A picture containing text, container, box

Description automatically generated

Submitted by

Name: B. Tarun Kumar

Name: G. Jay Prakash Narayana

Name: K. Satwik

Roll. No: 19131A1213

Roll. No: 19131A1231

Roll. No: 19131A1248

**CREDIT CARD FRAUD DETECTION**

1. **INTRODUCTION**

Online shopping is growing day to day .Credit cards are used for purchasing goods and services with the help of virtual card and physical card whereas virtual card for online transaction and physical card for offline transaction. In a physical-card based purchase, the cardholder presents his card physically to a merchant for making a payment. To carry out fraudulent transactions in this kind of purchase, an attacker has to steal the credit card. If the cardholder does not realize the loss of the card, it can lead to a substantial financial loss to the credit card company.

In online payment mode, attackers need only little information for doing fraudulent transactions (secure code, card number, expiration date etc.). In this purchase method, mainly transactions will be done through the Internet or telephone. To commit fraud in these types of purchases, a fraudster simply needs to know the card details. Most of the time, the genuine cardholder is information about the typical purchase category, the time since the last purchase, the amount of money spent, etc. Deviation from such patterns is a potential threat to the system not aware that someone else has seen or stolen his card information. The only way to detect this kind of fraud is to analyze the spending patterns on every card and to figure out any inconsistency with respect to the “usual” spending patterns. Fraud detection based on the analysis of existing purchase data of cardholders is a promising way to reduce the rate of successful credit card frauds. Since humans tend to exhibit specific behavioristic profiles, every cardholder can be represented by a set of patterns containing information about the typical purchase category, the time since the last purchase, the amount of money spent, etc. Deviation from such patterns is a potential threat to the system.

**EXISTING SYSTEM**

The computational capacity of the existing system is not sufficient due to the increasing amount of online transactions. The size of historical transactions can reach PBs and even EBs as the number of transactions coming into the system reaches million per second. Scalability of the system is also at stake. The Visa Company could only analyze 2% of its historical transactions, and make one update of its detection model every 2 or 3 days.

**PROPOSED SYSTEM**

The main objective of this system is to achieve the challenges of the drawbacks of existing systems like scalability, high response time, efficiency, imbalance of data, false predictions etc. The proposed system uses Machine learning techniques to detect whether a card is genuine (or) not. We are implementing this view in an web page that we can analyze that one card with 5-6 algorithms to view and analyze which algorithm is more efficient for that particular transaction.

**2.LITERATURE REVIEW**

1. Maes, S., Tuyls, K., Vanschoenwinkel, B. and Manderick, B., (2002). Credit card fraud detection using Bayesian and neural networks. Pro- ceeding International NAISO Congress on Neuro Fuzzy Technologies.
2. Singh G., Gupta R., Rastogi A., Chandel M.D.S., Riyaz A.A Machine Learning Approach for Detection of Fraud based on SVM, International Journal of Scientific Engineering and Technology, 1 (3) (2012), pp. 194-198 ISSN: 2277-1581
3. Patil S., Somavanshi H., Gaikwad J., Deshmane A., Badgujar R.Credit Card Fraud Detection Using Decision Tree Induction Algorithm International Journal of Computer Science and Mobile Computing (IJCSMC), 4 (4)(2015), pp. 92-95

ISSN: 2320-088X

1. Haibo He, Yang Bai, E. A. Garcia and Shutao Li, ”ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), Hong Kong, 2008, pp. 1322-1328. doi: 10.1109/IJCNN.2008.4633969
2. Machine Learning For Credit Card Fraud Detection System, Lakshmi S V S , Selvani Deepthi Kavila, november 2018.
3. Credit Card Fraud Detection using Data science and Machine learning, S P Maniraj, Aditya Saini , Shadab Ahmed, Swarna Deep Sarkar, September 2019.
4. A. Mishra, C. Ghorpade, “Credit Card Fraud Detection on the Skewed Data Using Various Classification and Ensemble Techniques” 2018 IEEE International Students' Conference on Electronics ,Electrical and Computer Science (SCEECS) pp. 1-5. IEEE.

**3.PROBLEM SPECIFICATION**

**3.1 Problem Description:**

Talking in terms of e-commerce transactions the major problem faced due to these fraudulent activities is so similar to legal ones. Hence having an efficient and complex fraud detection system is a must to prevent these fraudulent activities. The challenging section of this problem is to detect frauds in a huge dataset where the legal transactions are more and the fraudulent transactions are bare minimum or close to negligible.

**Problems faced by the users by using the existing system:**

* Difficult to predict the number of clusters (K-Value).
* It has a less accurate ratio.
* KNN is ineffective when number of features are quite large
* Difficult to choose which algorithm is suitable for detecting fraud cards using machine learning.

**4. SYSTEM REQUIREMENTS AND SPECIFICATION**

System analysis is the first technical step in the software engineering process. It is at this point that a general statement of software scope is refined into concrete specification that becomes the foundation for all software engineering activities that follow. Analysis must focus on the information, functional and behavioral domain of the problem. To better understand what is required, models are created and the problem is partitioned. In many cases it is not possible to completely specify a problem at an early stage.

**4.1 System Requirement Specification (SRS)**

A software requirement specification is developed as a consequence of analysis. Review is essential to ensure that the developer and customers have the same perception.

Software Requirement Specification is the starting point of the software development activity. The Software Requirement Specification is produced at the culmination of the analysis task. The introduction of the software requirement specification states the goals and objectives of the software, describing it in the context of the computer-based system. The SRS includes an information description, functional description, behavioral description, validation criteria.

The purpose of this document is to present the software requirements in a precise and easily understood manner. This document provides the functional, performance, design and verification requirements of the software to be developed.

**4.2. Functional Requirements**

The functional requirements of the system defines a function of software system or its

components. A function describes a set of inputs, behaviour of a system and output.

The following are the functional requirements of proposed system

**INPUT:**

Credit card Transaction data. **PROCESS:**

Transactions are passed to the proposed model which uses Logistic Regression, Decision Tree,RandomForest,SupportVectorMachine(S.V.M),KNearestNeighbour(K.N.N)and Naïve Bayesian Classifier. The fraud scores obtained by each model are merged into a single score *f(t)* and checked with the threshold values (*θU*-Lower bound of genuine transactions, *θL*-Upper bound of fraud transactions) . If *f(t)*> *θU*then the transaction is genuine and if *f(t)*< *θL*then the transaction is fraud. If *θL* < *f(t)* < *θU* then the transaction is passed to the decision tree classifier for further classification.

**OUTPUT:**

The prediction whether the transaction is genuine or fraud.

**4.3. Non-Functional Requirements**

Non-functional requirements are constraints that must be adhered to during development. They limit what resources can be used and set bounds on aspects of the software’s quality.

One of the most important things about non-functional requirements is to make them verifiable. The verification is normally done by measuring various aspects of the system and seeing if the measurements conform to the requirements. Non-Functional Requirements are divided into several groups. The first group of categories reflects the five qualities attributes

* Usability
* Efficiency
* Reliability
* Maintainability
* Reusability

The second group of non-functional requirements categories constraints the environment and technology of the system

**1. Platform**: It is quite important to make it clear on what hardware and operating system of the software must work on. Normally such requirements specify the least powerful platforms and declare that it must work on anything more recent or more powerful.

**Hardware Requirements**:

* 4GB RAM
* 1.7-2.4 GHz processor speed
* 500 GB HDD

**2. Technology to be used**: While it is wise to give the designers as much flexibility as possible to choose how to implement the system, sometimes constraints must be imposed. Requirements are normally stated to ensure that all systems in an organization use the same technology – this reduces the need to train people in different technologies.

**Software Requirements**:

* Python 3.6
* Anaconda 5.0.1
* Streamlit 0.79.0

The third group of non-functional requirements categories constraint the project plan and development methods

* **Development process (methodology) to be used**: In order to ensure quality, some requirements documents specify that certain processes be followed; for example, particular approaches to testing. The details of the process should not be included in the requirements; instead a reference should be made to other documents that describe the process.
* **Cost and delivery date:** One of the biggest challenges in software engineering is accurately forecasting how much time and effort it will take either to develop a system or to make a specific set of changes. All software developers have to participate in cost estimation

**4.4 Feasibility Study**

Preliminary investigation examines project feasibility; the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical operational and Economical feasibility for adding new modules and debugging old running system. All systems are feasible if they are given unlimited resources and infinite time.

There are aspects in the feasibility study portion of the preliminary investigation.

* Technical Feasibility
* Operational Feasibility
* Economic Feasibility

**4.4.1 Technical Feasibility**

Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements. Technical feasibility also performs the following tasks.

1. Analyzes the technical skills and capabilities of the software development team members
2. Determines whether the relevant technology is stable and established
3. Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

**4.4.2 Operational Feasibility**

Operational feasibility assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed. Operational feasibility also performs the following tasks.

* Determines whether the problems anticipated in user requirements are of high priority
* Determines whether the solution suggested by the software development team is acceptable
* Analyzes whether users will adapt to a new software
* Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

**4.4.3 Economic Feasibility**

Economic feasibility determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software. Software is said to be economically feasible if it focuses on the issues listed below.

* Cost incurred on software development to produce long-term gains for an organization
* Cost required conducting full software investigation (such as requirements elicitation and requirements analysis)
* Cost of hardware, software, development team, and training.

**5. DESIGN**

**5.1. UML DIAGRAMS**

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system. It allows software to be visualised in multiple dimensions, so that a computer system can be completely understood before construction begins.

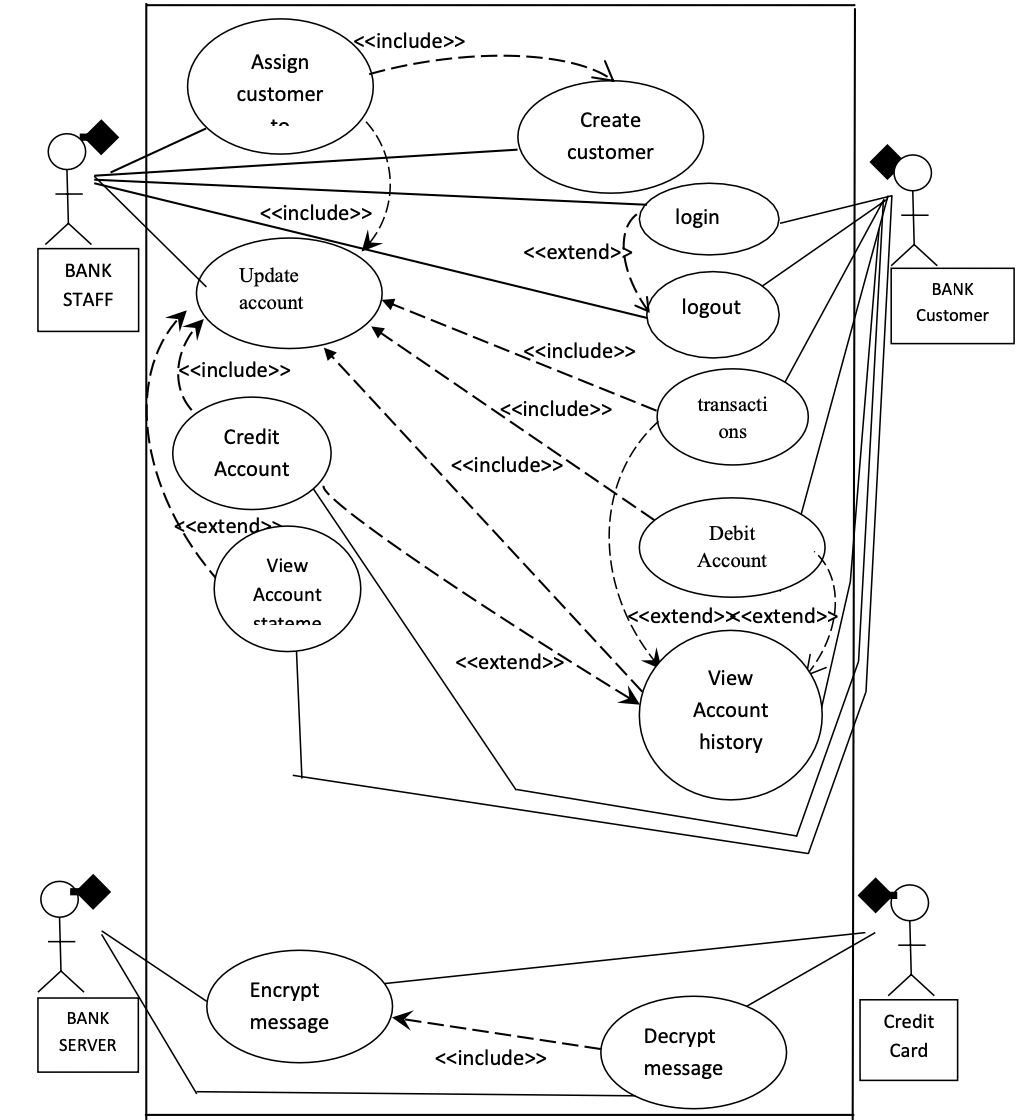
Conceptual model of UML can be mastered by learning the following three major elements:

