MACHINE LEARNING

(Predicting Amount of Purchase)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer science Engineering

By

K. Siddharth venkat

221710305019

Under the Guidance of

Mr. M. Venkateswarlu

Assistant Professor

Department Of Computer science Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

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 ofTechnology" 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 otherUniversity or Institute for the award of any degree or diploma.

Place: HYDERABAD **k. Siddharth venkat**

Date: 12/07/2020 **221710305019**

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled“**credit card fraud detection**” is being submitted by K. Siddharth Venkat in partial fulfillment of the requirement for the award ofBachelor of Technology **computerscienceEngineering** at GITAM(Deemed To Be University), Hyderabad during the academic year 2019-20 .

It is faithful record work carried out by her at the computer science Engineering Department, GITAM University Hyderabad Campusunder my guidance and supervision.

Dr.phani kumar sir

Professor and HOD

Department of CSE

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.

K. Siddharth venkat

221710305019

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms models of higher accuracy to predict the values is the primary task to be performed on Cereals data set My perception of understanding the given data set has been in the view of undertaking a client's requirement of overcoming the stagnant point of sales of the products being manufactured by clients.

To get a better understanding and work on a strategic approach for solution of the client, I have adapted the viewpoint of looking at ratings of the products and for further deep understanding of the problem, I have taken the stance of a consumer and reasoned out the various factors of choice of the products and they purchase , and my primary objective of this case study was to look up the factors which were dampening the sale of products and relate them to ratings of products and draft out an outcome report to client regarding the various accepts of a product manufacturing marketing and sale point determination

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.

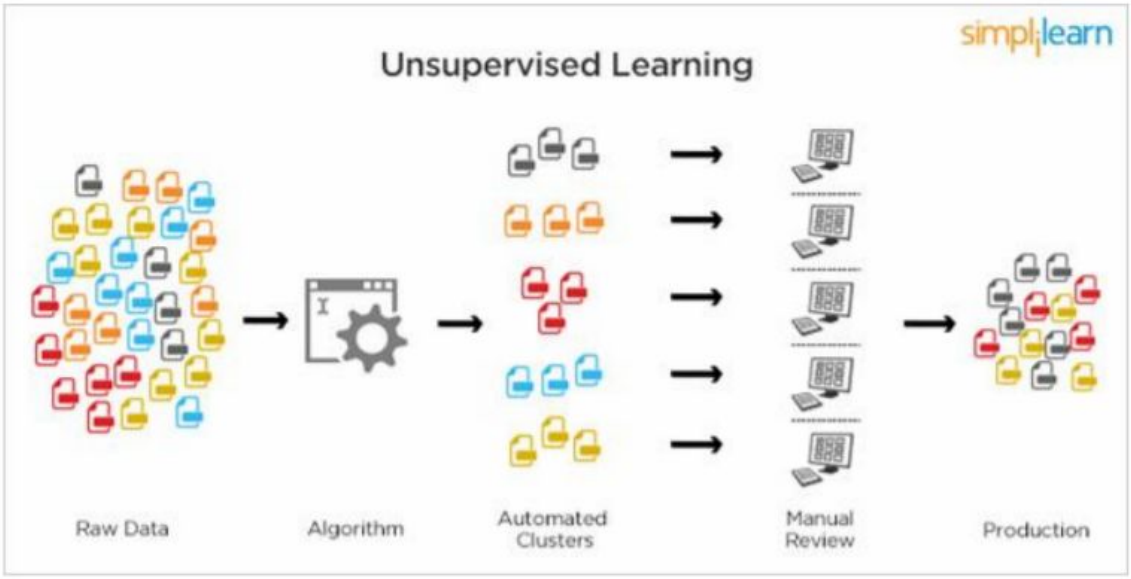


Figure 2 : Unsupervised Learning

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.

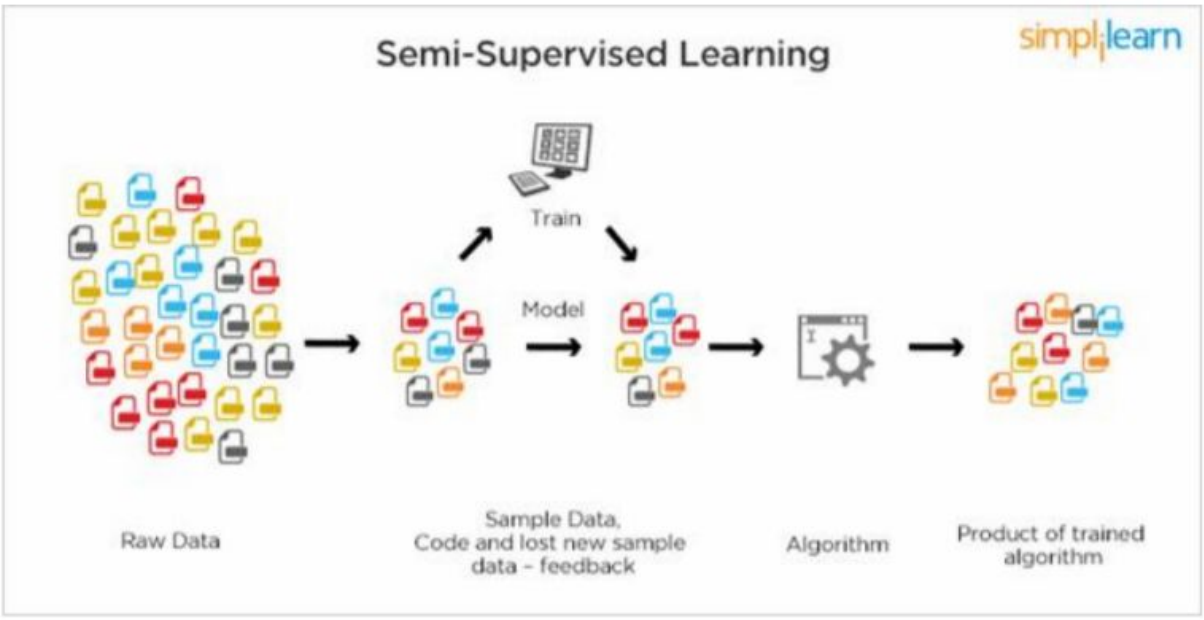


Figure 3 : Semi Supervised Learning

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.

