MACHINE LEARNING
(Credit card Fraud detection)

Summer Internship Report Submitted in partial fulfillment of

the requirement for undergraduate degree of **Bachelor of**

Technology

In

Computer Science Engineering

By

Sanjana Achampeta

221710304051

Under the Guidance of

Assistant Professor

Department Of Computer Science and Engineering

GITAM School of Technology

GITAM (Deemed to be University), Hyderabad-502329

DECLARATION

I submit this industrial training work entitled “CREDIT CARD FRAUD DETECTION” to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “Bachelor of Technology” in “Computer Science Engineering”. I declare that it was carried out independently by me under the guidance of , Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD **Sanjana Achampeta**

Date:13-07-2020 **221710304051**

CERTIFICATE

This is to certify that the Industrial Training Report entitled "CREDIT CARD FRAUD DETECTION" is being submitted by Sanjana Achampeta (221710304051) in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19.

It is faithful record work carried out by her at the Computer Science Engineering Department, GITAM University Hyderabad Campus under my guidance and supervision.

Assistant Professor	Professor and HOD
Department of ECE	Department of ECE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor,GITAM Hyderabad and **,** Principal, GITAM Hyderabad .

I would like to thank respected , Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

SANJANA ACHAMPETA

221710304051

ABSTRACT

Financial fraud is an ever growing menace with far consequences in the financial industry. Data mining had played an imperative role in the detection of credit card fraud in online transactions. Credit card fraud detection, which is a data mining problem, becomes challenging due to two major reasons - first, the profiles of normal and fraudulent behaviors change constantly and secondly, credit card fraud data sets are highly skewed. The performance of fraud detection in credit card transactions is greatly affected by the sampling approach on dataset, selection of variables and detection technique(s) used.

This paper investigates the performance of naïve bayes, k-nearest neighbor and logistic regression on highly skewed credit card fraud data. Dataset of credit card transactions is sourced from European cardholders containing 284,807 transactions. A hybrid technique of under-sampling and oversampling is carried out on the skewed data. The three techniques are applied on the raw and preprocessed data. The work is implemented in Python. The performance of the techniques is evaluated based on accuracy, sensitivity, specificity,precision, coefficient and balanced classification rate.

SANJANA ACHAMPETA

221710304051

Table of Contents

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence (AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and "more items to consider" and "get yourself a little something" on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today's data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