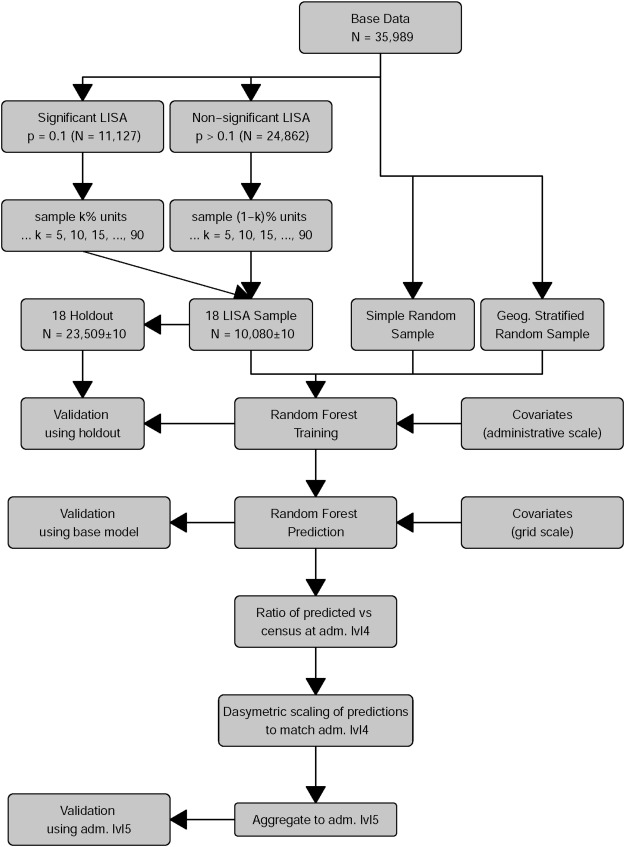
* UML building blocks
* Rules to connect the building blocks
* Common mechanisms of UML

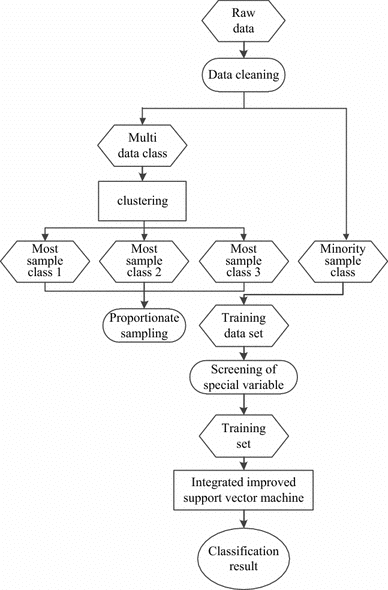
**5.1.1 ACTIVITY DIAGRAM**

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system.





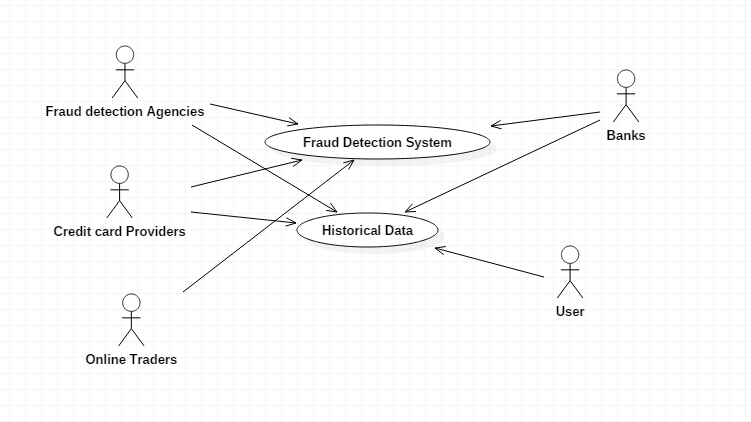




**5.1.2 USE CASE DIAGRAM**

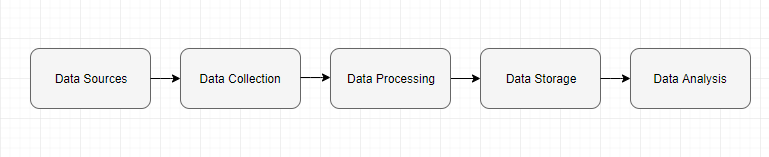
To model a system the most important aspect is to capture thedynamic behavior.

These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.



**5.1.3 DATA FLOW DIAGRAM**

A data flow diagram (DFD) is a graphical representation of the flow of data through an [information system](https://en.wikipedia.org/wiki/Information_system), modelling its process aspects. It is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. It can also be used for the [visualization](https://en.wikipedia.org/wiki/Data_visualization) of [data processing](https://en.wikipedia.org/wiki/Data_processing). A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored.



**5.2 ALGORITHMS AND METHODOLOGY**

**5.2.1 LOGISTIC REGRESSION**

Step1: Select a Dataset.

Step2: Preprocessing the dataset.

Step3: Performance analysis on each and every attribute of the dataset.

Step4: Split the dataset into train and test dataset.

Step5: Train the model using Logistic Regression on the train dataset.

Step6: Load the test data.

Step7: Apply Logistic regression on testing dataset.

Step8: Predict output class.

Step9: Construct a confusion matrix and evaluate performance measures like accuracy, precision.

It is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable. The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit transformation* of the probability of presence of the characteristic of interest:

Logistic regression equation

where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

Odds=p/(1-p)

and

Logit(p)=ln(p/(1-p))

**5.2.2 KNearestNeighbour(KNN/K-Means):**

Step1: Select a Dataset.

Step2: Preprocessing the dataset.

Step3: Performance analysis on each and every attribute of the dataset.

Step4: Split the dataset into train and test dataset.

Step5: Train the model using KNN on the train dataset.

Step6: Load the test data.

Step7: Apply KNN on testing dataset.

Step8: Predict output class.

Step9: Construct a confusion matrix and evaluate performance measures like accuracy, precision.

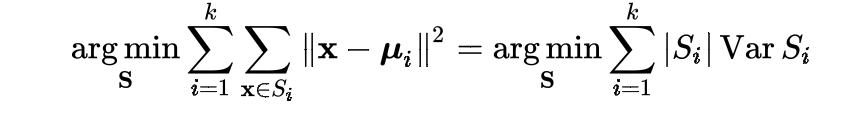
**K Means** algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster’s centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way k means algorithm works is as follows:

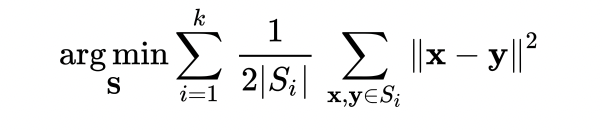
1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.

* Compute the sum of the squared distance between data points and all centroids.
* Assign each data point to the closest cluster (centroid).
* Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

Given a set of observations (**x**1, **x**2, ..., **x***n*), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into *k* (≤ *n*) sets **S** = {*S*1, *S*2, ..., *Sk*} so as to minimize the within-cluster sum of squares (WCSS) (i.e. [variance](https://en.wikipedia.org/wiki/Variance)). Formally, the objective is to find:



where ***μ****i* is the mean of points in *Si*. This is equivalent to minimizing the pairwise squared deviations of points in the same cluster:



The equivalence can be deduced from identity



. Because the total variance is constant, this is equivalent to maximizing the sum of squared deviations between points in *different* clusters (between-cluster sum of squares, BCSS),[[1]](https://en.wikipedia.org/wiki/K-means_clustering#cite_note-:12-1) which follows from the [law of total variance](https://en.wikipedia.org/wiki/Law_of_total_variance).

**5.2.3 Decision Tree:**

Step1: Select a Dataset.

Step2: Preprocessing the dataset.

Step3: Performance analysis on each and every attribute of the dataset.

Step4: Split the dataset into train and test dataset.

Step5: Train the model using the Decision Tree on the train dataset.

Step6: Load the test data.

Step7: Apply Decision Tree on testing dataset.

Step8: Predict output class.

Step9: Construct a confusion matrix and evaluate performance measures like accuracy, precision.

It is a type of supervised learning algorithm that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets based on most significant splitter or differentiator in input variables. A tree can be learned by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of a decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery.

**5.2.4 Random Forest:**

Step1: Select a Dataset.

Step2: Preprocessing the dataset.

Step3: Performance analysis on each and every attribute of the dataset.

Step4: Split the dataset into train and test dataset.

Step5: Train the model using the Random Forest on the train dataset.

Step6: Load the test data.

Step7: Apply Random Forest on testing dataset.

Step8: Predict output class.

Step9: Construct a confusion matrix and evaluate performance measures like accuracy, precision.

Random forests or random decision forests are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set). Random forests generally outperform [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning), but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by [Tin Kam Ho](https://en.wikipedia.org/wiki/Tin_Kam_Ho) using the [random subspace method](https://en.wikipedia.org/wiki/Random_subspace_method), which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by [Leo Breiman](https://en.wikipedia.org/wiki/Leo_Breiman) and [Adele Cutler](https://en.wikipedia.org/wiki/Adele_Cutler), who registered "Random Forests" as a [trademark](https://en.wikipedia.org/wiki/Trademark) in 2006 (as of 2019, owned by [Minitab, Inc.](https://en.wikipedia.org/wiki/Minitab)). The extension combines Breiman's "[bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating)" idea and random selection of features, introduced first by Ho and later independently by Amit and [Geman](https://en.wikipedia.org/wiki/Donald_Geman) in order to construct a collection of decision trees with controlled variance.

Random forests are frequently used as "blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

**5.2.5 Support Vector Machine(SVM):**

Step 1: Select a Dataset

Step 2: Partition the dataset into training dataset and testing dataset.

Step 3: Generate the support vectors for the training dataset

Step 4: Form a linear system for the given training dataset using Support Vectors.

Step 5: If the dataset generates a nonlinear system then convert into linear by applying kernel methods.

Step 6: Solve the linear system by using statistical methods

Step 7: Generate the hyperplane

Step 8: Construct the model to identify the classes by plotting a hyperplane

Step 9: Apply testing dataset on the model.

Step 10: Predict the output class

Step 11: Construct confusion matrix and evaluate performance measures.