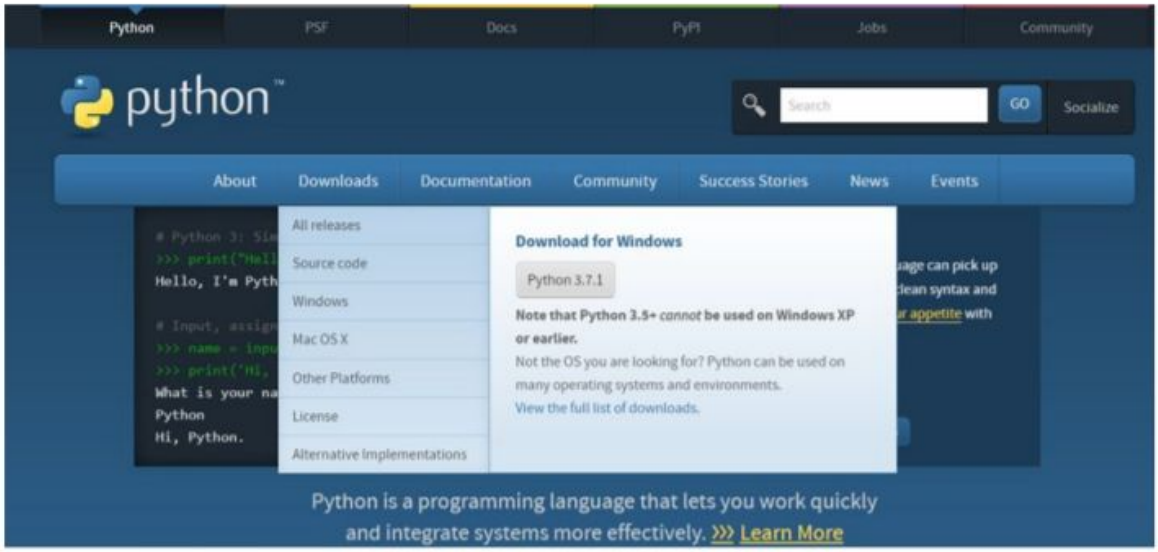


Figure 4 : Python download

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 3.4) next click install and next click finish
- Step 5: Open jupyter notebook (it opens in default browser)

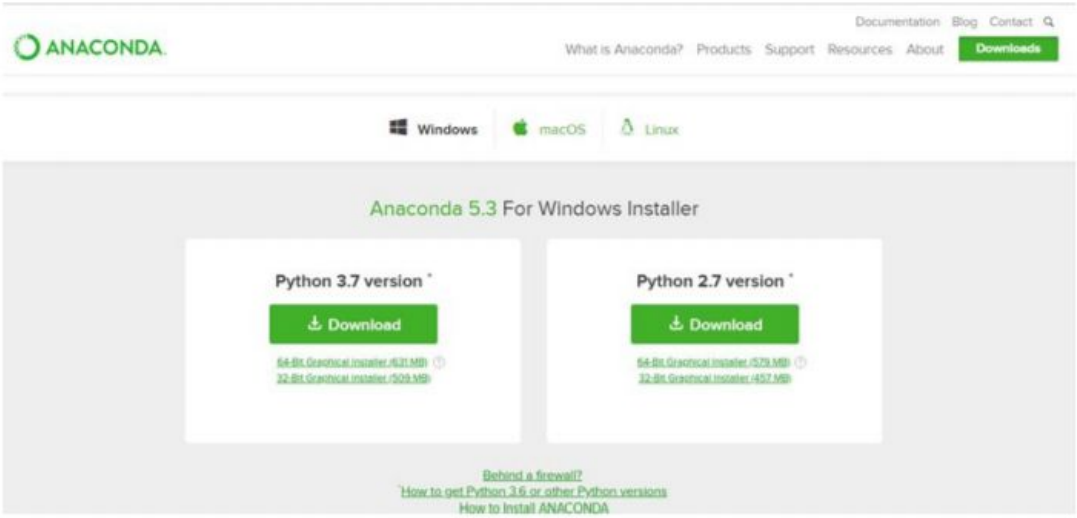


Figure 5 : Anaconda download

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 –

- o Numbers
- o Strings
- o Lists
- o Tuples
- o 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 11 ([]).
● 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 12 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 14 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:

in this project ,we detect the fraud made using credit card by using different techniques/algorithms

3.2 DATA SET:

The given data set consists of the following parameters:

- Time
- V1
- V2
- V3
- V4
- V5
- V6
- V7
- V8
- V9
- V10
- V11
- V12
- V13
- V14
- V15
- V16
- V17
- V18
- V19
- V20
- V21
- V22
- V23
- V24
- V35

- V26
- V27
- V28
- Amount
- Class

3.3 OBJECTIVE OF THE CASE STUDY:

The main objective of the case study is to look up at the factors that are responsible for fraud transactions using credit card and to visualize them and find a solution to that problem using the required models and to bring awareness to the people about the ongoing problem in the society.

CHAPTER 4
MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

The dataset is gained through database
Provided by our mentors

4.1.2 IMPORTING THE LIBRARIES:

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

importing libraries

```
In [6]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
```

Fig 4.1.2

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.

importing and reading the dataset

```
In [7]: df= pd.read_csv('creditcard.csv')
        df.head()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
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	0.1285
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	0.1671
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	-0.3276
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	0.6473
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	-0.2060

5 rows × 31 columns

Fig 4.1.3

4.1.4 finding the datatypes

```
In [10]: df.dtypes

Out[10]: 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
```

Plotting histograms:

```
In [11]: # plot the histogram of each parameter
         df.hist(figsize = (20, 20))
         plt.show()
```

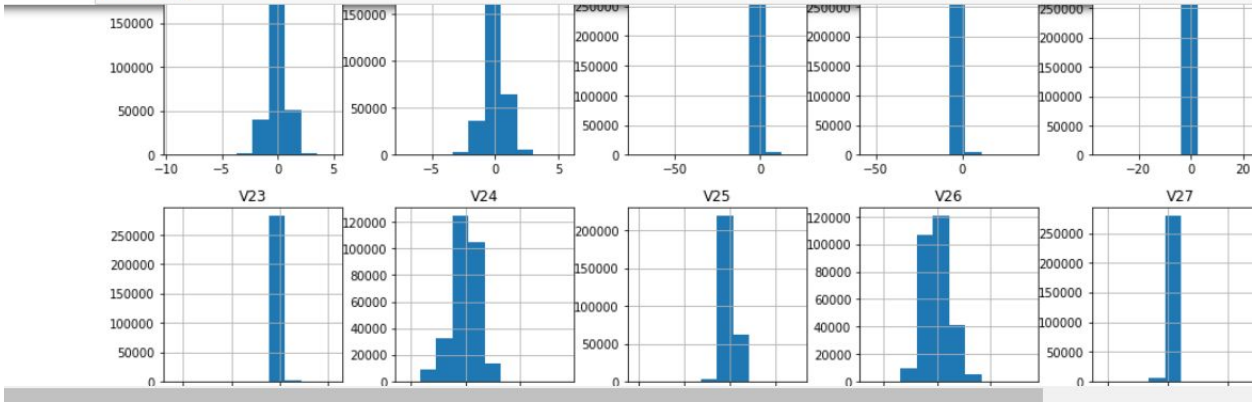


fig4.1.4

4.1.5 finding the null values