Figure 1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

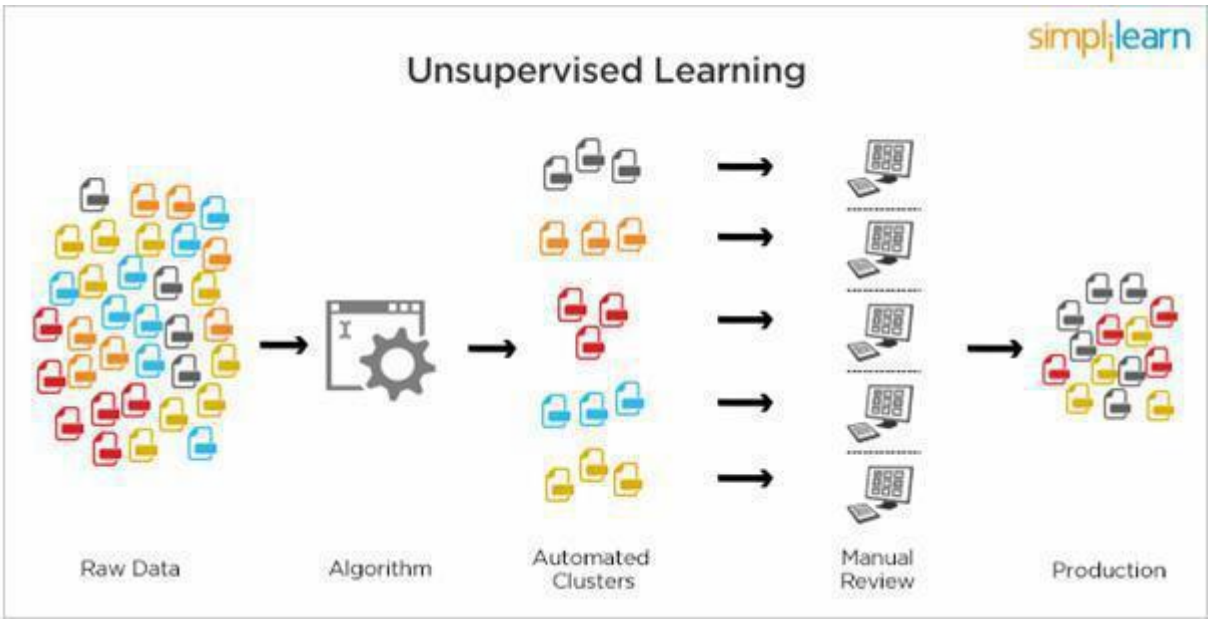


Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

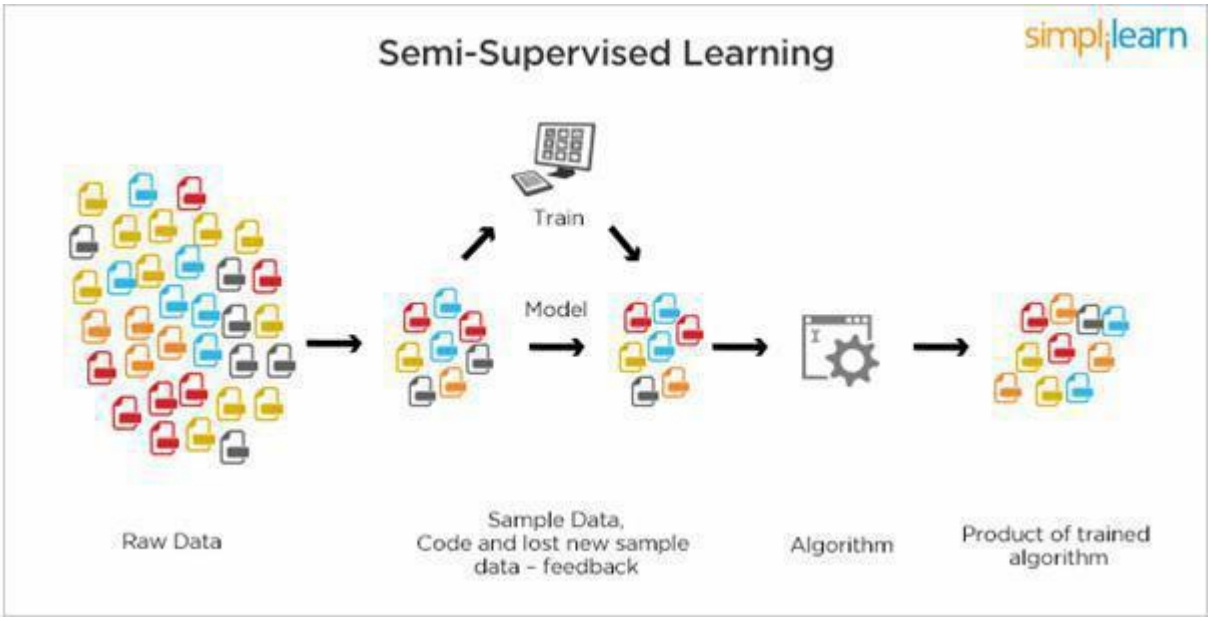


Figure 3 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

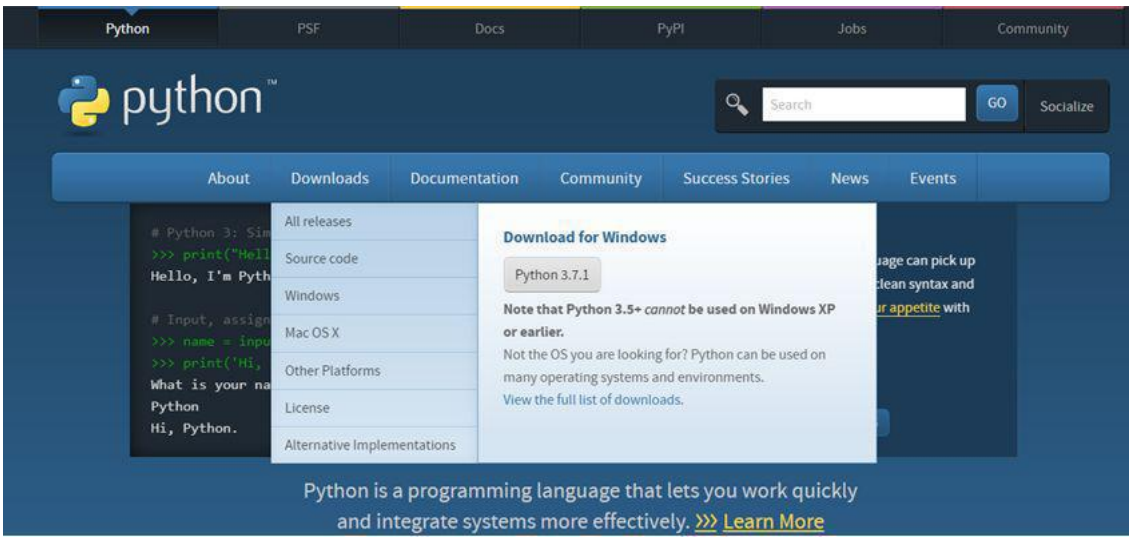


Figure 4 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:
- In windows
    - Step 1: Open Anaconda.com/downloads in web browser.
    - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 - bit graphic installer)
    - Step 3: select installation type(all users)
    - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
    - Step 5: Open jupyter notebook (it opens in default browser)