It is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

**5.3.1 DATASET**

Thedatasetscontaintransactionsmadebycredit cardsinSeptember2013byEuropeancardholders

which was taken from KAGGLE. This dataset presents transactions that occurred in two days,

where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the

positive class (frauds) account for 0.172% of all transactions. It contains only numeric input variables which are the result of a PCA transformation. Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of distinct principal components is equal to the smaller of the number of original variables or the number of observations minus one Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning.Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

**6. DEVELOPMENT**

**6.1 SAMPLE CODE**

**IMPORTING THE DEPENDENCIES**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.metrics import accuracy\_score**

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.svm import SVC**

**from xgboost import XGBClassifier**

**from sklearn.metrics import f1\_score**

**from sklearn.metrics import accuracy\_score**

**from sklearn.metrics import confusion\_matrix**

**from matplotlib import pyplot as plt**

**import seaborn as sns**

**from sklearn.metrics import f1\_score**

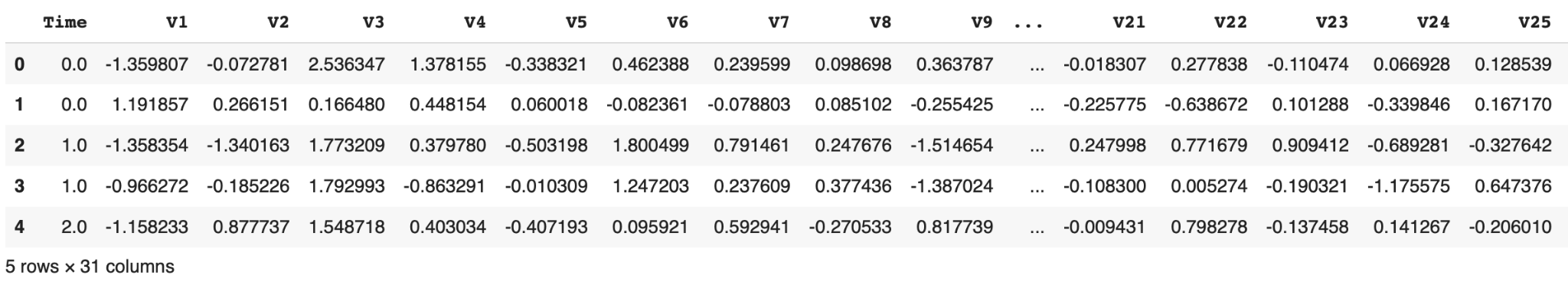
**# loading the dataset to a Pandas DataFrame**

