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

Vaishnavi Batchu

221710304062

Under the Guidance of

Mr. M. Venkateswarlu

Assistant Professor

Department Of ComputerScience 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 fulfilment 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 Mr. M.Venkateswarlu, 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 **Vaishnavi Batchu**

Date: 13-07-2020 **221710304062**

CERTIFICATE

This is to certify that the Industrial Training Report entitled "CREDIT CARD FRAUDDETECTION" is being submitted by Vaishnavi Batchu(221710304062) in partial fulfilment 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.

Mr. M. Venkateswarlu	**Dr.K.Manjunathachari**
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 **Dr. CH. Sanjay,** Principal, GITAM Hyderabad.

I would like to thank respected **Dr. K. Manjunathachari**, 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 **Mr. M. Venkateswarlu** 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.

VAISHNAVI BATCHU

221710304062

ABSTRACT

With the advent of fast-moving information technology everybody wishes to keep his information in the public domain i.e. either on internet or on intranet. We usually ignore the security threats and important network safety concerns with the informative data transmitted through the web. In the last two years many cases of credit card thefts have been reported to the cyber security department in London. We all use internet banking and credit card for online shopping. It is possible for someone standing a meter away to note your password without your knowledge. There are several solutions available in the market to ensure your cyber safety.

An internet user must be cautious about his private security from any cyber offense, scam and personal identity theft. We should be cautious of any small transaction done through our card. Anti-Glare Frameless Privacy Filters can help you get saved from credit card frauds. These screens are available for notebooks, laptops and flat screen monitors. These screens also reduce glare and protect the computer screen from any possible scratches.

This paper investigates the performance of naïve Bayes,random forest classifier, k-nearest neighbour and logistic regression on highly skewed credit card fraud data. Dataset of credit card transactions is sourced from European cardholders containing 284,079 transactions. A hybrid technique of under-sampling and oversampling is carried out on the skewed data. The four techniques are applied on the raw and pre-processed data. The work is implemented in Python. The performance of the techniques is evaluated based on accuracy, precision, f1-score and AUCROC curve.

VAISHNAVI BATCHU

221710304062

Table of Contents

List of Figures:

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

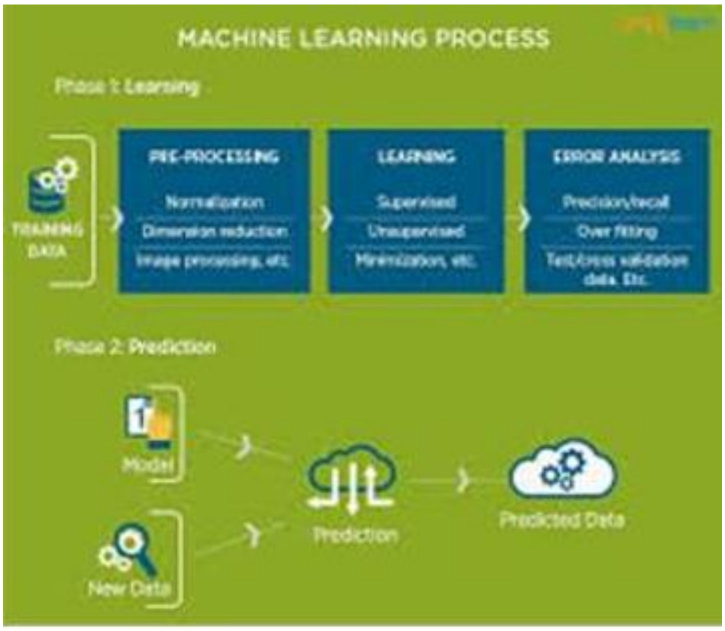


Figure 1 : The Process Flow

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.

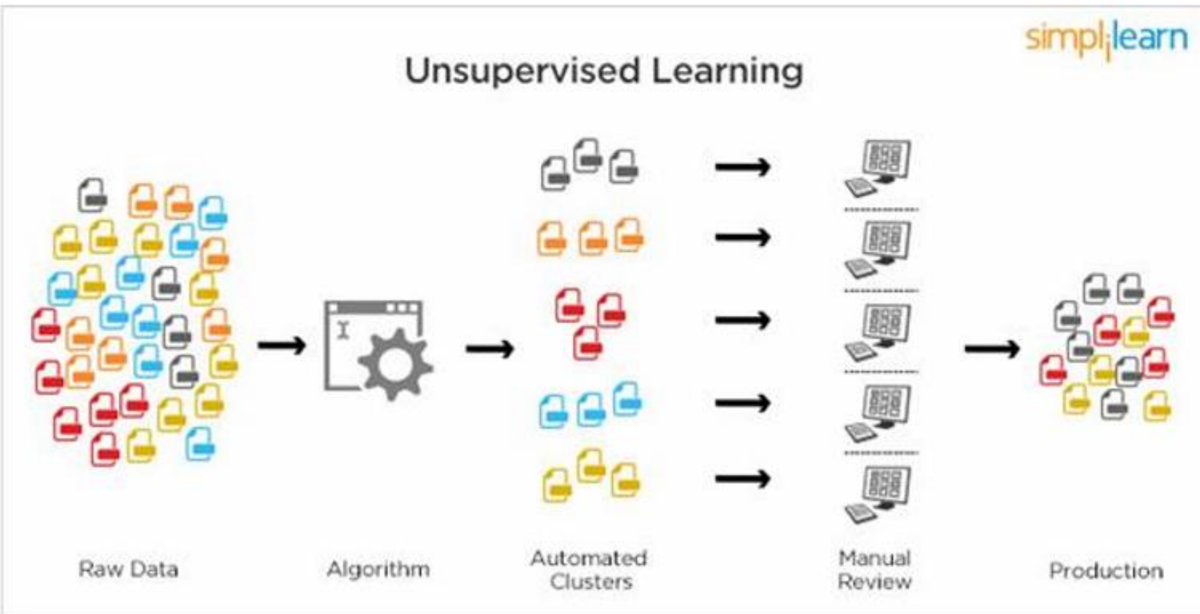


Figure 2 : Unsupervised Learning.

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.

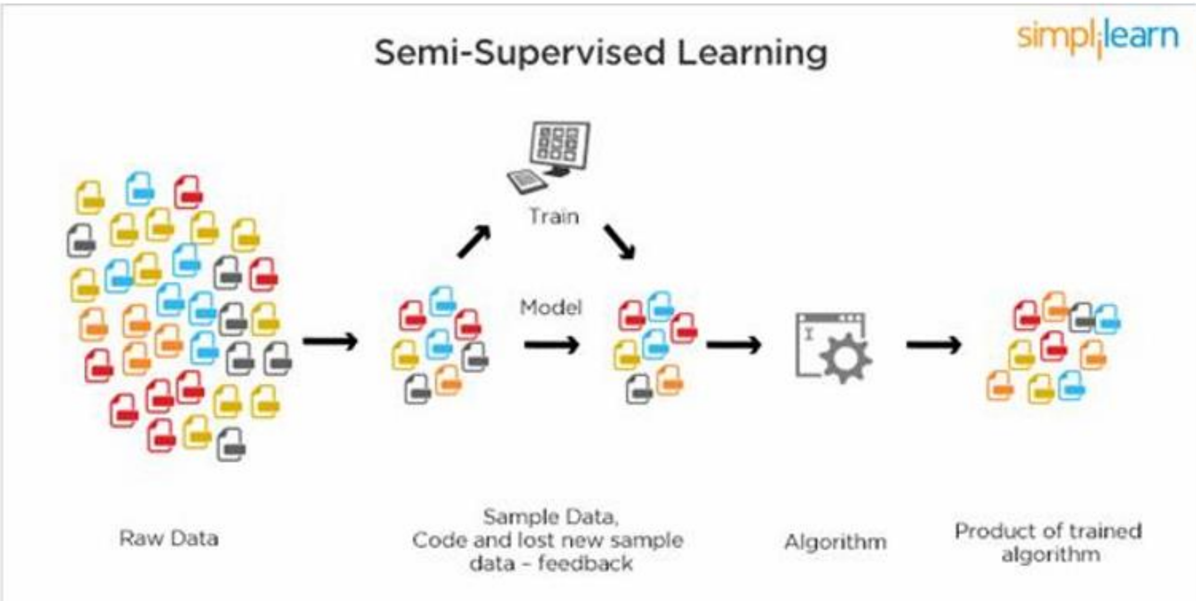


Figure 3 : Semi Supervised Learning

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.

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

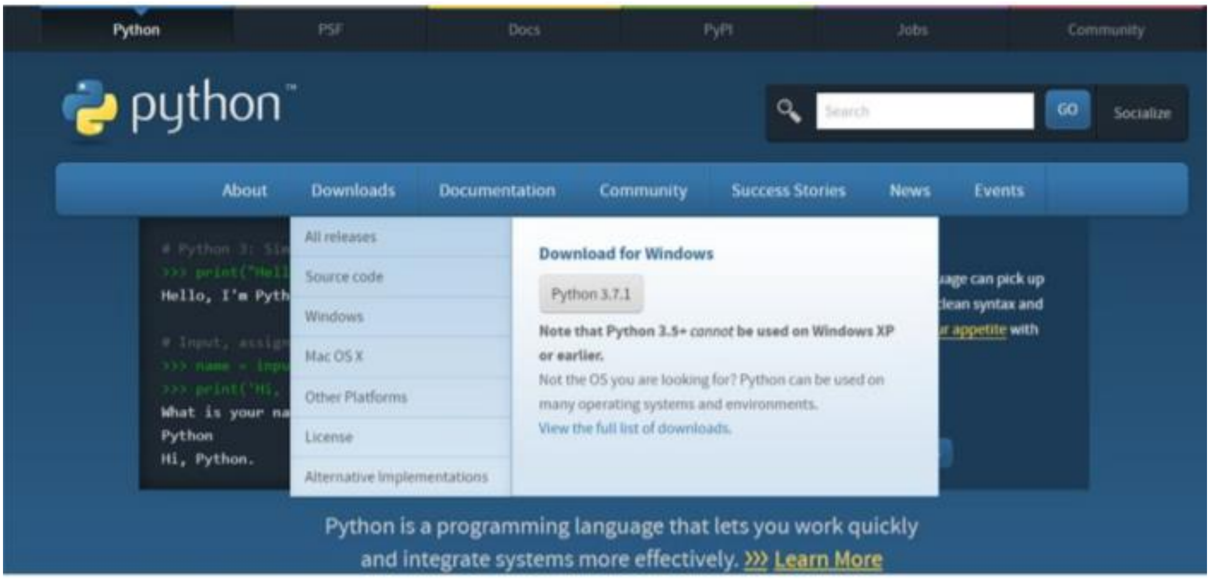


Figure 4: Python download

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:
- Step 1: Open Anaconda.com/downloads in a 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)

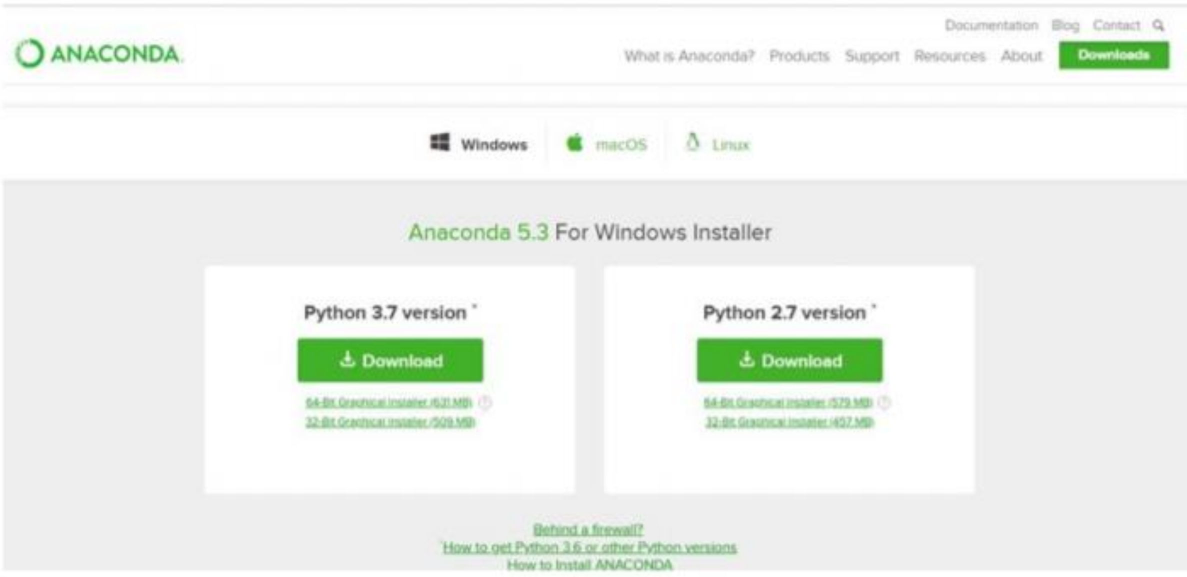


Figure 5: Anaconda download

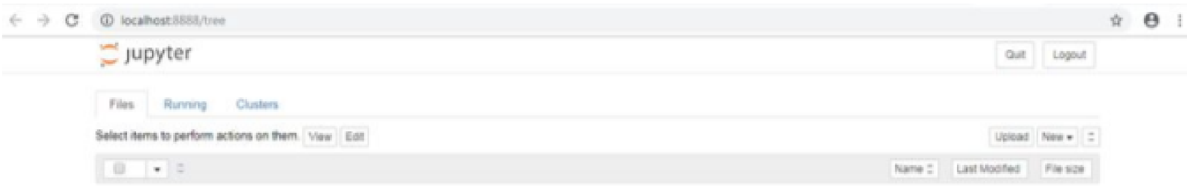


Figure 6: Jupyter notebook

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: o We define a class in a very similar way how we define a function. o 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 function 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__().

