  |1.0                         |0.0                                 |0.0               |
|0.102989694                         |48.0|0.0                                 |3029.0             |6712.6611111111115|9.0                            |0.0
  |1.0                         |0.0                                 |0.0               |
|0.010861113                         |75.0|0.0                                 |0.183106999        |6700.0            |18.0                           |0.0
  |1.0                         |0.0                                 |0.0               |
|1.200799201                         |26.0|1.0                                 |504.0              |6712.6611111111115|3.0                            |3.0
  |0.0                         |1.0                                 |1.0               |

only showing top 5 rows
```

Screenshot 02. Top 5 rows after Imputation

One of the modeling methods that I will be using is Logistic Regression and its advisable to standardize the data before modeling in regression as magnitude influences the regression coefficients. Thus to bring all the variables on the same scale I standardized the data using StandardScaler() function. Before standardizing the data needs to be vectorized for easier access to the variables while modelling and faster transformation.



```
 ---------------------------------- Standardizing our feature variables -------------------------

|features                                                         |Scaled_features
                                                                  |

|(10,[0,1,3,4,5],[0.13843888099999999,51.0,0.11841599400000001,2600.0,4.0])|(10,[0,1,3,4,5],[0.010675318106805217,3.3967354575631865,1.1850428988480877E-4,0.582996972511665
8,0.7003713810884121])                                            |
|[0.18435311199999999,62.0,0.0,0.332816337,7736.0,7.0,0.0,1.0,0.0,0.0]     |[0.014215862627345928,4.129364673900344,0.0,3.330644986879746E-4,1.734640222827018,1.2256499169
47212,0.0,0.9485797838557486,0.0,0.0]                             |
|[0.102989694,48.0,0.0,3029.0,6712.6611111111115,9.0,0.0,1.0,0.0,0.0]      |[0.007941755504167423,3.1969274894712343,0.0,3.0312585482421044,1.505177348182528,1.575835607448
9273,0.0,0.9485797838557486,0.0,0.0]                              |
|[0.010861113,75.0,0.0,0.183106999,6700.0,18.0,0.0,1.0,0.0,0.0]            |[8.375236453186699E-4,4.995199202298804,0.0,1.8324353118577368E-4,1.5023383522416003,3.151671214
8978545,0.0,0.9485797838557486,0.0,0.0]                           |
|[1.200799201,26.0,1.0,504.0,6712.6611111111115,3.0,3.0,0.0,1.0,0.0]       |[0.09259619378946395,1.7316690567969186,1.5289106187522137,0.5043758033390626,1.505177348182528,
0.5252785358163091,6.582431850299159,0.0,2.7254924001965977,0.0]|

only showing top 5 rows
```

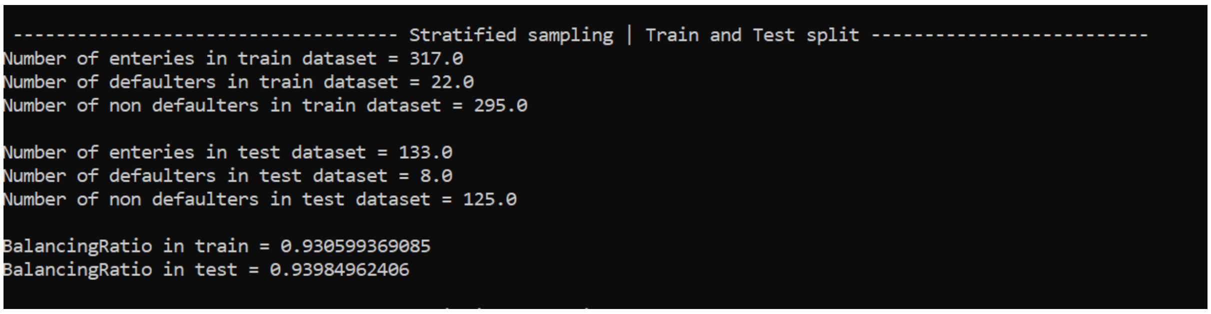
Screenshot 03. Scaled vs Unscaled features

The vector representation of our data after using Vector Assembler is in sparse vector format. The first entry in out features Vector is (10,[],[]) where the first value 10 is the length of the row, second is the index values where the values isn't zero and the last is the list of corresponding non-zero values. I think this representation is convenient for displaying your data and saves memory.

Modeling

Train and Test data sampling

Around 94% of our users are non-defaulters, which is realistic as very few people default on credit payment. We have a class imbalance issue in our data; thus, we will be using stratified sampling rather than random sampling.

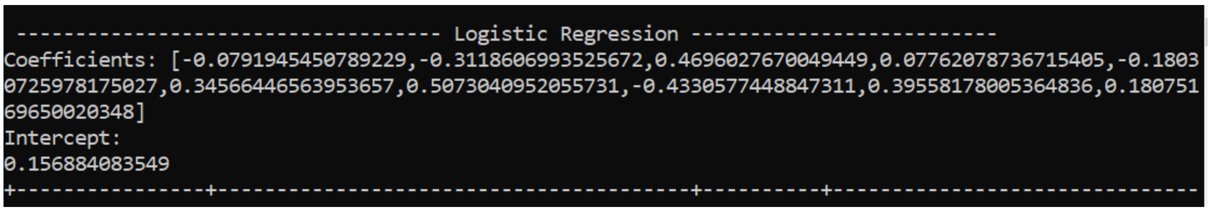


