Project Report

Apache Spark Machine Learning: Credit Risk Management and Maximizing Profit using Predictive Analysis

Introduction

Recently, financial institutions profits are on rise, especially in the area where the economic conditions are good. These institutions will continue to focus on growth of revenue, decreasing risk factors and improving customer relations. They are also in developing and improving new revenue streams by entering in new market areas and for this they require information and data regarding the customers and stream. This augments the amount of data as data is collected often and complexity of data is also increasing. Thus, Big Data is feasible solution for these banking and financial institutions to respond to the requirements in a quick and efficient manner. This project dives in that how adoption Big Data and analytics can meet the requirements of the dynamic financial services. This project specifically works on one of the major key challenges of today namely, Risk and Capital Management.

Traditional architectures and solutions have served banks and institutions well to manage credit, market liquidity and operational risks. Nowadays, we require a credit and behavioral scoring system to classify and approve of loan to new or existing customers and it needs significant analysis to give the worthiness of the applicant. This is one of the key challenges faced by the banks and financial institutions and can be solved using predictive modeling by taking data attributes such as Account Balance, Purpose, Age, Occupation and many such more attributes. Banking analysts are now building data and predictive models that can predict that (i) if the customer is likely to repay the loan in time and so not approving of loan can be a loss and (ii) if the customer is not likely to repay the loan in time given the following analysis then, approving of loan would not be profitable. This project covers the same model which help in giving efficient predictive model to approve of a loan of applicant based on the analysis of the given attributes and guide the bank officials to make an informed decision. With the help of this project, the institutions will have a clear view of applicant's ability to repay loan in time and will help in identifying future profits

Literature Review

Apache Spark

It is an open-source distributed cluster computing framework which was developed at University of California, Berkley in their AMPLab and was later given to the Apache group. It is an interface for programming entire clusters with implicit data parallelism and fault tolerance. It provides flexibility and extensibility of MapReduce but at significantly higher speeds.

Spark facilitates the implementation of both iterative algorithms and interactive data analysis. The latency of such applications may be reduced by several orders of magnitude compared to Apache Hadoop. It allows users to read, transform and aggregate data as well as train and deploy statistical models with ease. The Spark APIs are accessible to JAVA, Scala, Python, R and SQL. Apache Spark constitutes of

(i) Spark Core
(ii) Spark SQL

(iii) Spark Streaming
(iv) MLlib Machine Learning Library

Spark Core

It is the constitutional base for the entire project and provides distributed task dispatching, scheduling and basic I/O functions. The supported programming interface are JAVA, Python, Scala and R based on the abstraction of RDD

- RDD (Resilient Distributed Datasets):

RDDs belong to the core of API of Spark and is the basic data structure in Spark which can be created in two ways:

(i) Parallelizing an existing collection in driver program
(ii) Referring to a dataset in an external storage system, such as a shared filesystem, dataframe or any data source having hadoop input

Spark SQL

It is top component in Spark Core that introduced a data abstraction called DataFrames which can be described as follows:

- DataFrame:

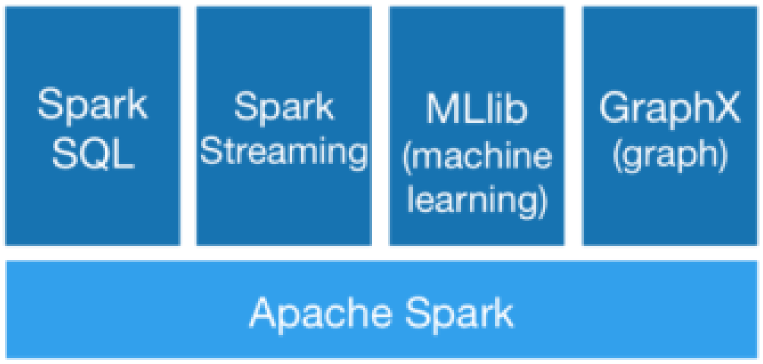
It is a distributed storage of data organized into named columns. It is similar to properties of relational database or a data frame R/Python but with higher speed and efficient streaming of data. It is available with Scala, Java, R and Python. Some of the features of DataFrame are: (i) Processing of structured and semi-structured data, (ii) Optimized querying, (iii) Representation in tabular format and (iv) Scalability and flexible to varied data formats such as JSON, XML and many more. DataFrames keep record of the schema and support various relational operations that helps in more efficient results unlike RDDs. It also constitutes all the properties of RDDs

Spark Streaming

It utilizes Spark Core's fast scheduling capability to perform streaming data and analytics. It performs on the data which are ingested in mini- batches and perform RDD transformation on it. The design generally lacks the latency factor which is equal to the mini-batch duration. Other streaming data engines that process event by event instead of mini-batches are Strom and Fink

MLlib Machine Learning Library

It is a distributed machine-learning framework on top of Spark core. It provides efficient functions for a wide range of learning algorithms and has many underlying statistical, optimization, and linear algebra primitives. It is exponentially increasing because of its varied open-source community. It consists of following tools: (i) ML Algorithms (ii) Featurization (iii) Pipelines (iv) Persistence and (v) Utilities. The popular ML algorithms ad utilities in Spark MLlib are: (i) Basic Statistics (ii) Regression (iii) Classification (iv)Recommendation (v) Clustering (vi) Dimensionality Reduction and (vii) Feature Extraction



<u>Comparison between Spark MLlib and Tensorflow</u>

Spark MLlib:

Spark is not specifically for machine learning; however, it has machine learning library called MLlib that contains common basic algorithms, utilities and linear algebra operations. But, for large scale machine learning applications the model parameters may not fit into driver node and they would need to be regularly maintained as RDD which in turn gives a lot of overhead. This is because for each iteration an RDD is to be created to maintain the RDD parameters. Thus, this will increase the expected time and since updating the model also includes splitting and shuffling of data to obtain outputs, this is where the Spark has its limitations

Tensorflow:

It is a first-generation distributed parameter-server system and an open-source platform. Unlike relational dataflow, TensorFlow let the nodes to represent computations that own or update flexible status. It has three main components: client, master, worker. Unlike Spark, it does not need to create or maintain an RDD so the limitations of Spark can overcome with TensorFlow. Comparatively TensorFlow provides with more APIs and primitives which allows the user to use the existing packages and algorithms of TensorFlow or build their own algorithms from base using different APIs. But, CPU utilization for TensorFlow is more than that of Spark which make of the limitations of TensorFlow but can be reduced using different cluster size

<u>Scala or Python. Which one would be a better choice for project?</u>

When we are working with DataFrames, there is not much difference between Python and Scala, but we do have to take in account the User Defined Functions (UDF) which are less efficient than its Scala equivalents. That is why we should use the Python platform while working with DataFrame and RDDs which is observed in this project. Python is better in terms of learning curve and easy to use as it is understandable and readable than Scala. Also, for data projects as Python provides more tools for machine learning and natural language processing in the SparkMLlib, it becomes a clear choice.

Dataset Description

For this project, I am going to use the dataset named German Credit Data which is being loaded from the following site and then converted to csv format for further analysis:
https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

Dataset Attribute Information:

Attribute	Description	Example
Credibility	Label	0,1
Account Balance	Status of existing checking account	1,2,3,4
Credit Duration	Duration in months (numerical)	Continuous Variable
History	Credit History (qualitative)	0,1,2,3,4
Purpose	Purpose for loan application (qualitative)	0,1,2,3,4,5,6,7,8,9,10
Credit Amount	Amount of credit requested (numerical)	Continuous Variable
Savings	Value of savings (qualitative)	1,2,3,4,5
Employment	Present employment status (qualitative)	1,2,3,4,5
Installment Percent	Installment rate in percentage (qualitative)	1,2,3,4
Gender and Status	Personal marital status and sex (qualitative)	1,2,3,4,5
Guarantors	Other debtors/guarantors (qualitative)	1,2,3
Residence Time	Present resident since (numerical)	1,2,3,4
Assets	Property possessions (qualitative)	1,2,3,4
Age	Age of the person (numerical)	Continuous Variable
Concurrent Credit	Concurrent Credits (qualitative)	1,2,3
Housing	Type of Apartment (qualitative)	1,2,3
Credit	Number of existing credits at this bank (numerical)	1,2,3,4
Job	Job (qualitative)	1,2,3,4
Dependents	Number of dependents on applicant (numerical)	1,2
Telephone	Possess Telephone (qualitative)	1,2
Foreign Worker	Is a foreign worker (qualitative)	1,2

Model Review

Logistic Regression:

It is the in-demand method for predicting a categorical response and special case of Generalized Linear Models which predicts the probability of outcomes. It is found in the pyspark.ml.classification package. This method is generally utilized to predict a binary outcome with binomial logistic regression or multiclass outcome with multinomial logistic regression. As we have binary target variable, we will be using the binomial logistic regression for this project instead of linear regression

Random Forest Classifier:

Random Forests are basically the amalgamation of decision trees in order to reduce the possibility of overfitting the model. This classifier can also be found in pyspark.ml.classification package. For this classifier you have to provide the input columns as inputCols and as a result of output columns we get the predictionCol, rawPredictionCol and probabilityCol. This classifier also supports both the binary outcome and multiclass labels for prediction analysis of data

Gradient Boosted Trees:

Recently, gradient boosted trees are becoming popular when it comes to the problems of classification and regression method using combination of decision trees similar to the random forest. The only change in this classifier is that it works with weights. It works in iterations and for the first iteration it is initialized with equal weights and from the second iteration it starts assigning more weights to misclassified points and less weights to correctly classified labels.

Evaluation of Model:

For the evaluation of all the models, we will be using the BinaryClassificationEvaluator() for Area under Curve/ ROC and MultiClassificationEvaluator() for Accuracy metric. These two evaluators are found under the pysparl.ml.evaluation package. By using these for all the models we will obtain Area Under Curve/ ROC and Accuracy metric which will the help us in finding that which model performs better for the given data.

Analysis

Use Case:

When a bank gets a loan application, the bank must make a decision to approve the loan or not by factors of customer like account balance, purpose, account balance, credit amount, occupation and many such attributes. The risks included in bank decision are:

- If the customer is likely to repay the loan in timely manner, then not approving loan would be loss to bank
- If the customer is not likely to repay the loan in time, then approving loan for that person in not profitable

Objective:

Risk Minimization and Maximizing Profit for Financial Institutions

As per the given name, the objective of this project is to reduce the losses incurred by the financial institutions and for that the organization requires a decision method to provide the loan approval based on certain attributes of the applicant. Thus, all the social, demographic and economic attirbutes are considered for this project as seen in the data description. This project aims to help by giving an efficient predictive model to approve a loan to prospective applicant and guide the bank manager in making a profitable decision and less risk to banks.