```
In [15]: ## finding the null values
         df.isna().sum()
```

```
Out[15]: 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
```

Fig 4.1.5

From the above we can say that there are no null values in my dataset.

4.1.6 data visualization

correlation

In [19]:
```
corr = df.corr()
corr
```

Out[19]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
Time	1.000000	1.173963e-01	-1.059333e-02	-4.196182e-01	-1.052602e-01	1.730721e-01	-6.301647e-02	8.471437e-02	-3.694943e-02	-8.660434e-03	...	4.473
V1	0.117396	1.000000e+00	4.697350e-17	-1.424390e-15	1.755316e-17	6.391162e-17	2.398071e-16	1.991550e-15	-9.490675e-17	2.169581e-16	...	-1.7
V2	-0.010593	4.697350e-17	1.000000e+00	2.512175e-16	-1.126388e-16	-2.039868e-16	5.024680e-16	3.966486e-16	-4.413984e-17	-5.728718e-17	...	8.444
V3	-0.419618	-1.424390e-15	2.512175e-16	1.000000e+00	-3.416910e-16	-1.436514e-15	1.431581e-15	2.168574e-15	3.433113e-16	-4.233770e-16	...	-2.9
V4	-0.105260	1.755316e-17	-1.126388e-16	-3.416910e-16	1.000000e+00	-1.940929e-15	-2.712659e-16	1.556330e-16	5.195643e-16	3.859585e-16	...	-9.9
V5	0.173072	6.391162e-17	-2.039868e-16	-1.436514e-15	-1.940929e-15	1.000000e+00	7.926364e-16	-4.209851e-16	7.589187e-16	4.205206e-16	...	-1.3
V6	-0.063016	2.398071e-16	5.024680e-16	1.431581e-15	-2.712659e-16	7.926364e-16	1.000000e+00	1.429426e-16	-1.707421e-16	1.114447e-16	...	-1.5
V7	0.084714	1.991550e-15	3.966486e-16	2.168574e-15	1.556330e-16	-4.209851e-16	1.429426e-16	1.000000e+00	-8.691834e-17	7.933251e-16	...	1.938
V8	-0.036949	-9.490675e-17	-4.413984e-17	3.433113e-16	5.195643e-16	7.589187e-16	-1.707421e-16	-8.691834e-17	1.000000e+00	2.900829e-16	...	-2.4

In [20]:
```
corr = df.corr()
plt.figure(figsize=(14,14))
heat = sns.heatmap(data=corr)
plt.title('Heatmap of Correlation')
```

Out[20]: `Text(0.5, 1, 'Heatmap of Correlation')`

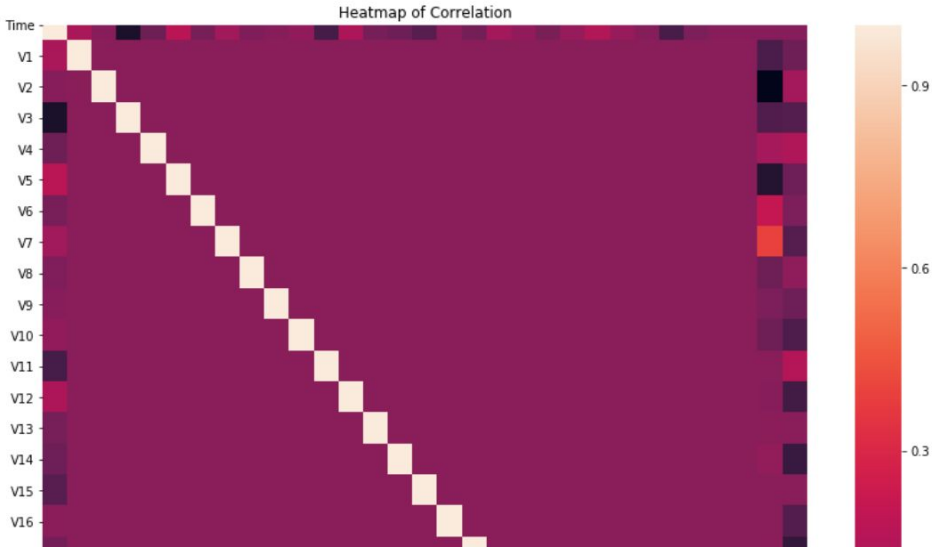


fig:correlation

Finding fraud vs normal transactions:

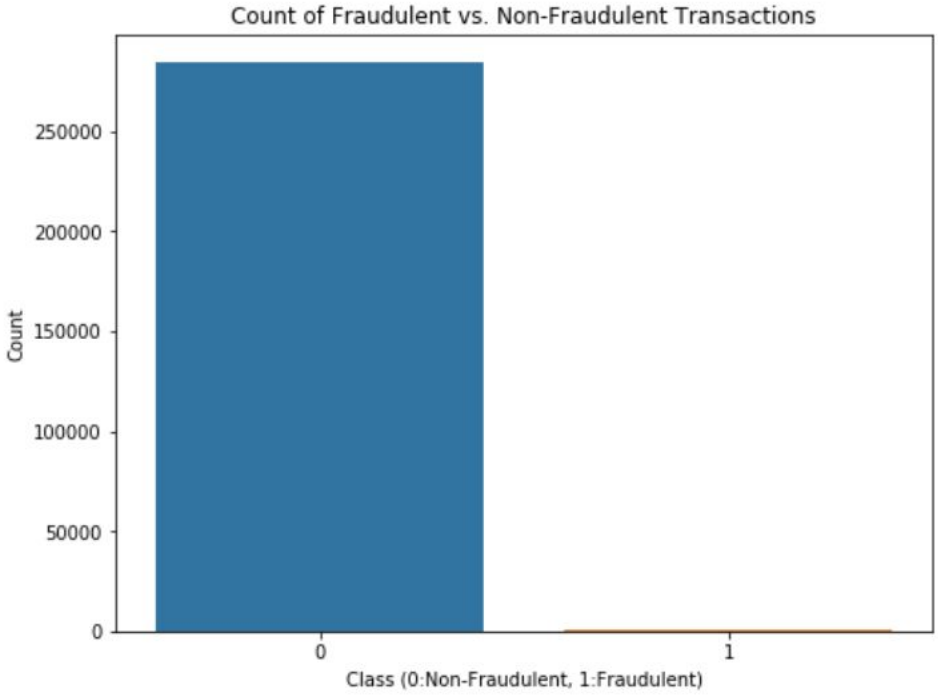
data visualization

[17]:
```
#fraud vs. normal transactions
counts = df.Class.value_counts()
normal = counts[0]
fraudulent = counts[1]
perc_normal = (normal/(normal+fraudulent))*100
perc_fraudulent = (fraudulent/(normal+fraudulent))*100
print('There were {} non-fraudulent transactions ({:.3f}%) and {} fraudulent transactions ({:.3f}%).'.format(normal, perc_normal
```