```
------------------------------------ Stratified sampling | Train and Test split -------------------------
Number of enteries in train dataset = 317.0
Number of defaulters in train dataset = 22.0
Number of non defaulters in train dataset = 295.0

Number of enteries in test dataset = 133.0
Number of defaulters in test dataset = 8.0
Number of non defaulters in test dataset = 125.0

BalancingRatio in train = 0.930599369085
BalancingRatio in test = 0.93984962406
```

Screenshot 04. Stratified sampling into train and test

Where the Balancing Ratio is non-defaulters divided by total users. I have assigned a class weightage (Balancing ratio) to our dependent variable, such that high weightage is assigned to default entries. We will be using weightage criterion only for Logistic regression and Naïve Bayes model.

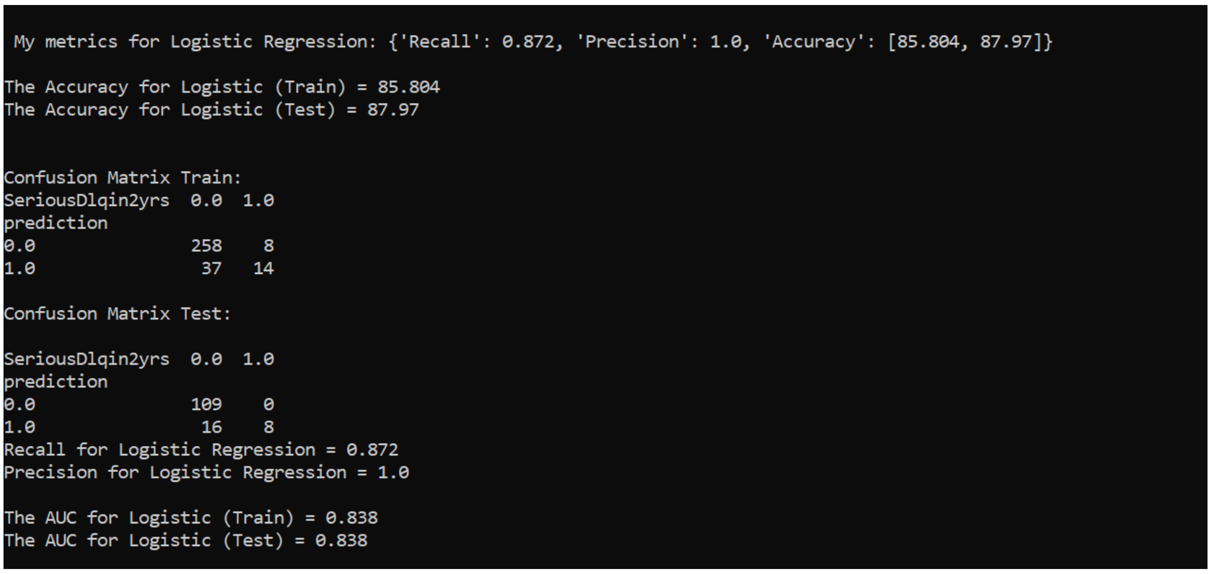
<u>Logistic Regression</u>
The input variables for this model are the scaled variables and we will also have the weightage criterion applied to our target variable.