3.CASE STUDY

3.1 PROBLEM STATEMENT:

The Credit Card Fraud Detection Problem includes modeling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not. Our aim here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

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 284,079 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.173% 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:

The goal of the credit card fraud detection system is to maximize true positive and minimize false positive predictions of legitimate transactions. The main contribution of this research is to identify the frequently occurred credit card frauds and methods committed to obtain credit card information illegally.

Our objective here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification. In this process, we have focused on analysing and pre-processing data sets as well as the deployment of multiple anomaly detection algorithms such as Local Outlier Factor and Isolation Forest algorithm on the PCA transformed Credit Card Transaction data.

4.MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Pre-processing 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.

```
#importing the required packages
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Versions of packages

```
#Checking the versions of the packages imported
import numpy
import matplotlib
print('numpy:',numpy.__version__)
print('pandas:',pd.__version__)
print('seaborn:',sns.__version__)
print('matplotlib:',matplotlib.__version__)
```

```
numpy: 1.16.5
pandas: 0.25.1
seaborn: 0.9.0
matplotlib: 3.1.1
```

Figure 8: Importing Libraries

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 Data Frame 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 dataset using pandas
data = pd.read_csv("creditcard1.csv")
data.head()  #To display top 5 rows of the dataset.
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270

5 rows × 31 columns

Figure 9: Reading the dataset

```
#checking the total number of columns and rows in the given dataset.
data.shape
```

```
(284079, 31)
```

```
#To display all the column names in the dataset
data.columns
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1
0',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V
20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Figure 10: Analysing the features of the dataset

```
#To display the information of the columns.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284079 entries, 0 to 284078
Data columns (total 31 columns):
Time      284079 non-null float64
V1        284079 non-null float64
V2        284079 non-null float64
V3        284079 non-null float64
V4        284079 non-null float64
V5        284079 non-null float64
V6        284079 non-null float64
V7        284079 non-null float64
V8        284079 non-null float64
V9        284079 non-null float64
V10       284079 non-null float64
V11       284079 non-null float64
V12       284079 non-null float64
V13       284079 non-null float64
V14       284079 non-null float64
V15       284079 non-null float64
V16       284079 non-null float64
V17       284079 non-null float64
V18       284079 non-null float64
V19       284079 non-null float64
V20       284079 non-null float64
V21       284079 non-null float64
V22       284079 non-null float64
V23       284079 non-null float64
V24       284079 non-null float64
V25       284079 non-null float64
V26       284079 non-null float64
V27       284079 non-null float64
V28       284079 non-null float64
Amount    284079 non-null float64
Class     284079 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.2 MB
```

Figure 11: Information of the columns

```
#To display the statistical information of the columns
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Time	284079.0	9.461489e+04	47385.817047	0.000000	54116.000000	84558.000000	139096.000000	172136.000000
V1	284079.0	1.664778e-04	1.958508	-56.407510	-0.920166	0.018583	1.315363	2.454930
V2	284079.0	-1.035871e-03	1.651394	-72.715728	-0.598942	0.065034	0.803043	22.057729
V3	284079.0	1.886829e-03	1.515581	-48.325589	-0.887673	0.181830	1.028302	9.382558
V4	284079.0	3.243236e-04	1.415728	-5.683171	-0.848578	-0.019310	0.744225	16.875344
V5	284079.0	-5.864321e-04	1.378626	-113.743307	-0.692343	-0.055104	0.611071	34.801666
V6	284079.0	2.464160e-04	1.331494	-26.160506	-0.767913	-0.273709	0.398966	73.301626
V7	284079.0	-4.844189e-04	1.234843	-43.557242	-0.554172	0.039755	0.570017	120.589494
V8	284079.0	-3.234414e-04	1.194679	-73.216718	-0.208497	0.022384	0.327114	20.007208
V9	284079.0	-1.336285e-04	1.098633	-13.434066	-0.643442	-0.051715	0.597270	15.594995
V10	284079.0	-5.556518e-05	1.088098	-24.588262	-0.535205	-0.092777	0.454155	23.745136
V11	284079.0	1.033724e-03	1.020768	-4.797473	-0.761462	-0.031380	0.740437	12.018913
V12	284079.0	-5.616045e-04	0.999805	-18.683715	-0.406279	0.139628	0.618200	7.848392
V13	284079.0	6.002220e-05	0.995510	-5.791881	-0.648567	-0.013626	0.662532	7.126883
V14	284079.0	3.602050e-04	0.958662	-19.214325	-0.425077	0.050806	0.493186	10.526766
V15	284079.0	4.598833e-04	0.915541	-4.498945	-0.582347	0.048756	0.649746	8.877742
V16	284079.0	-4.323618e-05	0.876327	-14.129855	-0.468306	0.066318	0.523414	17.315112
V17	284079.0	9.132299e-05	0.849565	-25.162799	-0.483593	-0.065539	0.399722	9.253526
V18	284079.0	-3.831342e-04	0.838333	-9.498746	-0.498850	-0.003997	0.500487	5.041069
V19	284079.0	8.474311e-06	0.814041	-7.213527	-0.456434	0.003772	0.459145	5.591971
V20	284079.0	2.406212e-07	0.771349	-54.497720	-0.211748	-0.062421	0.133133	39.420904
V21	284079.0	-1.140360e-05	0.735091	-34.830382	-0.228332	-0.029452	0.186196	27.202839
V22	284079.0	-1.834385e-04	0.725602	-10.933144	-0.542194	0.006675	0.528046	10.503090
V23	284079.0	-1.130614e-04	0.624914	-44.807735	-0.161900	-0.011293	0.147478	22.528412
V24	284079.0	1.417645e-05	0.605713	-2.836627	-0.354549	0.041082	0.439421	4.584549
V25	284079.0	3.324193e-04	0.521114	-10.295397	-0.316752	0.017122	0.350947	7.519589
V26	284079.0	-2.255503e-05	0.482266	-2.604551	-0.327054	-0.052293	0.241202	3.517346
V27	284079.0	-5.674385e-06	0.403565	-22.565679	-0.070835	0.001368	0.091007	31.612198
V28	284079.0	1.189142e-05	0.330164	-15.430084	-0.052941	0.011292	0.078257	33.847808
Amount	284079.0	8.838021e+01	249.624128	0.000000	5.600000	22.000000	77.300000	25691.160000
Class	284079.0	1.731913e-03	0.041580	0.000000	0.000000	0.000000	0.000000	1.000000

Figure 12: Statistical information of the data

```
#Barplot showing the frequency of normal and fraud transactions.
LABELS = ["Normal" , "Fraud"]
count_classes = pd.value_counts(data['Class'],sort=True)
count_classes.plot(kind = 'barh')
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("frequency")
```

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

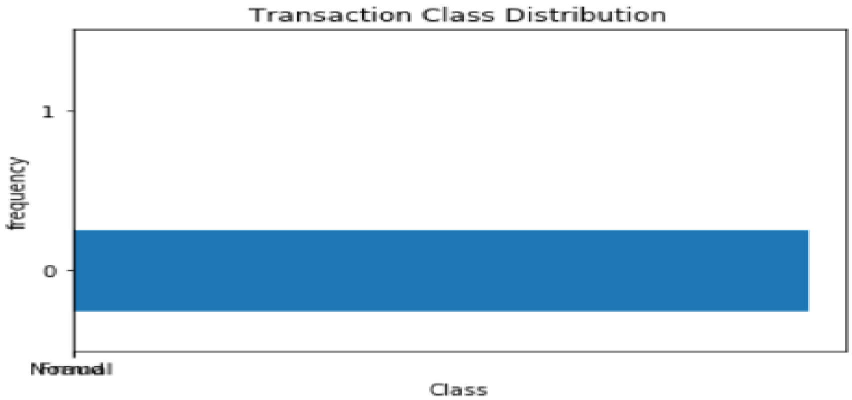


Figure 13: Horizontal Bar plot for class and frequency.

- From above graph we can see that the data is highly unbalanced.

```
#Relation plot for normal and fraud transactions
sns.relplot(x='Amount',y='Time',hue='Class',data=data)
```

```
<seaborn.axisgrid.FacetGrid at 0x2459c879d48>
```

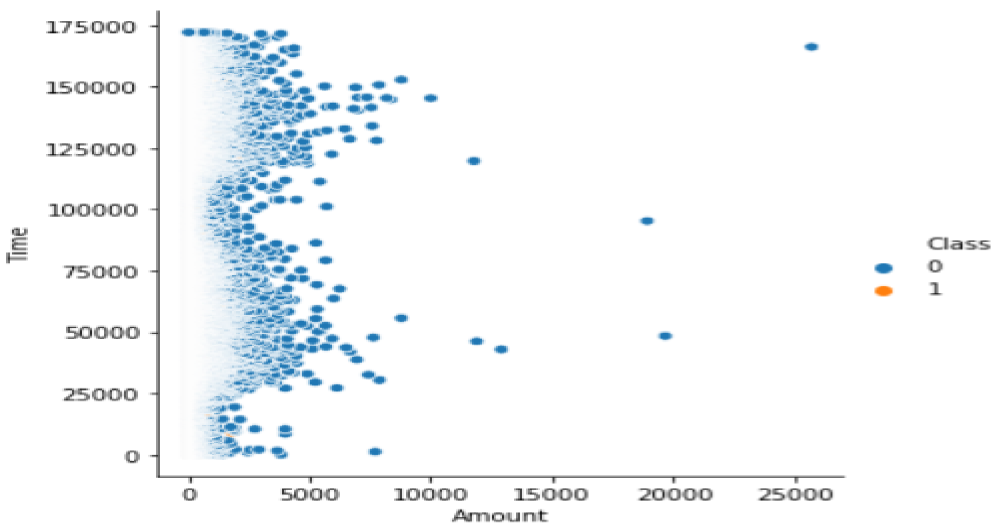


Figure 14: Relation plot for fraud and normal classes

- 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.

```
fig = plt.figure(figsize = (15, 12))