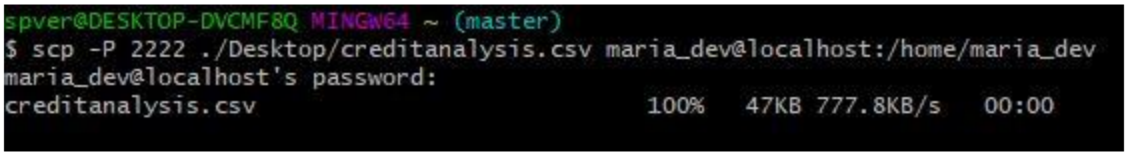
Method:

- The data will be loaded in form of csv file to home of the cluster and then copied to the user directory
- There will exploratory data analysis to learn more about how the attributes affect the credibility of applicant
- Then the data will be preprocessed and converted to format to give to the models for the prediction
- After this, the data will be split in training and testing data used to train and test the given model respectively
- After implementing the models, evaluation would be done on the basis of Area Under Curve/ROC and Accuracy and the best model will be chosen based on these metrics.
- Platform and tools used for this project:
    - i. Spark csv module (Microsoft Azure services platform) and DataFrame
    - ii. Python API (pyspark)
    - iii. MLLib packages
    - iv. ML API
    - v. Horton works Sandbox

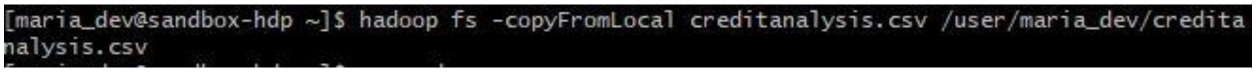
Methodology

Platform and Data Preparation:

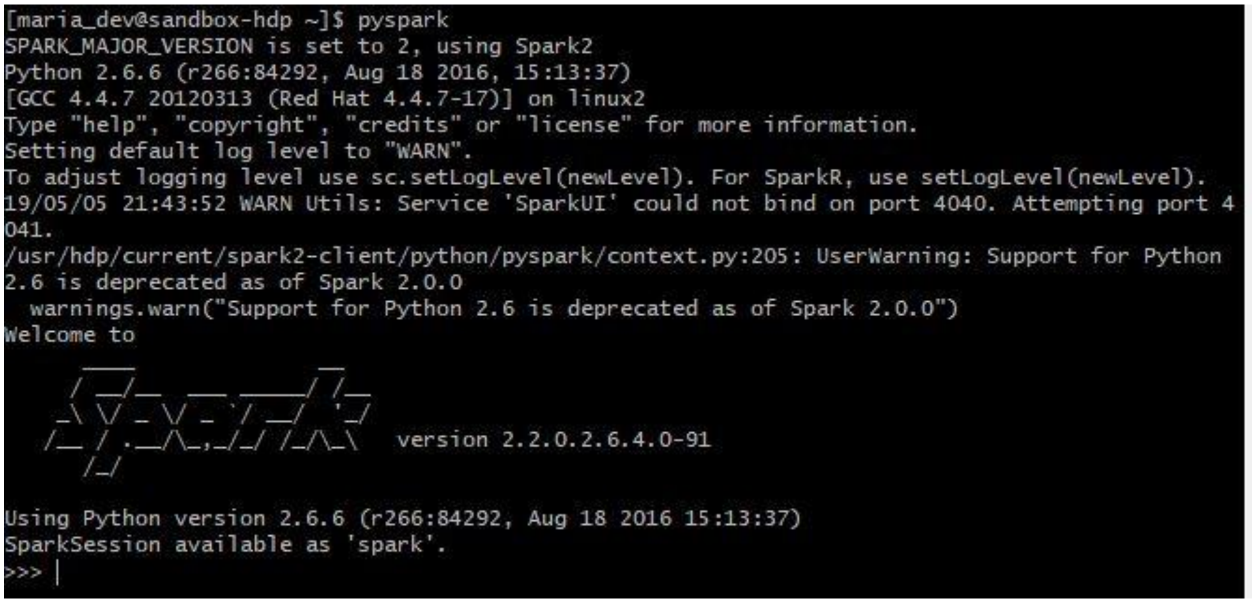
- Here, first we will load the dataset which is already converted to csv file from raw data to the cluster
- So, the csv file is copied to the hadoop cluster and then from the home it is then moved to the user directory to proceed with further analysis on data
- For this project we are going to use the pyspark platform provided on the cluster with the version details given the screenshot below



```
spver@DESKTOP-DVCMF8Q MINGW64 ~ (master)
$ scp -P 2222 ./Desktop/creditanalysis.csv maria_dev@localhost:/home/maria_dev
maria_dev@localhost's password:
creditanalysis.csv                          100%   47KB 777.8KB/s   00:00
```

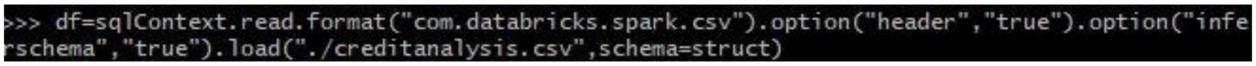


```
[maria_dev@sandbox-hdp ~]$ hadoop fs -copyFromLocal creditanalysis.csv /user/maria_dev/credita
nalysis.csv
```



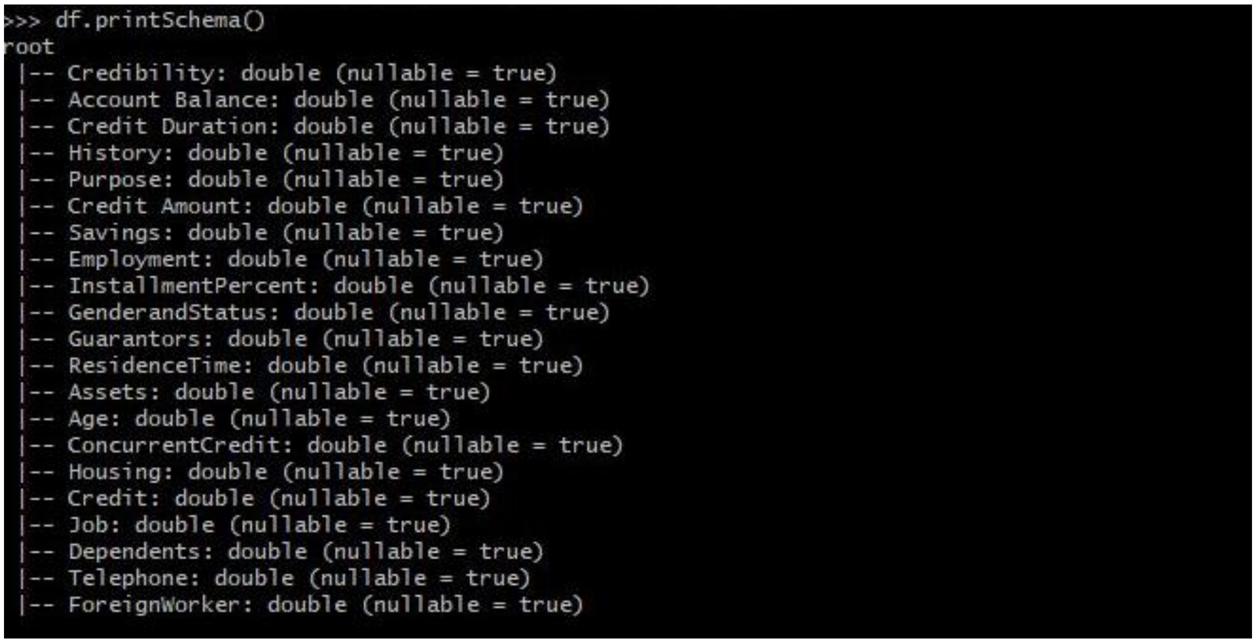
```
[maria_dev@sandbox-hdp ~]$ pyspark
SPARK_MAJOR_VERSION is set to 2, using Spark2
Python 2.6.6 (r266:84292, Aug 18 2016, 15:13:37)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-17)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/05/05 21:43:52 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4
041.
/usr/hdp/current/spark2-client/python/pyspark/context.py:205: UserWarning: Support for Python
2.6 is deprecated as of Spark 2.0.0
  warnings.warn("Support for Python 2.6 is deprecated as of Spark 2.0.0")
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.2.0.2.6.4.0-91
      /_/

Using Python version 2.6.6 (r266:84292, Aug 18 2016 15:13:37)
SparkSession available as 'spark'.
>>> |
```

- Now, moving with creating the struct for the reading the data and defined the datatype of each attribute of the data. It is then saved in form of the dataframe
- For loading the data in form of dataframe, I have created the schema for loading data using StructType and StructField and then it is given to the spark_csv module for loading the data as DataFrame



```
>>> df=sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("infe
rschema","true").load("./creditanalysis.csv",schema=struct)
```

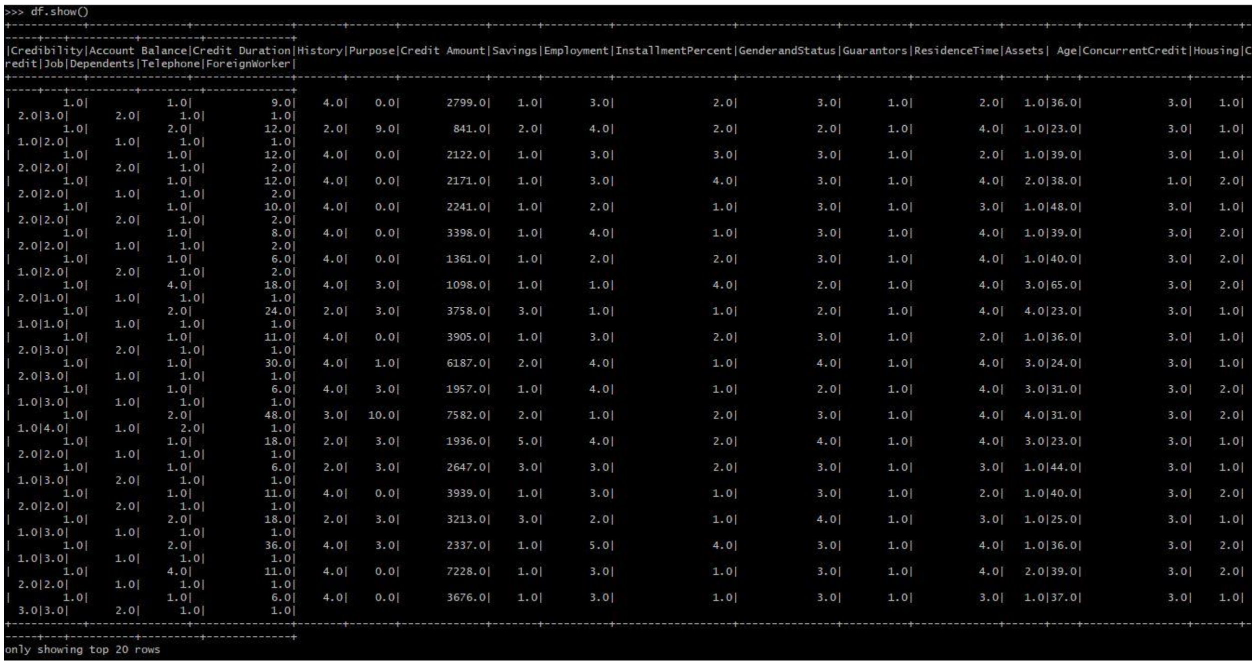
- The schema for the this dataframe can be given like:



```
>>> df.printSchema()
root
 |-- Credibility: double (nullable = true)
 |-- Account Balance: double (nullable = true)
 |-- Credit Duration: double (nullable = true)
 |-- History: double (nullable = true)
 |-- Purpose: double (nullable = true)
 |-- Credit Amount: double (nullable = true)
 |-- Savings: double (nullable = true)
 |-- Employment: double (nullable = true)
 |-- InstallmentPercent: double (nullable = true)
 |-- GenderandStatus: double (nullable = true)
 |-- Guarantors: double (nullable = true)
 |-- ResidenceTime: double (nullable = true)
 |-- Assets: double (nullable = true)
 |-- Age: double (nullable = true)
 |-- ConcurrentCredit: double (nullable = true)
 |-- Housing: double (nullable = true)
 |-- Credit: double (nullable = true)
 |-- Job: double (nullable = true)
 |-- Dependents: double (nullable = true)
 |-- Telephone: double (nullable = true)
 |-- ForeignWorker: double (nullable = true)
```