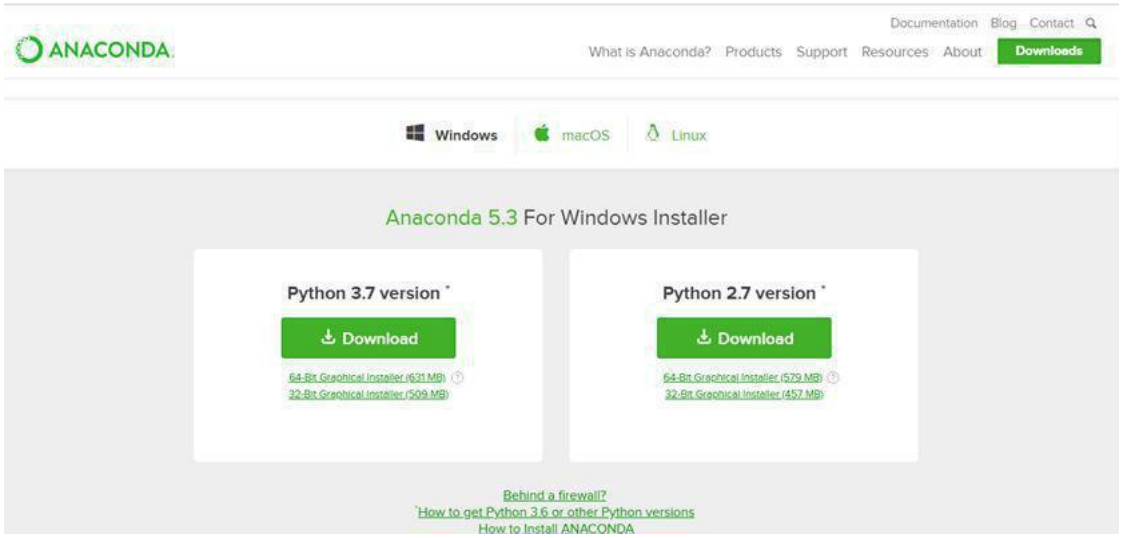


Figure 5 : Anaconda download

Figure 6: Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists

- Tuples
- Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays

or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION:

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOP's CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
    - We define a class in a very similar way how we define a function.
    - Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is

indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 7: Defining a Class

2.7.2 __init__ method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores:__init__().

CHAPTER 3

CASE STUDY

3.1 PROBLEM STATEMENT:

To predict whether the performed transaction is fraud or normal.

3.2 DATA SET:

The given dataset contains following parameters:

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. 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-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

3.3 OBJECTIVE OF THE CASE STUDY:

Due to confidentiality issues, we cannot provide the original features and more background information about the data.

- Features V1, V2, … V28 → The principal components obtained with PCA
- Time → Contains the seconds elapsed between each transaction and the first transaction in the dataset
- Amount → Is the transaction Amount; this feature can be used for example-dependant cost-sensitive learning.

- Class → Is the Target variable and it takes value 1 in case of fraud and 0 for normal transactions

CHAPTER 4

DATA PREPROCESSING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

We can get the data set from the database or we can get the data from the client.

4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.

```
1  # importing packages
2
3  import pandas as pd
4  import numpy as np
5  import seaborn as sns
6  import matplotlib.pyplot as plt
```

Figure 8: Importing Packages

4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as

columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

```
# reading the data

df = pd.read_csv("creditcard.csv")
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0

5 rows × 31 columns

Figure 9: Reading the dataset

4.1.4 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

1. dropna()
2. fillna()
3. interpolate()
4. mean imputation and median imputation .

1. <u>dropna():</u>

dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN).

dropna() function has a parameter called how which works as follows:

- if how = ’all’ is passed then it drops the rows where all the columns of the particular row are missing.
- if how = ’any’ is passed then it drops the rows where all the columns of the particular row are missing.

2. <u>fillna():</u>

fillna() is a function which replaces all the missing values using different ways

fillna() also have parameters called method and axis.

- if we use method = 'ffill' where ffill is a method called forward fill, which carry forwards the previous row's value .
- if we use method = 'bfill' where bfill is a method called backward fill, which carry backward the next row's value .
- if we use method = 'ffill' , axis = 'columns' then it carry forwards the previous column's value .
- if we use method = 'bfill' , axis = 'columns' then it carry backward the next column's value .

3. <u>interpolate():</u>

- interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values .

4. <u>mean and median imputation</u>

- mean and median imputation can be performed by using fillna().
- mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.
- median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

Missing values can be checked using isna() or isnull() functions which returns the output in a boolean format.

Total number of missing values in each column can be calculated using isna().sum() or isnull().sum().

```
df.isnull()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284803	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284804	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284805	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284806	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

284807 rows × 31 columns

Figure 10: Checking Missing Values

```
df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Figure 11 : Total number of missing values in each column.

```
sns.heatmap(df.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1af8a7d2488>
```

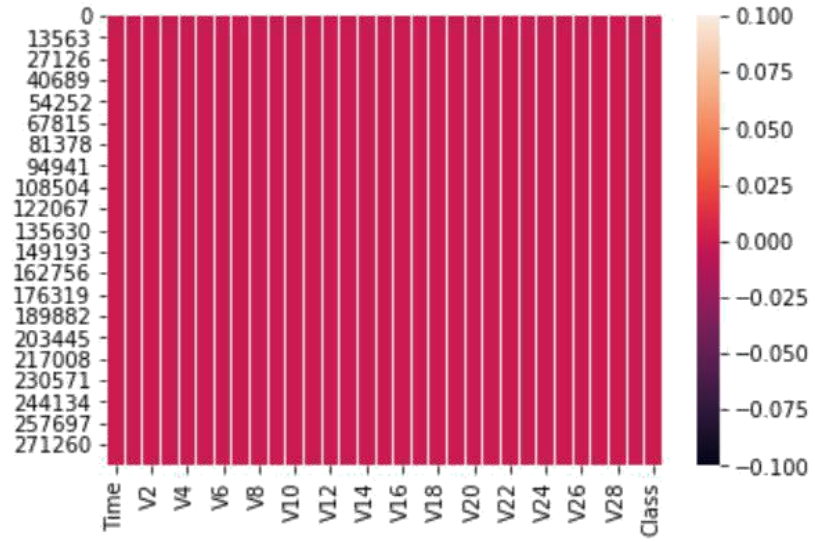


Figure 12: Visualizing the missing values

➔ Observation: From the above Graph we can say that there are no missing values in the heat map

From the above output we can observe that the given dataset do not contain any missing values.