plt.subplot(5, 6, 1) ; plt.plot(data.V1) ; plt.subplot(5, 6, 15) ; plt.plot(data.V15)
plt.subplot(5, 6, 2) ; plt.plot(data.V2) ; plt.subplot(5, 6, 16) ; plt.plot(data.V16)
plt.subplot(5, 6, 3) ; plt.plot(data.V3) ; plt.subplot(5, 6, 17) ; plt.plot(data.V17)
plt.subplot(5, 6, 4) ; plt.plot(data.V4) ; plt.subplot(5, 6, 18) ; plt.plot(data.V18)
plt.subplot(5, 6, 5) ; plt.plot(data.V5) ; plt.subplot(5, 6, 19) ; plt.plot(data.V19)
plt.subplot(5, 6, 6) ; plt.plot(data.V6) ; plt.subplot(5, 6, 20) ; plt.plot(data.V20)
plt.subplot(5, 6, 7) ; plt.plot(data.V7) ; plt.subplot(5, 6, 21) ; plt.plot(data.V21)
plt.subplot(5, 6, 8) ; plt.plot(data.V8) ; plt.subplot(5, 6, 22) ; plt.plot(data.V22)
plt.subplot(5, 6, 9) ; plt.plot(data.V9) ; plt.subplot(5, 6, 23) ; plt.plot(data.V23)
plt.subplot(5, 6, 10) ; plt.plot(data.V10) ; plt.subplot(5, 6, 24) ; plt.plot(data.V24)
plt.subplot(5, 6, 11) ; plt.plot(data.V11) ; plt.subplot(5, 6, 25) ; plt.plot(data.V25)
plt.subplot(5, 6, 12) ; plt.plot(data.V12) ; plt.subplot(5, 6, 26) ; plt.plot(data.V26)
plt.subplot(5, 6, 13) ; plt.plot(data.V13) ; plt.subplot(5, 6, 27) ; plt.plot(data.V27)
plt.subplot(5, 6, 14) ; plt.plot(data.V14) ; plt.subplot(5, 6, 28) ; plt.plot(data.V28)
plt.subplot(5, 6, 29) ; plt.plot(data.Amount)
plt.show()
```

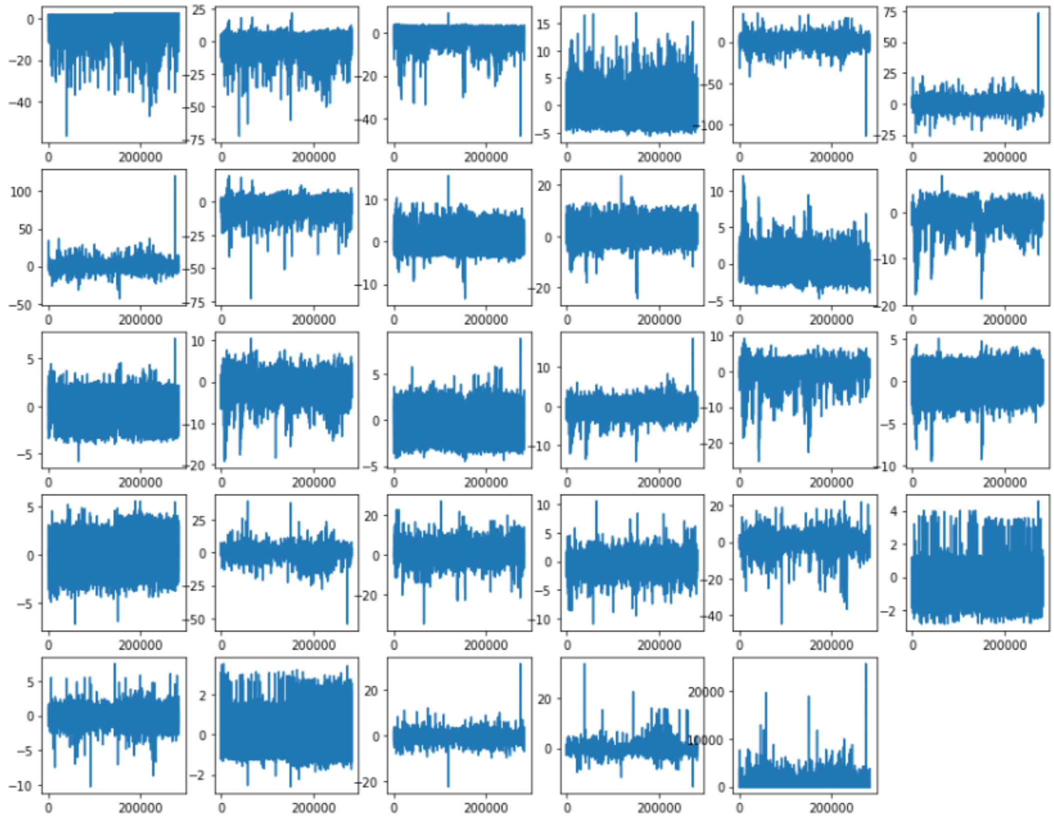
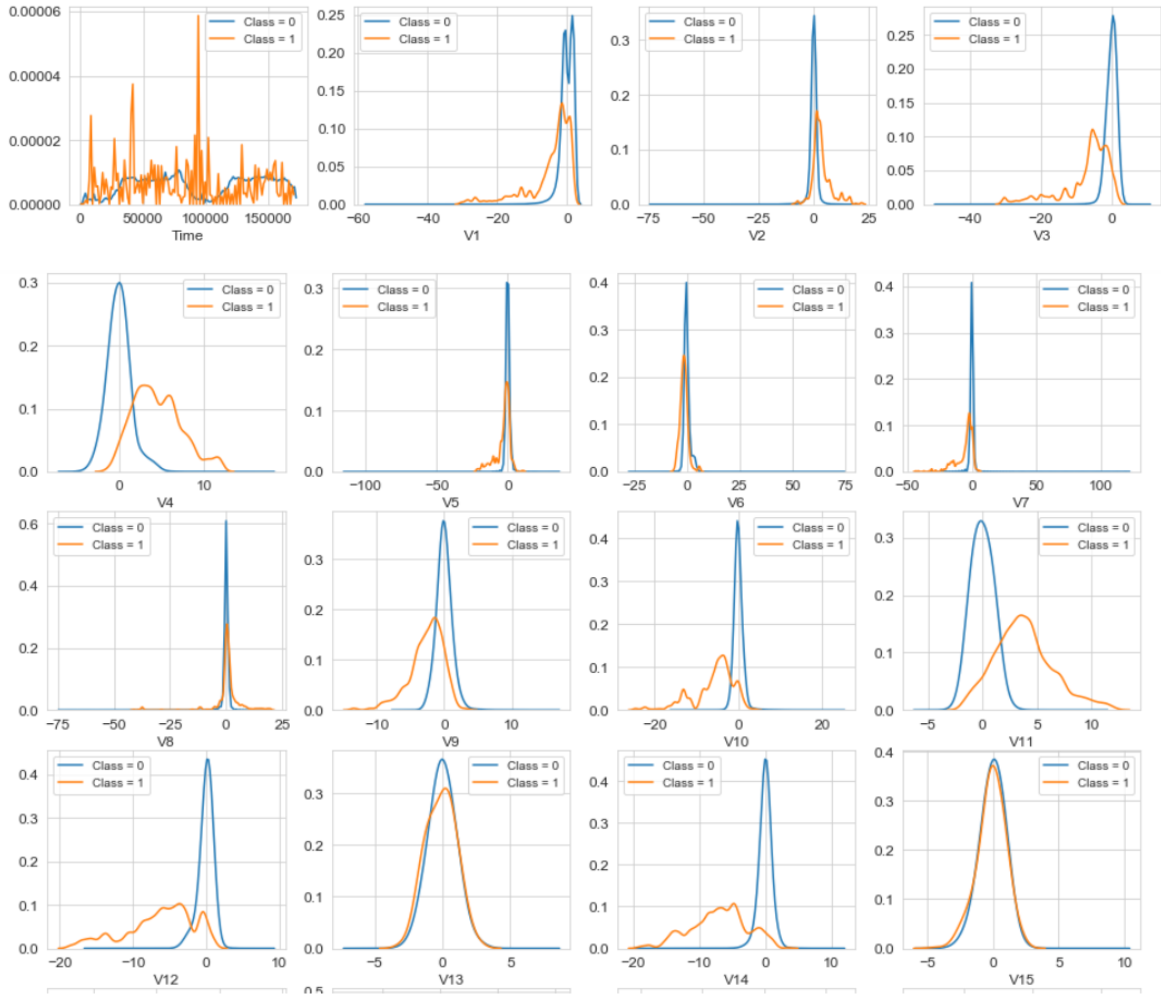


Figure 15: Subplots visualizing all the V features of the transactions.

```
var = data.columns.values

i = 0
normal = data.loc[data['Class'] == 0]
fraud = data.loc[data['Class'] == 1]

sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:
    i += 1
    plt.subplot(8,4,i)
    sns.kdeplot(normal[feature], bw=0.5,label="Class = 0")
    sns.kdeplot(fraud[feature], bw=0.5,label="Class = 1")
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show();
```

<Figure size 432x288 with 0 Axes>



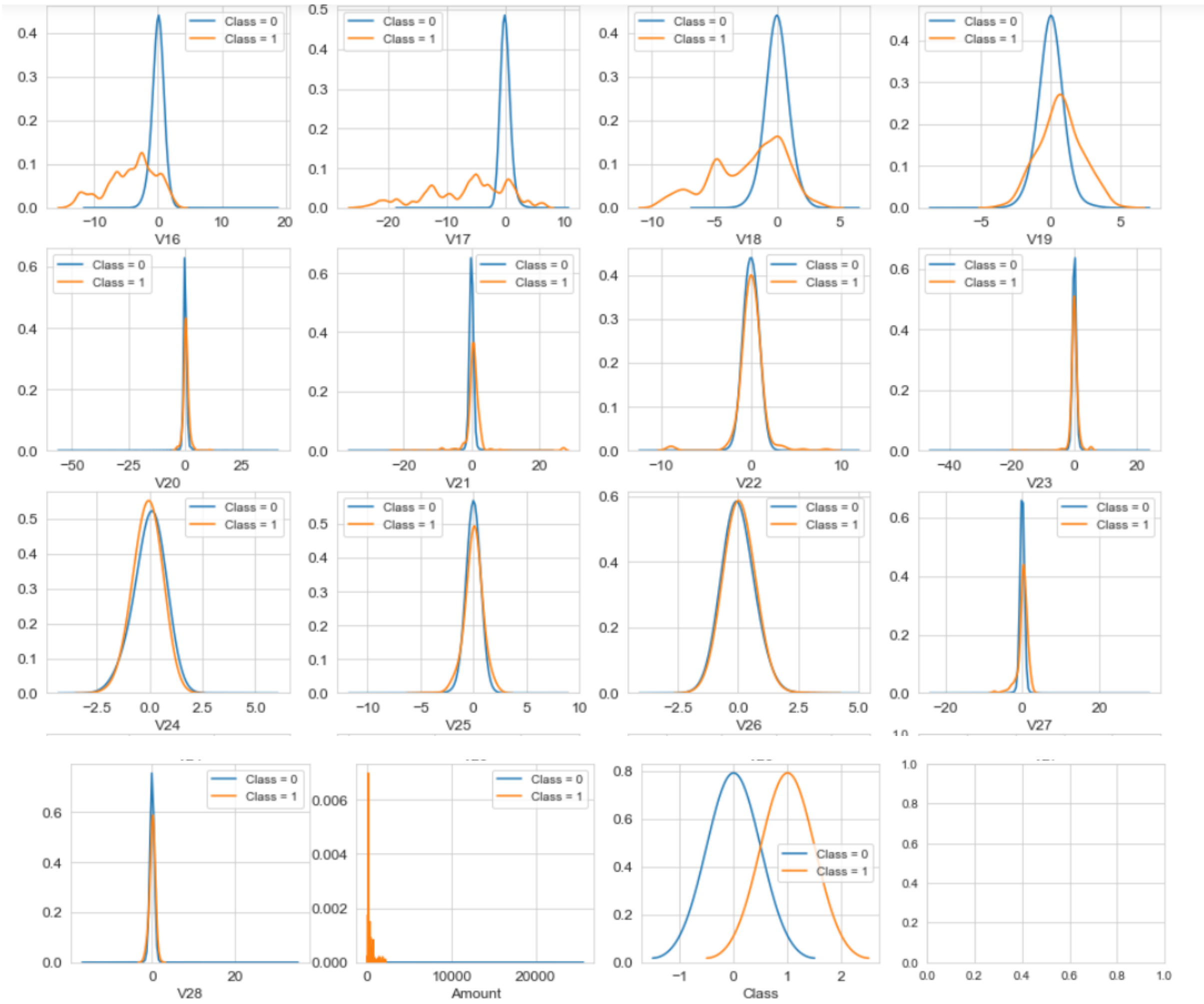


Figure 16: Feature density plot

- For some of the features we can observe a good selectivity in terms of distribution for the two values of Class: V4, V11 have clearly separated distributions for Class values 0 and 1, V12, V14, V18 are partially separated, V1, V2, V3, V10 have a quite distinct profile, whilst V25, V26, V28 have similar profiles for the two values of Class.
- In general, with just few exceptions (Time and Amount), the features distribution for legitimate transactions (values of Class = 0) is centred around 0, sometime with a long queue at one of the extremities. In the same time, the fraudulent transactions (values of Class = 1) have a skewed (asymmetric) distribution.

How different are the amount of money used in different transaction classes?

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

```
(492, 31) (283587, 31)
```

```
fraud.Amount.describe()
```

```
count      492.000000
mean       122.211321
std        256.683288
min          0.000000
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
```

```
Normal.Amount.describe()
```

```
count    283587.000000
mean         88.321519
std         249.608188
min           0.000000
25%           5.665000
50%          22.000000
75%          77.140000
max       25691.160000
Name: Amount, dtype: float64
```

Figure 17: Description of normal and fraud amounts.