Data exploration and feature extraction:

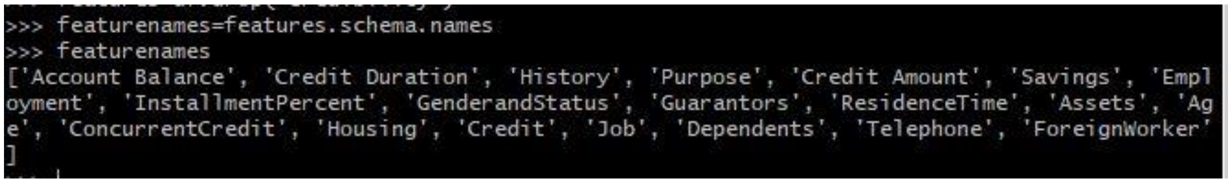
- For this section, we will be exploring the contents of data and study which attributes of the data affect much on the target variables
- The data in the csv file can be shown as follows (showing only 20 rows from 1000 rows):



```
>>> df.show()
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+---------------+----------+-------------+------+----+----------------+-------+-
-----+---+----------+---------+-------------+
|Credibility|Account Balance|Credit Duration|History|Purpose|Credit Amount|Savings|Employment|InstallmentPercent|GenderandStatus|Guarantors|ResidenceTime|Assets| Age|ConcurrentCredit|Housing|C
redit|Job|Dependents|Telephone|ForeignWorker|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+---------------+----------+-------------+------+----+----------------+-------+-
-----+---+----------+---------+-------------+
|        1.0|            1.0|            9.0|    4.0|    0.0|       2799.0|    1.0|       3.0|               2.0|            3.0|       1.0|          2.0|   1.0|36.0|             3.0|    1.0| 
  2.0|3.0|       2.0|      1.0|          1.0|
|        1.0|            2.0|           12.0|    2.0|    9.0|        841.0|    2.0|       4.0|               2.0|            2.0|       1.0|          4.0|   1.0|23.0|             3.0|    1.0| 
  1.0|2.0|       1.0|      1.0|          1.0|
|        1.0|            1.0|           12.0|    4.0|    0.0|       2122.0|    1.0|       3.0|               3.0|            3.0|       1.0|          2.0|   1.0|39.0|             3.0|    1.0| 
  2.0|2.0|       2.0|      1.0|          2.0|
|        1.0|            1.0|           12.0|    4.0|    0.0|       2171.0|    1.0|       3.0|               4.0|            3.0|       1.0|          4.0|   2.0|38.0|             1.0|    2.0| 
  2.0|2.0|       1.0|      1.0|          2.0|
|        1.0|            1.0|           10.0|    4.0|    0.0|       2241.0|    1.0|       2.0|               1.0|            3.0|       1.0|          3.0|   1.0|48.0|             3.0|    1.0| 
  2.0|2.0|       2.0|      1.0|          2.0|
|        1.0|            1.0|            8.0|    4.0|    0.0|       3398.0|    1.0|       4.0|               1.0|            3.0|       1.0|          4.0|   1.0|39.0|             3.0|    2.0| 
  2.0|2.0|       1.0|      1.0|          2.0|
|        1.0|            1.0|            6.0|    4.0|    0.0|       1361.0|    1.0|       2.0|               2.0|            3.0|       1.0|          4.0|   1.0|40.0|             3.0|    2.0| 
  1.0|2.0|       2.0|      1.0|          2.0|
|        1.0|            4.0|           18.0|    4.0|    3.0|       1098.0|    1.0|       1.0|               4.0|            2.0|       1.0|          4.0|   3.0|65.0|             3.0|    2.0| 
  2.0|1.0|       1.0|      1.0|          1.0|
|        1.0|            2.0|           24.0|    2.0|    3.0|       3758.0|    3.0|       1.0|               1.0|            2.0|       1.0|          4.0|   4.0|23.0|             3.0|    1.0| 
  1.0|1.0|       1.0|      1.0|          1.0|
|        1.0|            1.0|           11.0|    4.0|    0.0|       3905.0|    1.0|       3.0|               2.0|            3.0|       1.0|          2.0|   1.0|36.0|             3.0|    1.0| 
  2.0|3.0|       2.0|      1.0|          1.0|
|        1.0|            1.0|           30.0|    4.0|    1.0|       6187.0|    2.0|       4.0|               1.0|            4.0|       1.0|          4.0|   3.0|24.0|             3.0|    1.0| 
  2.0|3.0|       1.0|      1.0|          1.0|
|        1.0|            1.0|            6.0|    4.0|    3.0|       1957.0|    1.0|       4.0|               1.0|            2.0|       1.0|          4.0|   3.0|31.0|             3.0|    2.0| 
  1.0|3.0|       1.0|      1.0|          1.0|
|        1.0|            2.0|           48.0|    3.0|   10.0|       7582.0|    2.0|       1.0|               2.0|            3.0|       1.0|          4.0|   4.0|31.0|             3.0|    2.0| 
  1.0|4.0|       1.0|      2.0|          1.0|
|        1.0|            1.0|           18.0|    2.0|    3.0|       1936.0|    5.0|       4.0|               2.0|            4.0|       1.0|          4.0|   3.0|23.0|             3.0|    1.0| 
  2.0|2.0|       1.0|      1.0|          1.0|
|        1.0|            1.0|            6.0|    2.0|    3.0|       2647.0|    3.0|       3.0|               2.0|            3.0|       1.0|          3.0|   1.0|44.0|             3.0|    1.0| 
  1.0|3.0|       2.0|      1.0|          1.0|
|        1.0|            1.0|           11.0|    4.0|    0.0|       3939.0|    1.0|       3.0|               1.0|            3.0|       1.0|          2.0|   1.0|40.0|             3.0|    2.0| 
  2.0|2.0|       2.0|      1.0|          1.0|
|        1.0|            2.0|           18.0|    2.0|    3.0|       3213.0|    3.0|       2.0|               1.0|            4.0|       1.0|          3.0|   1.0|25.0|             3.0|    1.0| 
  1.0|3.0|       1.0|      1.0|          1.0|
|        1.0|            2.0|           36.0|    4.0|    3.0|       2337.0|    1.0|       5.0|               4.0|            3.0|       1.0|          4.0|   1.0|36.0|             3.0|    2.0| 
  1.0|3.0|       1.0|      1.0|          1.0|
|        1.0|            4.0|           11.0|    4.0|    0.0|       7228.0|    1.0|       3.0|               1.0|            3.0|       1.0|          4.0|   2.0|39.0|             3.0|    2.0| 
  2.0|2.0|       1.0|      1.0|          1.0|
|        1.0|            1.0|            6.0|    4.0|    0.0|       3676.0|    1.0|       3.0|               1.0|            3.0|       1.0|          3.0|   1.0|37.0|             3.0|    1.0| 
  3.0|3.0|       2.0|      1.0|          1.0|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+---------------+----------+-------------+------+----+----------------+-------+-
-----+---+----------+---------+-------------+
only showing top 20 rows
```

- Grouping some of the predictor attributes and observing how it affects the target variable

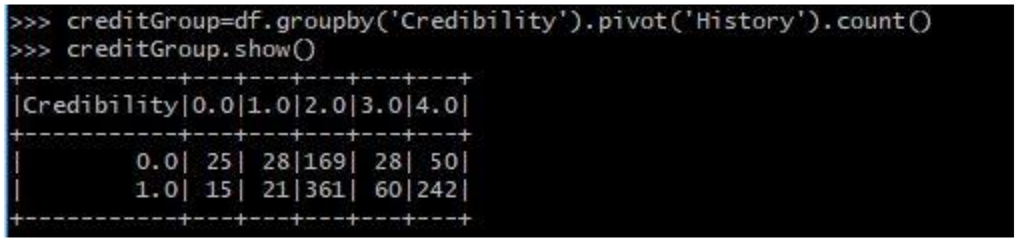
Attributes:



```
>>> featurenames=features.schema.names
>>> featurenames
['Account Balance', 'Credit Duration', 'History', 'Purpose', 'Credit Amount', 'Savings', 'Empl
oyment', 'InstallmentPercent', 'GenderandStatus', 'Guarantors', 'ResidenceTime', 'Assets', 'Ag
e', 'ConcurrentCredit', 'Housing', 'Credit', 'Job', 'Dependents', 'Telephone', 'ForeignWorker'
]
```

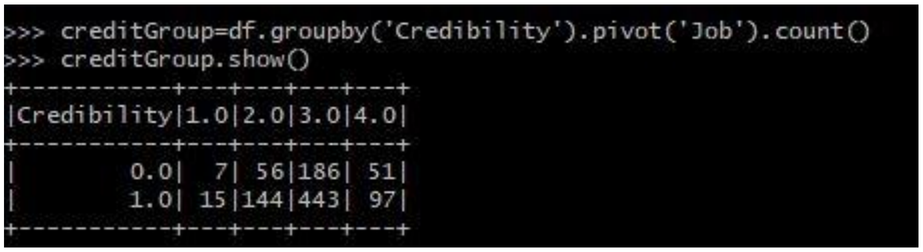
Target variable: 'Credibility' column of data

History Attribute:



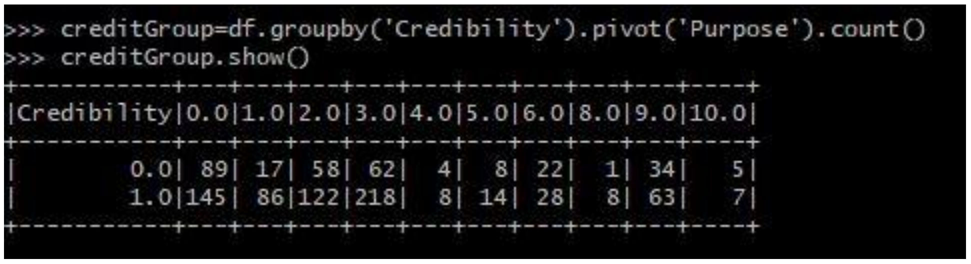
```
>>> creditGroup=df.groupby('Credibility').pivot('History').count()
>>> creditGroup.show()
+-----------+---+---+---+---+---+
|Credibility|0.0|1.0|2.0|3.0|4.0|
+-----------+---+---+---+---+---+
|        0.0| 25| 28|169| 28| 50|
|        1.0| 15| 21|361| 60|242|
+-----------+---+---+---+---+---+
```

Job Attribute:



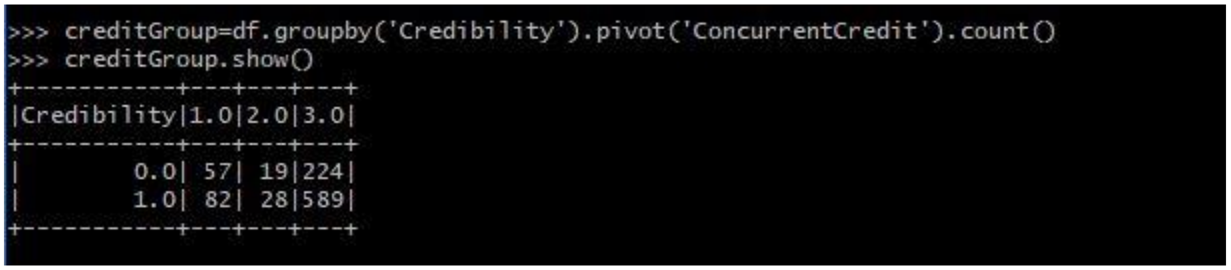
```
>>> creditGroup=df.groupby('Credibility').pivot('Job').count()
>>> creditGroup.show()
+-----------+---+---+---+---+
|Credibility|1.0|2.0|3.0|4.0|
+-----------+---+---+---+---+
|        0.0|  7| 56|186| 51|
|        1.0| 15|144|443| 97|
+-----------+---+---+---+---+
```