```
------------------------------------ Logistic Regression --------------------------
Coefficients: [-0.0791945450789229,-0.3118606993525672,0.4696027670049449,0.07762078736715405,-0.1803
0725978175027,0.34566446563953657,0.5073040952055731,-0.4330577448847311,0.39558178005364836,0.180751
69650020348]
Intercept:
0.156884083549
+---------------+-------------------------------------+----------+------------------------------
```

Screenshot 05. Coefficients for Logistic Regression

It is difficult to interpret these coefficients as we don't have p values associated with them, which showcases the confidence in that variables. We see that the first and the fourth variable, Revolving Utilization of Unsecured line and Debt Ratio, have very small coefficients associated with them. During EDA, we got a counterintuitive insight from the Dept ratio box plot, which can be one justification of small magnitude.

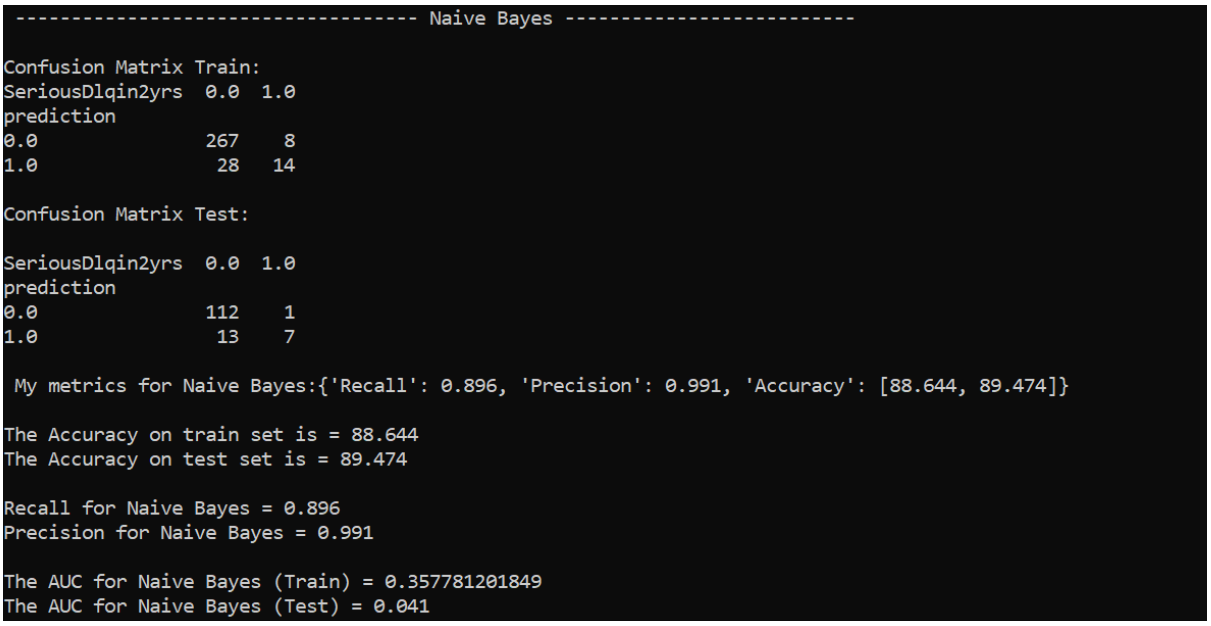


```
 My metrics for Logistic Regression: {'Recall': 0.872, 'Precision': 1.0, 'Accuracy': [85.804, 87.97]}

The Accuracy for Logistic (Train) = 85.804
The Accuracy for Logistic (Test) = 87.97


Confusion Matrix Train:
SeriousDlqin2yrs  0.0  1.0
prediction
0.0               258    8
1.0                37   14

Confusion Matrix Test:

SeriousDlqin2yrs  0.0  1.0
prediction
0.0               109    0
1.0                16    8
Recall for Logistic Regression = 0.872
Precision for Logistic Regression = 1.0