```
# We Will check Do fraudulent transactions occur more often based on  certain amount ?
#Let us find out with a visual representation.
f, (ax1, ax2) = plt.subplots(2, 1,sharex = True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```

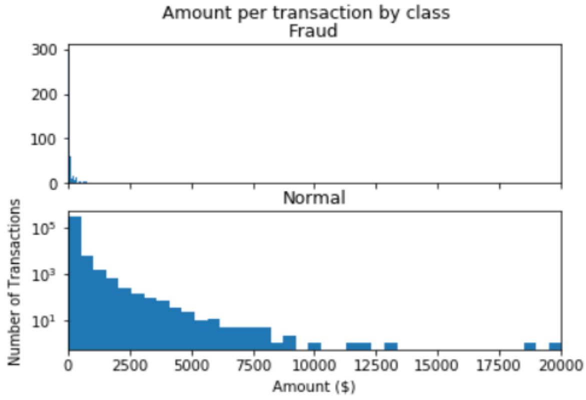


Figure 18: Graphical representation of normal and fraud amounts

```
f, (ax1, ax2) = plt.subplots(2, 1,sharex=True)
f.suptitle('Amount per transaction by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```

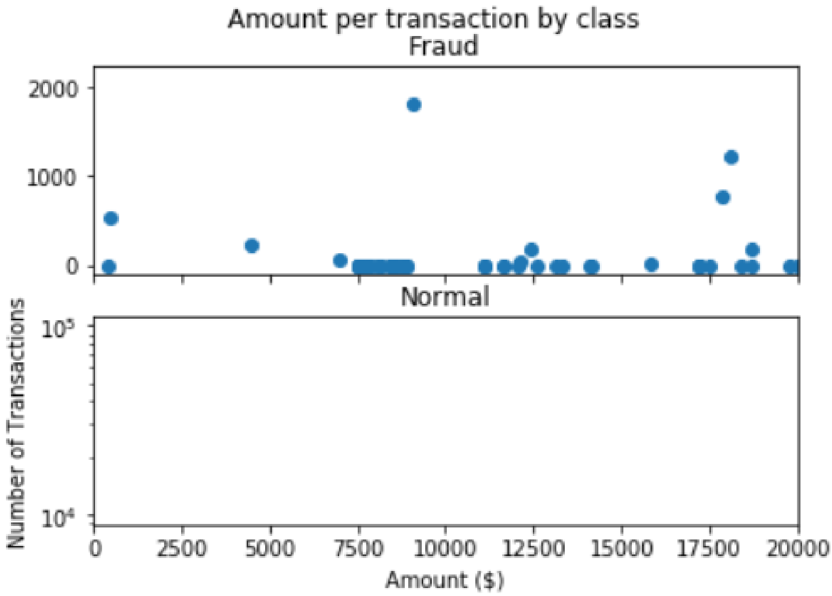


Figure 19: Scatter plot representation of fraud and normal amounts

- **We Will check Do fraudulent transactions occur more often during certain time frame ?**

```
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()
```

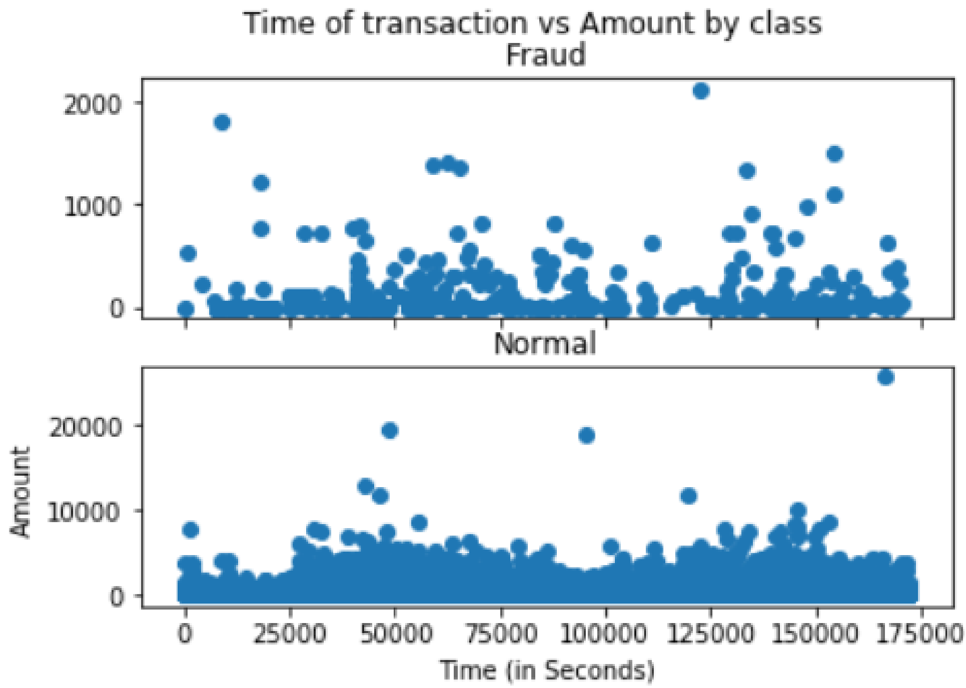
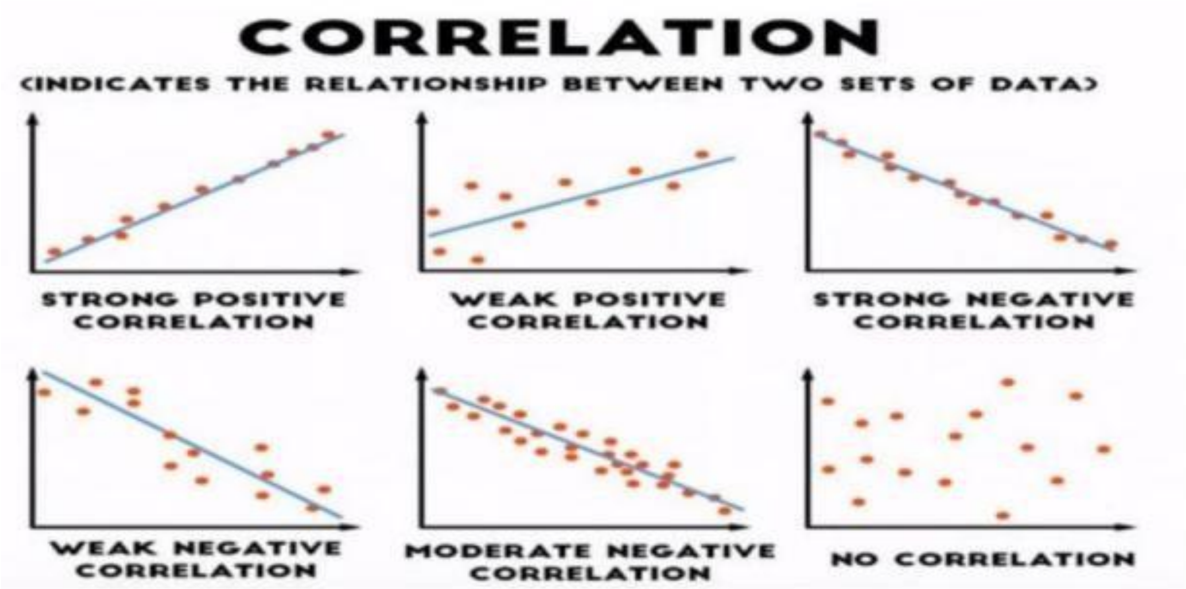


Figure 20: Scatter plot for amount and time by class

Correlation:

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. Correlation is described as the analysis which lets us know the association or the absence of the relationship between two variables ‘x’ and ‘y’. It is a statistical measure that represents the strength of the connection between pairs of variables.


CORRELATION
(INDICATES THE RELATIONSHIP BETWEEN TWO SETS OF DATA)
STRONG POSITIVE CORRELATION
WEAK POSITIVE CORRELATION
STRONG NEGATIVE CORRELATION
WEAK NEGATIVE CORRELATION
MODERATE NEGATIVE CORRELATION
NO CORRELATION

```python
fig = plt.subplots (figsize = (25, 25))
sns.heatmap(data.corr (), square = True, cbar = True, annot = True, annot_kws = {'size': 8})
plt.title('Correlations between Attributes')
plt.show ()
```

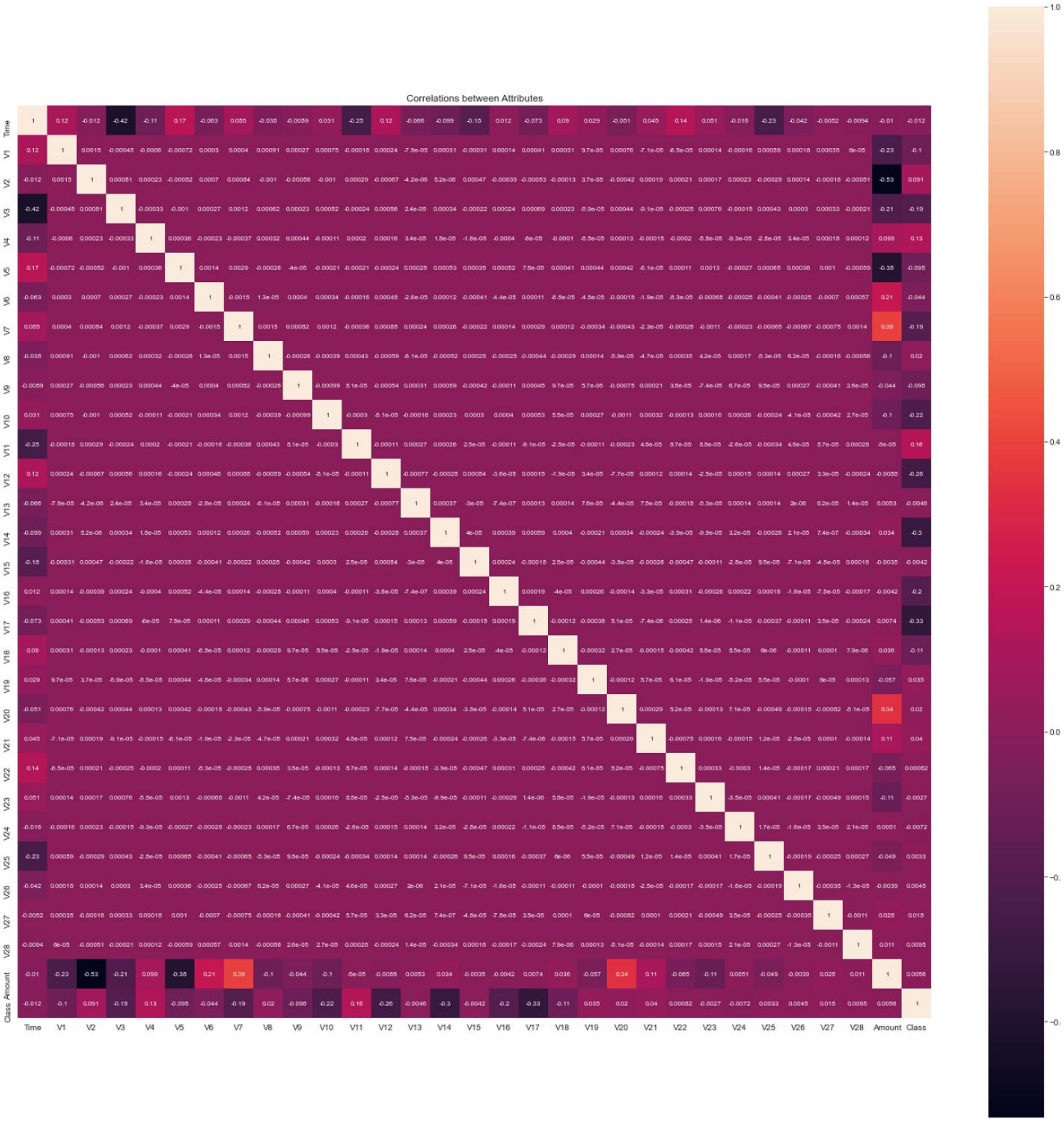


Figure 21: Correlation of features

- The above correlation matrix shows that none of the V1 to V28 PCA components have any correlation to each other however if we observe Class has some form positive and negative correlations with the V components but has no correlation with Time and Amount.

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. dropna():

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. fillna():

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. interpolate():

- 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. mean and median imputation

- 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().

```
data.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 22: Total number of missing values in each column.

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

```
#Visualization of nullvalues using heatmap
sns.heatmap(data.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2458ec3ef08>
```

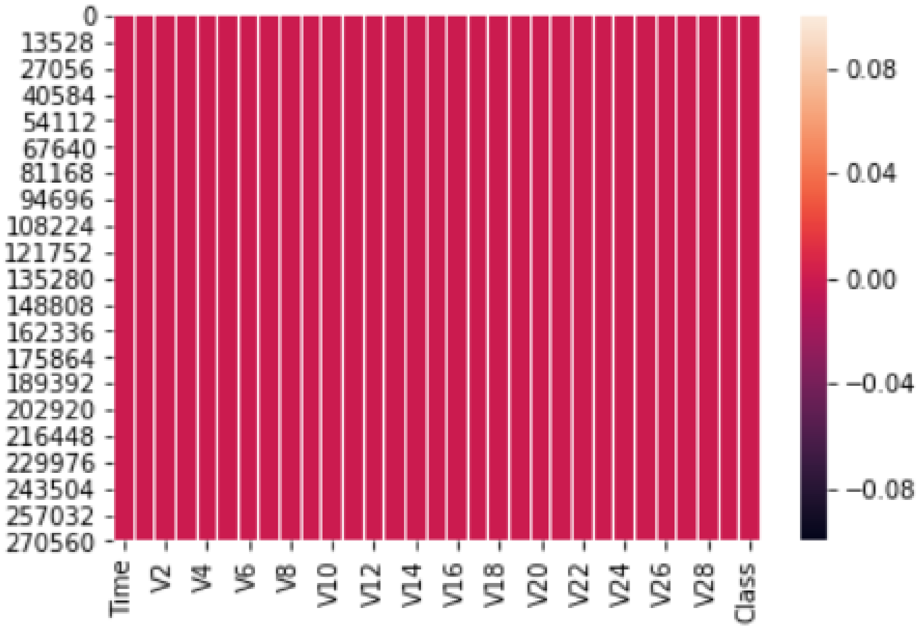


Figure 23: Visualising the missing values.