**credit\_card\_data = pd.read\_csv('/Users/tarunkumar/Desktop/creditcard.csv')**

**# first 5 rows of the dataset**

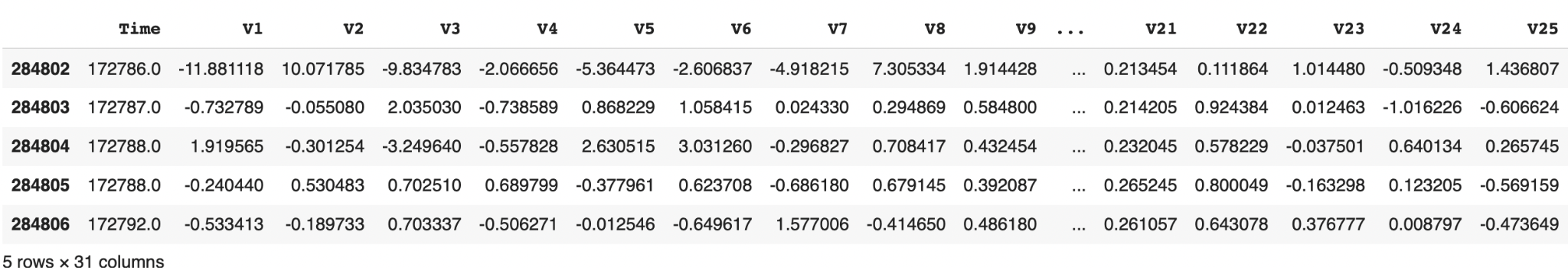
**credit\_card\_data.head()**

**Output:**

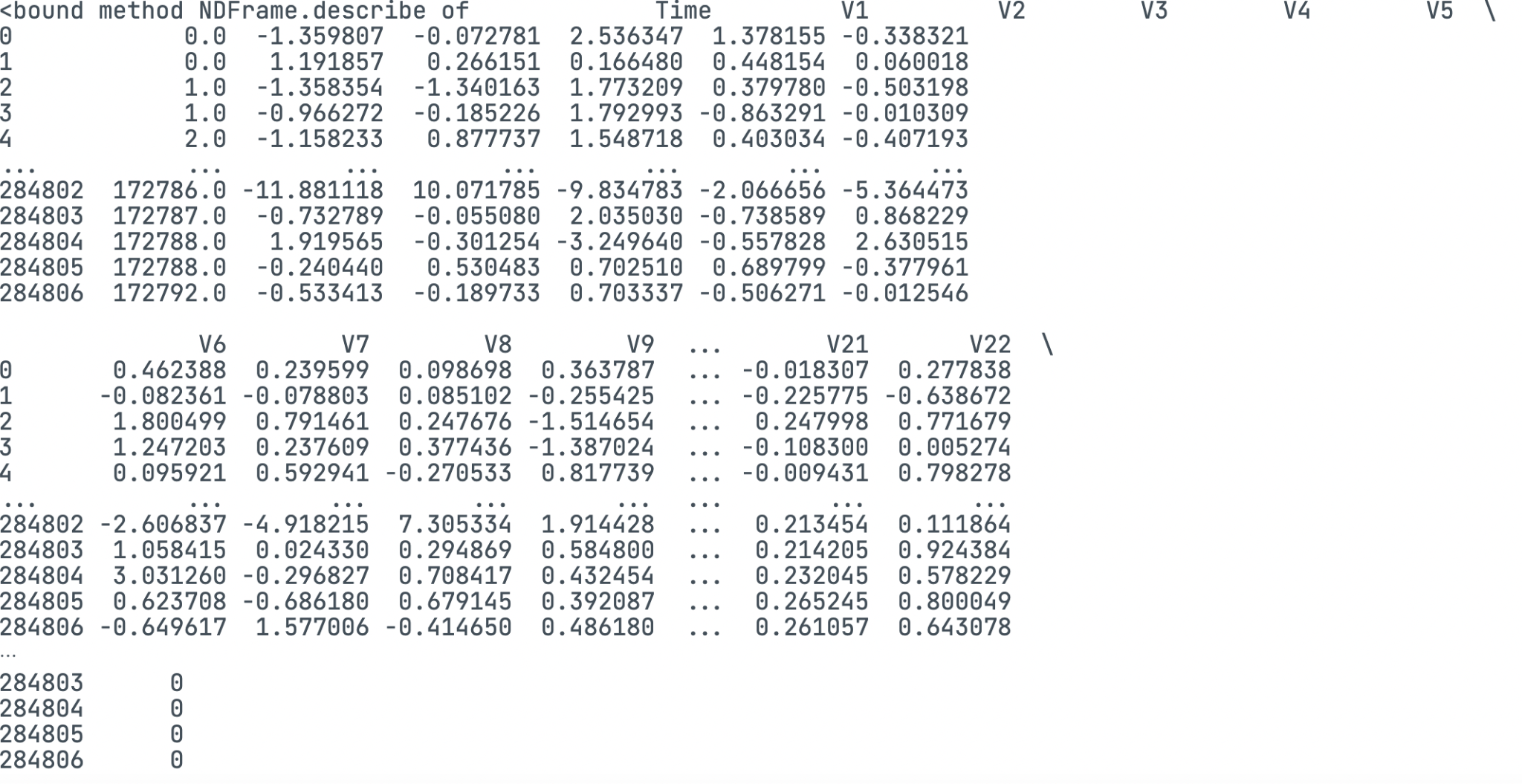
****

**credit\_card\_data.tail()**

**Output:**

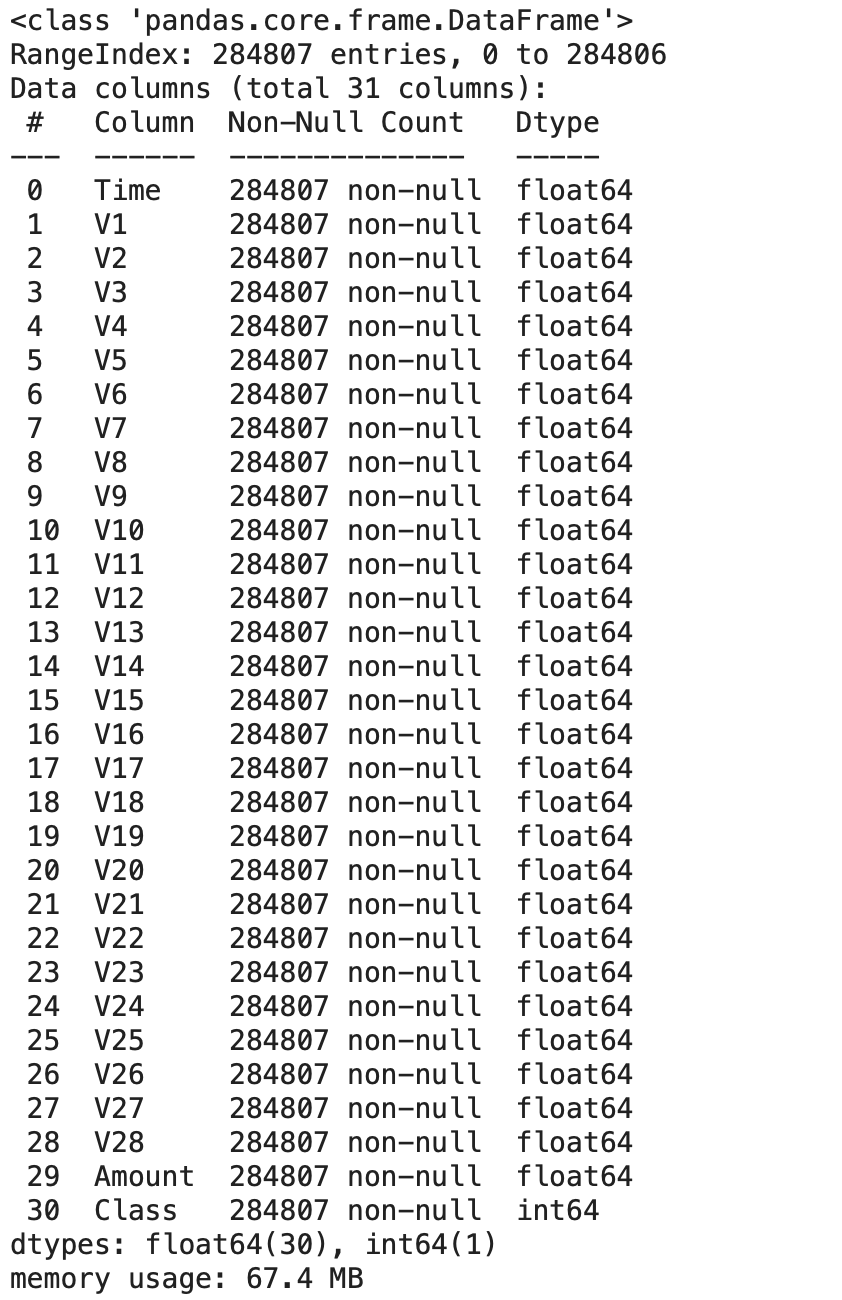
****

**credit\_card\_data.describe**

****

**# dataset informations**

**credit\_card\_data.info()**

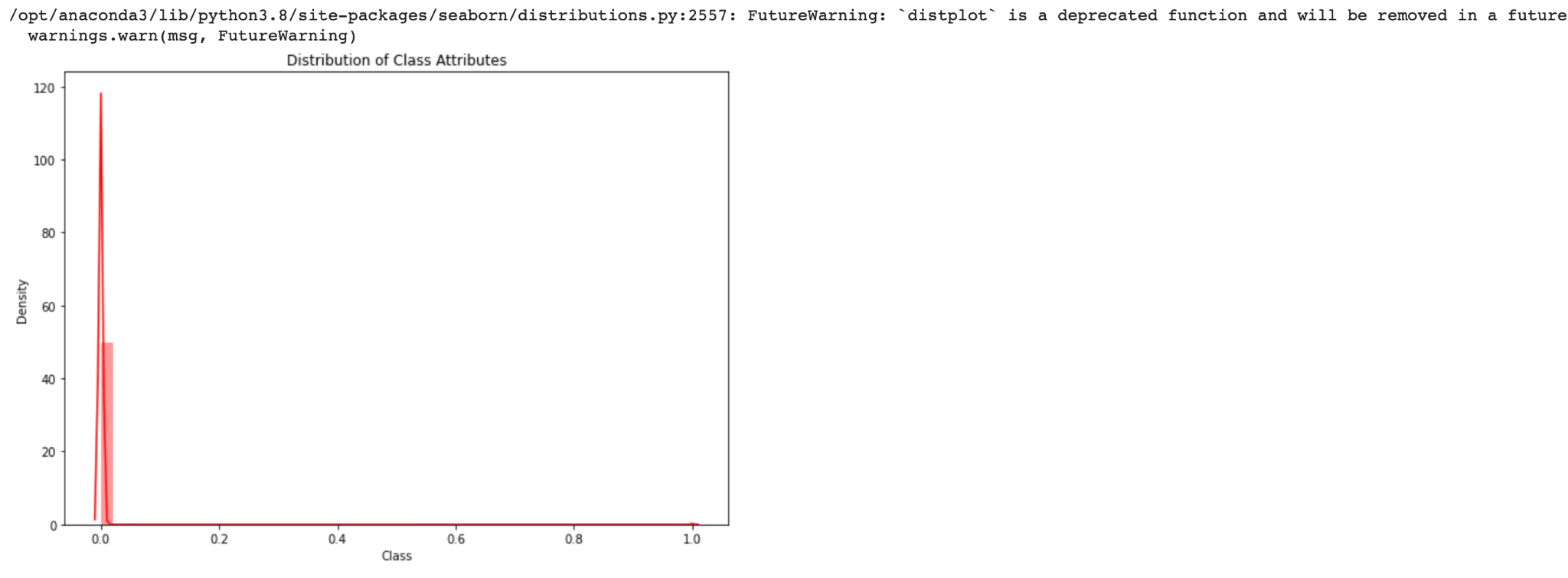
****

**#VISUALIZE THE FEATURES**

**plt.figure(figsize=(10,7))**

**plt.title("Distribution of Class Attributes")**

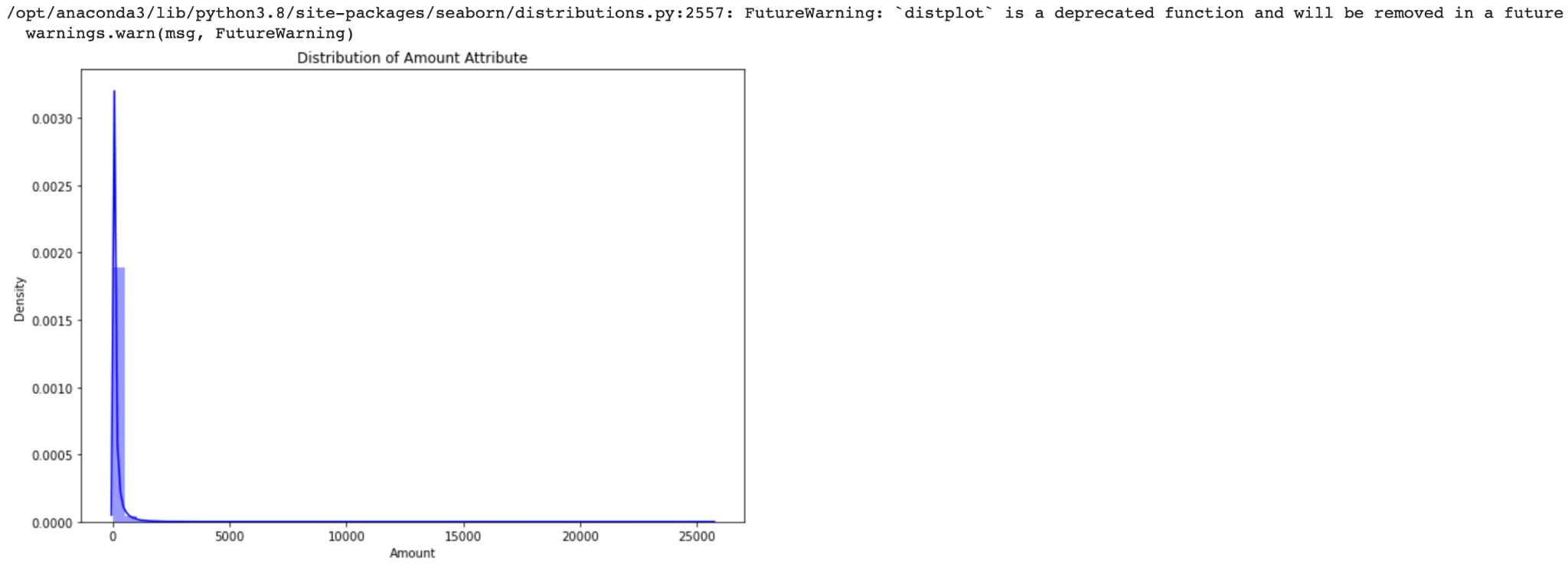
**sns.distplot(credit\_card\_data['Class'], color='red');**

****

**plt.figure(figsize=(10,7))**

**plt.title("Distribution of Amount Attribute")**

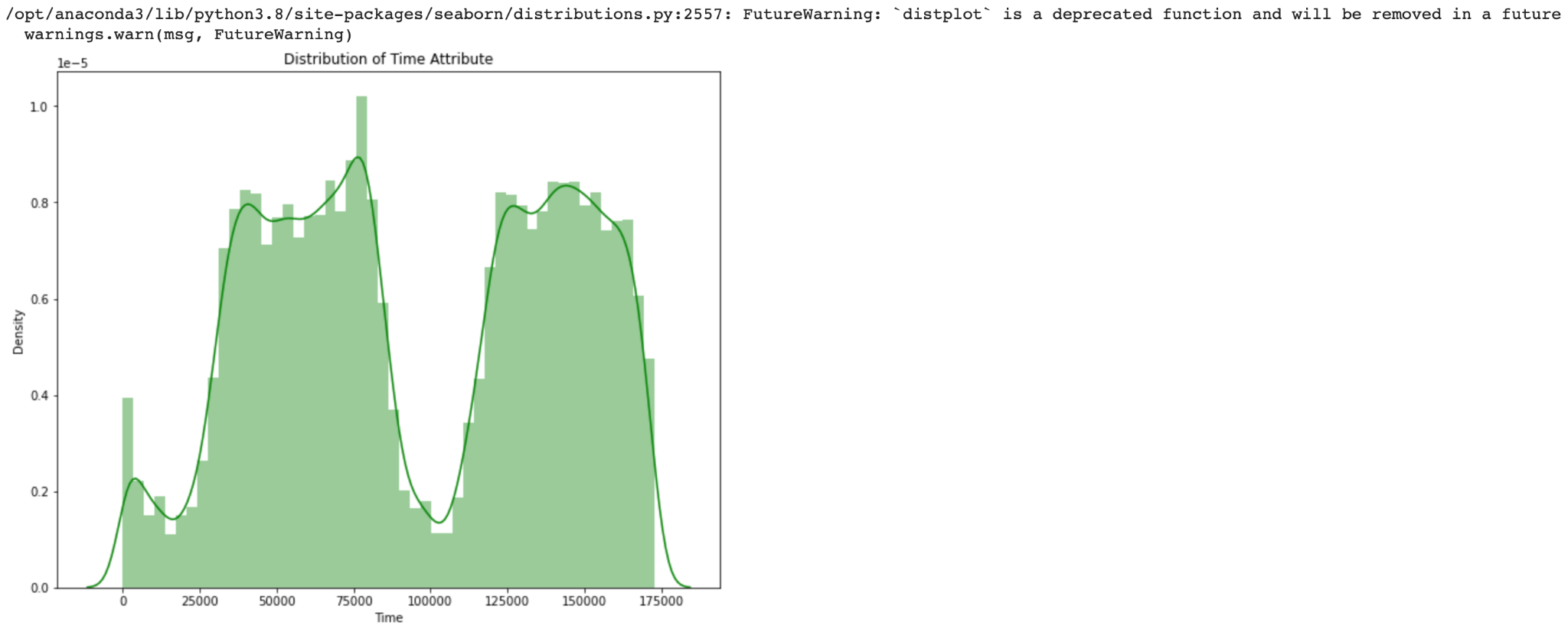
**sns.distplot(credit\_card\_data['Amount'], color='blue');**

****

**plt.figure(figsize=(10,8))**

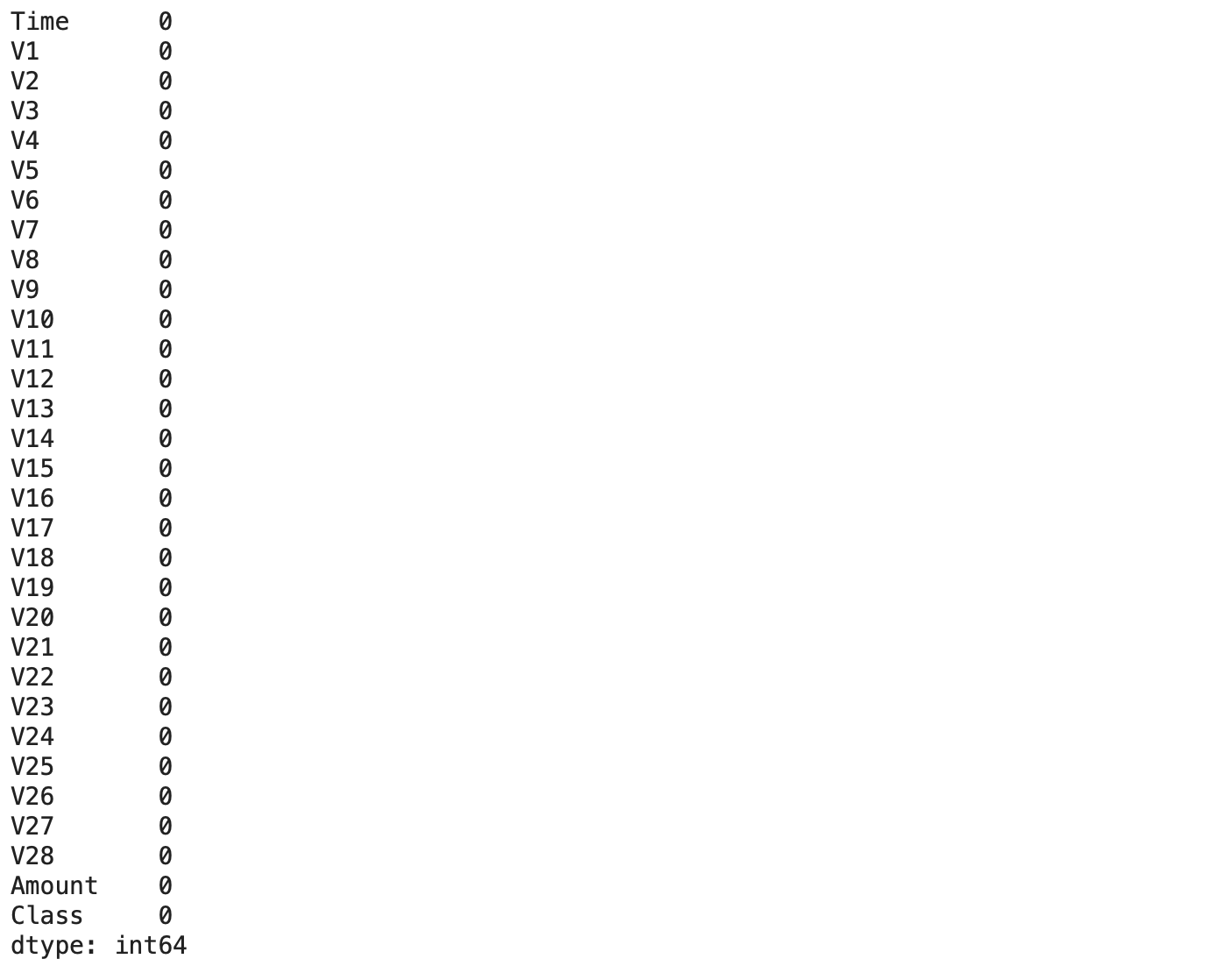
**plt.title("Distribution of Time Attribute")**