Purpose Attribute:



```
>>> creditGroup=df.groupby('Credibility').pivot('Purpose').count()
>>> creditGroup.show()
+-----------+---+---+---+---+---+---+---+---+---+----+
|Credibility|0.0|1.0|2.0|3.0|4.0|5.0|6.0|8.0|9.0|10.0|
+-----------+---+---+---+---+---+---+---+---+---+----+
|        0.0| 89| 17| 58| 62|  4|  8| 22|  1| 34|   5|
|        1.0|145| 86|122|218|  8| 14| 28|  8| 63|   7|
+-----------+---+---+---+---+---+---+---+---+---+----+
```

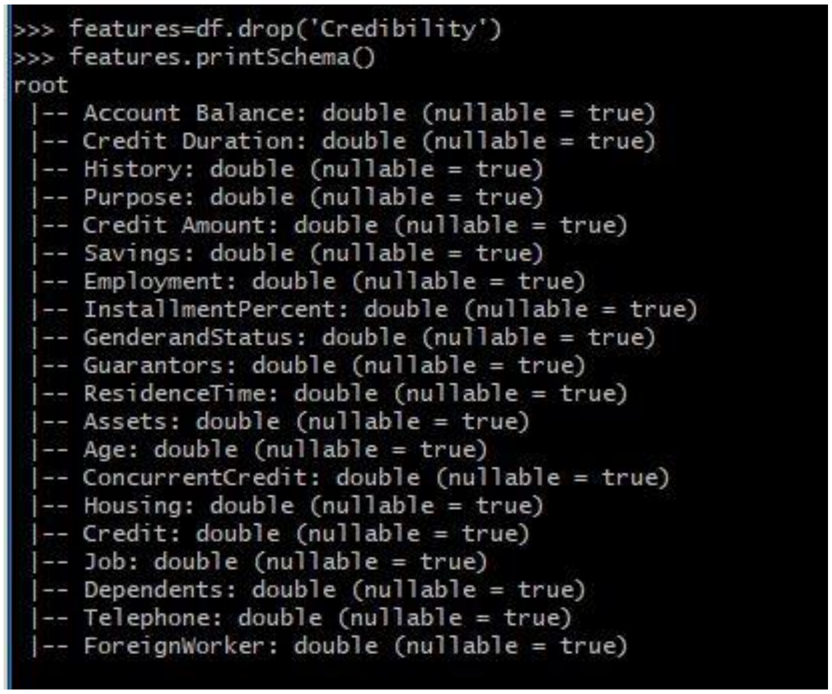
Concurrent Credit Attribute:



```
>>> creditGroup=df.groupby('Credibility').pivot('ConcurrentCredit').count()
>>> creditGroup.show()
+-----------+---+---+---+
|Credibility|1.0|2.0|3.0|
+-----------+---+---+---+
|        0.0| 57| 19|224|
|        1.0| 82| 28|589|
+-----------+---+---+---+
```

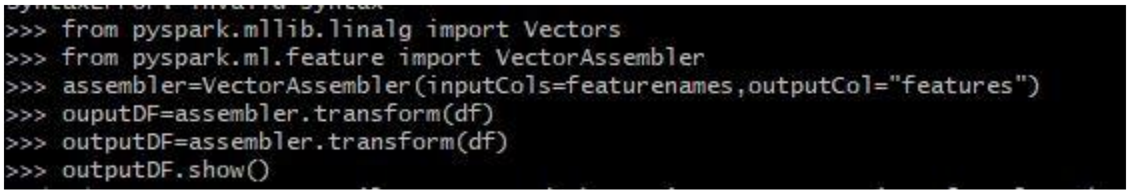
- Now, as we have explored the data and its attributes, we need to extract features from the data to load it into model and make predictions

- So, dropping the 'Credibility' column of data we get the features of the data



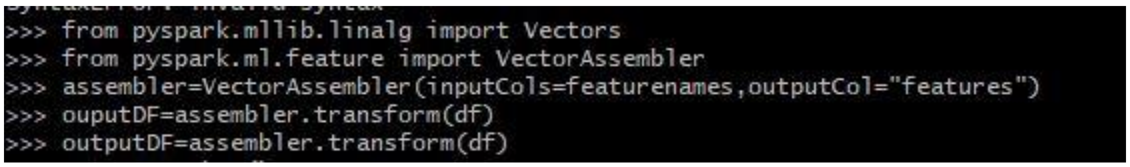
```
>>> features=df.drop('Credibility')
>>> features.printSchema()
root
 |-- Account Balance: double (nullable = true)
 |-- Credit Duration: double (nullable = true)
 |-- History: double (nullable = true)
 |-- Purpose: double (nullable = true)
 |-- Credit Amount: double (nullable = true)
 |-- Savings: double (nullable = true)
 |-- Employment: double (nullable = true)
 |-- InstallmentPercent: double (nullable = true)
 |-- GenderandStatus: double (nullable = true)
 |-- Guarantors: double (nullable = true)
 |-- ResidenceTime: double (nullable = true)
 |-- Assets: double (nullable = true)
 |-- Age: double (nullable = true)
 |-- ConcurrentCredit: double (nullable = true)
 |-- Housing: double (nullable = true)
 |-- Credit: double (nullable = true)
 |-- Job: double (nullable = true)
 |-- Dependents: double (nullable = true)
 |-- Telephone: double (nullable = true)
 |-- ForeignWorker: double (nullable = true)
```

- Vector Assembler: It is a transformer which combines columns provided as inputCols into a single vector and give it as form of outputCol. This single vector is then used to train the models like logistic regression and tree classifier.
- So, now moving forward with using the Vector assembler we obtain all the feature of the data as a single vector by giving the inputCols as features.schema.names and outputCol as "features"

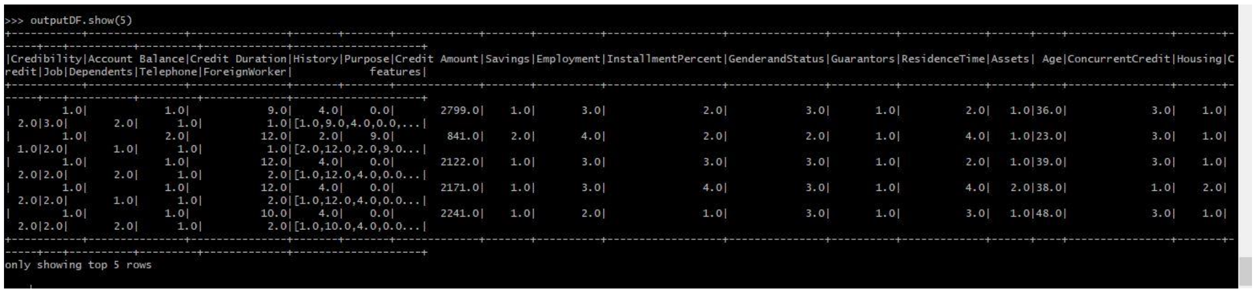


```
>>> from pyspark.mllib.linalg import Vectors
>>> from pyspark.ml.feature import VectorAssembler
>>> assembler=VectorAssembler(inputCols=featurenames,outputCol="features")
>>> ouputDF=assembler.transform(df)
>>> outputDF=assembler.transform(df)
>>> outputDF.show()
```

- Then by using the defined Vector Assembler we transform our original dataframe which is something like this

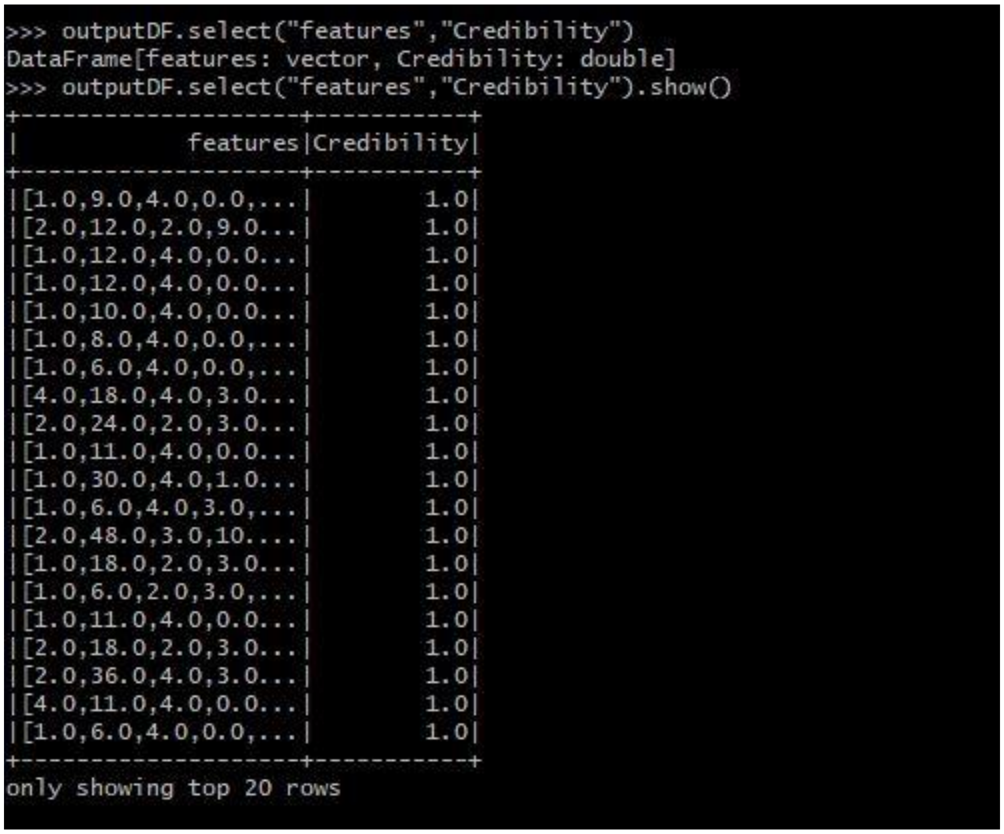


```
>>> from pyspark.mllib.linalg import Vectors
>>> from pyspark.ml.feature import VectorAssembler
>>> assembler=VectorAssembler(inputCols=featurenames,outputCol="features")
>>> ouputDF=assembler.transform(df)
>>> outputDF=assembler.transform(df)
```



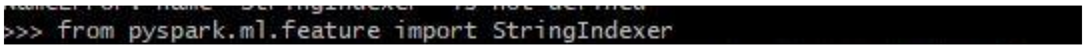
```
>>> outputDF.show(5)
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+---------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+
|Credibility|Account Balance|Credit Duration|History|Purpose|Credit Amount|Savings|Employment|InstallmentPercent|GenderandStatus|Guarantors|ResidenceTime|Assets| Age|ConcurrentCredit|Housing|C
redit|Job|Dependents|Telephone|ForeignWorker|            features|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+---------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+
|        1.0|            1.0|            9.0|    4.0|    0.0|       2799.0|    1.0|       3.0|              2.0|            3.0|       1.0|          2.0|   1.0|36.0|             3.0|    1.0|
  2.0|3.0|       2.0|      1.0|          1.0|[1.0,9.0,4.0,0.0,...|
|        1.0|            2.0|           12.0|    2.0|    9.0|        841.0|    2.0|       4.0|              2.0|            2.0|       1.0|          4.0|   1.0|23.0|             3.0|    1.0|
  1.0|2.0|       1.0|      1.0|          1.0|[2.0,12.0,2.0,9.0...|
|        1.0|            1.0|           12.0|    4.0|    0.0|       2122.0|    1.0|       3.0|              3.0|            3.0|       1.0|          2.0|   1.0|39.0|             3.0|    1.0|
  2.0|2.0|       2.0|      1.0|          2.0|[1.0,12.0,4.0,0.0...|
|        1.0|            1.0|           12.0|    4.0|    0.0|       2171.0|    1.0|       3.0|              4.0|            3.0|       1.0|          4.0|   2.0|38.0|             1.0|    2.0|
  2.0|2.0|       1.0|      1.0|          2.0|[1.0,12.0,4.0,0.0...|
|        1.0|            1.0|           10.0|    4.0|    0.0|       2241.0|    1.0|       2.0|              1.0|            3.0|       1.0|          3.0|   1.0|48.0|             3.0|    1.0|
  2.0|2.0|       2.0|      1.0|          2.0|[1.0,10.0,4.0,0.0...|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+---------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+
only showing top 5 rows
```