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

4.1.5 OUTLIERS:

An outlier is a data point in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset.

```
Fraud = data[data['Class']==1]
Valid = data[data['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))
```

```
print(outlier_fraction)
print("Fraud Cases : {}".format(len(Fraud)))
print("Valid Cases : {}".format(len(Valid)))
```

```
0.0017349173269578647
Fraud Cases : 492
Valid Cases : 283587
```

Figure 24: Outlier fraction

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

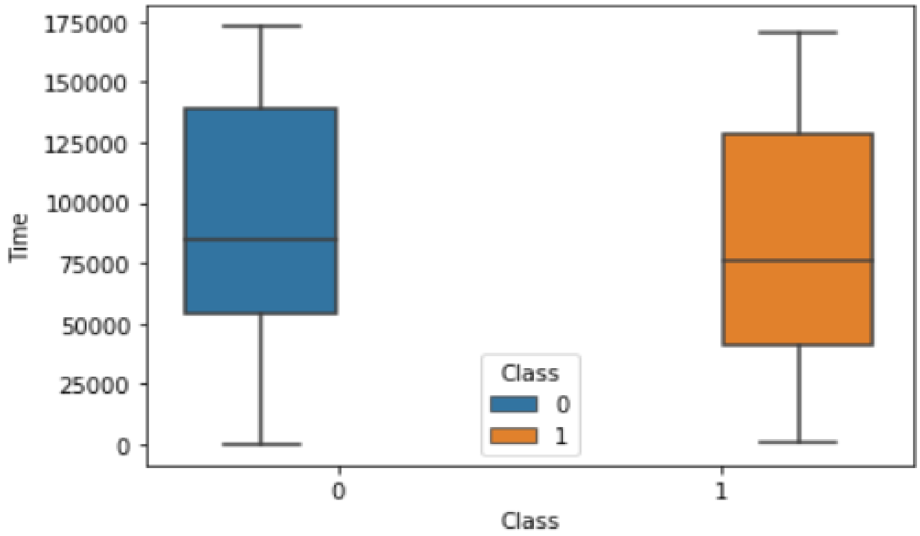


Figure 25: Box plot for class and time.

- 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 = data)
plt.ylim(0, 2500)
plt.show()
```

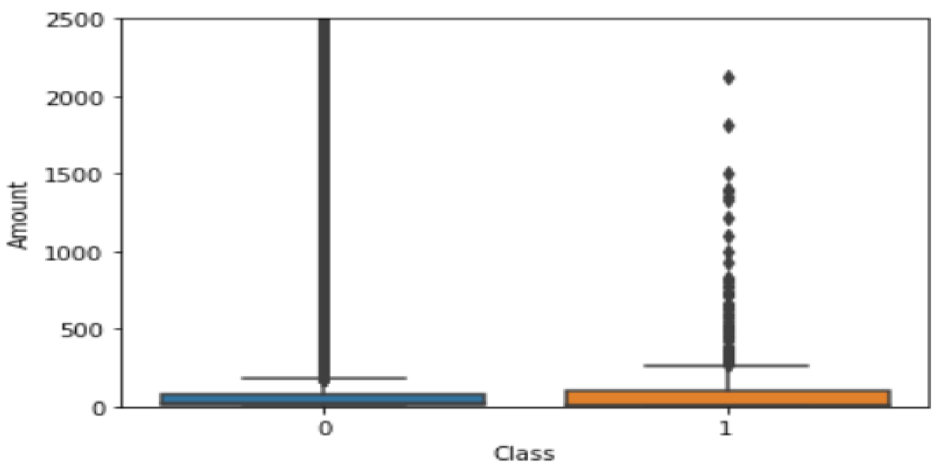


Figure 26: Boxplot for class and amount

- From above box plot we can easily infer that there are no fraud transactions occur above the transaction amount of 2300. All of the fraud transactions have transaction amount less than 2300. However, there are many transactions which have a transaction amount greater than 2300 and all of them are normal

```
data.hist(figsize=(20,20))
plt.show()
```

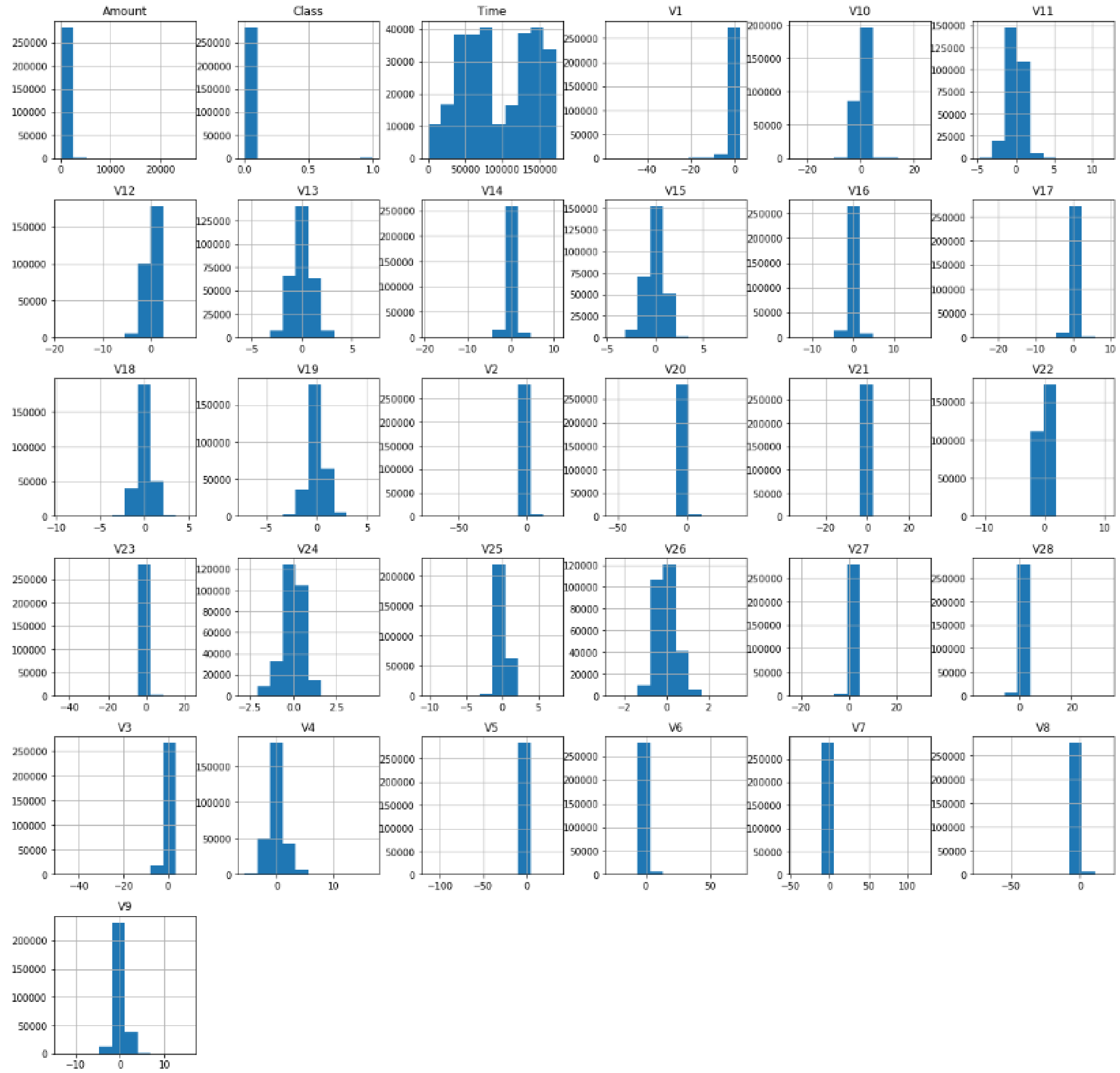


Figure 27: Histograms visualizing all the features of the dataset.

4.1.6 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.

```
data.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 28: Description about the type of each feature in the dataset.(Categorical or Numerical).

4.2 TRAINING THE MODEL:

```
#To calculate number of fraud and normal transactions
fraud = data[data['Class']==1]
Normal = data[data['Class']==0]
print(fraud.shape,Normal.shape)
```

```
(492, 31) (283587, 31)
```

Figure 29: 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(data.drop(['Class'],axis=1),data['Class'])
```

```
y.value_counts()
```

```
1    283045
0    283045
Name: Class, dtype: int64
```

Figure 30 :Balancing the dataset

4.2.1 Splitting the data.

- **Splitting the data :** after the pre-processing 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)
```

```
(452872, 30)
(113218, 30)
(452872,)
(113218,)
```

Figure 31: 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 .

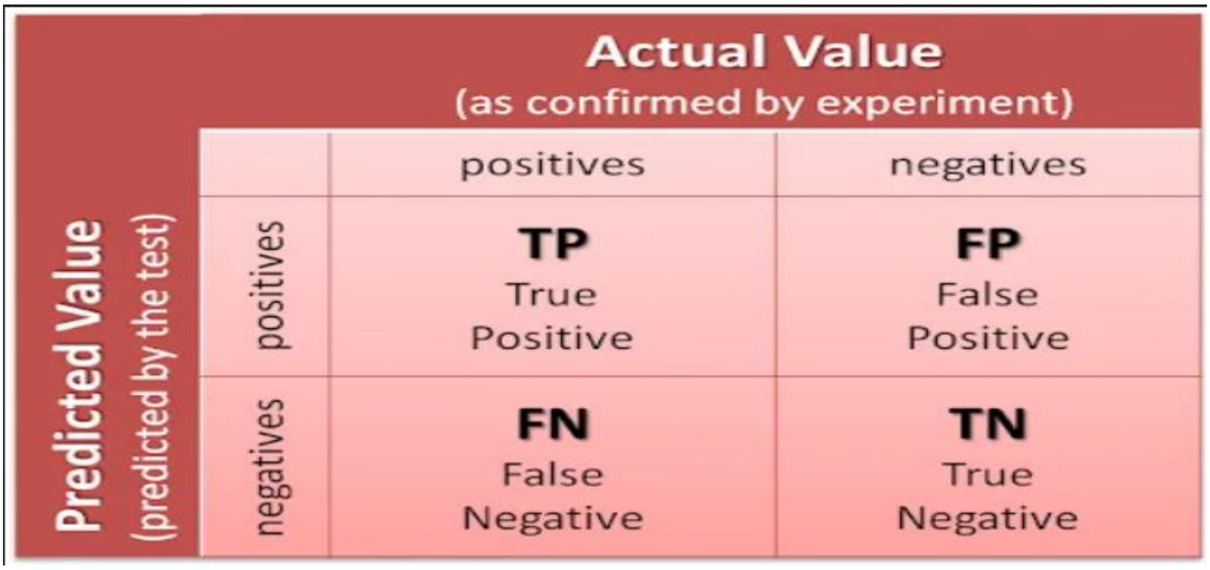
4.2.2 Metrics:

Classification Report:

A Classification report is used to measure the quality of predictions from a classification algorithm. How many predictions are True and how many are False. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report .

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.



- Positive (P): Observation is positive (for example: is an apple).
- Negative (N): Observation is not positive (for example: is not an apple).
- True Positive (TP): Observation is positive, and is predicted to be positive.
- False Negative (FN): Observation is positive, but is predicted negative.
- True Negative (TN): Observation is negative, and is predicted to be negative.

- False Positive (FP) : Observation is negative, but is predicted positive.

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.

- **Accuracy->**Accuracy represents the number of correctly classified data instances over the total number of data instances.

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

Accuracy may not be a good measure if the dataset is not balanced (both negative and positive classes have different number of data instances).