```
LABELS = ["Normal" , "Fraud"]
count_classes = pd.value_counts(df['Class'],sort=True)
count_classes.plot(kind = 'bar')
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("frequency")
```

```
Text(0, 0.5, 'frequency')
```

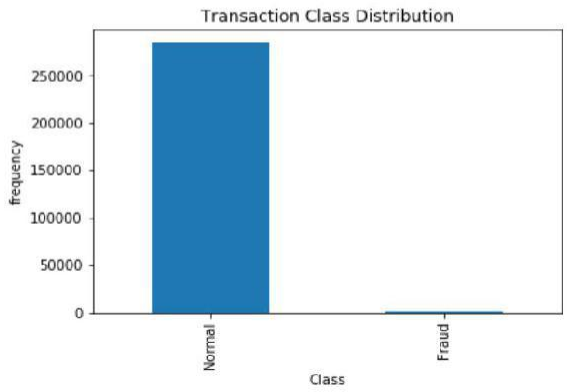


Figure 13: Bar graph for class and frequency

→Observations: From above graph we can see that the data is highly unbalanced i.e it has very high number of normal transactions when compared to that of fraud transactions

```
sns.boxplot(x = "Class", y = "Time",hue='Class', data = df)
plt.show()
```

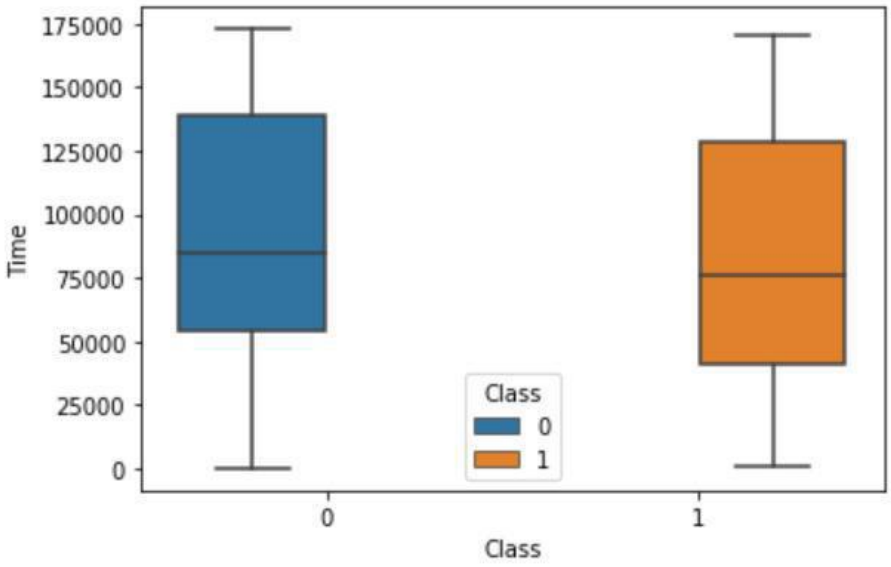


Figure 14:Box plot for class and time.

➔ Observation: By looking at the above box plot we can say that both fraud & genuine transactions occur throughout time and there is no distinction between them.

By looking at the above box plot we can say that both fraud & genuine transactions occur throughout time and there is no distinction between them.

```
sns.boxplot(x = "Class", y = "Amount", data = df)
plt.ylim(0, 5000)
plt.show()
```

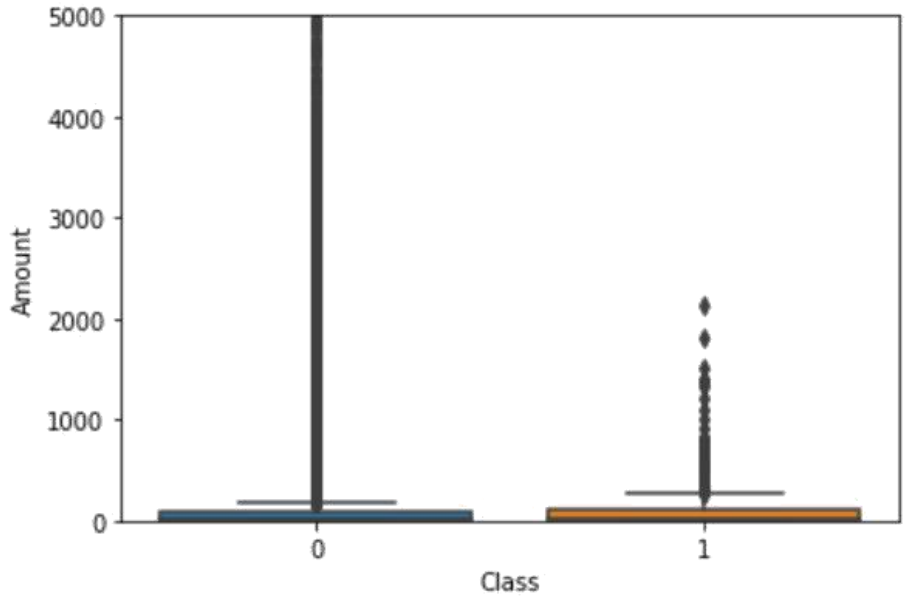


Figure 15:Boxplot for class and amount

➔ Observation: From above box plot we can easily infer that there are no fraud transactions occur above the transaction amount of 3000. All of the fraud transactions have transaction amount less than 3000. However, there are many transactions which have a transaction amount greater than 3000 and all of them are normal.