**sns.distplot(credit\_card\_data['Time'], color='green');**

****

**# checking the number of missing values in each column**

**credit\_card\_data.isnull().sum()**

****

**# distribution of legit transactions & fraudulent transactions**

**credit\_card\_data['Class'].value\_counts()**

**Output:**

****

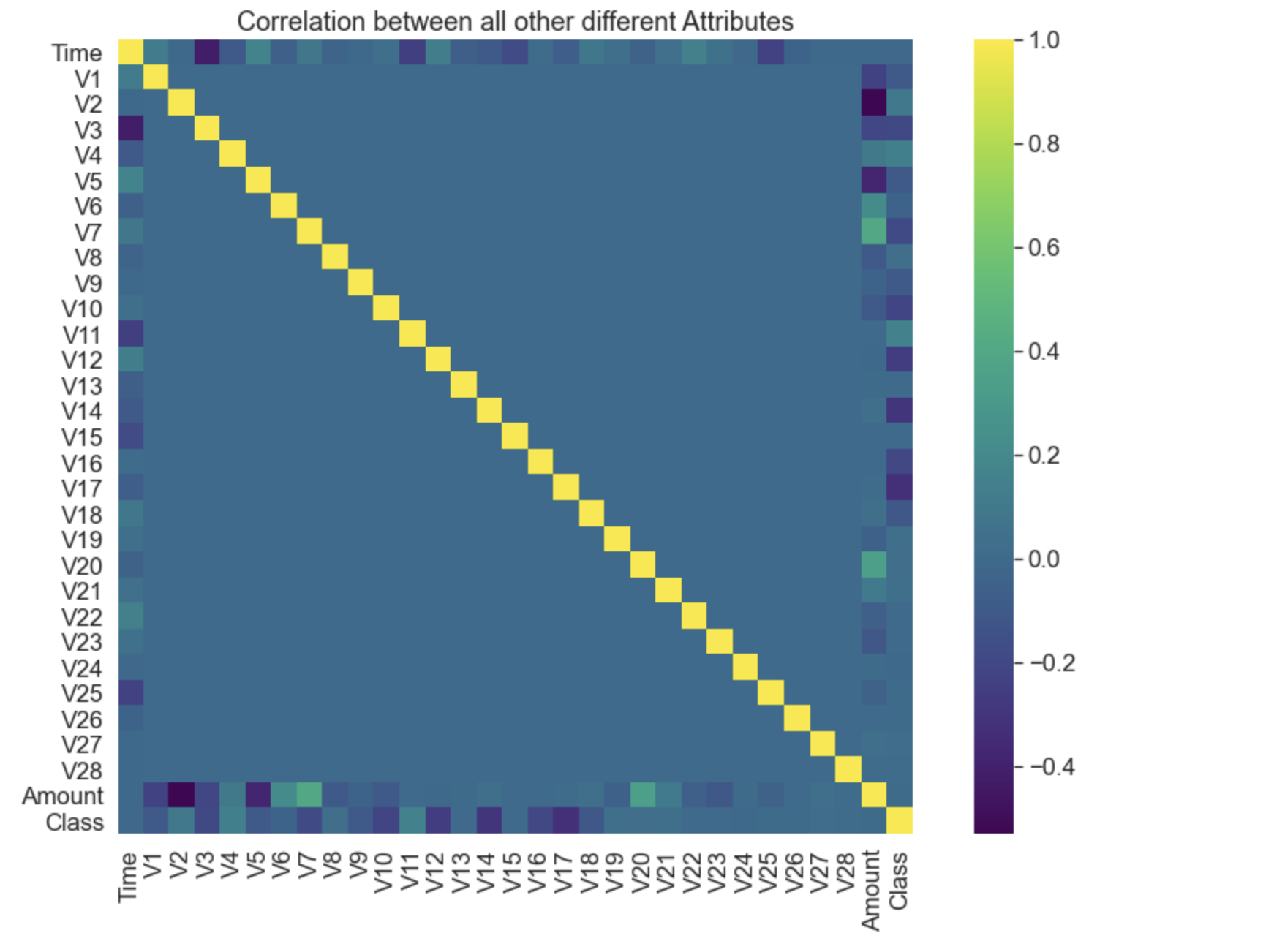
**#correlation of given dataset**

**plt.figure(figsize=(15,10))**

**sns.heatmap(credit\_card\_data.corr(), vmax=1, square=True, cmap='viridis')**

**plt.title("Correlation between all other different Attributes")**

**plt.show()**

****

**fruad = len(credit\_card\_data[credit\_card\_data['Class']==1])**

**notfruad = len(credit\_card\_data[credit\_card\_data['Class']==0])**

**## Data to plot**

**labels = 'Fraud','Not Fraud'**

**sizes = [fruad, notfruad]**

**#plot**

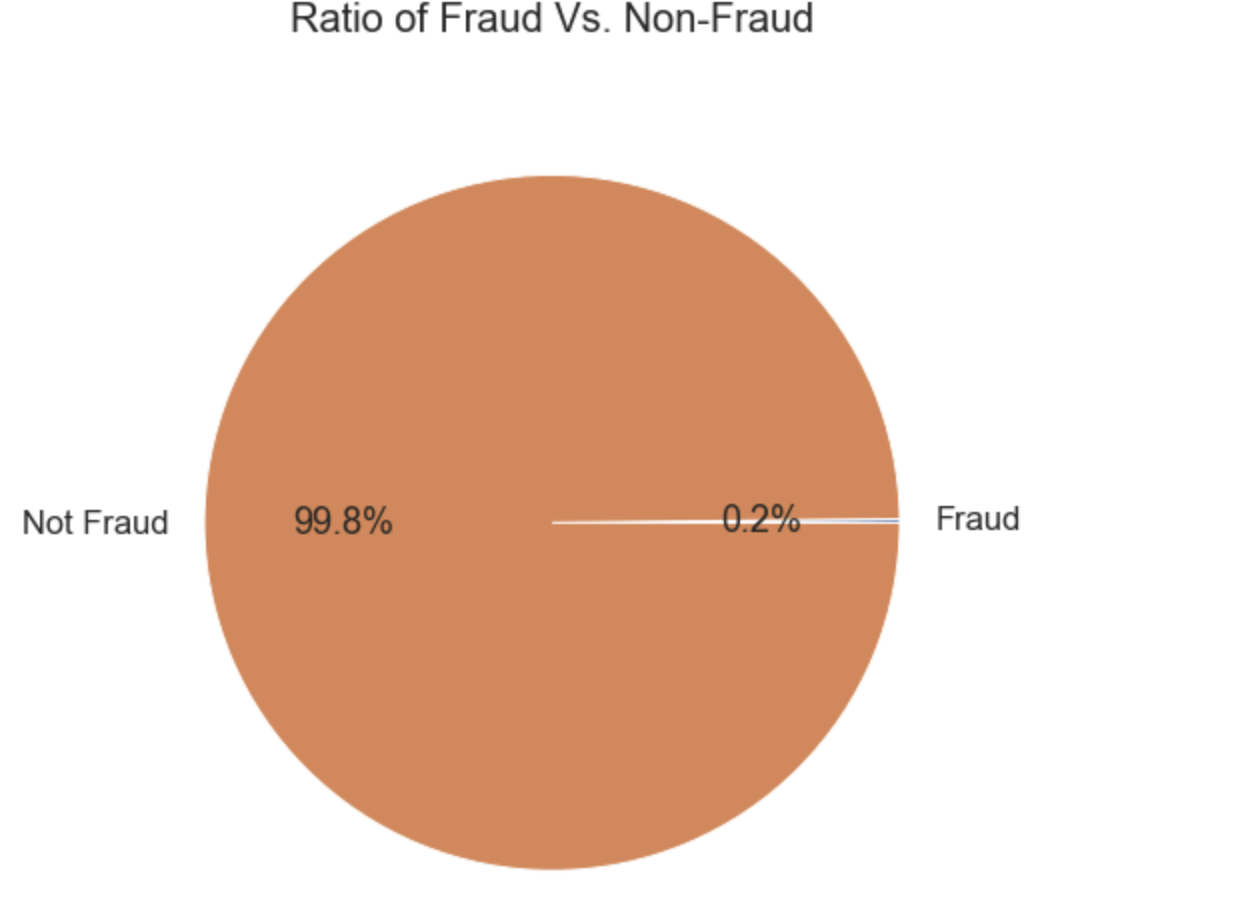
**plt.figure(figsize=(10,8))**

**plt.pie(sizes,labels=labels,autopct='%1.1f%%', startangle=0)**

**plt.title('Ratio of Fraud Vs. Non-Fraud\n', fontsize=20)**

**sns.set\_context("paper", font\_scale=2)**

**Output:**

****

**#This Dataset is highly unbalanced**

**0 --> Normal Transaction**

**1 --> fraudulent transaction**

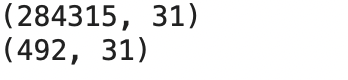
**# separating the data for analysis**

**legit = credit\_card\_data[credit\_card\_data.Class == 0]**

**fraud = credit\_card\_data[credit\_card\_data.Class == 1]**

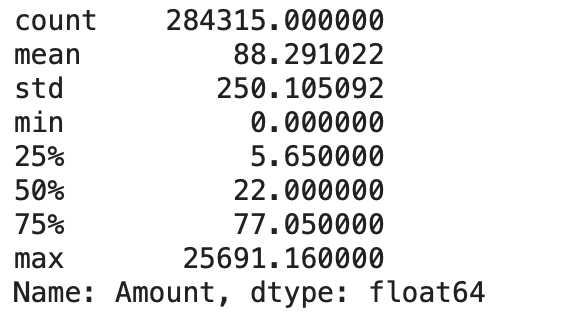
**print(legit.shape)**

**print(fraud.shape)**

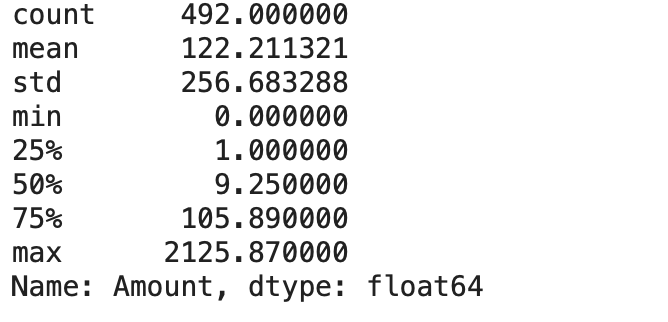
****

**# statistical measures of the data**

**legit.Amount.describe()**

****

**fraud.Amount.describe()**

****

**#Under-Sampling**

**#Build a sample dataset containing similar distribution of normal transactions and #Fraudulent Transactions**

**#Number of Fraudulent Transactions --> 492**

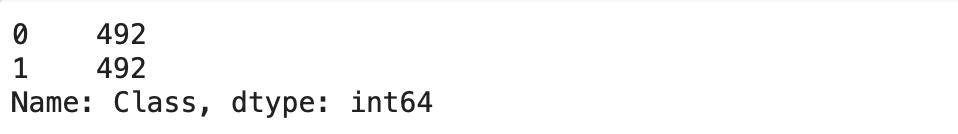
**legit\_sample = legit.sample(n=492)**

**#Concatenating two DataFrames**

**new\_dataset = pd.concat([legit\_sample, fraud], axis=0)**

**new\_dataset['Class'].value\_counts()**

**output:**

****

**#Splitting the data into Features & Targets**

**X = new\_dataset.drop(columns='Class', axis=1)**

**Y = new\_dataset['Class']**

**#Split the data into Training data & Testing Data**

**X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)**

**print(X.shape, X\_train.shape, X\_test.shape)**

**Output:**

****

**Model Training**

**Logistic Regression:**

**model = LogisticRegression(random\_state=2)**

**model.fit(X\_train, Y\_train)**

**X\_train\_prediction = model.predict(X\_train)**

**training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)**

**print('Accuracy on Training data : ', training\_data\_accuracy)**

**X\_test\_prediction = model.predict(X\_test)**

**test\_data\_accuracy1 = accuracy\_score(X\_test\_prediction, Y\_test)**

**print('Accuracy score on Test Data : ', test\_data\_accuracy1)**

**test\_data\_f1score1 = f1\_score(Y\_test, X\_test\_prediction, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score1)**