- **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.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

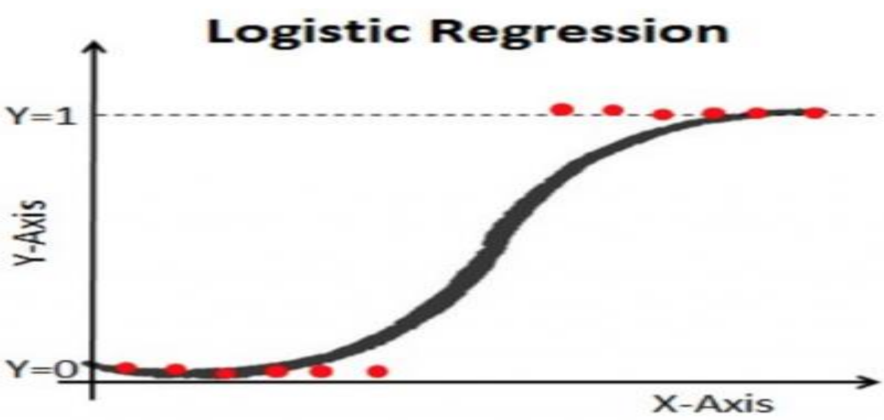
4.3 Model Building and Evaluation

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 data
##objectName.fit(InputData, outputData)
```

Figure 32: Import, initialize and fitting the logistic regression model 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

```
syntax: objectName.predict(Input)
```

```
y_train_pred = log_reg.predict(X_train) #Predicting on train data
y_train_pred
```

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

–

Figure 33: Predicting on train data

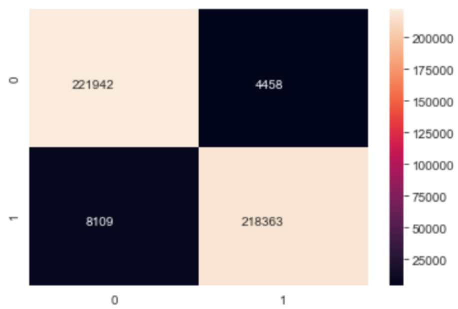
```
y_train == y_train_pred # comparing original data o/p and model predicted o/p
```

```
40281     True
200590    True
49273     True
322477    True
332369    True
          ...
371403    True
491263    True
470924    True
491755    True
128037    True
Name: Class, Length: 452872, dtype: bool
```

Figure 34: comparing the predicted value with the original one .

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_train, y_train_pred), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20cc194f708>
```




```
#Accuracy score for training data
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred)
```

```
0.972250437209631
```

```
#f1-score for training data
from sklearn.metrics import f1_score
f1_score(y_train, y_train_pred)
```

```
0.9720293883946557
```

Figure 35: 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, 0, 1], dtype=int64)
```

Figure 36: Predicting on test data.

```
y_test == y_test_pred  # comparing original data o/p and model predicted o/p
```

```
327747    True
421980    True
216522    True
561063    True
271054    True
          ...
24611     True
451136    True
322330    True
157770    True
377635    True
Name: Class, Length: 113218, dtype: bool
```

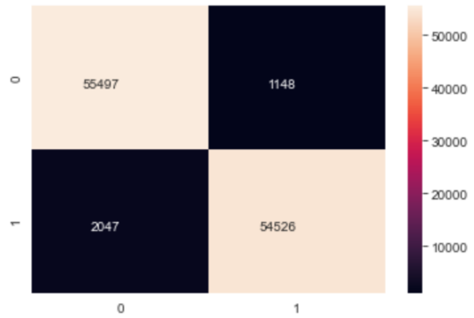
Figure 37: comparing the predicted value with the original test data.

```
#applying metrics on test data
confusion_matrix(y_test,y_test_pred)
```

```
array([[55497,  1148],
       [ 2047, 54526]], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_test, y_test_pred), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20cc2ff3108>
```




```
#Accuracy score for test data
accuracy_score(y_test,y_test_pred)
```

```
0.9717801056369129
```

```
#f1-score for test data
from sklearn.metrics import f1_score
f1_score(y_test, y_test_pred)
```

```
0.9715359875987777
```

Figure 38: 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    226400
           1       0.98      0.96      0.97    226472

    accuracy                           0.97    452872
   macro avg       0.97      0.97      0.97    452872
weighted avg       0.97      0.97      0.97    452872

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

           0       0.96      0.98      0.97     56645
           1       0.98      0.96      0.97     56573

    accuracy                           0.97    113218
   macro avg       0.97      0.97      0.97    113218
weighted avg       0.97      0.97      0.97    113218
```

Figure 39: Overall performance of the logistic regression model on training and test data.

```
models = ['training','testing']
f1_scores = [0.9720293883946557,0.9715359875987777]
plt.bar(models, f1_scores, color=['pink', 'grey' ])
plt.ylabel("f1_scores")
plt.title("train vs test for logistic regression")
plt.show()
```

Figure 39:Visualization on f1-score on training and testing data in logistic regression

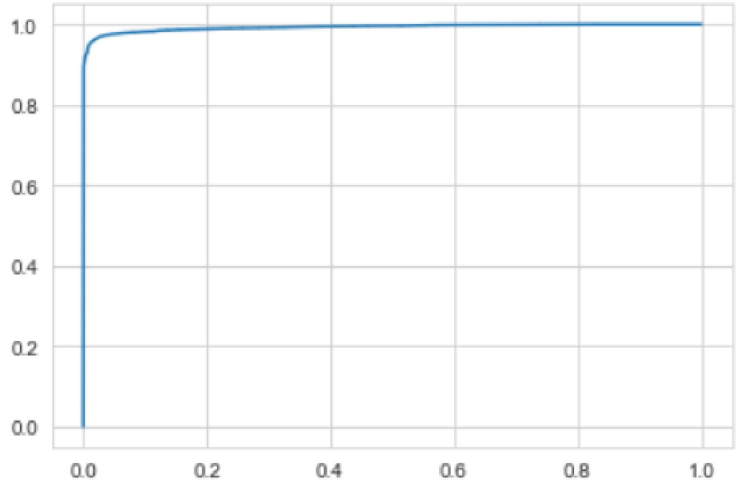
Observations:

- Training data f1-score in logistic regression: 97.2%
- Testing data f1-score in logistic regression:97.1%

```
#import the roc_auc_score, roc_curve from sklearn.metrics
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) #roc_curve for logistic regression
```

```
[<matplotlib.lines.Line2D at 0x21135a4a2c8>]
```



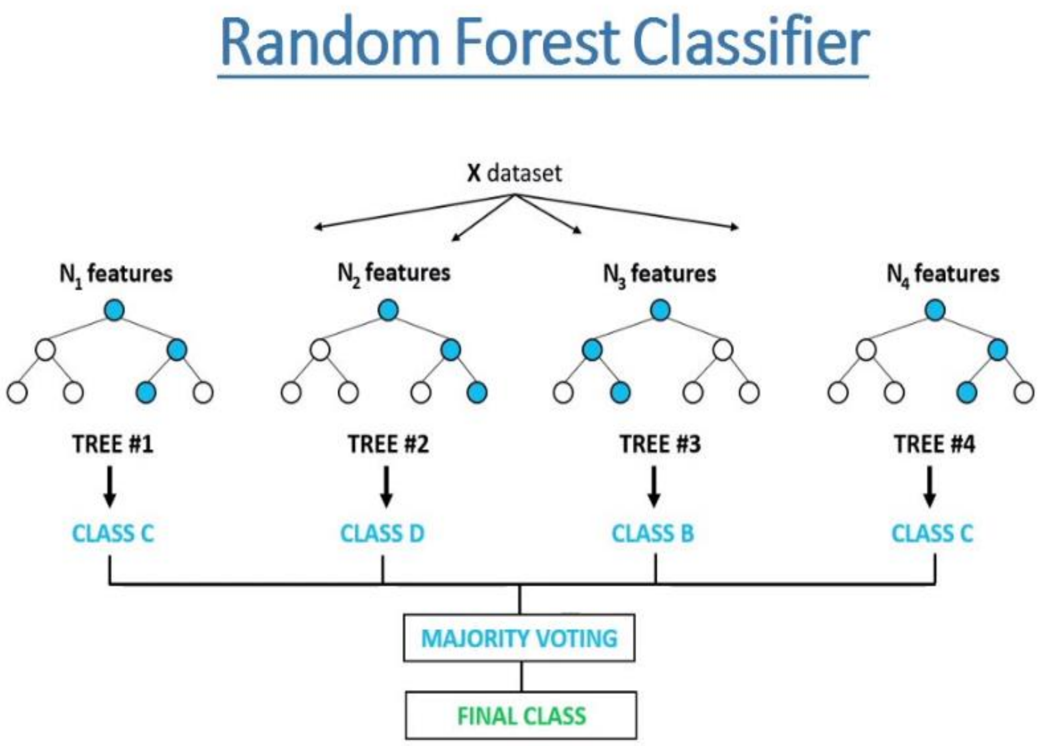

```
roc_auc_score(y_test, fraud_prob1)
```

```
0.9930221374247054
```

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

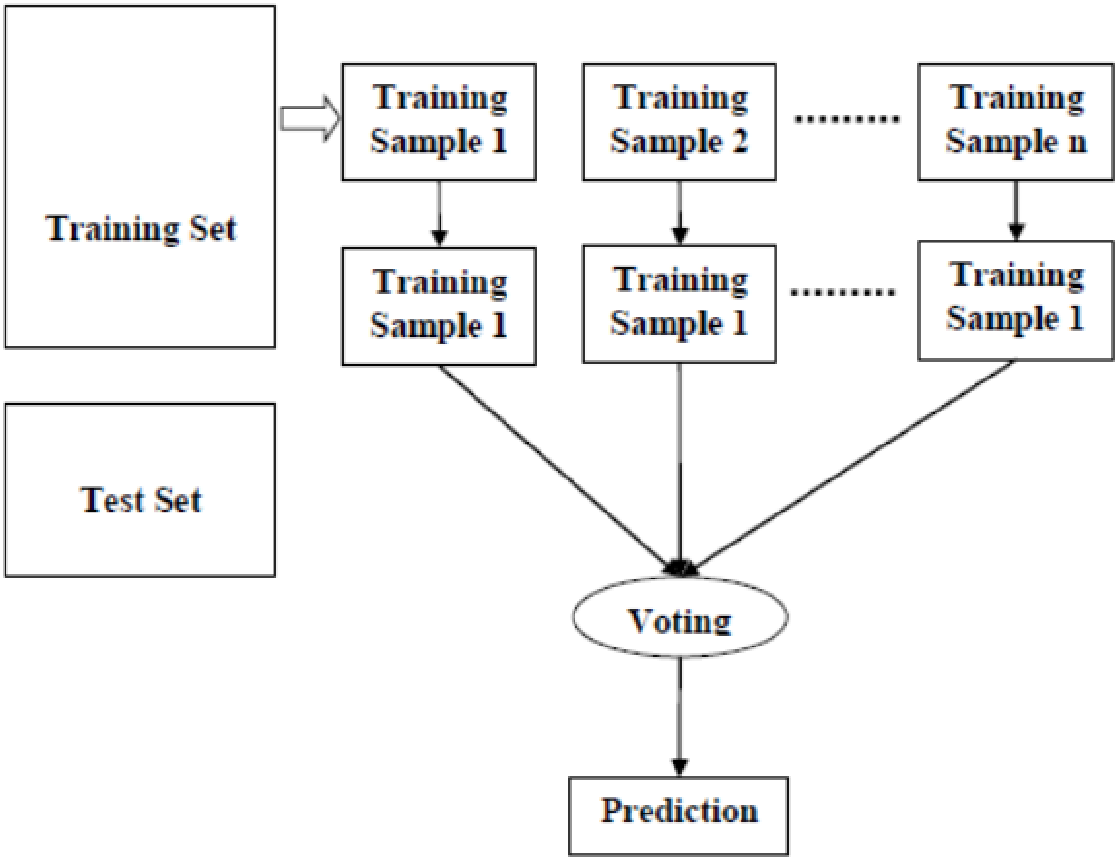
4.3.2 Random forest classification

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 forestalgorithm 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 41: Import ,initialize and fitting the random forest classifier on the training data.

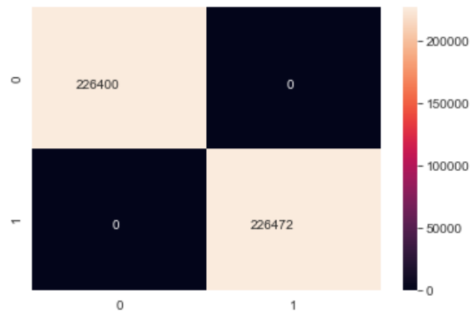
<u>Predicting on training data</u>

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

Figure 42: Prediction on train data.