```
There were 284315 non-fraudulent transactions (99.827%) and 492 fraudulent transactions (0.173%).
```

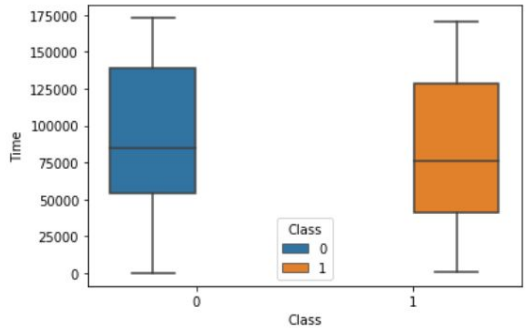
In [18]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=counts.index, y=counts)
plt.title('Count of Fraudulent vs. Non-Fraudulent Transactions')
plt.ylabel('Count')
plt.xlabel('Class (0:Non-Fraudulent, 1:Fraudulent)')
```

Out[18]:
```
Text(0.5, 0, 'Class (0:Non-Fraudulent, 1:Fraudulent)')
```



Using boxplot:

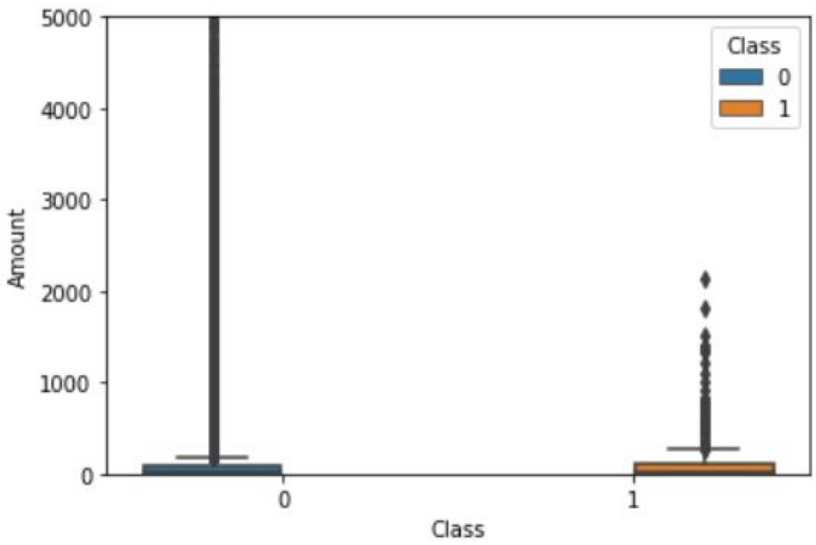
In [22]:
```
sns.boxplot(x="Class",y="Time",hue='Class',data=df)
plt.show()
```



-> by looking at the box plot we can say that there is no certain distinction between the fraud and normal transcations

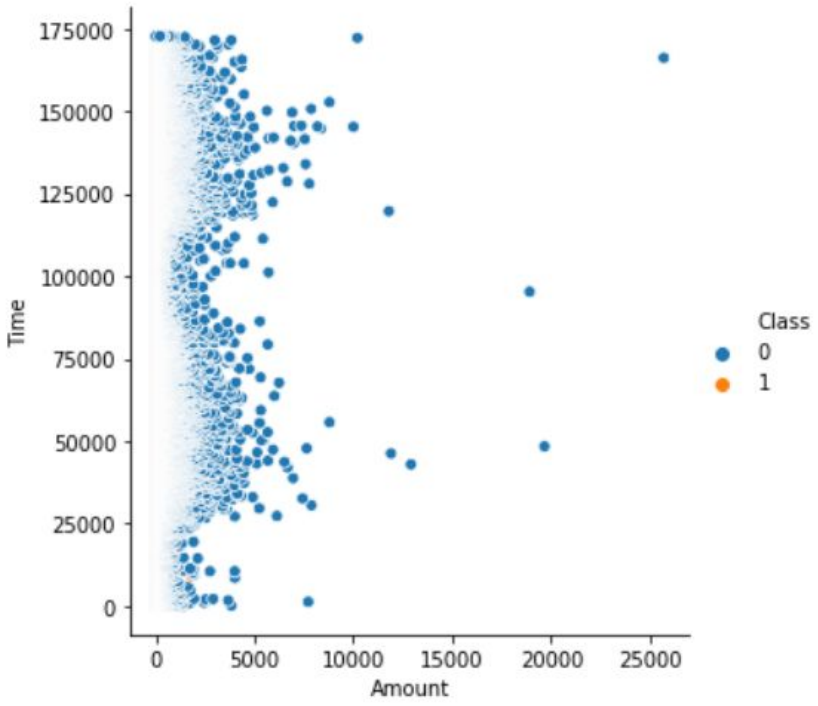
Fig boxplot

In [23]:
```
sns.boxplot(x="Class",y="Amount",hue="Class",data=df)
plt.ylim(0,5000)
plt.show()
```



Using relplot:

```
In [24]: sns.relplot(x='Amount',y='Time',hue='Class',data=df)
Out[24]: <seaborn.axisgrid.FacetGrid at 0x14c115fd948>
```



4.2 TRAINING THE MODEL:

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 data

```
In [25]: # input and output
         X=df.drop('Class',axis=1)
         y=df.Class
```

```
In [53]: # splitting into train and test
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
                                                             , random_state=1)
```

Scaling :

