CREDIT CARD FRAUD DETECTION

PROJECT REPORT

SUBMITTED AS PART OF THE COURSE:**EXPLORATORY DATA ANALYSIS**(EDA)

CSE3040

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT CHENNAI

WINTER 2020-2021

COURSE FACULTY: DR.SUBHRA RANI PATRA

SUBMITTED BY:

NALLA VEDAVATHI – 19MIA1106

SNEHA M – 19MIA1101

NITHYA SHARMA – 19MIA1028

K NIHARIKA SAMYUKTHA – 19MIA1083

AKSHITHA BACHU – 19MIA1096

ABSTRACT:

WE TOOK THIS DATA SET AS OUR ASSIGNMENT AND TRIED TO PERFORM THE EDA TO THE BEST OF OUR CAPABILITY! IT IS IMPORTANT THAT CREDIT CARD COMPANIES ARE ABLE TO RECOGNIZE FRAUDULENT CREDIT CARD TRANSACTIONS SO THAT CUSTOMERS ARE NOT CHARGED FOR ITEMS THAT THEY DID NOT PURCHASE. "FRAUD DETECTION IS A SET OF ACTIVITIES THAT ARE TAKEN TO PREVENT MONEY OR PROPERTY FROM BEING OBTAINED THROUGH FALSE PRETENCES."

FRAUD IS A MAJOR PROBLEM FOR THE WHOLE CREDIT CARD INDUSTRY THAT GROWS BIGGER WITH THE INCREASING POPULARITY OF ELECTRONIC MONEY TRANSFERS. TO EFFECTIVELY PREVENT THE CRIMINAL ACTIONS THAT LEAD TO THE LEAKAGE OF BANK ACCOUNT INFORMATION LEAK, SKIMMING, COUNTERFEIT CREDIT CARDS, THE THEFT OF BILLIONS OF DOLLARS ANNUALLY, AND THE LOSS OF REPUTATION AND CUSTOMER LOYALTY, CREDIT CARD ISSUERS SHOULD CONSIDER THE IMPLEMENTATION OF ADVANCED CREDIT CARD FRAUD PREVENTION AND FRAUD DETECTION METHODS. MACHINE LEARNING-BASED METHODS CAN CONTINUOUSLY IMPROVE THE ACCURACY OF FRAUD PREVENTION BASED ON INFORMATION ABOUT EACH CARDHOLDER'S BEHAVIOUR.

INTRODUCTION:

FRAUD CAN BE COMMITTED IN DIFFERENT WAYS AND IN MANY INDUSTRIES. THE MAJORITY OF DETECTION METHODS COMBINE A VARIETY OF FRAUD DETECTION DATASETS TO FORM A CONNECTED OVERVIEW OF BOTH VALID AND NON-VALID PAYMENT DATA TO MAKE A DECISION. THIS DECISION MUST CONSIDER IP ADDRESS, GEOLOCATION, DEVICE IDENTIFICATION, "BIN" DATA, GLOBAL LATITUDE/LONGITUDE, HISTORIC TRANSACTION PATTERNS, AND THE ACTUAL TRANSACTION INFORMATION. IN PRACTICE, THIS MEANS THAT MERCHANTS AND ISSUERS DEPLOY ANALYTICALLY BASED RESPONSES THAT USE INTERNAL AND EXTERNAL DATA TO APPLY A SET OF BUSINESS RULES OR ANALYTICAL ALGORITHMS TO DETECT FRAUD.

CREDIT CARD FRAUD IS USUALLY CAUSED EITHER BY CARD OWNER'S NEGLIGENCE WITH HIS DATA OR BY A BREACH IN A WEBSITE'S SECURITY. HERE ARE SOME EXAMPLES:

- A CONSUMER REVEALS HIS CREDIT CARD NUMBER TO UNFAMILIAR INDIVIDUALS.
- A CARD IS LOST OR STOLEN AND SOMEONE ELSE USES IT.
- MAIL IS STOLEN FROM THE INTENDED RECIPIENT AND USED BY CRIMINALS.
- BUSINESS EMPLOYEES COPY CARDS OR CARD NUMBERS OF ITS OWNER.
- MAKING A COUNTERFEIT CREDIT CARD.

WHEN YOUR CARD IS LOST OR STOLEN, AN UNAUTHORIZED CHARGE CAN HAPPEN; IN OTHER WORDS, THE PERSON WHO FINDS IT USES IT FOR A PURCHASE. CRIMINALS CAN ALSO FORGE YOUR NAME AND USE THE CARD OR ORDER SOME GOODS THROUGH A MOBILE PHONE OR COMPUTER. ALSO, THERE IS THE PROBLEM OF USING A COUNTERFEIT CREDIT CARD – A FAKE CARD THAT HAS THE REAL ACCOUNT INFORMATION THAT WAS STOLEN FROM HOLDERS. THAT IS ESPECIALLY DANGEROUS BECAUSE THE VICTIMS HAVE THEIR REAL CARDS, BUT DO NOT KNOW THAT SOMEONE HAS COPIED THEIR CARD. SUCH FRAUDULENT CARDS LOOK QUITE LEGITIMATE AND HAVE THE LOGOS AND ENCODED MAGNETIC STRIPS OF THE ORIGINAL ONE. FRAUDULENT CREDIT CARDS ARE USUALLY DESTROYED BY THE CRIMINALS AFTER SEVERAL SUCCESSFUL PAYMENTS, JUST BEFORE A VICTIM REALIZES THE PROBLEM AND REPORTS IT.