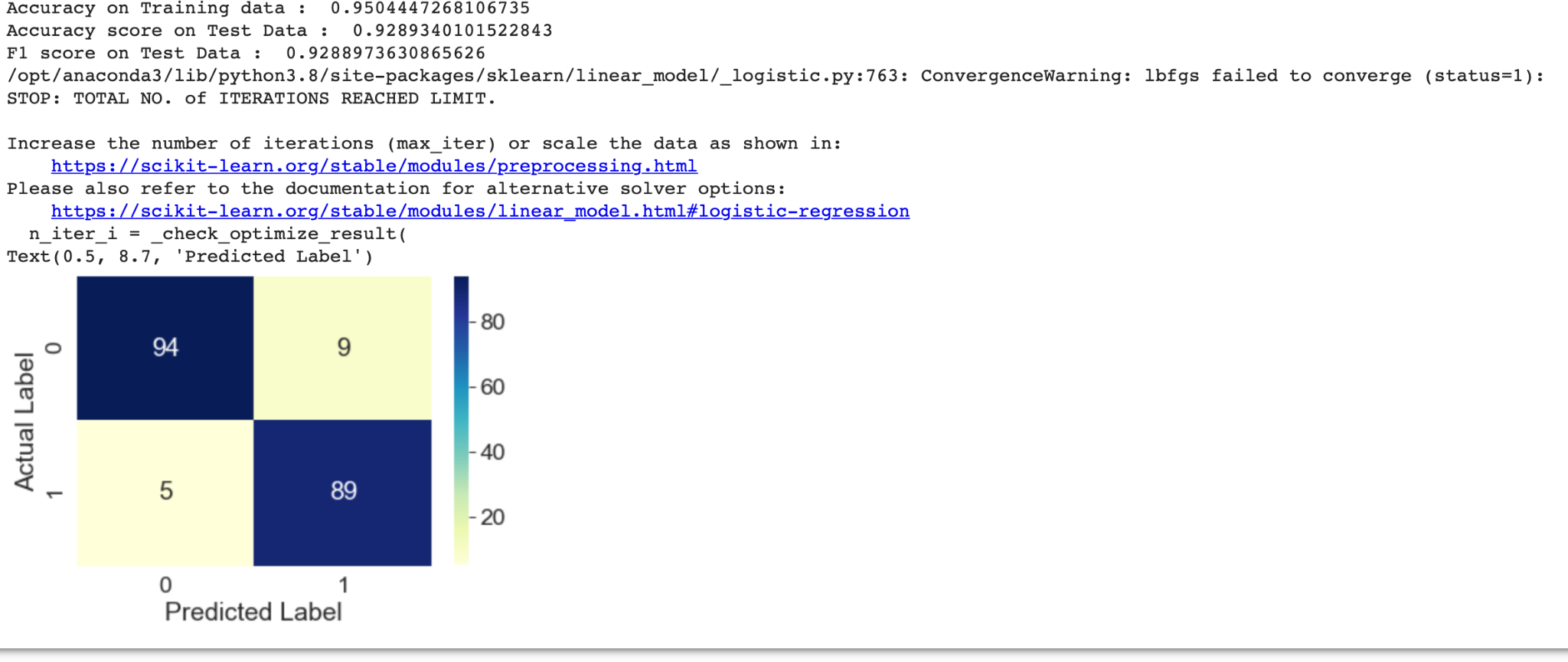
**cnf\_matrix =confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**#Pickle the model that you have trained**

**import pickle**

**pickle\_out = open("lr\_class.pkl", "wb")**

**pickle.dump(model, pickle\_out)**

**pickle\_out.close()**

**Decision trees:**

**model1 = DecisionTreeClassifier(max\_depth = 4, criterion = 'entropy')**

**model1.fit(X\_train, Y\_train)**

**# accuracy on training data**

**X\_train\_prediction = model.predict(X\_train)**

**training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)**

**print('Accuracy on Training data : ', training\_data\_accuracy)**

**# accuracy on test data**

**X\_test\_prediction = model1.predict(X\_test)**

**test\_data\_accuracy2 = accuracy\_score(X\_test\_prediction, Y\_test)**

**print('Accuracy score on Test Data : ', test\_data\_accuracy2)**

**test\_data\_f1score2 = f1\_score(Y\_test, X\_test\_prediction, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score2)**

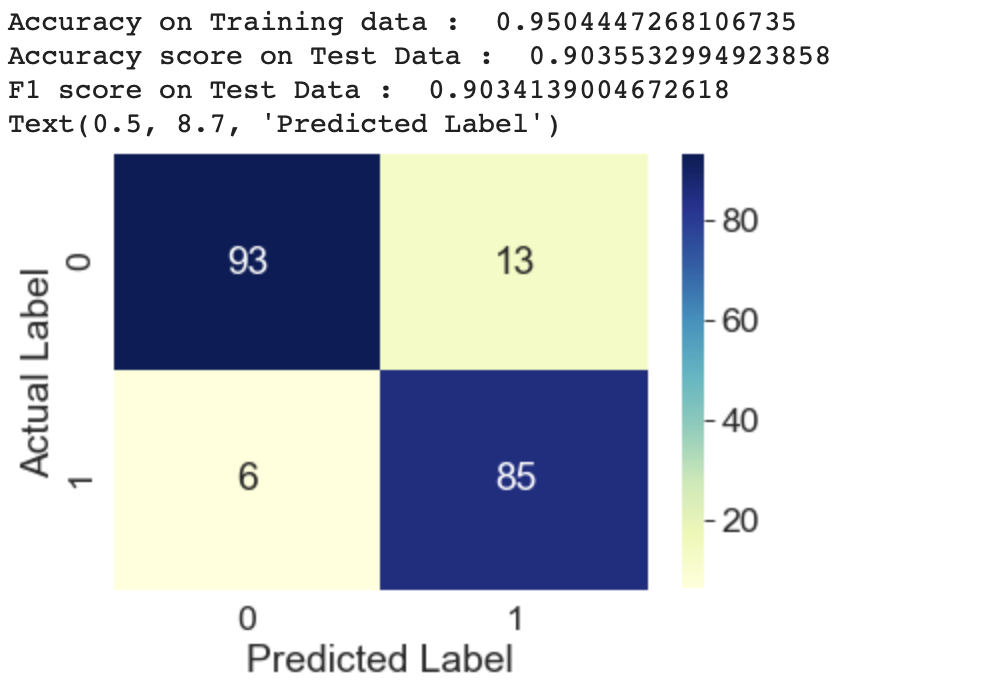
**cnf\_matrix = confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**pickle\_out = open("dt\_class.pkl", "wb")**

**pickle.dump(model1, pickle\_out)**

**pickle\_out.close()**

**K NEAREST NEIGHBOUR:**

**n = 5**

**KNN = KNeighborsClassifier(n\_neighbors = n)**

**KNN.fit(X\_train, Y\_train)**

**# accuracy on training data**

**X\_train\_prediction = KNN.predict(X\_train)**

**training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)**

**print('Accuracy on Training data : ', training\_data\_accuracy)**

**# accuracy on test data**

**X\_test\_prediction = KNN.predict(X\_test)**

**test\_data\_accuracy3 = accuracy\_score(X\_test\_prediction, Y\_test)**

**print('Accuracy score on Test Data : ', test\_data\_accuracy3)**

**test\_data\_f1score3 = f1\_score(Y\_test, X\_test\_prediction, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score3)**

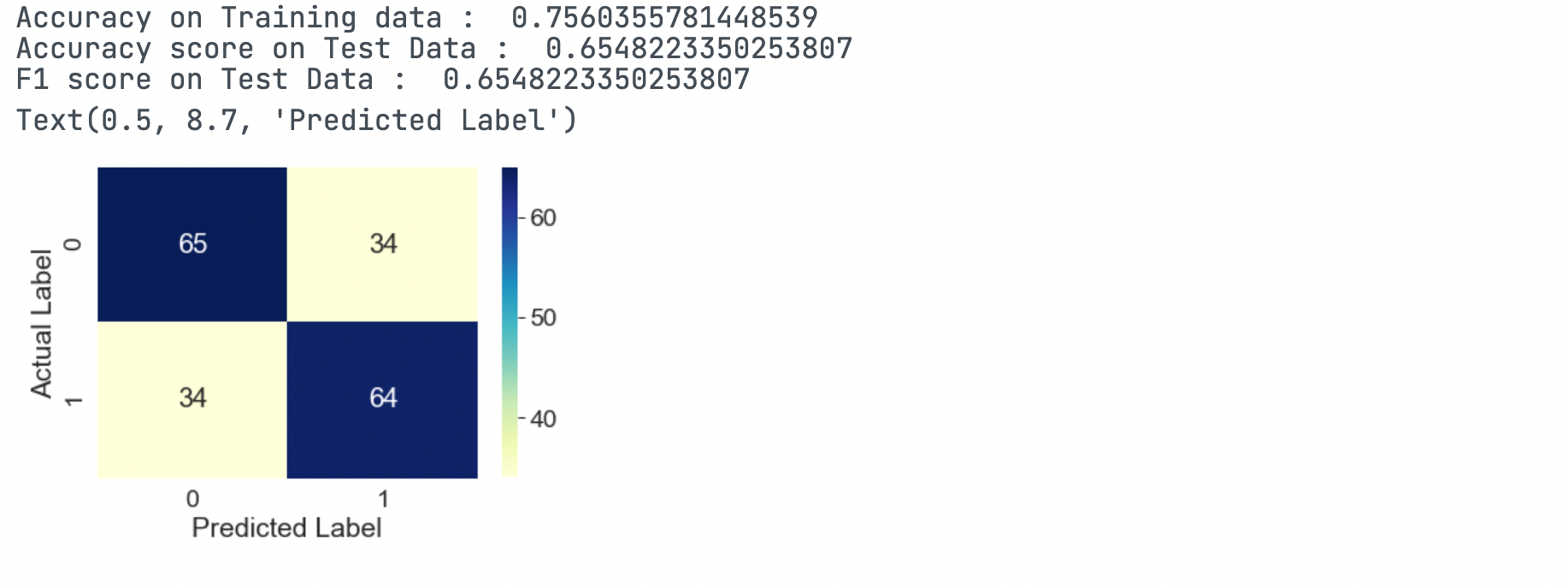
**cnf\_matrix = confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**pickle\_out = open("knn\_class.pkl", "wb")**