The AUC for Logistic (Train) = 0.838
The AUC for Logistic (Test) = 0.838
```

Screenshot 05. KPIs for Logistic Regression

Logistic Regression performs better on the test rather than the train data, which says that the model is not overfitting, but the way data is split influences our evaluation measures. I also performed a K-fold cross validation model for Logistic Regression to tune the hyper parameter, to make sure that our model is not sensitive to training data. But the metrics even after fine tuning the parameters are the same. In future I would divide the dataset into multiple train and test sections for better evaluation.

Naïve Bayes

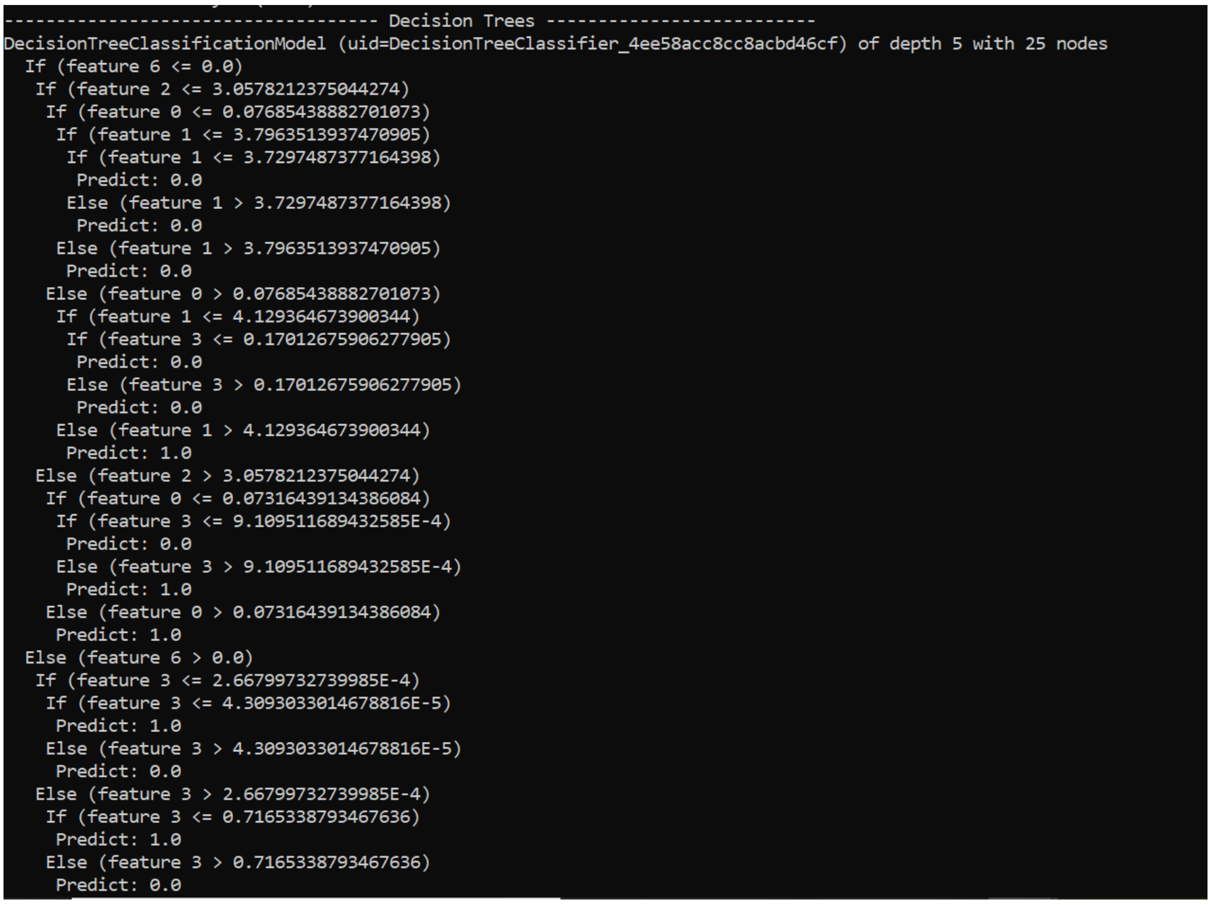


```
 ----------------------------------- Naive Bayes -------------------------

Confusion Matrix Train:
SeriousDlqin2yrs  0.0  1.0
prediction
0.0               267    8
1.0                28   14

Confusion Matrix Test:

SeriousDlqin2yrs  0.0  1.0
prediction
0.0               112    1
1.0                13    7

 My metrics for Naive Bayes:{'Recall': 0.896, 'Precision': 0.991, 'Accuracy': [88.644, 89.474]}

The Accuracy on train set is = 88.644
The Accuracy on test set is = 89.474

Recall for Naive Bayes = 0.896
Precision for Naive Bayes = 0.991

The AUC for Naive Bayes (Train) = 0.357781201849
The AUC for Naive Bayes (Test) = 0.041
```

Screenshot 06. KPIs for Naïve Bayes

Naïve Bayes outperforms Logistic Regression in all the metric but AUC, which is unreasonably low. I would have liked to investigate this nature of NB in future. Even for this modeling approach we have given more weightage to default user entries.

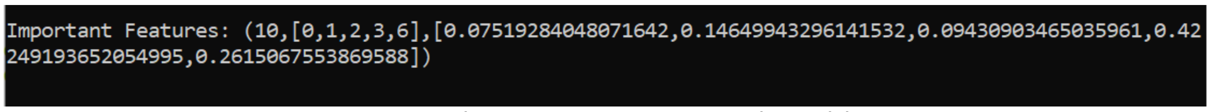
Decision Tree