```
# We Will check Do fraudulent transactions occur more often during certain time frame ? Let us find out with a visual represe

f, (ax1, ax2) = plt.subplots(2, 1,sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```

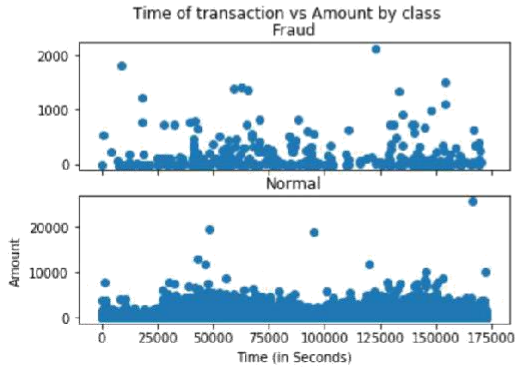


Figure 16: Scatter plot comparing normal and fraud transactions withrespect to amount.

```
sns.relplot(x='Amount',y='Time',hue='Class',data=data)
```

```
<seaborn.axisgrid.FacetGrid at 0x1f6800e1988>
```

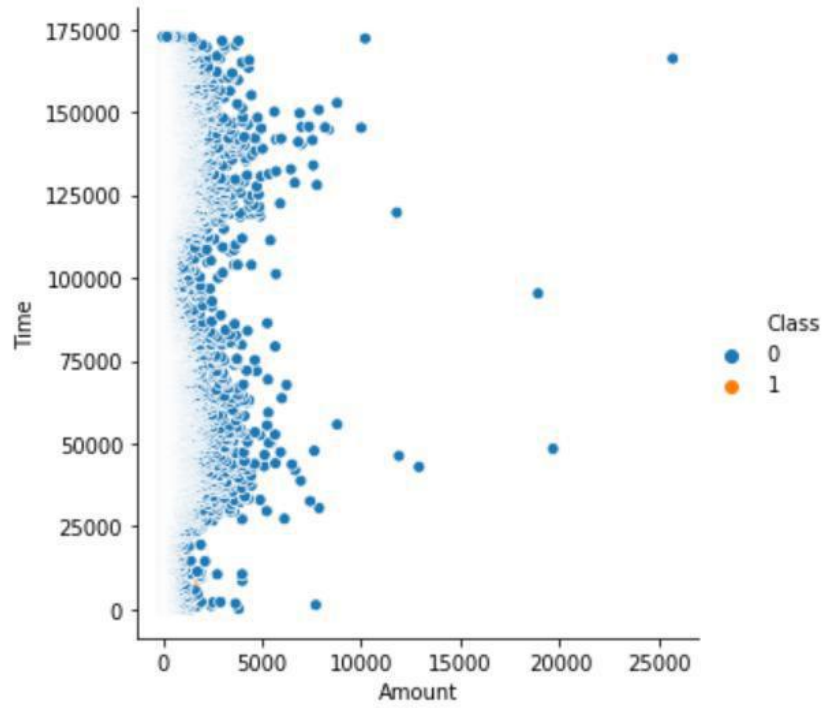


Figure 17: Relation plot for amount and time.

➔ Observations: From the above graph we can observe that we have very less number of fraud transactions when compared to normal transactions ,which states that it is an imbalanced dataset

4.1.5 CATEGORICAL DATA:

- Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.

  Categorical Variables are of two types: Nominal and Ordinal

- **Nominal:**

  The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour

- **Ordinal:**

  The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium

- Categorical data can be handled by using dummy variables, which are also called as indicator variables.
- Handling categorical data using dummies: In pandas library we have a method called get_dummies() which creates dummy variables for those categorical data in the form of 0's and 1's. Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

```
df.dtypes
```

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Figure 18: Description about the type of each feature in the dataset.(Categorical or Numerical).

4.2 TRAINING THE MODEL:

```
fraud = df[df['Class']==1]
Normal = df[df['Class']==0]
print(fraud.shape,Normal.shape)
```

```
(492, 31) (284315, 31)
```

Figure 19: Imbalanced data

Since the dataset is imbalanced, it is balanced using SMOTE.