**pickle.dump(KNN, pickle\_out)**

**pickle\_out.close()**

**SUPPORT VECTOR MACHINE:**

**svm = SVC()**

**svm.fit(X\_train, Y\_train)**

**# accuracy on training data**

**X\_train\_prediction = svm.predict(X\_train)**

**training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)**

**print('Accuracy on Training data : ', training\_data\_accuracy)**

**# accuracy on test data**

**X\_test\_prediction = svm.predict(X\_test)**

**test\_data\_accuracy4 = accuracy\_score(X\_test\_prediction, Y\_test)**

**#print('confusion matrix of test data'.format(confusion\_matrix(X\_test\_prediction,Y\_test,labels=[0,1])))**

**print('Accuracy score on Test Data : ', test\_data\_accuracy4)**

**test\_data\_f1score4 = f1\_score(Y\_test, X\_test\_prediction, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score4)**

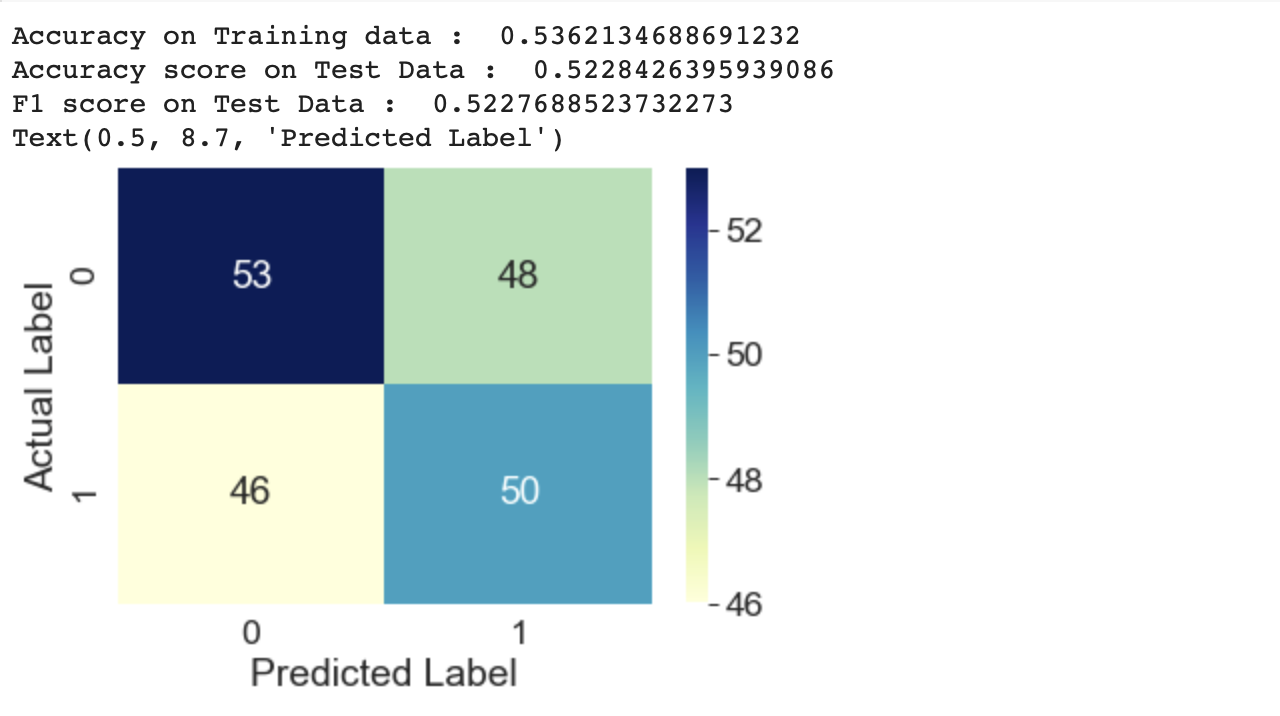
**cnf\_matrix = confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**pickle\_out = open("svm\_class.pkl", "wb")**

**pickle.dump(svm, pickle\_out)**

**pickle\_out.close()**

**RANDOM FOREST ALGORITHM:**

**rf = RandomForestClassifier(max\_depth = 4)**

**rf.fit(X\_train, Y\_train)**

**# accuracy on training data**

**X\_train\_prediction = rf.predict(X\_train)**

**training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)**

**print('Accuracy on Training data : ', training\_data\_accuracy)**

**# accuracy on test data**

**X\_test\_prediction = rf.predict(X\_test)**

**test\_data\_accuracy5 = accuracy\_score(X\_test\_prediction, Y\_test)**

**print('Accuracy score on Test Data : ', test\_data\_accuracy5)**

**test\_data\_f1score5 = f1\_score(Y\_test, X\_test\_prediction, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score5)**

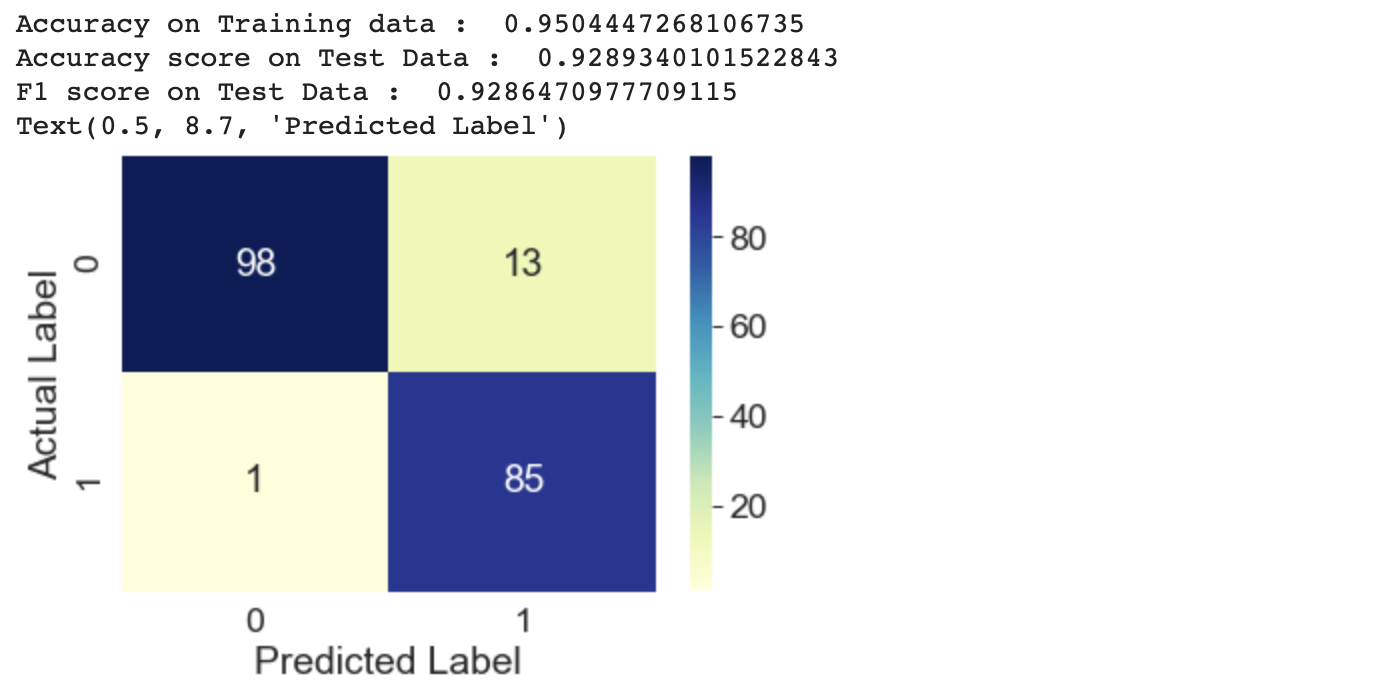
**cnf\_matrix = confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**pickle\_out = open("rf\_class.pkl", "wb")**

**pickle.dump(rf,pickle\_out)**

**pickle\_out.close()**

**MULTI LAYER PERCEPTRON ALGORITHM:**

**from sklearn.metrics import recall\_score**

**from sklearn.neural\_network import MLPClassifier**

**MLPC = MLPClassifier(hidden\_layer\_sizes=(200,), max\_iter=10000)**

**MLPC.fit(X\_train, Y\_train)**

**y\_pred = MLPC.predict(X\_test)**

**recall\_acc = recall\_score (Y\_test,y\_pred)**

**recall\_acc**

**test\_data\_accuracy7 = accuracy\_score(y\_test, y\_pred)**

**print('Accuracy score on Test Data : ', test\_data\_accuracy7)**

**test\_data\_f1score7 = f1\_score(y\_pred, Y\_test, average='weighted')**

**print('F1 score on Test Data : ', test\_data\_f1score7)**

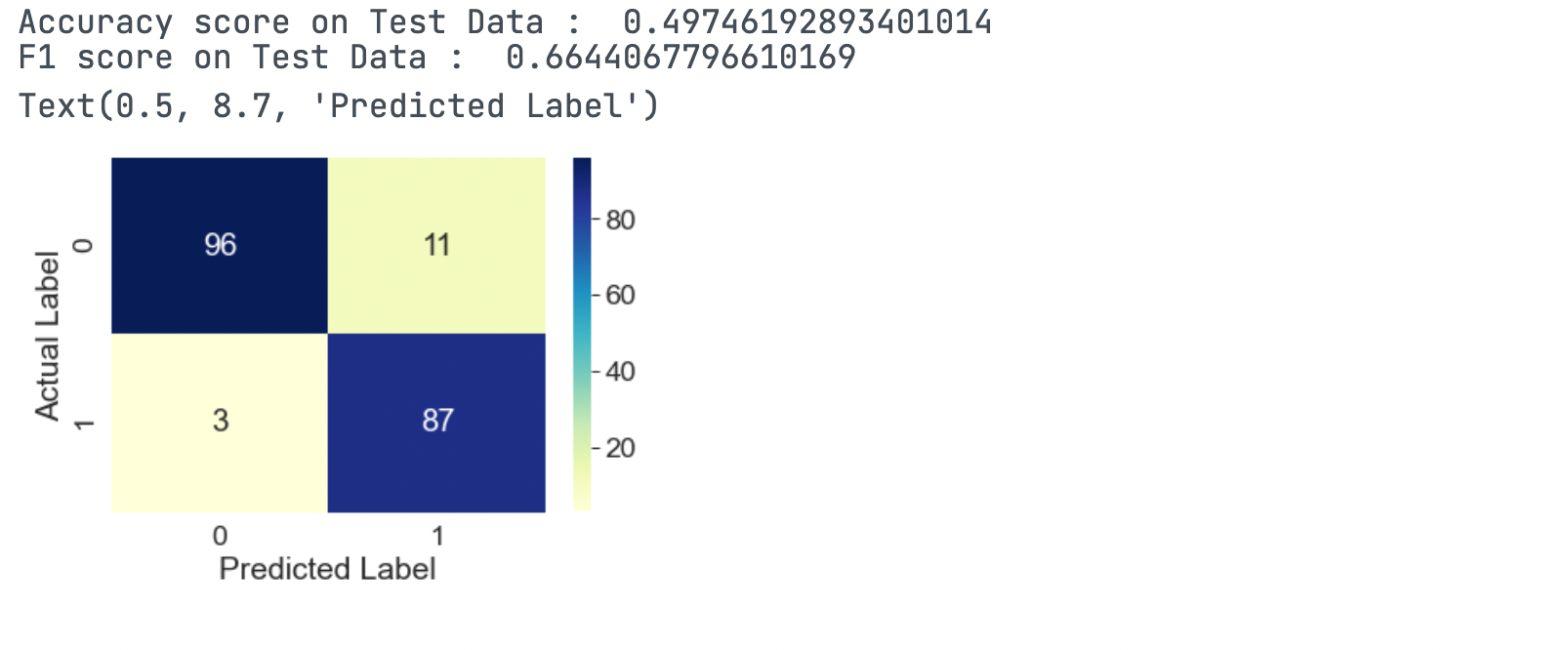
**cnf\_matrix = confusion\_matrix(X\_test\_prediction,Y\_test)**