```
----------------------------------- Decision Trees --------------------------
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_4ee58acc8cc8acbd46cf) of depth 5 with 25 nodes
  If (feature 6 <= 0.0)
   If (feature 2 <= 3.0578212375044274)
    If (feature 0 <= 0.07685438882701073)
     If (feature 1 <= 3.7963513937470905)
      If (feature 1 <= 3.7297487377164398)
       Predict: 0.0
      Else (feature 1 > 3.7297487377164398)
       Predict: 0.0
     Else (feature 1 > 3.7963513937470905)
      Predict: 0.0
    Else (feature 0 > 0.07685438882701073)
     If (feature 1 <= 4.129364673900344)
      If (feature 3 <= 0.17012675906277905)
       Predict: 0.0
      Else (feature 3 > 0.17012675906277905)
       Predict: 0.0
     Else (feature 1 > 4.129364673900344)
      Predict: 1.0
   Else (feature 2 > 3.0578212375044274)
    If (feature 0 <= 0.07316439134386084)
     If (feature 3 <= 9.109511689432585E-4)
      Predict: 0.0
     Else (feature 3 > 9.109511689432585E-4)
      Predict: 1.0
    Else (feature 0 > 0.07316439134386084)
     Predict: 1.0
  Else (feature 6 > 0.0)
   If (feature 3 <= 2.66799732739985E-4)
    If (feature 3 <= 4.3093033014678816E-5)
     Predict: 1.0
    Else (feature 3 > 4.3093033014678816E-5)
     Predict: 0.0
   Else (feature 3 > 2.66799732739985E-4)
    If (feature 3 <= 0.7165338793467636)
     Predict: 1.0
    Else (feature 3 > 0.7165338793467636)
     Predict: 0.0
```

Screenshot 07. Decision Tree

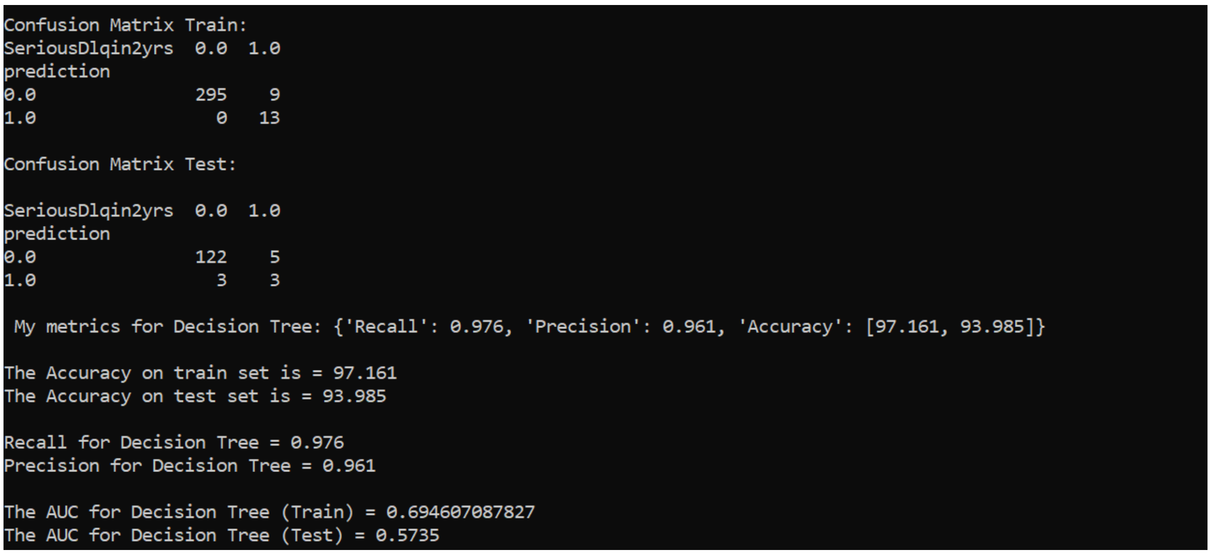
Screenshot 7 showcases the decision tree, with decision criterion at each split. We can also extract the feature importance using '.featureImportances' parameter.