In Machine Learning and Data Science we often come across a term called Imbalanced Data Distribution, generally happens when observations in one of the class are much higher or lower than the other classes. As Machine Learning algorithms tend to increase accuracy by reducing the error, they do not consider the class distribution. This problem is prevalent in examples such as Fraud Detection, Anomaly Detection, Facial recognition etc.

Standard ML techniques such as Decision Tree and Logistic Regression have a bias towards the majority class, and they tend to ignore the minority class. They tend only to predict the majority class, hence, having major misclassification of the minority class in comparison with the majority class. In more technical words, if we have imbalanced data distribution in our dataset then our model becomes more prone to the case when minority class has negligible or very lesser recall.

Imbalanced Data Handling Techniques: There are mainly 2 mainly algorithms that are widely used for handling imbalanced class distribution.

1. SMOTE
2. Near Miss Algorithm

```
from imblearn.combine import SMOTETomek
smk = SMOTETomek(random_state=120)
X,y = smk.fit_sample(df.drop(['Class'],axis=1),df['Class'])
```

```
y.value_counts()
```

```
1    283765
0    283765
Name: Class, dtype: int64
```

Figure 20: Balancing the dataset

4.2.1 Splitting the data.

- **Splitting the data :** after the preprocessing is done then the data is split into train and test sets.
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)

- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%) .
- First we need to identify the input and output variables and we need to separate the input set and output set.
- In scikit learn library we have a package called model_selection in which train_test_split method is available .we need to import this method.
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables) .

```
#Splitting the dataset into training and test data.
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(454024, 30)
(113506, 30)
(454024,)
(113506,)
```

Figure 21: importing train_test_split and splitting the data.

- Then we need to import logistic regression method from linear_model package from scikit learn library
- We need to train the model based on our train set (that we have obtained from splitting)
- Then we have to test the model for the test set ,that is done as follows
    - We have a method called predict , using this method we need to predict the output for the input test set and we need to compare the output with the output test data.
    - If the predicted values and the original values are close then we can say that model is trained with good accuracy .

4.2.2 Metrics:

Confusion Matrix:

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

	Class 1 Predicted	*Class 2 Predicted*
Class 1 Actual	**TP**	**FN**
Class 2 Actual	**FP**	**TN**

From In the project we can see that that data is highly unbalanced and there are more number of normal (Genuine) transactions than the Fraud transactions so in this case we are considering "F1-Score" as the metrics .

- **F1-Score**→ The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal.

The formula for the F1 score is: **F1 = 2 * (Precision * Recall) / (Precision + Recall)**

4.3 Model Building and Evaluation

4.3.1 Logistic regression

Logistic Regression is used when the dependent variable (target) is categorical. Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability

Example: Yes or No, get a disease or not, pass or fail, defective or non-defective, etc.,

Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression() # creating an object for Logistic Regression
# we have to apply this object(Log_reg) to the training data
final_model1 = log_reg.fit(X_train,y_train) # with the help of fit method we are fitting logistic regression with training da
##objectName.fit(InputData, outputData)
```

Figure 22: Applying logistic regression on training data.

Instead of directly predicting on test data, let us see how well the model predicts the training data.

Predicting on training data

```
y_train_pred = log_reg.predict(X_train)
y_train_pred
```

```
array([0, 0, 1, ..., 1, 1, 0], dtype=int64)
```

Figure 23: Predicting on train data

```
y_train == y_train_pred # comparing original data o/p and model predicted o/p
```

```
119440    True
96043     True
369660    True
207082    True
433513    True
          ...
371403    True
491263    True
470924    True
491755    True
128037    True
Name: Class, Length: 454024, dtype: bool
```

Figure 24: comparing the predicted value with the original one

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_train,y_train_pred)
```

```
array([[222404,   4496],
       [  8081, 219043]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred)
```

```
0.9722988212076895
```

Figure 25: Applying the metrics on training data.

Predicting on test data

```
# Predicting the model on test data
y_test_pred = log_reg.predict(X_test)
```

```
y_test_pred
```

```
array([1, 1, 0, ..., 1, 1, 1], dtype=int64)
```

Figure 26: Predicting on test data.

```
y_test == y_test_pred
```

```
395737    True
473850    True
50426     True
344637    True
148214    True
          ...
234610    True
169248    True
508560    True
336336    True
504712    True
Name: Class, Length: 113506, dtype: bool
```

Figure 27: Comparing the predicted value with the original test data.

```
confusion_matrix(y_test,y_test_pred)
```

```
array([[55727,  1138],
       [ 1959, 54682]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred)
```

```
0.9727150987612989
```

Figure 28: Applying metrics on test data

```
#classification report on training and test data
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred))
print("-------------------------------------------------------")
print(classification_report(y_test,y_test_pred))
```