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_train, y_train_pred1), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20cc1718b48>
```




```
#accuracy score for training data
accuracy_score(y_train,y_train_pred1)
```

```
1.0
```

```
#f1-score for training data
from sklearn.metrics import f1_score
f1_score(y_train, y_train_pred1)
```

```
1.0
```

Figure 43 :Applying metrics on training data

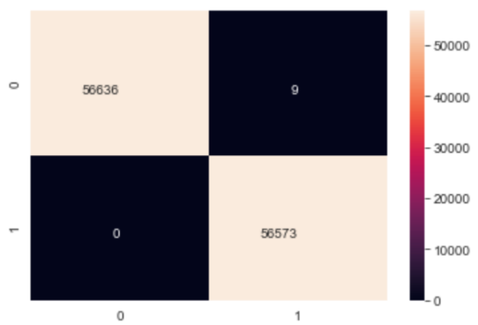
Predicting on test data

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

Figure 44: Prediction on test data.

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_test, y_test_pred1), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20cc2e81b48>
```



```
#accuracy score for test data
accuracy_score(y_test,y_test_pred1)
```

```
0.9999205073398223
```

```
#f1-score for test data
from sklearn.metrics import f1_score
f1_score(y_test, y_test_pred1)
```

```
0.9999204630816138
```

Figure45:Applying metrics on test data

```
#Applying metrics on training and test data and generating classification report.
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    226400
           1       1.00      1.00      1.00    226472

    accuracy                           1.00    452872
   macro avg       1.00      1.00      1.00    452872
weighted avg       1.00      1.00      1.00    452872

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

           0       1.00      1.00      1.00     56645
           1       1.00      1.00      1.00     56573

    accuracy                           1.00    113218
   macro avg       1.00      1.00      1.00    113218
weighted avg       1.00      1.00      1.00    113218
```

Figure 46: Overall performance of the random forest classifier model on training and test data.

```
models = ['training','testing']
f1_scores = [1.0,0.9999204630816138]
plt.bar(models, f1_scores, color=['pink', 'grey' ])
plt.ylabel("f1-scores")
plt.title("train vs test for randomforest classifier")
plt.show()
```

Figure 47:Visualization on f1-score on training and testing data in random forest classifier.

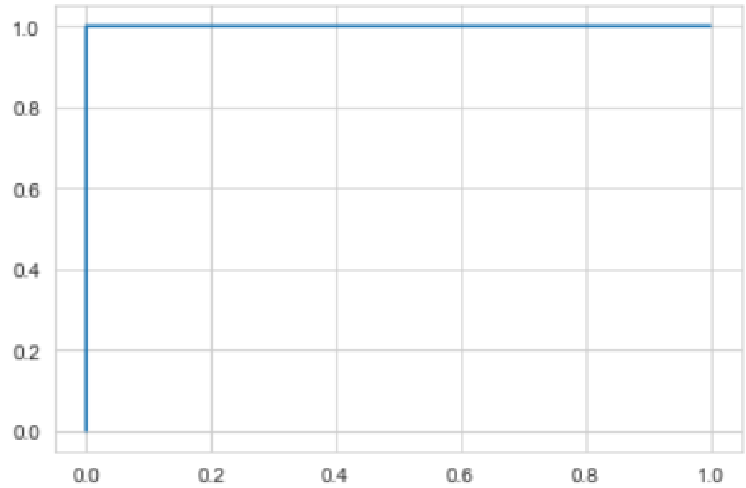
Observations:

- Training data f1-score in Random forest classifier: 100%
- Testing data f1-score in Random forest classifier:99%

```
#import the roc_auc_score, roc_curve from sklearn.metrics
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)    #roc_curve for Random forest
```

```
[<matplotlib.lines.Line2D at 0x211011c0fc8>]
```




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

```
0.9999989004791096
```

Figure 48: Measuring the accuracy of a random forest classifier model using the Area Under the Precision-Recall Curve (AUPRC).

4.3.3 Naive Bayes

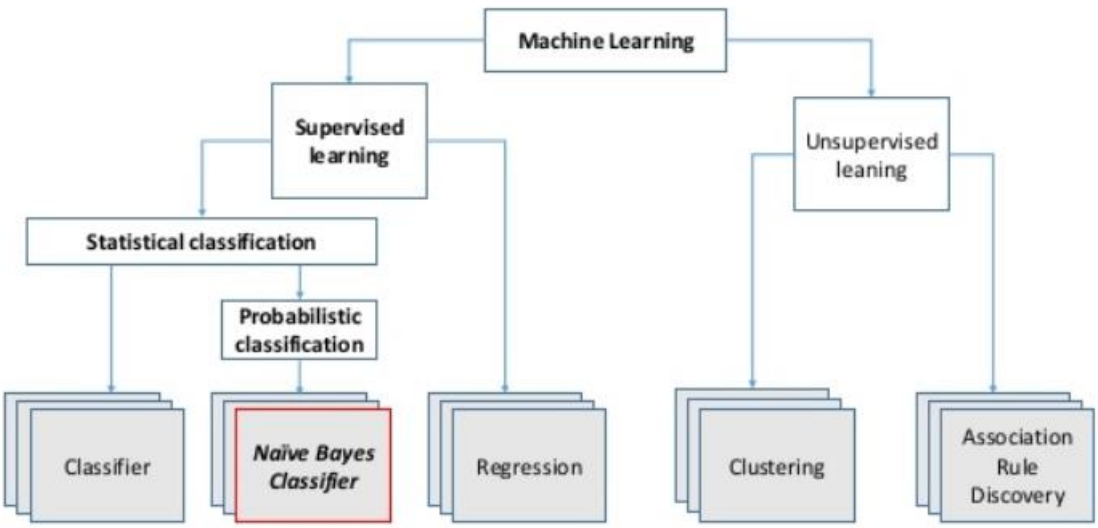
Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of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



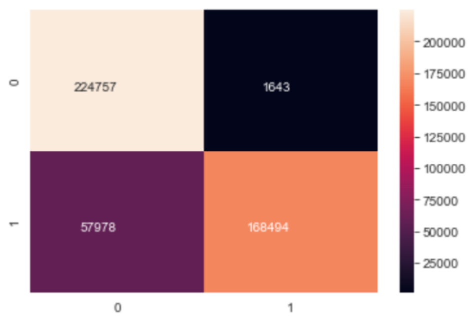
```
#import initialize and fit
#import the GaussianNB from sklearn.naive_bayes
from sklearn.naive_bayes import GaussianNB

#initialize the object for GaussianNB
gn = GaussianNB()

#fit GaussianNB to dataset
final_model3 = gn.fit(X_train,y_train)

#Predicting on training data
y_train_pred2 = gn.predict(X_train)
```

Figure 49: Import ,initialize and fit the Naïve Bayes model on the training data.

Predicting on training data

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_train, y_train_pred2), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c8723afc8>
```




```
#accuracy score for training data
from sklearn.metrics import accuracy_score
accuracy_score(y_train,y_train_pred2)
```

```
0.8683491140984649
```

```
#f1-score for training data
from sklearn.metrics import f1_score
f1_score(y_train, y_train_pred2)
```

```
0.8496731037369297
```

Figure 50:Applying metrics on training data.

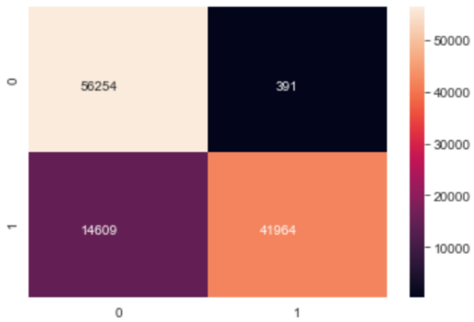
Predicting on test data

```
y_test_pred2 = gn.predict(X_test) #Predicting on test data
```

Figure 51: Prediction on training data

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_test, y_test_pred2), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c87902a08>
```




```
#accuracy score for test data
accuracy_score(y_test,y_test_pred2)
```

```
0.8675122330371495
```

```
#f1-score for test data
from sklearn.metrics import f1_score
f1_score(y_test, y_test_pred2)
```

```
0.8483745754488113
```

Figure 52: Applying metrics on test data.

```
#Applying metrics on training and test data and generating classification report.
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    226400
           1       0.99      0.74      0.85    226472

    accuracy                           0.87    452872
   macro avg       0.89      0.87      0.87    452872
weighted avg       0.89      0.87      0.87    452872

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

           0       0.79      0.99      0.88     56645
           1       0.99      0.74      0.85     56573

    accuracy                           0.87    113218
   macro avg       0.89      0.87      0.87    113218
weighted avg       0.89      0.87      0.87    113218
```

Figure 53: Overall performance of the naive bayes model on training and test data.

```
models = ['training','testing']
f1_scores = [0.8496731037369297,0.8483745754488113]
plt.bar(models, f1_scores, color=['pink', 'grey' ])
plt.ylabel("f1-scores")
plt.title("train vs test for Naive Bayes")
plt.show()
```

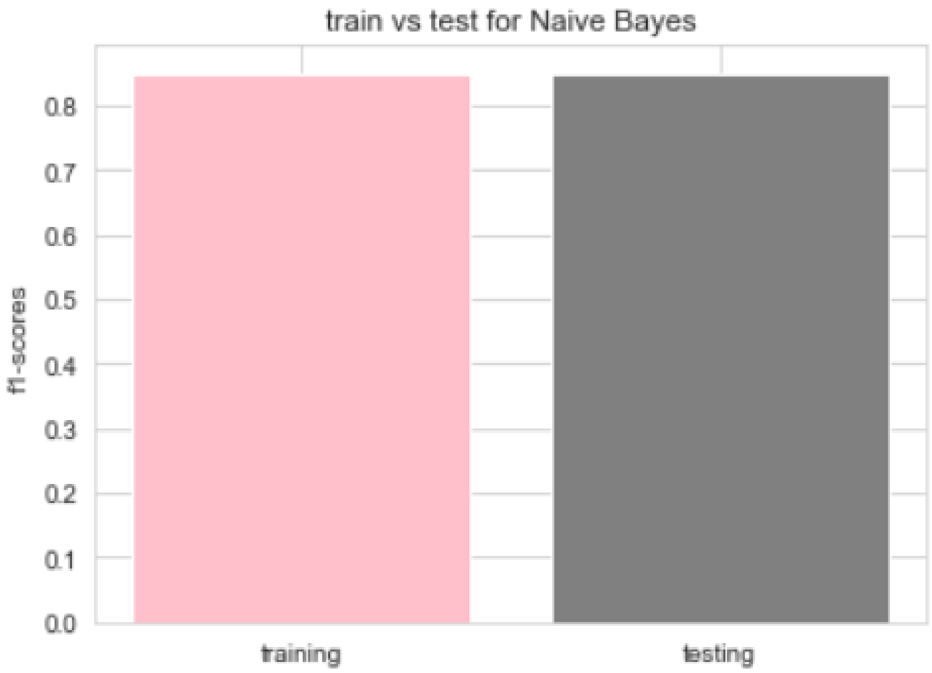


Figure 54: Visualization on f1-score on training and testing data in Naïve Bayes.

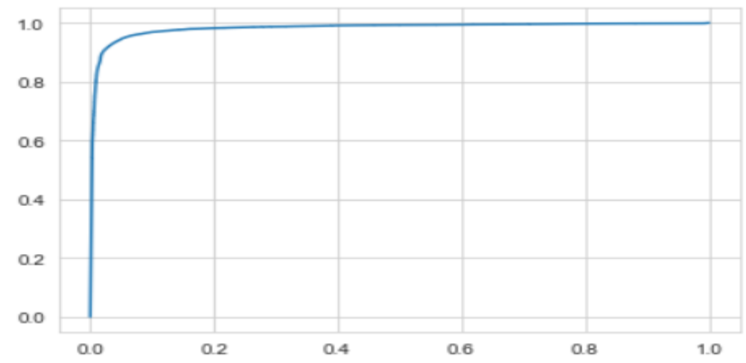
Observations:

- Training data f1-score in Naive Bayes: 84.9%
- Testing data f1-score in Naive Bayes:84.8%

```
#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)  #roc_curve for Random forest
```

```
[<matplotlib.lines.Line2D at 0x21137a556c8>]
```

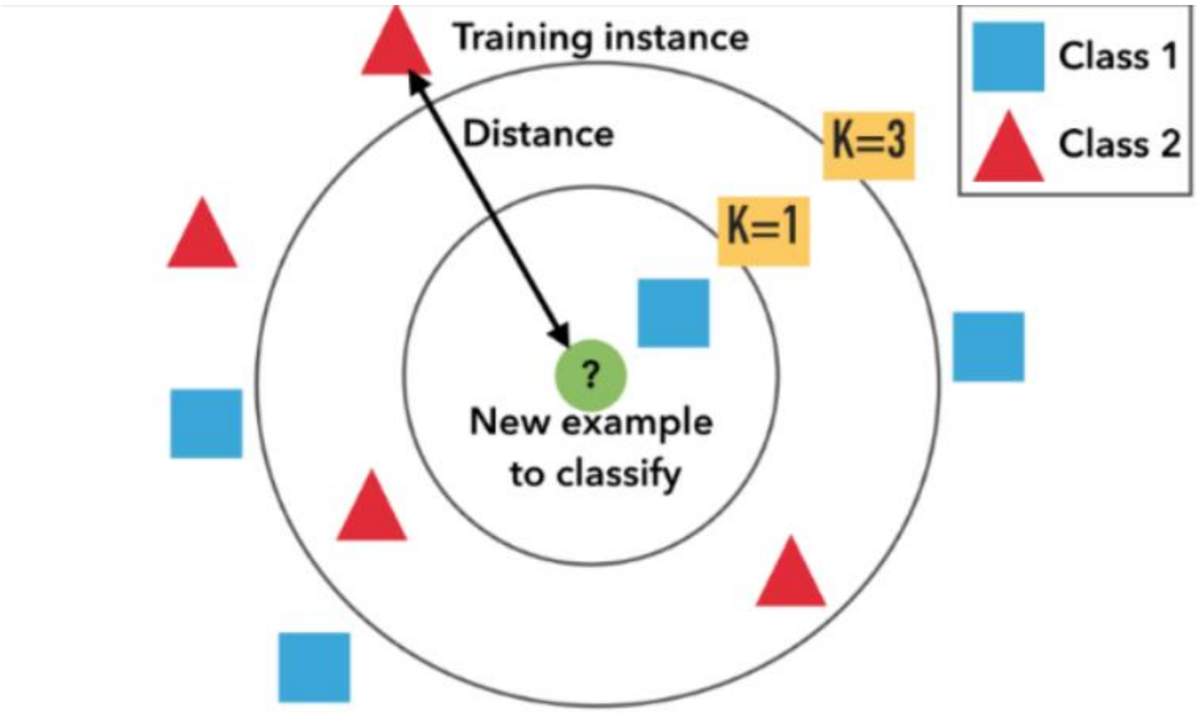



```
roc_auc_score(y_test, fraud_prob3)
```

```
0.9825882357284228
```

Figure 55: Measuring the accuracy of naive bayes model using the Area Under the Precision-Recall Curve (AUPRC).