```
Important Features: (10,[0,1,2,3,6],[0.07519284048071642,0.14649943296141532,0.09430903465035961,0.42
249193652054995,0.2615067553869588])
```

Screenshot 08. Feature Importance in Decision Tree

Variables: Revolving Utilization of Unsecured Lines, age, Number of times 30-59 days past due, Dept Ratio and Number of times 90 days late are important variables in deciding if a person would default in this decision tree. These variables being significant in deciding if a user will default is intuitive.

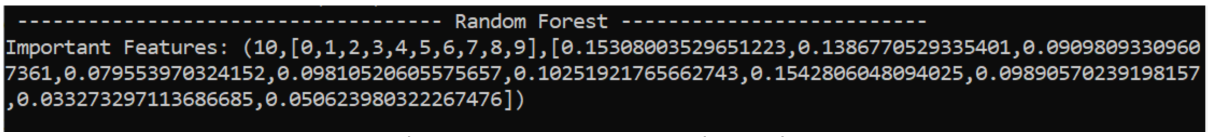


```
Confusion Matrix Train:
SeriousDlqin2yrs  0.0  1.0
prediction
0.0              295    9
1.0                0   13

Confusion Matrix Test:

SeriousDlqin2yrs  0.0  1.0
prediction
0.0              122    5
1.0                3    3

 My metrics for Decision Tree: {'Recall': 0.976, 'Precision': 0.961, 'Accuracy': [97.161, 93.985]}

The Accuracy on train set is = 97.161
The Accuracy on test set is = 93.985

Recall for Decision Tree = 0.976
Precision for Decision Tree = 0.961

The AUC for Decision Tree (Train) = 0.694607087827
The AUC for Decision Tree (Test) = 0.5735
```

Screenshot 10. KPIs for Decision Tree

The Recall and Accuracy has significantly increased as compared to Logistic and Naïve Bayes without reducing Precision significantly. We are still suffering with a low AUC value on train and test dataset.

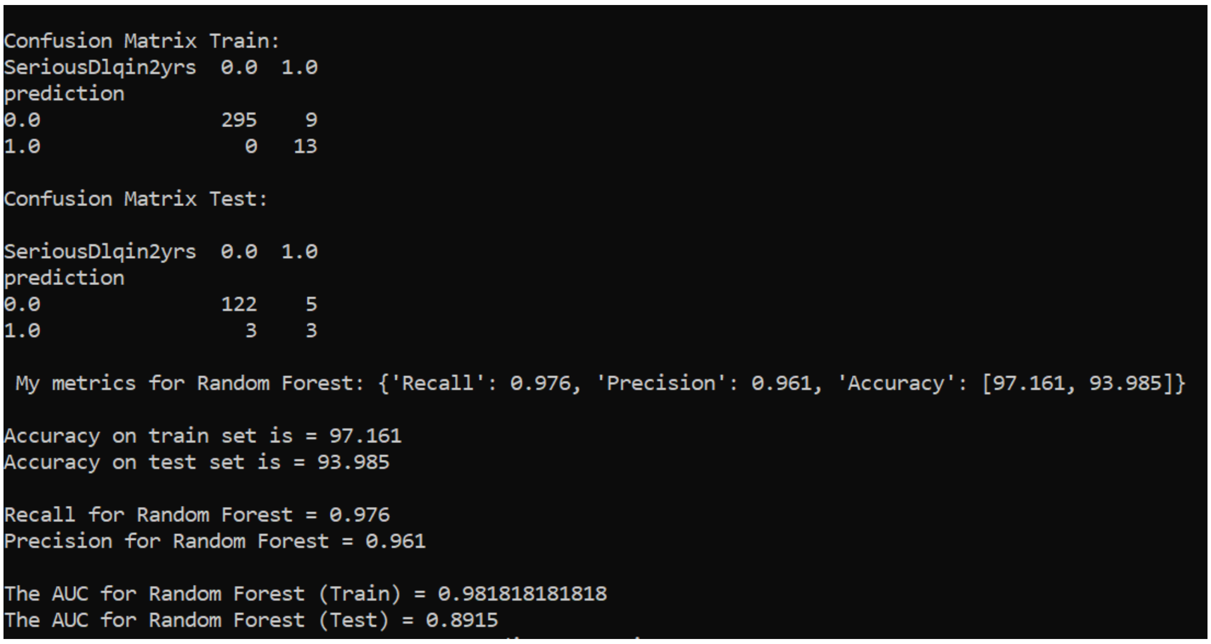
Random Forest