In [37]:
```
#Applying Standard Scaler in the model
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [38]:
```
#fitting the data set
X_std=sc.fit_transform(X)
X_std
```

Out[38]:
```
array([[-1.99658302, -0.69424232, -0.04407492, ...,  0.33089162,
        -0.06378115,  0.24496426],
       [-1.99658302,  0.60849633,  0.16117592, ..., -0.02225568,
         0.04460752, -0.34247454],
       [-1.99656197, -0.69350046, -0.81157783, ..., -0.13713686,
        -0.18102083,  1.16068593],
       ...,
       [ 1.6419735 ,  0.98002374, -0.18243372, ...,  0.01103672,
        -0.0804672 , -0.0818393 ],
       [ 1.6419735 , -0.12275539,  0.32125034, ...,  0.26960398,
         0.31668678, -0.31324853],
       [ 1.64205773, -0.27233093, -0.11489898, ..., -0.00598394,
         0.04134999,  0.51435531]])
```

In [26]:
```
# scaling for train data we use fit_transform
# cuz in fit when it comes to the train it finds mean and variance
# based on mean and variance it will apply it to the train data
scaled_X_train=pd.DataFrame(sc.fit_transform(X_train),
                            columns=X_train.columns)
scaled_X_train

# scaling for test data and for test we should only use transform
# cuz in fit when it comes to the train it finds mean and variance
# based on mean and variance it will apply it to the train data

scaled_X_test=pd.DataFrame(sc.transform(X_test),
                           columns=X_train.columns)
scaled_X_test
```

Out[26]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V2
0	0.526644	-0.311159	-0.466100	-0.098088	-0.158494	1.462291	-1.512549	0.232529	-0.434778	0.329303	...	-0.253144	-0.101400	0.063203	0.62053
1	-0.348861	-0.414691	0.798367	0.873992	0.019404	-0.208669	-0.490021	0.255854	0.363429	-0.637721	...	-0.011416	-0.173920	-0.507088	0.14733
2	-0.263726	-0.161440	0.676939	0.637703	-0.089477	0.408472	-0.399064	0.564634	-0.053523	-0.418408	...	0.266598	-0.413953	-1.066414	-0.20265
3	-1.330858	-0.676664	0.616262	1.167100	-1.110472	-0.087922	-0.343105	0.545062	-0.025785	0.352418	...	0.408527	-0.299103	-0.576565	-0.39076
4	-0.294457	0.652095	0.373373	-0.379528	0.620441	0.041657	-1.102397	0.297705	-0.238499	-0.073744	...	-0.135830	-0.216748	-0.592252	-0.12574
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
85438	-0.337066	0.648961	0.072533	-0.213683	0.608369	-0.028659	-0.193174	-0.161911	0.041939	0.804223	...	-0.174355	-0.357242	-0.803257	-0.32935
85439	0.595960	1.024352	-0.241517	-1.939784	-0.744871	1.833607	2.239866	-0.150349	0.548212	0.301066	...	-0.103696	-0.175443	-0.677665	0.40574
85440	-1.283046	-0.088930	0.661577	1.423288	1.435471	0.328697	0.406776	0.419864	-0.116955	-0.950739	...	0.306222	0.067209	0.484102	-0.22733
85441	0.788872	1.021513	-0.291041	-0.437183	0.162890	-0.123230	0.251870	-0.523884	0.034036	1.048756	...	-0.047959	0.265093	1.066138	0.08579
85442	-1.286711	0.542403	-0.186323	0.382986	0.478970	-0.553821	-0.268745	-0.266549	0.092194	0.529950	...	-0.138381	0.076936	-0.022481	-0.01779

85443 rows × 30 columns

Fig: 4.2

Chapter 5

MODEL BUILDING AND EVALUATION

Brief about the algorithms used

- **Logistic regression Model**

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of the target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

logistic regression

```
In [32]: from sklearn.linear_model import LogisticRegression
         reg=LogisticRegression()
         reg.fit(X_train,y_train)
```

5.1.1 predicting the model based on test data:

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

```
In [38]: #predicting the model on test data
         y_test_pred=reg.predict(X_test)

In [39]: y_test==y_test_pred

Out[39]: 169876    True
         127467    True
         137900    True
         21513     True
         134700    True
                   ...
         128956    True
         177494    True
         26287     True
         198160    True
         25893     True
         Name: Class, Length: 85443, dtype: bool

In [40]: #classification report
         from sklearn.metrics import classification_report,confusion_matrix
         print(classification_report(y_test,y_test_pred))

                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     85308
                    1       0.71      0.62      0.66       135

             accuracy                           1.00     85443
            macro avg       0.85      0.81      0.83     85443
         weighted avg       1.00      1.00      1.00     85443
```

Fig :5.1.1

Fig :5.1.2

➢ KNN Model

The *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric method proposed by Thomas Cover used for classification and regression.[1] In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression.

In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k*

nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

```
[45]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier()
      knn.fit(X_train, y_train)
```

5.1.3predicting the model:

```
[49]: y_test==y_test_pred

t[49]: 169876    True
       127467    True
       137900    True
       21513     True
       134700    True
                 ...
       128956    True
       177494    True
       26287     True
       198160    True
       25893     True
       Name: Class, Length: 85443, dtype: bool

[50]: from sklearn.metrics import classification_report
      print(classification_report(y_test,y_test_pred))

                    precision    recall  f1-score   support

                 0       1.00      1.00      1.00     85308
                 1       1.00      0.03      0.06       135

          accuracy                           1.00     85443
         macro avg       1.00      0.51      0.53     85443
      weighted avg       1.00      1.00      1.00     85443
```

Fig :5.1.3

➢ Random Forest Model

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction

(regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

```
: from sklearn.ensemble import RandomForestClassifier
  rfc = RandomForestClassifier()
  rfc.fit(X_train, y_train)
```

5.1.5 predicting the model:

```
In [48]: y_test_pred=rfc.predict(X_test)

In [49]: y_test==y_test_pred

Out[49]: 169876    True
         127467    True
         137900    True
         21513     True
         134700    True
                   ...
         128956    True
         177494    True
         26287     True
         198160    True
         25893     True
         Name: Class, Length: 85443, dtype: bool

In [50]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_test_pred))

                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     85308
                    1       0.90      0.75      0.82       135

             accuracy                           1.00     85443
            macro avg       0.95      0.87      0.91     85443
         weighted avg       1.00      1.00      1.00     85443
```

Fig:5.1.5

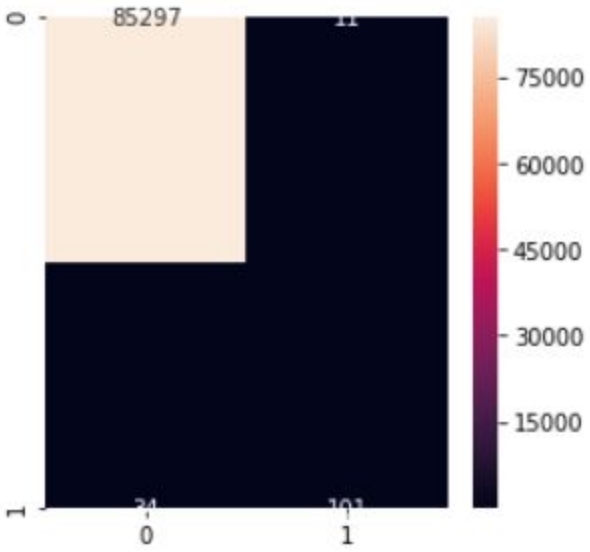
Confusion matrix:

In [51]:
```
#conusion matrix
matrix=confusion_matrix(y_test,y_test_pred)
print(matrix)
```