4.3.4 K Nearest Neighbor (KNN):



- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of K number of neighbors
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

```
# Model Building:
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')

# Apply the knn object on the dataset
# syntax: objectName.fit(Input, Output0)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=3)
```

Figure 56: Import, initialize and fit the K Nearest Neighbor model on the training data.

Predicting on training data

```
# Predictions on the data
# predict function--> gives the predicted values
# Syntax:objectname.predict(Input)
y_train_pred_knn = knn.predict(X_train)
y_train_pred_knn
```

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

Figure 57: Predicting on train data

```
# Check the accuracy, classification report
from sklearn.metrics import classification_report
print(classification_report(y_train, y_train_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    226400
           1       0.98      1.00      0.99    226472

    accuracy                           0.99    452872
   macro avg       0.99      0.99      0.99    452872
weighted avg       0.99      0.99      0.99    452872
```

Figure 58: Classification report on training data with n_neighbors=3

```
from sklearn.metrics import accuracy_score
# Checking for optimum k-value
# Build the models with multiple k values
scores=[]
for k in range(1, 20):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train, y_train)
    pred_test_knn = knn_model.predict(X_test)
    scores.append(accuracy_score(y_test, pred_test_knn))
scores
```

```
[0.9842692858026109,
 0.9796145489233161,
 0.9722305640445865,
 0.9700665971841933,
 0.9638661696903319,
 0.9620996661308272,
 0.9562613718666643,
 0.9549629917504284,
 0.9501316045151831,
 0.9488420569167447,
 0.9444434630535781,
 0.9431539154551396,
 0.939497253086965,
 0.9381723754173364,
 0.9347453585118974,
 0.9339415993923228,
 0.9308325531275945,
 0.9300552915614125,
 0.9272818809729901]
```

Figure 59: Accuracy scores for some range of multiple values(1 to 20)

```
# Plot of K values and scores
plt.plot(range(1,20), scores, marker='o', markerfacecolor='r', linestyle='--')
```

```
[<matplotlib.lines.Line2D at 0x211026bde48>]
```

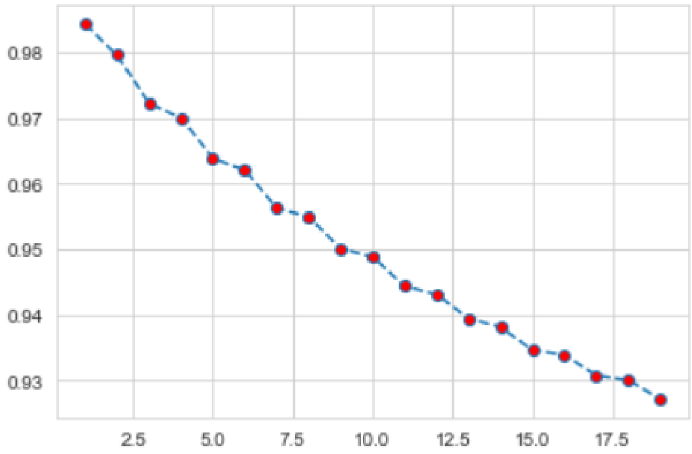


Figure 60: Determining optimum K value.

```
#  Optimum k value is 2
final_model4 = KNeighborsClassifier(n_neighbors=2, metric='euclidean')
final_model4.fit(X_train, y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=2)
```

Figure 61: fitting the model based on optimum K value

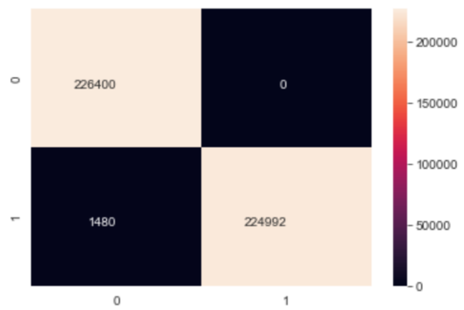
```
# Prediction on training data
final_train_pred = final_model4.predict(X_train)
final_train_pred
```

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

Figure 62: Prediction on training data

```
from sklearn.metrics import confusion_matrix;
sns.heatmap(confusion_matrix(y_train, final_train_pred), annot=True, fmt='d', annot_kws={'va':'top', 'ha':'right'})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c8a7e5688>
```



```
#accuracy score for test data
accuracy_score(y_train,final_train_pred)
```

```
0.9967319684149164
```

```
#f1-score for test data
from sklearn.metrics import f1_score
f1_score(y_train,final_train_pred)
```

```
0.9967217762656602
```

Figure 63: Applying metrics on training data

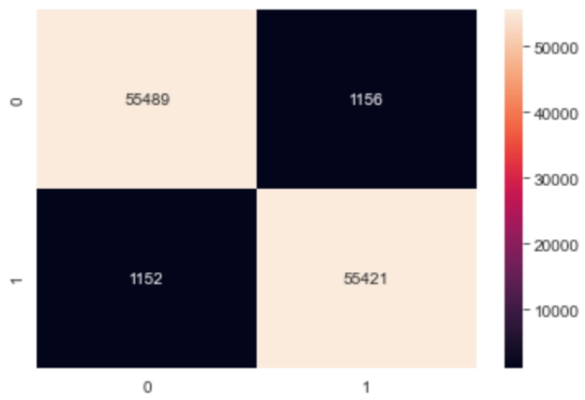
```
# Predictions on Test Data
final_test_pred = final_model4.predict(X_test)
final_test_pred
```

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

Figure 64: Prediction on test data

```
# compare actual values of test data(y_test) and final_test_pred(model predicted values)
sns.heatmap(confusion_matrix(y_test, final_test_pred), annot=True, fmt='d')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20c8a875c08>
```




```
#accuracy score for test data
accuracy_score(y_test,final_test_pred)
```

```
0.9796145489233161
```

```
#f1-score for test data
from sklearn.metrics import f1_score
f1_score(y_test, final_test_pred)
```

```
0.9796022978347326
```

Figure 65: Applying metrics on test data

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

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    226400
           1       1.00      0.99      1.00    226472

    accuracy                           1.00    452872
   macro avg       1.00      1.00      1.00    452872
weighted avg       1.00      1.00      1.00    452872

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

           0       0.98      0.98      0.98     56645
           1       0.98      0.98      0.98     56573

    accuracy                           0.98    113218
   macro avg       0.98      0.98      0.98    113218
weighted avg       0.98      0.98      0.98    113218
```

Figure 66 : Overall performance of the K Nearest Neighbor model on training and test data.

```
models = ['training','testing']
f1_scores = [0.9967217762656602,0.9796022978347326]
plt.bar(models, f1_scores, color=['pink', 'grey' ])
plt.ylabel("f1-scores")
plt.title("train vs test for KNN")
plt.show()
```

Figure 67: Visualization on f1-score on training and testing data in K Nearest Neighbor.

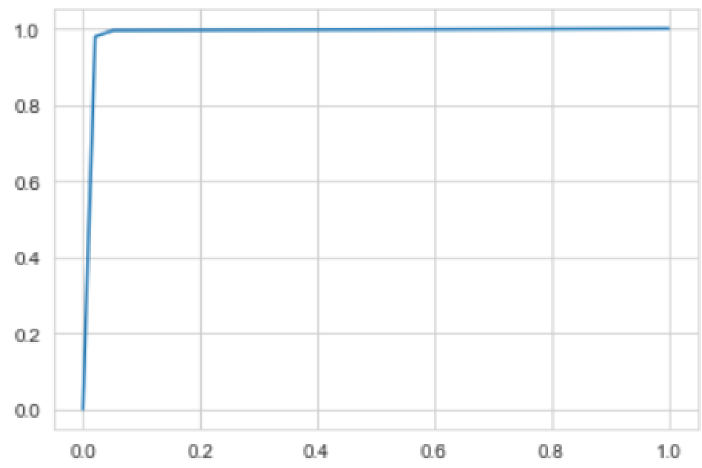
Observations:

- Training data f1-score in KNN: 99%
- Testing data f1-score in KNN:97%

```
#1-->fraud 0-->genuine
# Roc curve
## TPR, FPR, Threshold
from sklearn.metrics import roc_auc_score, roc_curve
fraud_prob_knn = final_model4.predict_proba(X_test)[:,1]
fpr4, tpr4, threshold4 = roc_curve(y_test, fraud_prob_knn)
```

```
plt.plot(fpr4, tpr4)  #roc_curve for KNN
```

```
[<matplotlib.lines.Line2D at 0x21101873b88>]
```




```
roc_auc_score(y_test, fraud_prob_knn)
```

```
0.9870395214662901
```

Figure 68: Measuring the accuracy of K Nearest Neighbor model using the Area Under the Precision-Recall Curve (AUPRC).

4.4 Visualising the best model among logistic regression, Random forest , Naive Bayes and K Nearest Neighbor.

```
models = ['Logistic Regression','Random forest','NaiveBayes','KNN']
f1_scores = [0.97,0.99,0.84,0.97]
plt.bar(models,f1_scores,color=['lightblue','pink','lightgrey','grey'])
plt.ylabel("f1-scores")
plt.title("which model has high f1score")
plt.show()
```

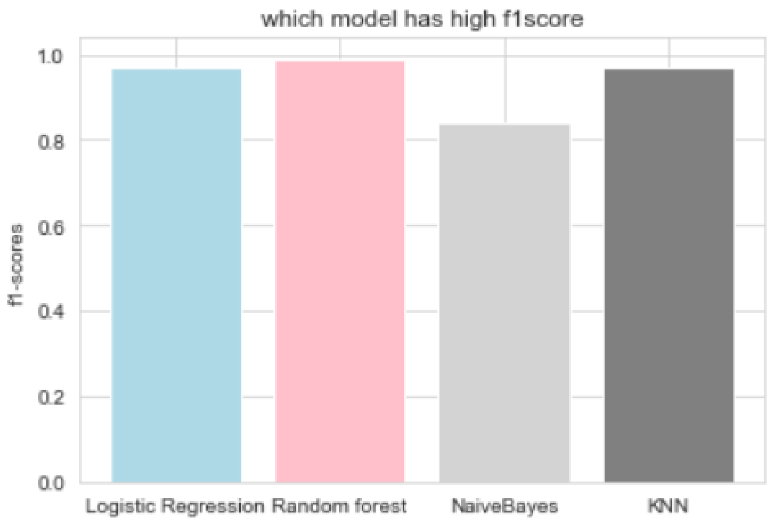


Figure 69: Comparison of the applied models based on f1-scores.

➢ ***From the above graph we can observe that random forest has high f1-score***

❖ **The best model is the RANDOM FOREST CLASSIFIER with high f1score(0.99),accuracy-score(0.99).**

5.Conclusion

For any such requirement of credit card fraud detection, card holders are at the right place for managing their card security. Credit Card Fraud Detection project not only reports but also smoothly handles the transactions in a very efficient and a highly consistent way.

The security aspect that is presented to cardholders by this site is highly efficient, and at the same time, very user-friendly, because such frauds can be identified with ease and cardholders can then access their cards easily.

The process that is available in this website has been defined very clearly, and is analyzed well too, so that the cardholders or users can use this particular site gradually with no hesitations as such.

6.References

- https://en.wikipedia.org/wiki/Machine_learning
- https://towardsdatascience.com/supervised-machine-learning-model-validation-a-step-by-step-approach-771109ae0253
- https://builtin.com/data-science/random-forest-algorithm
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://www.geeksforgeeks.org/confusion-matrix-machine-learning/