**sns.heatmap(pd.DataFrame(cnf\_matrix), annot=True, cmap="YlGnBu", fmt='g')**

**plt.ylabel('Actual Label')**

**plt.xlabel('Predicted Label')**

**Output:**

****

**pickle\_out = open("mlp\_class.pkl", "wb")**

**pickle.dump(MLPC, pickle\_out)**

**pickle\_out.close()**

**ANALYSING OR COMPARING THE RESULTS**

**results = pd.DataFrame({**

**'Model': ['Logistic Regression', 'Decision Tree', 'KNN', 'SVM', 'Random Forest', 'XGBoost', 'MLPClassifier'],**

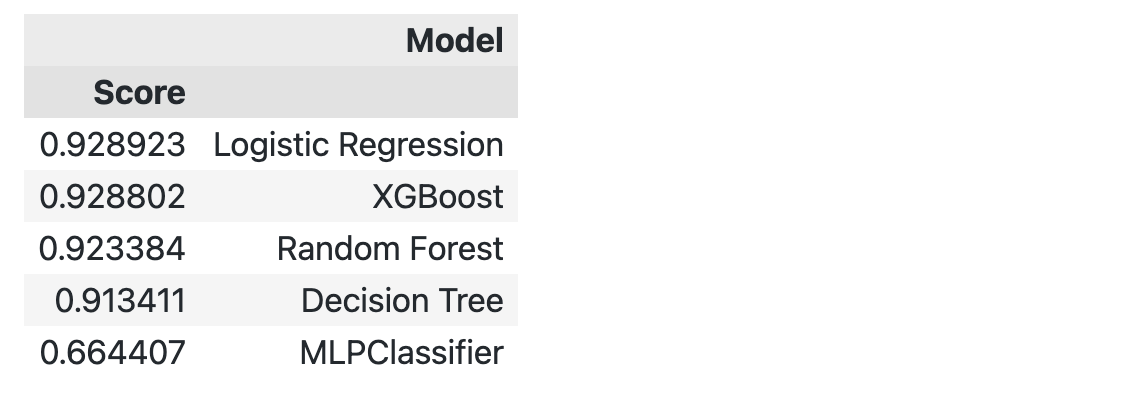
**'Score': [ test\_data\_f1score1,test\_data\_f1score2,test\_data\_f1score3,test\_data\_f1score4,test\_data\_f1score5,test\_data\_f1score6,test\_data\_f1score7]})**

**result\_df = results.sort\_values(by='Score', ascending=False)**

**result\_df = result\_df.set\_index('Score')**

**result\_df.head(5)**

**Output:**

****

**names = ['Logistic Regression','Decision Tree', 'KNN', 'SVM', 'Random Forest', 'XGBoost', 'MLPClassifier']**

**results = [test\_data\_f1score1,test\_data\_f1score2,test\_data\_f1score3,test\_data\_f1score4,test\_data\_f1score5,test\_data\_f1score6,test\_data\_f1score7]**

**plt.figure(figsize=(20,20))**

**plt.bar(names,results)**

**plt.xlabel("Models", fontsize=16) #setting the xtitle and size**

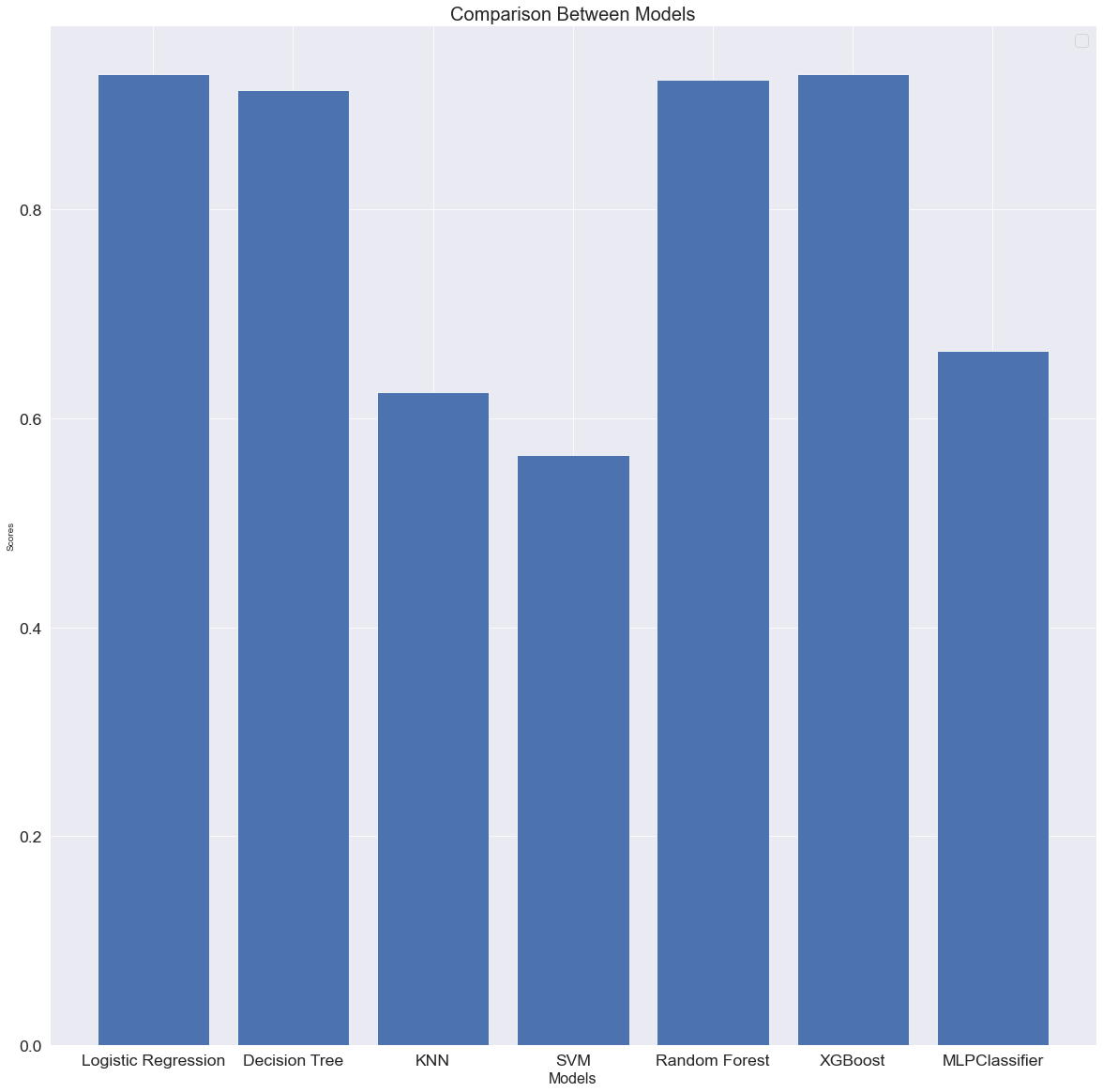
**plt.ylabel("Scores", fontsize=10) # Setting the title and size**

**plt.title("Comparison Between Models", fontsize=20)**

**plt.legend()**

**plt.show()**

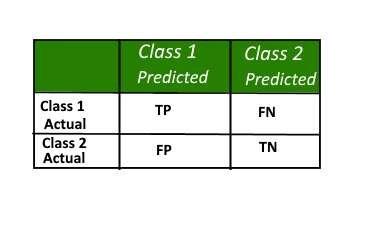
**Output:**

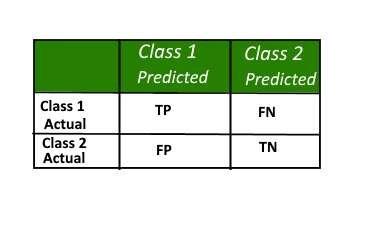
****

**CONFUSION MATRIX:**

A confusion matrix is a table that is often used to describe the performance of a classification

model (or "classifier") on a set of test data for which the true values are known.



****

**Definition of the terms:**

True Positive (TP)

False Negative (FN)

True Negative (TN)

False Positive (FP)

: Observation is positive, and is predicted to be positive.

: Observation is positive, but is predicted negative.

: Observation is negative, and is predicted to be negative.

: Observation is negative, but is predicted positive.

**Classification Rate/Accuracy:**

Classification Rate or Accuracy is given by the relation:

Accuracy = (TP+TN) / (TP+TN+FP+FN)

**Recall:**

Recall can be defined as the ratio of the total no of correctly classified positive examples divide to

the total number of positive examples. High Recall indicates the class is correctly recognized

(small number of FN).

Recall is given by the relation:

Recall = TP / (TP+FN)

**Precision:**

To get the value of precision we divide the total number of correctly classified positive examples

by the total number of predicted positive examples. High precision indicates an example labeled

as positive is indeed positive (small number of FP).

Precision = TP / (TP+FP)

**Specificity (True negative rate):**

Specificity (SP) is calculated by the number of correct negative predictions divided by the total

number of negatives. It is also called true negative rate (TNR).

Specificity = TN / (TN+FP)

**10. SYSTEM TESTING**

The purpose of testing is to discover errors. Testing is the process of trying to discover

every conceivable fault or weakness in a work product. It provides a way to check the functionality

of components, subassemblies or finished products. It is the process of exercising the software with

the intent of ensuring that the software system meets its requirements and user expectations and

does not fail in an unacceptable manner. There are various types of tests. Each test addresses a specific

testing requirements.

**Types of Testing:**

The primary objective of test case design is to derive a set of tests that have the highest

likelihood for uncovering errors in the software. To accomplish this objective two different

categories of test case design techniques are used.

**1. White-Box Testing**

White box testing sometimes called glass box testing is a test case designs that focus on the

program control structure. Test cases are derived to ensure that

1. Guarantee that all independent paths within a module have been exercised at least once.

2. Exercise all logical design on their true and false sides.

3. Executes all loops at their boundaries and within their operational boundaries.

4. Exercise internal data structure to ensure their validity. Several methods are used in the

white box testing.

**2. Black Box Testing**

Tests can be conducted at software interface by knowing the specified function that a product has

been designed to perform, tests can be conducted that demonstrate each function is fully

operational, at the same time searching for errors is called black box testing, sometimes called

behavioural testing. Black box testing is not an alternative to white box techniques. Rather it is a

complementary approach that is likely to uncover a different class of errors than white box

methods. Black box tests are designed to uncover errors in functional requirements without regard

to the internal workings of a program. Black box testing techniques focus on the information

domain of the software.

Black box testing attempts to find errors in the following categories.

1. Incorrect or missing functions.

2. Interface errors.

3. Errors in the data structures or external database access.

4. Performance errors.

5. Initialization and termination errors.

**3. System Testing**

System tests are designed to validate a fully developed system with a view to assuming that it

meets its requirements. System testing is actually a series of different tests, whose primary purpose

is to be fully exercising the computer-based system. In this system, although each test has a

different purpose, all the works are verified to ensure that all system elements have been properly

integrated and performed allocated functions.

There are three kinds of system testing:

1. Alpha Testing: Alpha testing refers to the system testing that is carried by the customer

within the organization along with the developer. The Alpha test are conducted in controlled

manners.

2. Beta Testing: Beta testing is the system performed by a selected group of customers, the

developer is not present at the site and the user will inform the problems that are

encountered during testing. The software developer makes the necessary changes and

submits to the customer.

3. Acceptance Testing: Acceptance testing is the system testing performed by the customer

to whether or not to accept the delivery of the system.

**4. Unit Testing**

Unit testing focuses verification effort on the smallest unit of the software design, the module.

Using the detailed design description as a guide, important control paths are tested to uncover

errors with the boundary of the module for the following modules. All the statements into module

are executed at least once. From this we can ensure that all independent paths through the control

structures are exercised.

**5. Integration Testing**

Integration testing is a systematic technique for constructing the program structures and to conduct

tests for uncovered errors with interfacing.

**TEST CASES:**