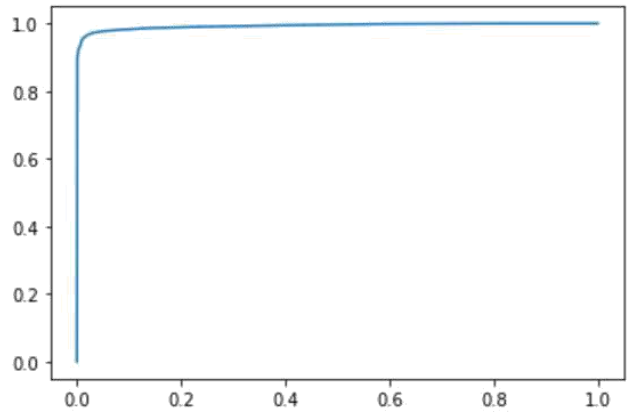
```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97    226900
           1       0.98      0.96      0.97    227124

    accuracy                           0.97    454024
   macro avg       0.97      0.97      0.97    454024
weighted avg       0.97      0.97      0.97    454024

-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     56865
           1       0.98      0.97      0.97     56641

    accuracy                           0.97    113506
   macro avg       0.97      0.97      0.97    113506
weighted avg       0.97      0.97      0.97    113506
```

Figure 29: Performance of the logistic regression model based on training and test data.

```
from sklearn.metrics import roc_auc_score, roc_curve
fraud_prob1 = final_model1.predict_proba(X_test)[:,1]
fpr1, tpr1, threshold1 = roc_curve(y_test, fraud_prob1)
```

```
plt.plot(fpr1, tpr1)
```

```
[<matplotlib.lines.Line2D at 0x2330973bc08>]
```




```
roc_auc_score(y_test, fraud_prob1)
```

```
0.9930467703750365
```

Figure 30: Measuring the accuracy of logistic regression model using the Area Under the Precision-Recall Curve (AUPRC).

4.3.2 Random forest classification

Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or the same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithms of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.

4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

```
#import initialize and fit
#import the RFC from sklearn
from sklearn.ensemble import RandomForestClassifier

#initialize the object for RFC
rfc = RandomForestClassifier()

#fit RFC to dataset
final_model2 = rfc.fit(X_train,y_train)
```

Figure 31: Applying random forest classifier on the training data.

Predicting on training data

```
y_train_pred1 = rfc.predict(X_train) #Predicting on training data
```

```
confusion_matrix(y_train,y_train_pred1)
```

```
array([[226900,      0],
       [     0, 227124]], dtype=int64)
```

```
accuracy_score(y_train,y_train_pred1)
```

```
1.0
```

Figure 32: Prediction and applying the metrics on train data.

Predicting on test data

```
y_test_pred1 = rfc.predict(X_test) #Predicting on test data
```

```
confusion_matrix(y_test,y_test_pred1)
```

```
array([[56851,    14],
       [    0, 56641]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred1)
```

```
0.9998766585026342
```

Figure 33: Prediction and applying the metrics on test data.

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred1))
print("-----------------------------------------------------")
print(classification_report(y_test,y_test_pred1))
```

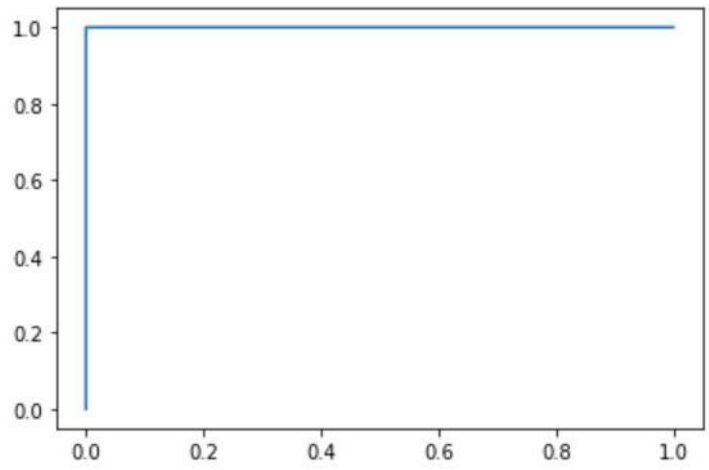
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226900
           1       1.00      1.00      1.00    227124

    accuracy                           1.00    454024
   macro avg       1.00      1.00      1.00    454024
weighted avg       1.00      1.00      1.00    454024

-----------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56865
           1       1.00      1.00      1.00     56641

    accuracy                           1.00    113506
   macro avg       1.00      1.00      1.00    113506
weighted avg       1.00      1.00      1.00    113506
```

Figure 34: Overall performance of the random forest classifier model based on training and test data.

```
from sklearn.metrics import roc_auc_score, roc_curve
fraud_prob2 = final_model2.predict_proba(X_test)[:,1]
fpr2, tpr2, threshold2 = roc_curve(y_test, fraud_prob2)
```

```
plt.plot(fpr2, tpr2)
```

```
[<matplotlib.lines.Line2D at 0x23309ceb788>]
```




```
roc_auc_score(y_test, fraud_prob2)
```

```
0.9999996836278628
```

Figure 35: Measuring the accuracy of a random forest classifier model using the Area Under the Precision-Recall Curve (AUPRC).

4.3.3 Naive Bayes

Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of probability for prediction of unknown class.

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location. Even if these features are interdependent, these features are still considered independently. This assumption simplifies

computation, and that's why it is considered as naive. This assumption is called class conditional independence.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D|h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

```
from sklearn.naive_bayes import GaussianNB
gn = GaussianNB()
final_model3 = gn.fit(X_train,y_train)
y_train_pred2 = gn.predict(X_train)
```

Figure 36: Applying naive bayes algorithm on training data.