The transformed data having only the features and target as columns and the first 20 rows:

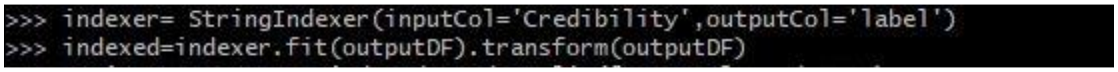


```
>>> outputDF.select("features","Credibility")
DataFrame[features: vector, Credibility: double]
>>> outputDF.select("features","Credibility").show()
+--------------------+-----------+
|            features|Credibility|
+--------------------+-----------+
|[1.0,9.0,4.0,0.0,...|        1.0|
|[2.0,12.0,2.0,9.0...|        1.0|
|[1.0,12.0,4.0,0.0...|        1.0|
|[1.0,12.0,4.0,0.0...|        1.0|
|[1.0,10.0,4.0,0.0...|        1.0|
|[1.0,8.0,4.0,0.0,...|        1.0|
|[1.0,6.0,4.0,0.0,...|        1.0|
|[4.0,18.0,4.0,3.0...|        1.0|
|[2.0,24.0,2.0,3.0...|        1.0|
|[1.0,11.0,4.0,0.0...|        1.0|
|[1.0,30.0,4.0,1.0...|        1.0|
|[1.0,6.0,4.0,3.0,...|        1.0|
|[2.0,48.0,3.0,10....|        1.0|
|[1.0,18.0,2.0,3.0...|        1.0|
|[1.0,6.0,2.0,3.0,...|        1.0|
|[1.0,11.0,4.0,0.0...|        1.0|
|[2.0,18.0,2.0,3.0...|        1.0|
|[2.0,36.0,4.0,3.0...|        1.0|
|[4.0,11.0,4.0,0.0...|        1.0|
|[1.0,6.0,4.0,0.0,...|        1.0|
+--------------------+-----------+
only showing top 20 rows
```

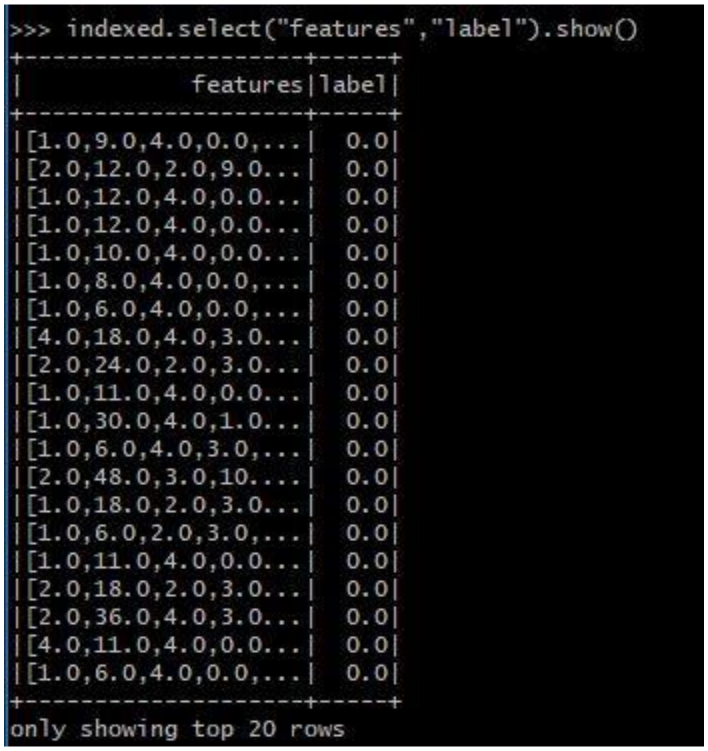
- Now using the StringIndexer we will change the target variable i.e the Credibility column to the label column from the obtained data after the vector assembler



```
>>> from pyspark.ml.feature import StringIndexer
```



```
>>> indexer= StringIndexer(inputCol='Credibility',outputCol='label')
>>> indexed=indexer.fit(outputDF).transform(outputDF)
```

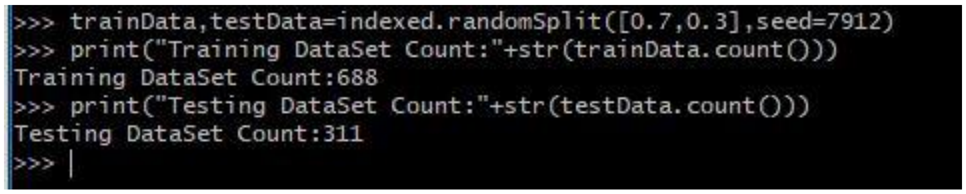


```
>>> indexed.select("features","label").show()
+--------------------+-----+
|            features|label|
+--------------------+-----+
|[1.0,9.0,4.0,0.0,...|  0.0|
|[2.0,12.0,2.0,9.0...|  0.0|
|[1.0,12.0,4.0,0.0...|  0.0|
|[1.0,12.0,4.0,0.0...|  0.0|
|[1.0,10.0,4.0,0.0...|  0.0|
|[1.0,8.0,4.0,0.0,...|  0.0|
|[1.0,6.0,4.0,0.0,...|  0.0|
|[4.0,18.0,4.0,3.0...|  0.0|
|[2.0,24.0,2.0,3.0...|  0.0|
|[1.0,11.0,4.0,0.0...|  0.0|
|[1.0,30.0,4.0,1.0...|  0.0|
|[1.0,6.0,4.0,3.0,...|  0.0|
|[2.0,48.0,3.0,10....|  0.0|
|[1.0,18.0,2.0,3.0...|  0.0|
|[1.0,6.0,2.0,3.0,...|  0.0|
|[1.0,11.0,4.0,0.0...|  0.0|
|[2.0,18.0,2.0,3.0...|  0.0|
|[2.0,36.0,4.0,3.0...|  0.0|
|[4.0,11.0,4.0,0.0...|  0.0|
|[1.0,6.0,4.0,0.0,...|  0.0|
+--------------------+-----+
only showing top 20 rows
```

- Next the data which obtained from this will be used for the splitting into train and test data

<u>Splitting the Data:</u>

- The data obtained after the preprocessing will now be used in splitting and then will be used to train the models and make predictions
- randomSplit with seed is used to split the data in training and testing as follows:



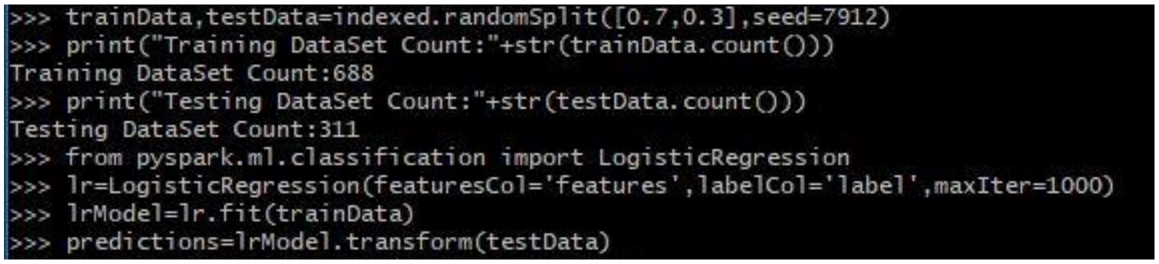
```
>>> trainData,testData=indexed.randomSplit([0.7,0.3],seed=7912)
>>> print("Training DataSet Count:"+str(trainData.count()))
Training DataSet Count:688
>>> print("Testing DataSet Count:"+str(testData.count()))
Testing DataSet Count:311
>>> |
```

- Now this data will be used in training the model and then making predictions

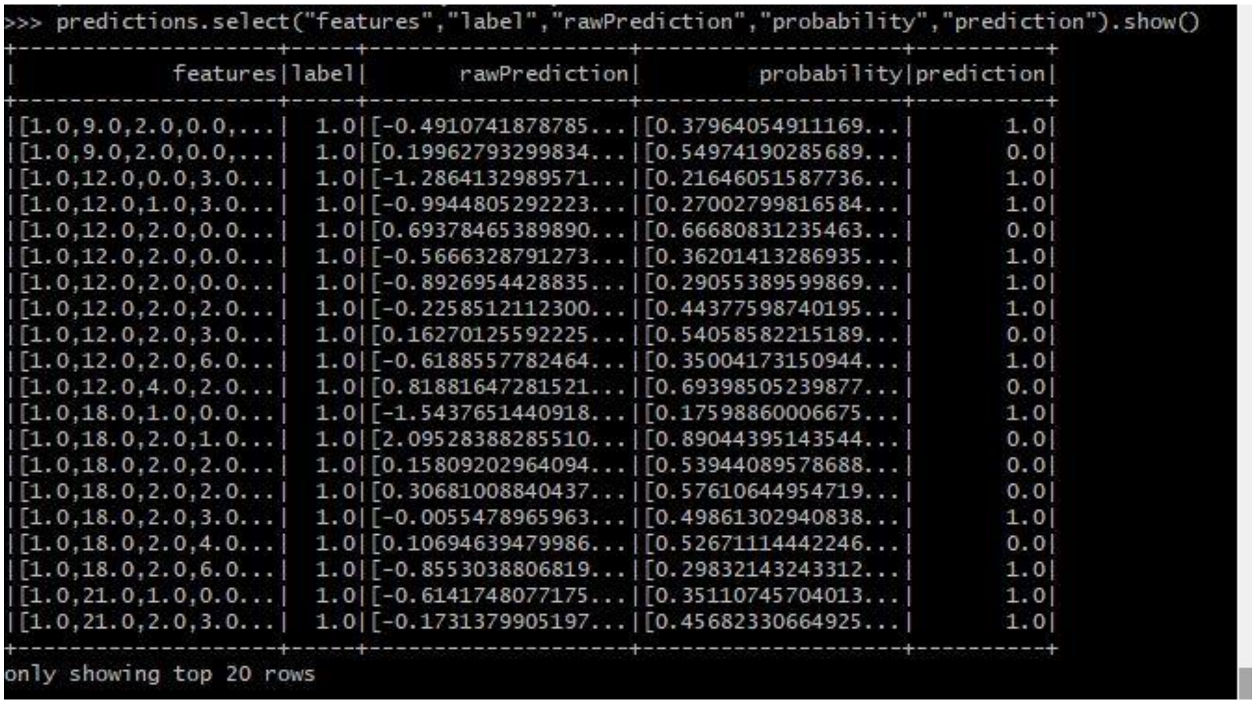
Model Performance

➢ **<u>Logistic Regression:</u>**

- This classifier is used when we want to work on the binary outcome and provide efficient results.
- Logistic Regression is used when we need to describe data and to explain the relationship between one dependent variable to another
- Here basically binomial logistic regression is being used because the target variable categorical outputs of two levels and for more than two levels multinomial regression is being used which is not required here
- In logistic regression, estimation is done by choosing parameters which maximizes the likelihood of observing sample observations