```
[[85297    11]
 [   34   101]]
```

In [52]:
```
plt.figure(figsize = (4,4))
sns.heatmap(matrix,annot=True,fmt="d")
```

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x24a0560ef48>



Applying gridsearchcv:

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=101)
```

```
param_grid = {
    'n_estimators':[200,500],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[4,5,6,7,8],
    'criterion':['gini','entropy']
}
```

:
```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5)
CV_rfc.fit(X_train, y_train)
```

```
CV_rfc.best_params_
```

- **After applying the grid search in jupiter and colab i am able to get the required output but if i wanted to do it again it's taking so much time ,hence i took the highest recall and f1-scores as my metrics and performed visualization on them.**

- **But I have done the random forest classifier model in another jupiter file and I took all the values same as the main jupiter file and performed gridsearch and the result turned out to be the highest precision value after applying gridsearchcv. The precision value increased from 0.91 to 0.93.and when i did it in my main jupiter file the recall and f1 score has increased.**

In other jupiter file:

random forest classifier

In [28]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(random_state=101)
```

In [29]:
```
param_grid = {
    'n_estimators':[12,14],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[4,5,6,7,8],
    'criterion':['gini','entropy']
}
```

In [30]:
```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5)
CV_rfc.fit(X_train, y_train)
```

Out[30]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=101),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [12, 14]})
```

In [31]:
```
CV_rfc.best_params_
```

Out[31]:
```
{'criterion': 'entropy',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 12}
```

In [32]:
```
rfc1=RandomForestClassifier(random_state=101, max_features='auto', n_estimators=500, max_depth=8, criterion='entropy')
```

In [33]:
```
rfc1.fit(X_train, y_train)
```

Out[33]:
```
RandomForestClassifier(criterion='entropy', max_depth=8, n_estimators=500,
                       random_state=101)
```

In [34]:
```
rfc1.score(X_train,y_train)
rfc1.score(X_test,y_test)
```

Out[34]:
```
0.9994967405170698
```

In [35]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [36]:
```
pred = rfc1.predict(X_test)
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85308
           1       0.93      0.74      0.82       135

    accuracy                           1.00     85443
   macro avg       0.96      0.87      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

In [ ]:
```
n
```

In main jupiter file:

In [61]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(random_state=1)
```

In [62]:
```
param_grid = {
    'n_estimators':[6,7],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[4,5,6,7,8],
    'criterion':['gini','entropy']
}
```

In [63]:
```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5)
CV_rfc.fit(X_train, y_train)
```

Out[63]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [6, 7]})
```

In [64]:
```
CV_rfc.best_params_
```

Out[64]:
```
{'criterion': 'entropy',
 'max_depth': 7,
 'max_features': 'log2',
 'n_estimators': 7}
```

In [65]:
```
rfc=RandomForestClassifier()
rfc.fit(X_train, y_train)
```

Out[65]:
```
RandomForestClassifier()
```

In [66]:
```
rfc.score(X_test,y_test)
```

Out[66]:
```
0.9994967405170698
```

```
In [68]: y_test_pred=rfc.predict(X_test)
```

```
In [69]: y_test==y_test_pred
```

```
Out[69]: 169876    True
         127467    True
         137900    True
         21513     True
         134700    True
                   ...
         128956    True
         177494    True
         26287     True
         198160    True
         25893     True
         Name: Class, Length: 85443, dtype: bool
```

```
In [70]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85308
           1       0.90      0.76      0.83       135

    accuracy                           1.00     85443
   macro avg       0.95      0.88      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

Chapter -6

Evaluation of Models

Here we have used 3 models i.e

- **Logistic regression**
- **KNN Model**
- **Random Forest Model**

So by comparing all the accuracies we can infer that rfc model has a good accuracy ,precision ,f1 score ,recall . so the RFC model is selected among the models for detecting the fraud done.

6.1.1 visualizing the models according to their recall values:

comparing the recall values

```
In [53]: x=[ 0.62  , 0.03,0.76 ]
         y=["Logistic Regression","knn"," Random Forest Classifier"]
         plt.figure(figsize=(10,10))
         plt.bar(x,y)
         plt.xlabel("Range")
         plt.ylabel("Models")
         plt.title("recall Value comparison")

Out[53]: Text(0.5, 1.0, 'recall Value comparison')
```