```
 ----------------------------------- Random Forest --------------------------
Important Features: (10,[0,1,2,3,4,5,6,7,8,9],[0.15308003529651223,0.1386770529335401,0.0909809330960
7361,0.079553970324152,0.09810520605575657,0.10251921765662743,0.1542806048094025,0.09890570239198157
,0.033273297113686685,0.050623980322267476])
```

Screenshot 11. Feature Importance in Random Forest

Unlike Decision tree Random Forest gives almost equal weightage to all variables, this signifies that our Decision Tree model was overfitting the dataset. Random Forest creates a tree for different bag of original dataset, thus making sure that it’s not overfitting.

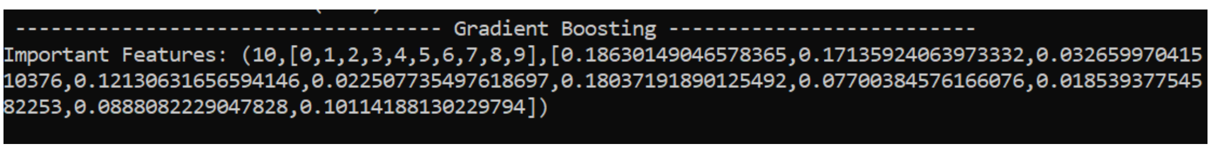


```
Confusion Matrix Train:
SeriousDlqin2yrs  0.0  1.0
prediction
0.0              295    9
1.0                0   13

Confusion Matrix Test:

SeriousDlqin2yrs  0.0  1.0
prediction
0.0              122    5
1.0                3    3

 My metrics for Random Forest: {'Recall': 0.976, 'Precision': 0.961, 'Accuracy': [97.161, 93.985]}

Accuracy on train set is = 97.161
Accuracy on test set is = 93.985

Recall for Random Forest = 0.976
Precision for Random Forest = 0.961

The AUC for Random Forest (Train) = 0.981818181818
The AUC for Random Forest (Test) = 0.8915
```

Screenshot 12. KPIs for Random Forest

As we can see that AUC for Random Forest is significantly better than decision tree, along with good accuracy, precision and recall. Random Forest has outperformed all the models we have discussed yet.

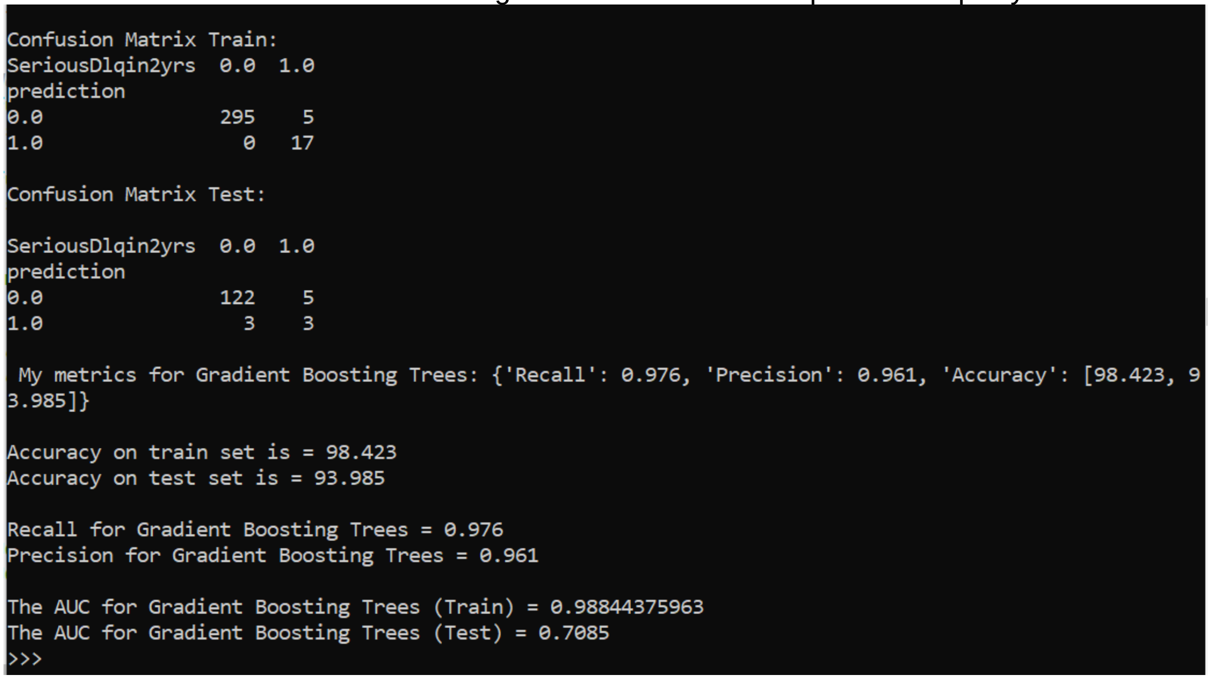
<u>Gradient Boosting Trees</u>