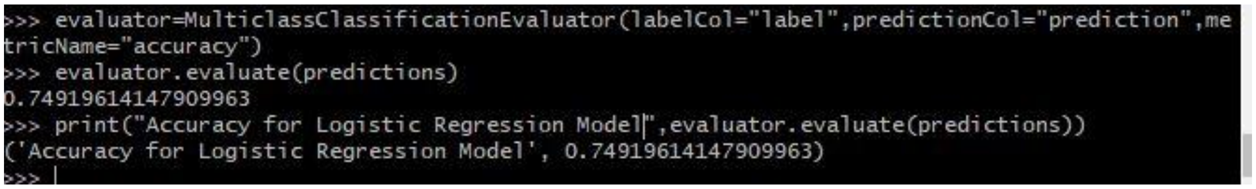
```
>>> trainData,testData=indexed.randomSplit([0.7,0.3],seed=7912)
>>> print("Training DataSet Count:"+str(trainData.count()))
Training DataSet Count:688
>>> print("Testing DataSet Count:"+str(testData.count()))
Testing DataSet Count:311
>>> from pyspark.ml.classification import LogisticRegression
>>> lr=LogisticRegression(featuresCol='features',labelCol='label',maxIter=1000)
>>> lrModel=lr.fit(trainData)
>>> predictions=lrModel.transform(testData)
```



```
>>> predictions.select("features","label","rawPrediction","probability","prediction").show()
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[1.0,9.0,2.0,0.0,...|  1.0|[-0.4910741878785...|[0.37964054911169...|       1.0|
|[1.0,9.0,2.0,0.0,...|  1.0|[0.19962793299834...|[0.54974190285689...|       0.0|
|[1.0,12.0,0.0,3.0...|  1.0|[-1.2864132989571...|[0.21646051587736...|       1.0|
|[1.0,12.0,1.0,3.0...|  1.0|[-0.9944805292223...|[0.27002799816584...|       1.0|
|[1.0,12.0,2.0,0.0...|  1.0|[0.69378465389890...|[0.66680831235463...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[-0.5666328791273...|[0.36201413286935...|       1.0|
|[1.0,12.0,2.0,0.0...|  1.0|[-0.8926954428835...|[0.29055389599869...|       1.0|
|[1.0,12.0,2.0,2.0...|  1.0|[-0.2258512112300...|[0.44377598740195...|       1.0|
|[1.0,12.0,2.0,3.0...|  1.0|[0.16270125592225...|[0.54058582215189...|       0.0|
|[1.0,12.0,2.0,6.0...|  1.0|[-0.6188557782464...|[0.35004173150944...|       1.0|
|[1.0,12.0,4.0,2.0...|  1.0|[0.81881647281521...|[0.69398505239877...|       0.0|
|[1.0,18.0,1.0,0.0...|  1.0|[-1.5437651440918...|[0.17598860006675...|       1.0|
|[1.0,18.0,2.0,1.0...|  1.0|[2.09528388285510...|[0.89044395143544...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[0.15809202964094...|[0.53944089578688...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[0.30681008840437...|[0.57610644954719...|       0.0|
|[1.0,18.0,2.0,3.0...|  1.0|[-0.0055478965963...|[0.49861302940838...|       1.0|
|[1.0,18.0,2.0,4.0...|  1.0|[0.10694639479986...|[0.52671114442246...|       0.0|
|[1.0,18.0,2.0,6.0...|  1.0|[-0.8553038806819...|[0.29832143243312...|       1.0|
|[1.0,21.0,1.0,0.0...|  1.0|[-0.6141748077175...|[0.35110745704013...|       1.0|
|[1.0,21.0,2.0,3.0...|  1.0|[-0.1731379905197...|[0.45682330664925...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

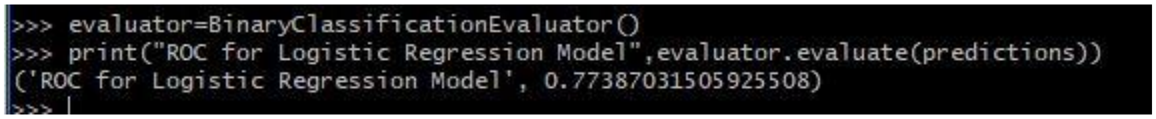
- As seen in figure above, import the LogisticRegression from the pyspark.ml.classification and then set the parameters featuresCol as “features” column of obtained data and labelCol as “label” with maximum of iterations as 1000

- Now as the binary classification evaluator does not support the metric accuracy so we will be using the MulticlassClassificationEvaluator() to get accuracy metric having the parameter tuned to labelCol as "label" and predictionCol="prediction" which can be shown as follows



```
>>> evaluator=MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction",me
tricName="accuracy")
>>> evaluator.evaluate(predictions)
0.74919614147909963
>>> print("Accuracy for Logistic Regression Model",evaluator.evaluate(predictions))
('Accuracy for Logistic Regression Model', 0.74919614147909963)
>>> |
```

- We are able to get a good accuracy using this model which is around 0.749196 or 74.92%
- Then using the BinaryClassificationEvaluator(), we get the Area Under Curve/ROC for the model and we seem to get some good performance as we are getting 0.77387 or 77.4% metric

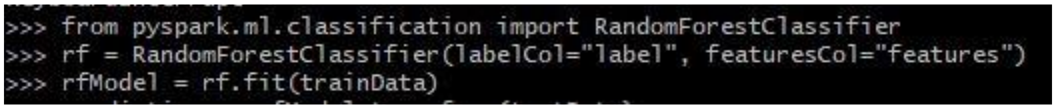


```
>>> evaluator=BinaryClassificationEvaluator()
>>> print("ROC for Logistic Regression Model",evaluator.evaluate(predictions))
('ROC for Logistic Regression Model', 0.77387031505925508)
>>> |
```

- Thus, Logistic Regression is providing us with some efficient results in analysis for this dataset

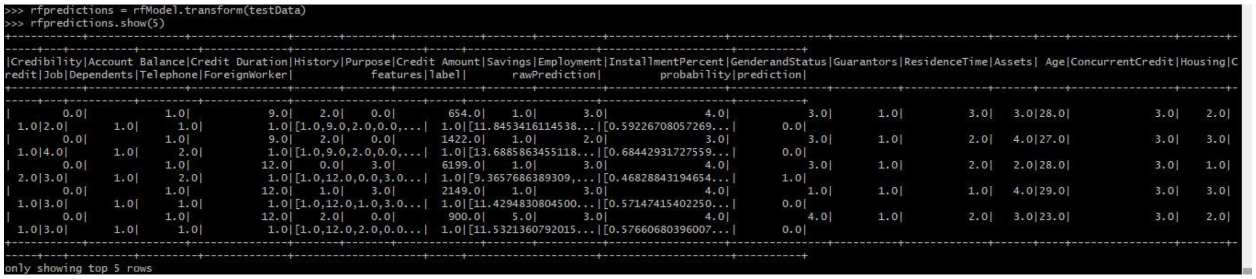
➢ Random Forest Classifier:

- This classifier includes combination of various decision trees and the output of these trees is mode of all the predicted classes or mean prediction
- Basically when we used only decision tree with very deep levels they tend to overfit the data and thus we then implement random forest which are the way of averaging many deep level decision trees trained on different parts of same training set so that we can help in reducing the variance.
- This helps in improving the performance of the final model

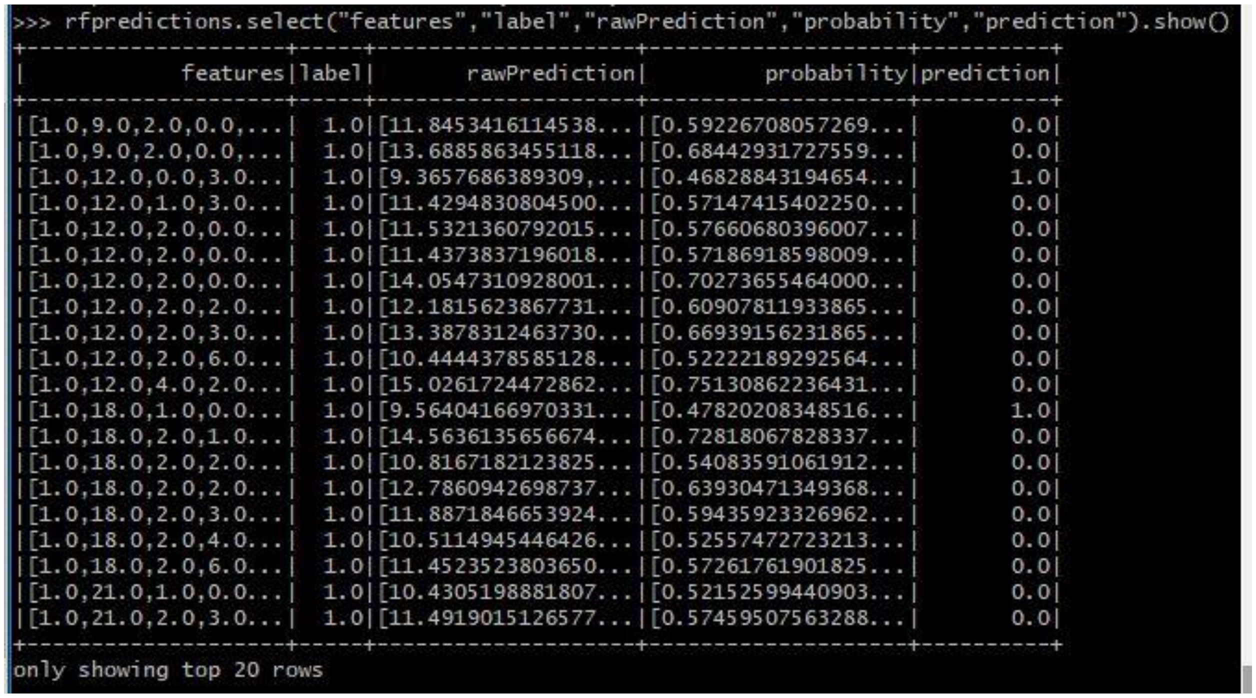


```
>>> from pyspark.ml.classification import RandomForestClassifier
>>> rf = RandomForestClassifier(labelCol="label", featuresCol="features")
>>> rfModel = rf.fit(trainData)
```

- As it can be seen in the figure above, RandomForestClassifier() is being imported from the pyspark.ml.classification package and then it is being defined with the parameters
- For this classifier, similar to Logistic Regression the labelCol is set to "label" and the featuresCol is set to "features" and then the model is used to fit the training data
- Now testing the model using the test data obtained

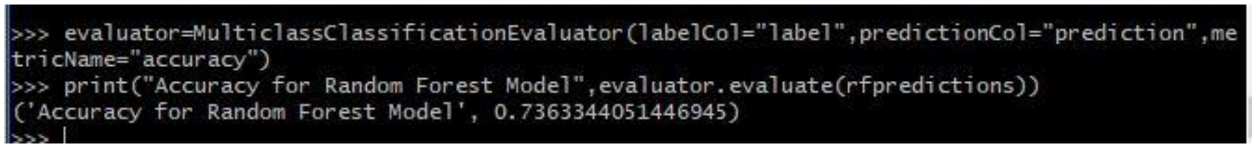