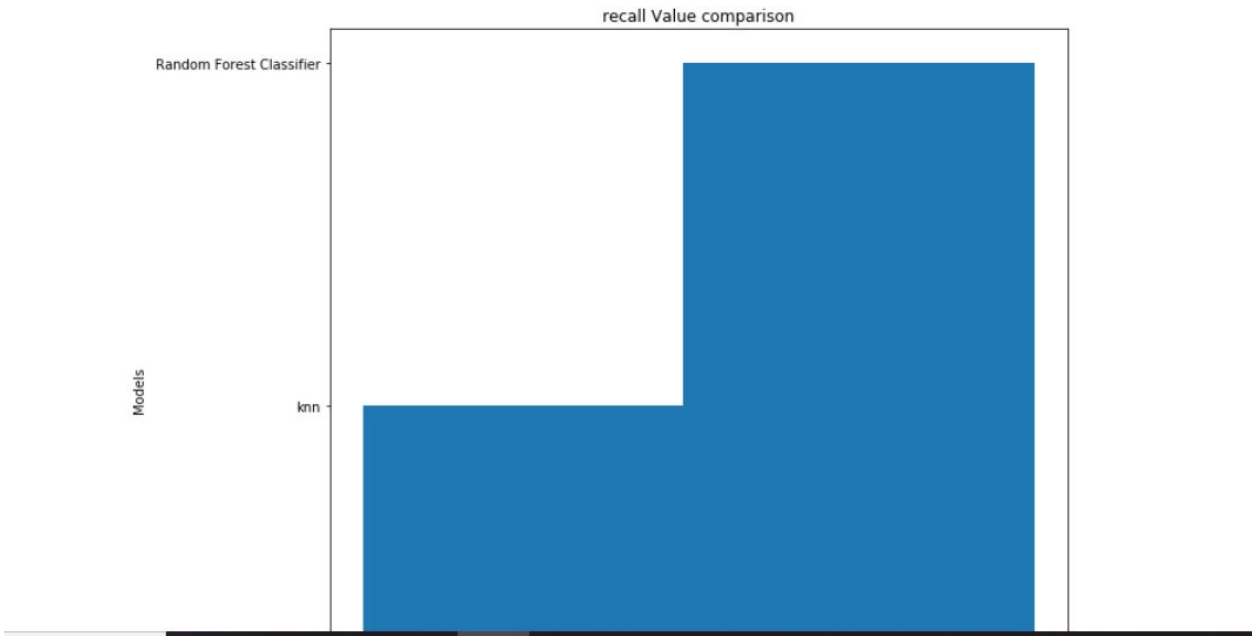


Fig :6.1.1

6.1.2 visualizing using barplot:

```
In [54]: #barplot of the recall values
         plt.figure(figsize=(10,10))
         plt.xlabel("Range")
         plt.ylabel("Models")
         sns.barplot(x,y)
```

```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x26b114ad808>
```

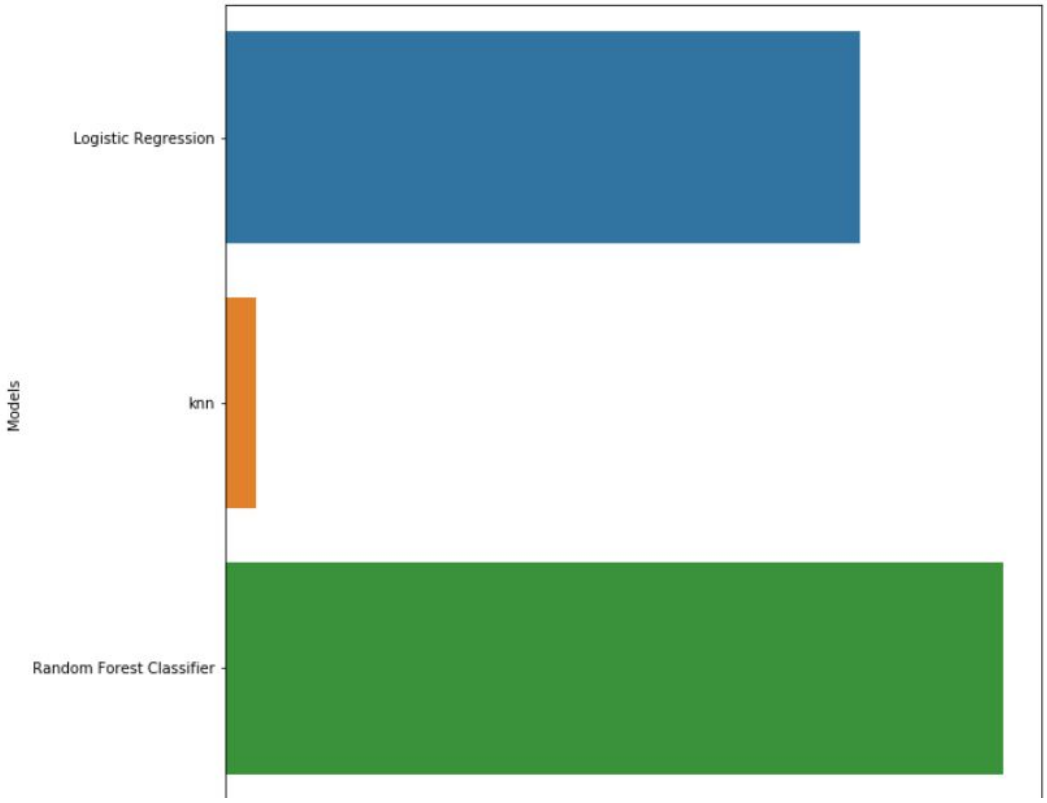


Fig :6.1.2

6.1.3 visualizing using countplot:

```
In [55]: #countplot of the recall values
         plt.figure(figsize=(10,10))
         sns.countplot(x)
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x26b11dcbfc8>
```

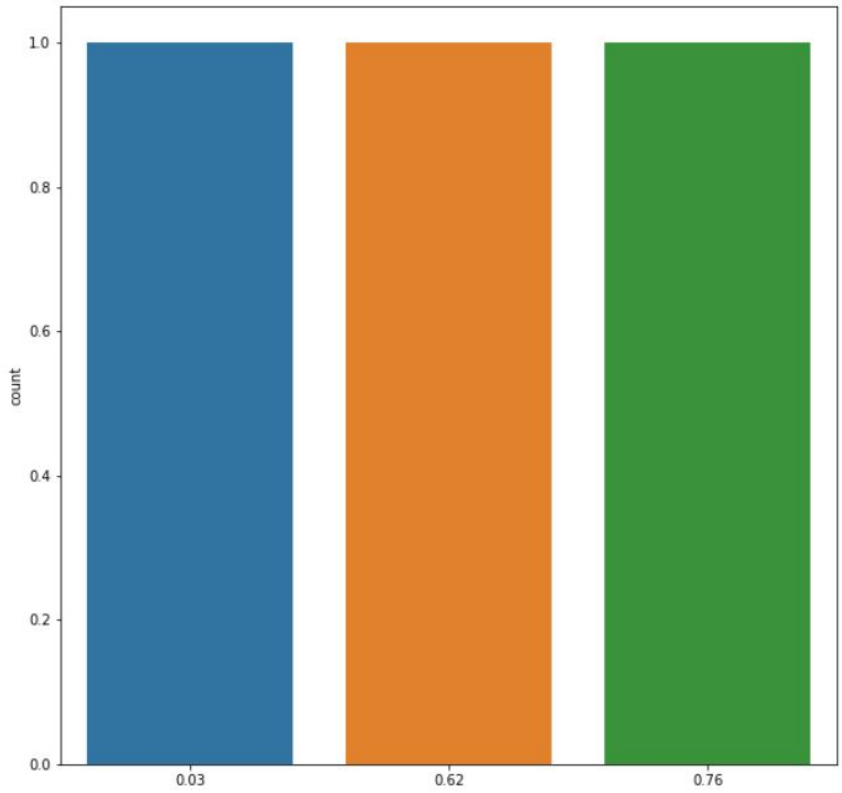
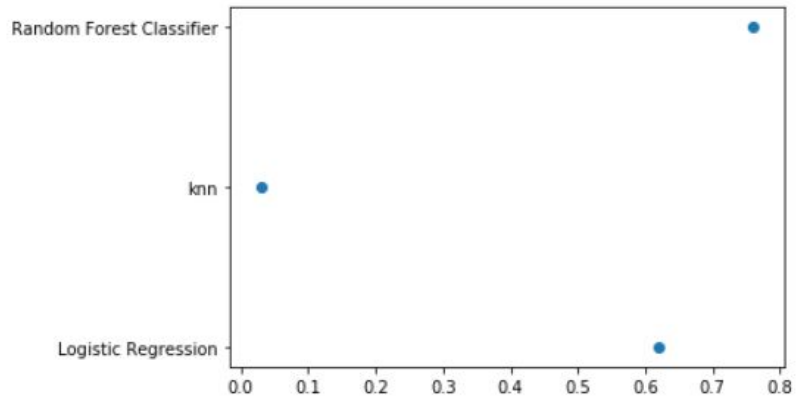