Predicting on training data

```
confusion_matrix(y_train,y_train_pred2)
```

```
array([[225269,   1631],
       [ 58254, 168870]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred2)
```

```
0.8681016862544711
```

Figure 37: Applying metrics on training data.

Predicting on training data

```
y_test_pred2 = gn.predict(X_test)
```

```
confusion_matrix(y_test,y_test_pred2)
```

```
array([[56433,   432],
       [14546, 42095]], dtype=int64)
```

```
accuracy_score(y_test,y_test_pred2)
```

```
0.8680422180325269
```

Figure 38: Applying metrics on test data.

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,y_train_pred2))
print("-------------------------------------------------------")
print(classification_report(y_test,y_test_pred2))
```

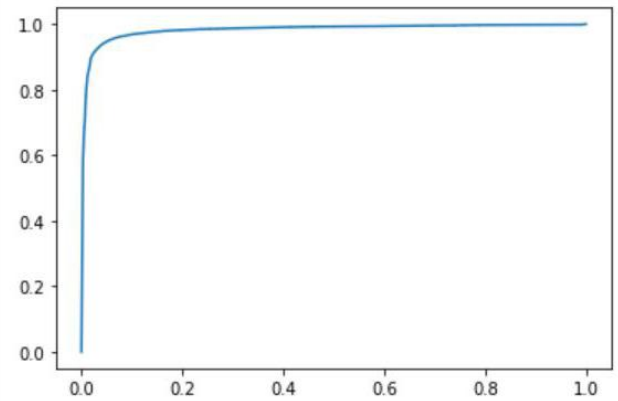
```
              precision    recall  f1-score   support

           0       0.79      0.99      0.88    226900
           1       0.99      0.74      0.85    227124

    accuracy                           0.87    454024
   macro avg       0.89      0.87      0.87    454024
weighted avg       0.89      0.87      0.87    454024

-------------------------------------------------------
              precision    recall  f1-score   support

           0       0.80      0.99      0.88     56865
           1       0.99      0.74      0.85     56641

    accuracy                           0.87    113506
   macro avg       0.89      0.87      0.87    113506
weighted avg       0.89      0.87      0.87    113506
```

Figure 39: Performance of the naive bayes model based on training and test data.

```
#1-->fraud 0-->genuine
# Roc curve
## TPR, FPR, Threshold
from sklearn.metrics import roc_auc_score, roc_curve
fraud_prob3 = final_model3.predict_proba(X_test)[:,1]
fpr3, tpr3, threshold3 = roc_curve(y_test, fraud_prob3)
```

```
plt.plot(fpr3, tpr3)
```

```
[<matplotlib.lines.Line2D at 0x2330b12ac48>]
```




```
roc_auc_score(y_test, fraud_prob3)
```

```
0.9825532882565753
```

Figure 40: Measuring the accuracy of naive bayes model using the Area Under the Precision-Recall Curve (AUPRC).

4.4 Visualizing the best model among logistic regression, Random forest and Naive Bayes

```
models = ['Logistic Regression','Random forest','NaiveBayes']
f1_scores = [0.972461564453455,0.9998764298827849,0.8489633752823492]
plt.bar(models,f1_scores,color=['lightblue','pink','lightgrey'])
plt.ylabel("f1 scores")
plt.title("which model has high accuracy")
plt.show()
```

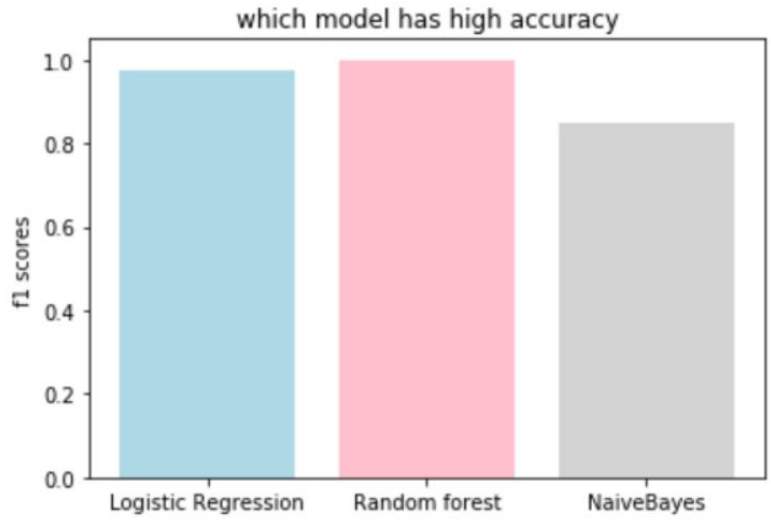


Figure 41: Comparison Of Applied Model

--> Observations: From the above graph we can say that the ***"Random Forest Classifier"*** is showing the highest accuracy

5. Conclusion:

From the Machine Learning model building and evaluation we can say that the "***Random Forest algorithm"*** is best predicting model for fraud and genuine transactions.

6. References:

https://en.wikipedia.org/wiki/Machine_learning

https://www.kaggle.com/mlg-ulb/creditcardfraud

https://builtin.com/data-science/random-forest-algorithm