```
>>> rfpredictions = rfModel.transform(testData)
>>> rfpredictions.show(5)
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
|Credibility|Account Balance|Credit Duration|History|Purpose|Credit Amount|Savings|Employment|InstallmentPercent|GenderandStatus|Guarantors|ResidenceTime|Assets| Age|ConcurrentCredit|Housing|C
redit|Job|Dependents|Telephone|ForeignWorker|            features|label|       rawPrediction|         probability|prediction|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
|        0.0|            1.0|            9.0|    2.0|    0.0|        654.0|    1.0|       3.0|              4.0|             3.0|       1.0|          3.0|   3.0|28.0|             3.0|    2.0|
  1.0|2.0|       1.0|      1.0|          1.0|[1.0,9.0,2.0,0.0,...|  1.0|[11.8453416114538...|[0.59226708057269...|       0.0|
|        0.0|            1.0|            9.0|    2.0|    0.0|       1422.0|    1.0|       2.0|              3.0|             3.0|       1.0|          2.0|   4.0|27.0|             3.0|    3.0|
  1.0|4.0|       1.0|      2.0|          1.0|[1.0,9.0,2.0,0.0,...|  1.0|[13.6885863455118...|[0.68442931727559...|       0.0|
|        0.0|            1.0|           12.0|    0.0|    3.0|       6199.0|    1.0|       3.0|              4.0|             3.0|       1.0|          2.0|   2.0|28.0|             3.0|    1.0|
  2.0|3.0|       1.0|      2.0|          1.0|[1.0,12.0,0.0,3.0...|  1.0|[9.3657686389309,...|[0.46828843194654...|       1.0|
|        0.0|            1.0|           12.0|    1.0|    3.0|       2149.0|    1.0|       3.0|              4.0|             1.0|       1.0|          1.0|   4.0|29.0|             3.0|    3.0|
  1.0|3.0|       1.0|      1.0|          1.0|[1.0,12.0,1.0,3.0...|  1.0|[11.4294830804500...|[0.57147415402250...|       0.0|
|        0.0|            1.0|           12.0|    2.0|    0.0|        900.0|    5.0|       3.0|              4.0|             4.0|       1.0|          2.0|   3.0|23.0|             3.0|    2.0|
  1.0|3.0|       1.0|      1.0|          1.0|[1.0,12.0,2.0,0.0...|  1.0|[11.5321360792015...|[0.57660680396007...|       0.0|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+-----------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
only showing top 5 rows
```

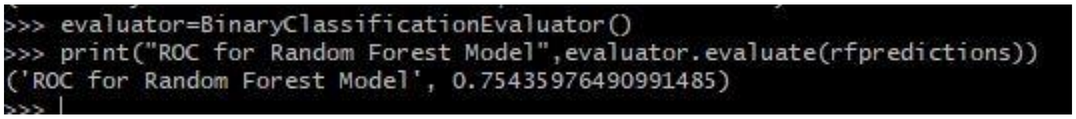


```
>>> rfpredictions.select("features","label","rawPrediction","probability","prediction").show()
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[1.0,9.0,2.0,0.0,...|  1.0|[11.8453416114538...|[0.59226708057269...|       0.0|
|[1.0,9.0,2.0,0.0,...|  1.0|[13.6885863455118...|[0.68442931727559...|       0.0|
|[1.0,12.0,0.0,3.0...|  1.0|[9.3657686389309,...|[0.46828843194654...|       1.0|
|[1.0,12.0,1.0,3.0...|  1.0|[11.4294830804500...|[0.57147415402250...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[11.5321360792015...|[0.57660680396007...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[11.4373837196018...|[0.57186918598009...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[14.0547310928001...|[0.70273655464000...|       0.0|
|[1.0,12.0,2.0,2.0...|  1.0|[12.1815623867731...|[0.60907811933865...|       0.0|
|[1.0,12.0,2.0,3.0...|  1.0|[13.3878312463730...|[0.66939156231865...|       0.0|
|[1.0,12.0,2.0,6.0...|  1.0|[10.4444378585128...|[0.52222189292564...|       0.0|
|[1.0,12.0,4.0,2.0...|  1.0|[15.0261724472862...|[0.75130862236431...|       0.0|
|[1.0,18.0,1.0,0.0...|  1.0|[9.56404166970331...|[0.47820208348516...|       1.0|
|[1.0,18.0,2.0,1.0...|  1.0|[14.5636135656674...|[0.72818067828337...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[10.8167182123825...|[0.54083591061912...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[12.7860942698737...|[0.63930471349368...|       0.0|
|[1.0,18.0,2.0,3.0...|  1.0|[11.8871846653924...|[0.59435923326962...|       0.0|
|[1.0,18.0,2.0,4.0...|  1.0|[10.5114945446426...|[0.52557472723213...|       0.0|
|[1.0,18.0,2.0,6.0...|  1.0|[11.4523523803650...|[0.57261761901825...|       0.0|
|[1.0,21.0,1.0,0.0...|  1.0|[10.4305198881807...|[0.52152599440903...|       0.0|
|[1.0,21.0,2.0,3.0...|  1.0|[11.4919015126577...|[0.57459507563288...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

- So, evaluating this model using the MulticlassClassificationEvaluator() and BinaryClassificationEvaluator() to obtain the Accuracy and Area Under Curve/ROC for the model



```
>>> evaluator=MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction",me
tricName="accuracy")
>>> print("Accuracy for Random Forest Model",evaluator.evaluate(rfpredictions))
('Accuracy for Random Forest Model', 0.7363344051446945)
>>> |
```

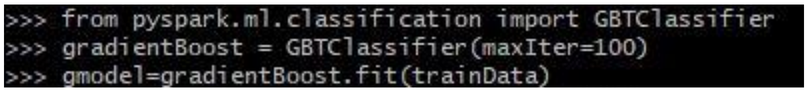


```
>>> evaluator=BinaryClassificationEvaluator()
>>> print("ROC for Random Forest Model",evaluator.evaluate(rfpredictions))
('ROC for Random Forest Model', 0.7543597649099148 5)
>>> |
```

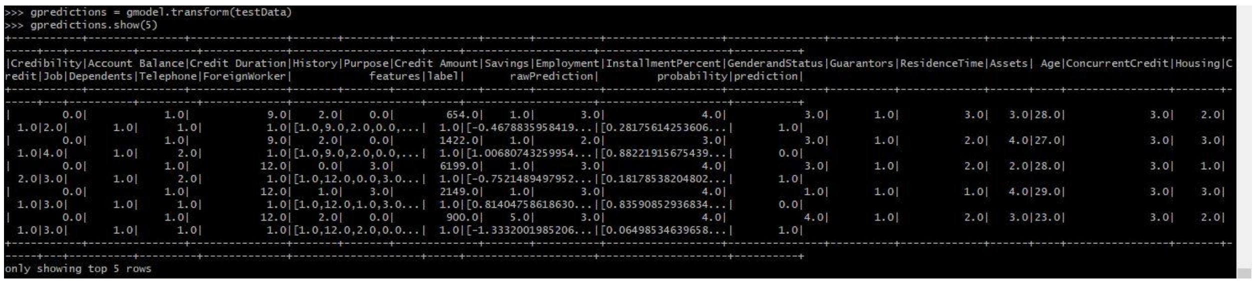
- The Accuracy and ROC for Random Forest model is around 0.73633 or 73.63% and 0.75436 or 75.44% respectively which is somewhat less from what we got in Logistic Regression model

➢ **Gradient Boosted Tree Model:**

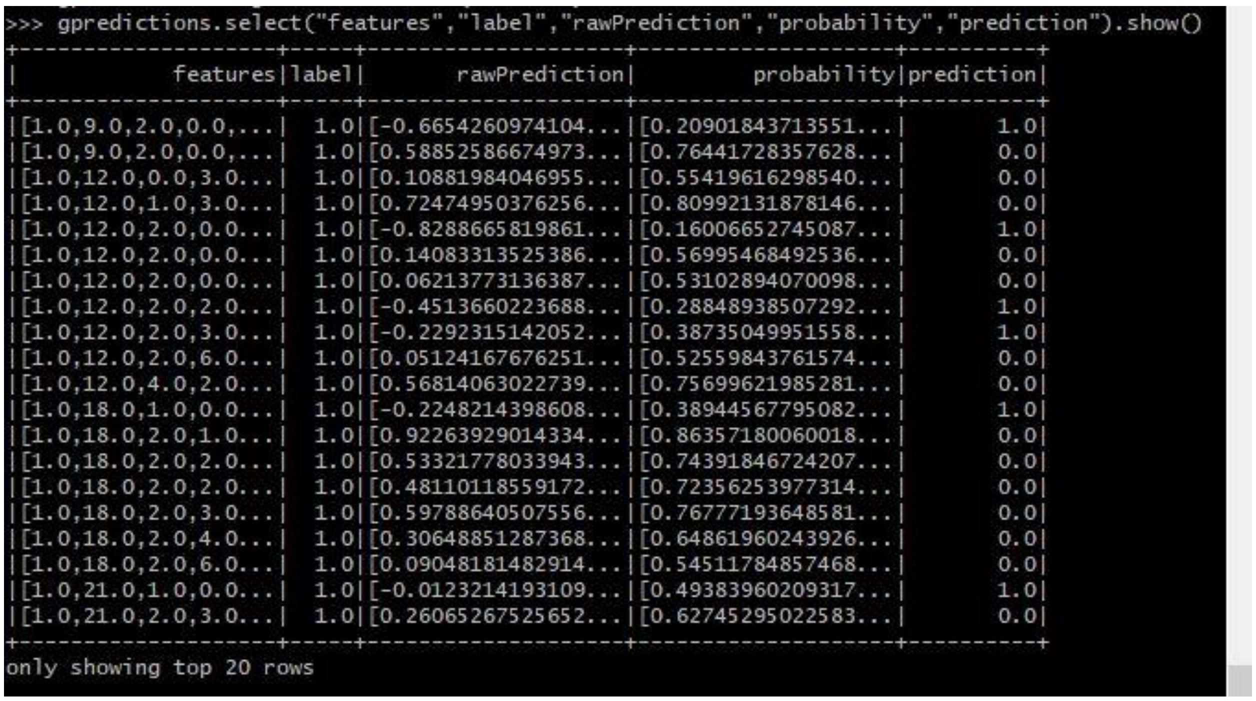
- This model uses decision trees but of a fixed equal size for the base iteration and then going forward it provides weights to change the prediction in every iteration
- Now, we will be using the GBTClassifier imported from pyspark.ml.classification package and is implemented with maximum of 100 iterations



```
>>> from pyspark.ml.classification import GBTClassifier
>>> gradientBoost = GBTClassifier(maxIter=100)
>>> gmodel=gradientBoost.fit(trainData)
```