Fig :6.1.3

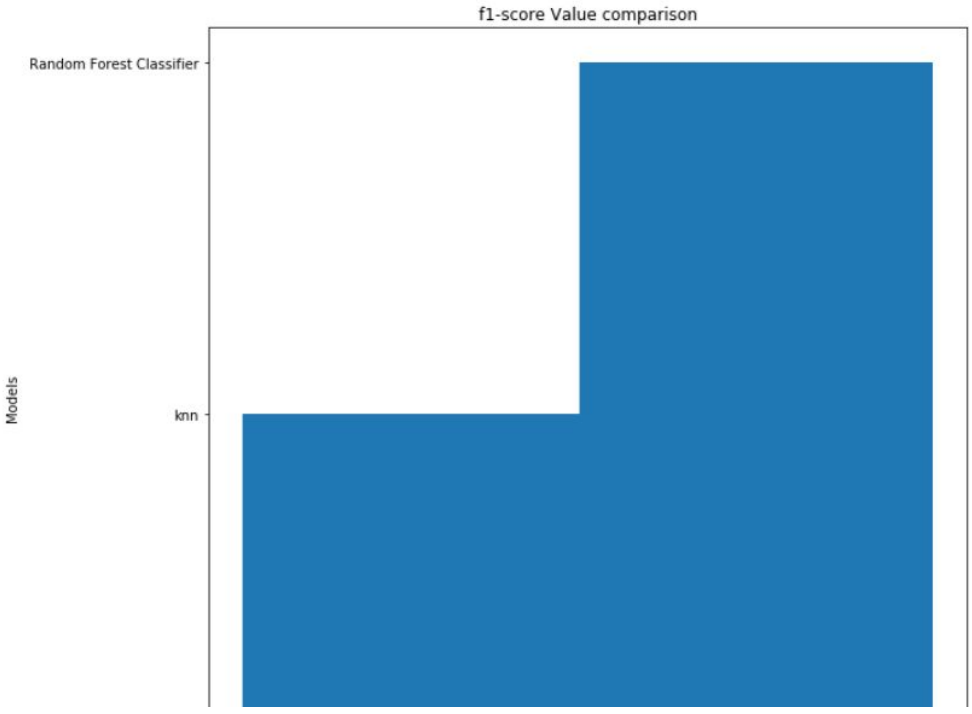
Visualizing using scatter plot;

```
In [56]: #by visualizing with scatter plot
         plt.scatter(x,y)
         plt.show()
```



6.1.4 visualizing the models according to their f1-score values:

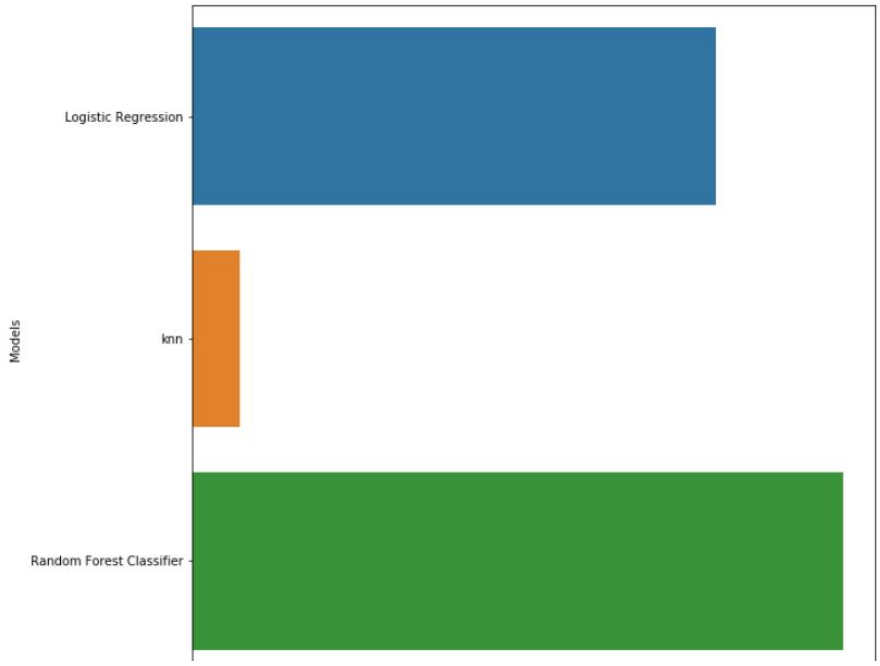
```
In [57]: x=[0.66, 0.06 ,0.82 ]
         y=["Logistic Regression","knn"," Random Forest Classifier"]
         plt.figure(figsize=(10,10))
         plt.bar(x,y)
         plt.xlabel("Range")
         plt.ylabel("Models")
         plt.title("f1-score Value comparison")
Out[57]: Text(0.5, 1.0, 'f1-score Value comparison')
```



6.1.5 visualizing using barplot:

```
In [58]: #barplot of the f1score values
         plt.figure(figsize=(10,10))
         plt.xlabel("Range")
         plt.ylabel("Models")
         sns.barplot(x,y)
```

```
Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x1a7685bca48>
```



->from the above visualizations by comparing precision,recall,f1-score values of the three methods(logistic regression,knn,random forest classifier)we can say that by using random forest classifier we can get the best output.

Note : Even after taking all three values of precision,recall,and f1 score,i had the best output of precision value for knn but recall and f1 score are very less for the knn model,so i didnt consider knn as ideal model ,hence i took random forest model as my best model which has the best metric values and also performed grid search on randomforest.

CONCLUSION:

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set .we can conclude that random first model is best suitable for detecting the frauds made using credit card transactions , and we could easily differentiate among the normal and fraud transactions using this model.

References:

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

https://github.com/topics/credit-card-fraud.