```
 ----------------------------------- Gradient Boosting --------------------------
Important Features: (10,[0,1,2,3,4,5,6,7,8,9],[0.18630149046578365,0.17135924063973332,0.032659970415
10376,0.12130631656594146,0.022507735497618697,0.18037191890125492,0.07700384576166076,0.018539377545
82253,0.0888082229047828,0.10114188130229794])
```

Screenshot 13. Feature Importance in Gradient Boosting Trees

Like Random Forest Gradient Boosting Tree distributes the importance equally



```
Confusion Matrix Train:
SeriousDlqin2yrs  0.0  1.0
prediction
0.0               295    5
1.0                 0   17

Confusion Matrix Test:

SeriousDlqin2yrs  0.0  1.0
prediction
0.0               122    5
1.0                 3    3

 My metrics for Gradient Boosting Trees: {'Recall': 0.976, 'Precision': 0.961, 'Accuracy': [98.423, 9
3.985]}

Accuracy on train set is = 98.423
Accuracy on test set is = 93.985

Recall for Gradient Boosting Trees = 0.976
Precision for Gradient Boosting Trees = 0.961

The AUC for Gradient Boosting Trees (Train) = 0.98844375963
The AUC for Gradient Boosting Trees (Test) = 0.7085
>>>
```

Screenshot 14. KPIs for Gradient Boosting Trees

The idea behind Gradient Boosting Trees (GBT) is to combine week learners to give better robust outputs. The AUC value for GBT decreases a bit from Random forest with same values for rest of the metrics.

Conclusion

Model Name	**AUC**	**Precision**	**Recall**	**Accuracy**
Logistic Regression	0.838	1.00	0.87	87.97
Naïve Bayes	0.041	0.99	0.89	89.47
Decision Tree	0.573	0.961	0.976	93.98
Random Forest	0.891	0.961	0.976	93.98
Gradient Boosting Trees	0.708	0.961	0.976	93.98

Table 1. KPIs for all the models

From the table above, we see that Random Forest performs better in most of the metrics. Even though Naïve Bayes has a better Precision and as I initially said that is important for our problem statement, it has a very low AUC. We conclude that Random Forests performs the best on our dataset, and we would be using this model for making flagging potential defaulters.

Discussion

- There is a significant difference in machine learning modeling in Python and in PySpark. The difference is not at the underlying methodology, but the way input is accepted, and output is generated.
- I struggled in extracting desired output from the models due to lack of experience in PySpark and inconsistency in output format. For example- Logistic Regression model has summary component, whereas Naïve Bayes doesn’t.
- Pre-processing steps like Imputation and Standardization are relatively faster in PySpark
- I was able to appreciate the sparse data storage of Vector Assembler and would like to learn more on storage of ‘null’ elements or a sparse data form like Tf idf.
- I would like to see in future a function that provided a detailed summary of a machine learning model irrespective of the machine learning technique.

Citations

(1) Hend Sayed, Manal A. Abdel-Fattah, Sherif Kholief, “Predicting Potential Banking Customer Churn using Apache Spark ML and MLlib Packages: A Comparative Study,” Information Systems Department Faculty of Computers and Information.

(2) Wei Zhang, Tong Mo, Weiping Li, Hanyu Huang, Xiaogang Tian, “The comparison of decision tree based Insurance Churn Prediction between Spark ML and SPSS,” 9th International Conference on Service Science.

(3) M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti, “Big data machine learning using apache spark MLlib,” Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017, vol. 2018-Janua, no.November, pp. 3492–3498, 2018.

(4) Diego García-Gil, Sergio Ramírez-Gallego, Salvador García and Francisco Herrera, “A comparison on scalability for batch big data processing on Apache Spark and Apache Flink,” Big Data Analytics, 2017, Volume 2

References

(1) https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa
(2) https://medium.com/@dhiraj.p.rai/logistic-regression-in-spark-ml-8a95b5f5434c
(3) https://towardsdatascience.com/healthcare-dataset-with-spark-6bf48019892b
(4) https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html
(5) https://spark.apache.org/mllib/

Codes

(1) Importing files from server to local system

```
scp -P 2222  maria_dev@localhost:/home/maria_dev/filename C:/Users/Sanchayni/Desktop
```

(2) Executing a .py file

```
execfile('/home/maria_dev/python_file_name.py')
```