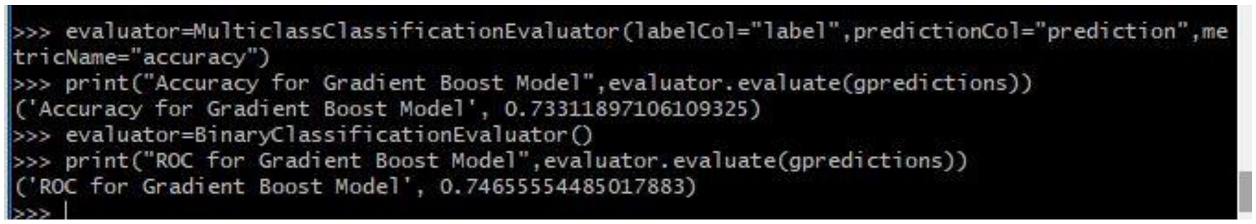


```
>>> gpredictions = gmodel.transform(testData)
>>> gpredictions.show(5)
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
|Credibility|Account Balance|Credit Duration|History|Purpose|Credit Amount|Savings|Employment|InstallmentPercent|GenderandStatus|Guarantors|ResidenceTime|Assets| Age|ConcurrentCredit|Housing|C
redit|Job|Dependents|Telephone|ForeignWorker|            features|label|       rawPrediction|         probability|prediction|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
|        0.0|            1.0|            9.0|    2.0|    0.0|        654.0|    1.0|       3.0|               4.0|             3.0|       1.0|          3.0|   3.0|28.0|             3.0|    2.0|
 1.0|2.0|       1.0|      1.0|          1.0|[1.0,9.0,2.0,0.0,...|  1.0|[-0.4678835958419...|[0.28175614253606...|       1.0|
|        0.0|            1.0|            9.0|    2.0|    0.0|       1422.0|    1.0|       2.0|               3.0|             3.0|       1.0|          2.0|   4.0|27.0|             3.0|    3.0|
 1.0|4.0|       1.0|      2.0|          1.0|[1.0,9.0,2.0,0.0,...|  1.0|[1.00680743259954...|[0.88221915675439...|       0.0|
|        0.0|            1.0|           12.0|    0.0|    3.0|       6199.0|    1.0|       3.0|               4.0|             3.0|       1.0|          2.0|   2.0|28.0|             3.0|    1.0|
 2.0|3.0|       1.0|      2.0|          1.0|[1.0,12.0,0.0,3.0...|  1.0|[-0.7521489497952...|[0.18178538204802...|       1.0|
|        0.0|            1.0|           12.0|    1.0|    3.0|       2149.0|    1.0|       3.0|               4.0|             1.0|       1.0|          1.0|   4.0|29.0|             3.0|    3.0|
 1.0|3.0|       1.0|      1.0|          1.0|[1.0,12.0,1.0,3.0...|  1.0|[0.81404758618630...|[0.83590852936834...|       0.0|
|        0.0|            1.0|           12.0|    2.0|    0.0|        900.0|    5.0|       3.0|               4.0|             4.0|       1.0|          2.0|   3.0|23.0|             3.0|    2.0|
 1.0|3.0|       1.0|      1.0|          1.0|[1.0,12.0,2.0,0.0...|  1.0|[-1.3332001985206...|[0.06498534639658...|       1.0|
+-----------+---------------+---------------+-------+-------+-------------+-------+----------+------------------+----------------+----------+-------------+------+----+----------------+-------+-
----+---+----------+---------+-------------+--------------------+-----+--------------------+--------------------+----------+
only showing top 5 rows
```



```
>>> gpredictions.select("features","label","rawPrediction","probability","prediction").show()
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|[1.0,9.0,2.0,0.0,...|  1.0|[-0.6654260974104...|[0.20901843713551...|       1.0|
|[1.0,9.0,2.0,0.0,...|  1.0|[0.58852586674973...|[0.76441728357628...|       0.0|
|[1.0,12.0,0.0,3.0...|  1.0|[0.10881984046955...|[0.55419616298540...|       0.0|
|[1.0,12.0,1.0,3.0...|  1.0|[0.72474950376256...|[0.80992131878146...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[-0.8288665819861...|[0.16006652745087...|       1.0|
|[1.0,12.0,2.0,0.0...|  1.0|[0.14083313525386...|[0.56995468492536...|       0.0|
|[1.0,12.0,2.0,0.0...|  1.0|[0.06213773136387...|[0.53102894070098...|       0.0|
|[1.0,12.0,2.0,2.0...|  1.0|[-0.4513660223688...|[0.28848938507292...|       1.0|
|[1.0,12.0,2.0,3.0...|  1.0|[-0.2292315142052...|[0.38735049951558...|       1.0|
|[1.0,12.0,2.0,6.0...|  1.0|[0.05124167676251...|[0.52559843761574...|       0.0|
|[1.0,12.0,4.0,2.0...|  1.0|[0.56814063022739...|[0.75699621985281...|       0.0|
|[1.0,18.0,1.0,0.0...|  1.0|[-0.2248214398608...|[0.38944567795082...|       1.0|
|[1.0,18.0,2.0,1.0...|  1.0|[0.92263929014334...|[0.86357180060018...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[0.53321778033943...|[0.74391846724207...|       0.0|
|[1.0,18.0,2.0,2.0...|  1.0|[0.48110118559172...|[0.72356253977314...|       0.0|
|[1.0,18.0,2.0,3.0...|  1.0|[0.59788640507556...|[0.76777193648581...|       0.0|
|[1.0,18.0,2.0,4.0...|  1.0|[0.30648851287368...|[0.64861960243926...|       0.0|
|[1.0,18.0,2.0,6.0...|  1.0|[0.09048181482914...|[0.54511784857468...|       0.0|
|[1.0,21.0,1.0,0.0...|  1.0|[-0.0123214193109...|[0.49383960209317...|       1.0|
|[1.0,21.0,2.0,3.0...|  1.0|[0.26065267525652...|[0.62745295022583...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```

- So, evaluating this model using MulticlassClassificationEvaluator() and BinaryClassificationEvaluator() to obtain the Accuracy and Area Under Curve/ROC of Gradient Boosted Tree model



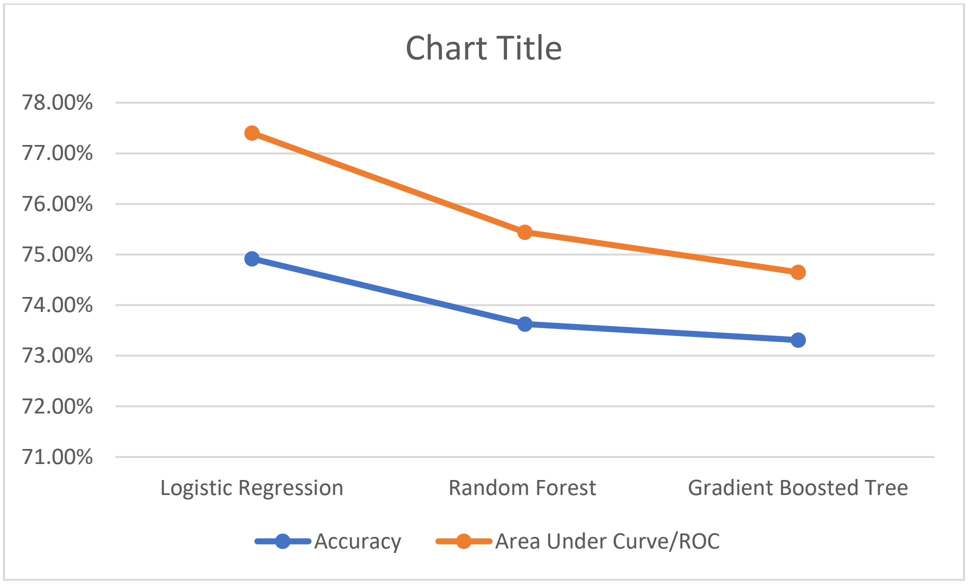
```
>>> evaluator=MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction",me
tricName="accuracy")
>>> print("Accuracy for Gradient Boost Model",evaluator.evaluate(gpredictions))
('Accuracy for Gradient Boost Model', 0.73311897106109325)
>>> evaluator=BinaryClassificationEvaluator()
>>> print("ROC for Gradient Boost Model",evaluator.evaluate(gpredictions))
('ROC for Gradient Boost Model', 0.74655554485017883)
>>> |
```

- Here, also it observed that the Accuracy and Area Under Curve/ROC is around 0.73312 or 73.31% and 0.74655 or 74.65% respectively is less than the Logistic Regression and near to Random Forest
- This is because like Random Forest, Gradient Boosted Tree also uses the combinations of Decision Trees just to reduce the chances of overfitting, but the gradient boosted model does apply weights on each iteration starting with equal weights and then classifying the weights on the basis of classification which is not observed in random forest model

➢ <u>Comparison of Models:</u>

Models	Accuracy	Area Under Curve/ROC
Logistic Regression	0.749196 or 74.92%	0.77387 or 77.4%
Random Forest	0.73633 or 73.63%	0.75436 or 75.44%
Gradient Boosted Tree	0.73312 or 73.31%	0.74655 or 74.65%

Plotting the metrics of the models



Alternatives

- **Spark RDD to Spark CSV Module**

I have used an alternative to Spark RDD for loading the csv file to sandbox which is Spark CSV Module which makes it much easier and time efficient for the users of Microsoft Azure platforms. By using the Spark csv module, the functionality found in the core Spark it becomes easy to load data to DataFrame as many steps are reduced. It is true thing that loading data directly from csv takes less time than loading data through RDD. So, to work efficiently I have used spark_csv module

- **MLlib with TensorFlow**

Recently Yahoo open sourced a project TensorFlowOnSpark, which is amalgamation of Spark and TensorFlow helpful for implementing deep learning frameworks specially for the ones who are in need to run on large clusters.

Now, Spark provides its own library and packages for Machine Learning named MLlib which uses in-memory computation and reduces the overhead of disk read and write. Spark MLlib has made it an easy job for many companies to work with big data efficiently, but we still face some problems for deep learning frameworks implementation. This is where Tensorflow comes in picture to provide improvements in the performance of numerical computations and neural networks and helps people in making deep learning networks with ease unlike MLlib library.

With this project, we do not need to make neural network or have a huge amount of data to process which makes it easy for us to work with MLlib package, but if there is huge data to run on the cluster, we should consider using TensorFlow.

Using TensorFlow package for this project would not be that beneficial because it will generate same type of results. But, TensorFlowOnSpark will surely come in help when we require to build neural networks and deep learning models which is currently not required for this project. Thus, using MLlib package of Spark is much better choice for this project and this specific dataset

Conclusion

- The objective of the project was to predict the credit risk and try to minimize the risk to maximize the profit of the financial institution and bank. This will help bank managers and officials to take an informed decision.

- With this project, I explored with different models for predictive analysis such as Logistic Regression, Random Forest Classifier and Gradient Boosting Tree Model.

- Being a classification-based dataset (given labeled data with binary target variable), we will not be working with supervised models used for continuous variables like Linear Regression and unsupervised models like clustering. It will not provide efficient results for this dataset for predictions.

- After preprocessing the data and exploring all the classification-based models, Logistic Regression comes out to be the best model in terms of Accuracy and Area Under Curve/ROC metrics.

- Looking to the other alternatives of MLlib package, which is TensorFlowOnSpark, we can conclude that for this project working with MLlib would be a better choice.

- Thus, I conclude that for further predictive analysis for this data and helping the bank manager we should work with Logistic Regression Model based on the utilized platform and methods.

References

- http://www.oracle.com/us/technologies/big-data/big-data-in-financial-services-wp-2415760.pdf
- https://en.wikipedia.org/wiki/Apache_Spark
- https://spark.apache.org/docs/2.2.0/sql-programming-guide.html
- https://datascience.stackexchange.com/questions/13123/import-csv-file-contents-into-pyspark-dataframes
- https://www.analyticsindiamag.com/tensorflow-vs-spark-differ-work-tandem/
- https://spark.apache.org/docs/2.1.1/mllib-linear-methods.html
- https://spark.apache.org/docs/latest/mllib-ensembles.html
- https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PySpark_SQL_Cheat_Sheet_Python.pdf
- https://mindfulmachines.io/blog/2018/4/3/spark-rdds-and-dataframes
- https://spark.apache.org/docs/2.3.0/sql-programming-guide.html