THIS DATA SET ON KAGGLE DEALING WITH CREDIT CARD FRAUD DETECTION. THE DATASET CONTAINS TRANSACTIONS MADE BY CREDIT CARDS IN SEPTEMBER 2013 BY EUROPEAN CARDHOLDERS. THE DATA SET HAS 31 FEATURES, 28 OF WHICH HAVE BEEN ANONYMIZED

AND ARE LABELLED V1 THROUGH V28. THE REMAINING THREE FEATURES ARE THE TIME AND THE AMOUNT OF THE TRANSACTION AS WELL AS WHETHER THAT TRANSACTION WAS FRAUDULENT OR NOT. THIS DATASET PRESENTS TRANSACTIONS THAT OCCURRED IN TWO DAYS, WHERE WE HAVE 492 FRAUDS OUT OF 284,807 TRANSACTIONS. THE DATASET IS HIGHLY UNBALANCED, THE POSITIVE CLASS (FRAUDS) ACCOUNT FOR 0.172% OF ALL TRANSACTIONS. BEFORE IT WAS UPLOADED TO KAGGLE, THE ANONYMIZED VARIABLES HAD BEEN MODIFIED IN THE FORM OF A PCA (PRINCIPAL COMPONENT ANALYSIS).

IT CONTAINS ONLY NUMERICAL INPUT VARIABLES WHICH ARE THE RESULT OF A PCA TRANSFORMATION. UNFORTUNATELY, DUE TO CONFIDENTIALITY ISSUES, WE CANNOT PROVIDE THE ORIGINAL FEATURES AND MORE BACKGROUND INFORMATION ABOUT THE DATA. 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.

GIVEN THE CLASS IMBALANCE RATIO, WE RECOMMEND MEASURING THE ACCURACY USING THE AREA UNDER THE PRECISION-RECALL CURVE (AUPRC). CONFUSION MATRIX ACCURACY IS NOT MEANINGFUL FOR UNBALANCED CLASSIFICATION.

SINCE NEARLY ALL PREDICTORS HAVE BEEN ANONYMIZED, I DECIDED TO FOCUS ON THE NON-ANONYMIZED PREDICTORS TIME AND AMOUNT OF THE TRANSACTION DURING MY EDA. THE DATA SET CONTAINS 284,807 TRANSACTIONS. THE MEAN VALUE OF ALL TRANSACTIONS IS $88.35 WHILE THE LARGEST TRANSACTION RECORDED IN THIS DATA SET AMOUNTS TO $25,691.16. HOWEVER, AS YOU MIGHT BE GUESSING RIGHT NOW BASED ON THE MEAN AND MAXIMUM, THE DISTRIBUTION OF THE MONETARY VALUE OF ALL TRANSACTIONS IS HEAVILY RIGHT-SKEWED. THE VAST MAJORITY OF TRANSACTIONS ARE RELATIVELY SMALL AND ONLY A TINY FRACTION OF TRANSACTIONS COMES EVEN CLOSE TO THE MAXIMUM.

CREDIT CARD FRAUD DETECTION IS A TYPICAL EXAMPLE OF CLASSIFICATION. IN THIS PROCESS, WE HAVE FOCUSED MORE ON ANALYSING THE FEATURE MODELLING AND POSSIBLE BUSINESS USE CASES OF THE ALGORITHM’S OUTPUT THAN ON THE ALGORITHM ITSELF.

PROBLEM STATEMENT:

THE CREDIT CARD FRAUD DETECTION PROBLEM INCLUDES MODELLING 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.

LITERATURE SURVEY/RELATED WORK:

- HTTPS://TOWARDSDATASCIENCE.COM/DETECTING-CREDIT-CARD-FRAUD-USING-MACHINE-LEARNING-A3D83423D3B8
- HTTPS://WWW.KAGGLE.COM/DATAENGEL/CREDIT-CARD-FRAUD-DETECTION-WITH-ML-AND-DP
- HTTPS://WWW.KAGGLE.COM/HAZRATNIT/CREDIT-FRAUD-DETECTION
- HTTPS://SPD.GROUP/MACHINE-LEARNING/CREDIT-CARD-FRAUD-DETECTION/
- HTTPS://WWW.KAGGLE.COM/RENJITHMADHAVAN/CREDIT-CARD-FRAUD-DETECTION-USING-PYTHON
- HTTPS://WWW.KAGGLE.COM/PARULPANDEY/A-GUIDE-TO-HANDLING-MISSING-VALUES-IN-PYTHON
- HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2021/01/A-QUICK-INTRODUCTION-TO-K-NEAREST-NEIGHBOR-KNN-CLASSIFICATION-USING-PYTHON/
- HTTPS://STACKABUSE.COM/K-NEAREST-NEIGHBORS-ALGORITHM-IN-PYTHON-AND-SCIKIT-LEARN/
- HTTPS://WWW.GEEKSFORGEEKS.ORG/DECISION-TREE-IMPLEMENTATION-PYTHON/
- HTTPS://WWW.W3SCHOOLS.COM/PYTHON/PYTHON_ML_DECISION_TREE.ASP

PROPOSED WORK

FINDING DATASET

FINDING PROBLEM STATEMENT

DATA UNDERSTANDING

HANDLING MISSING DATA

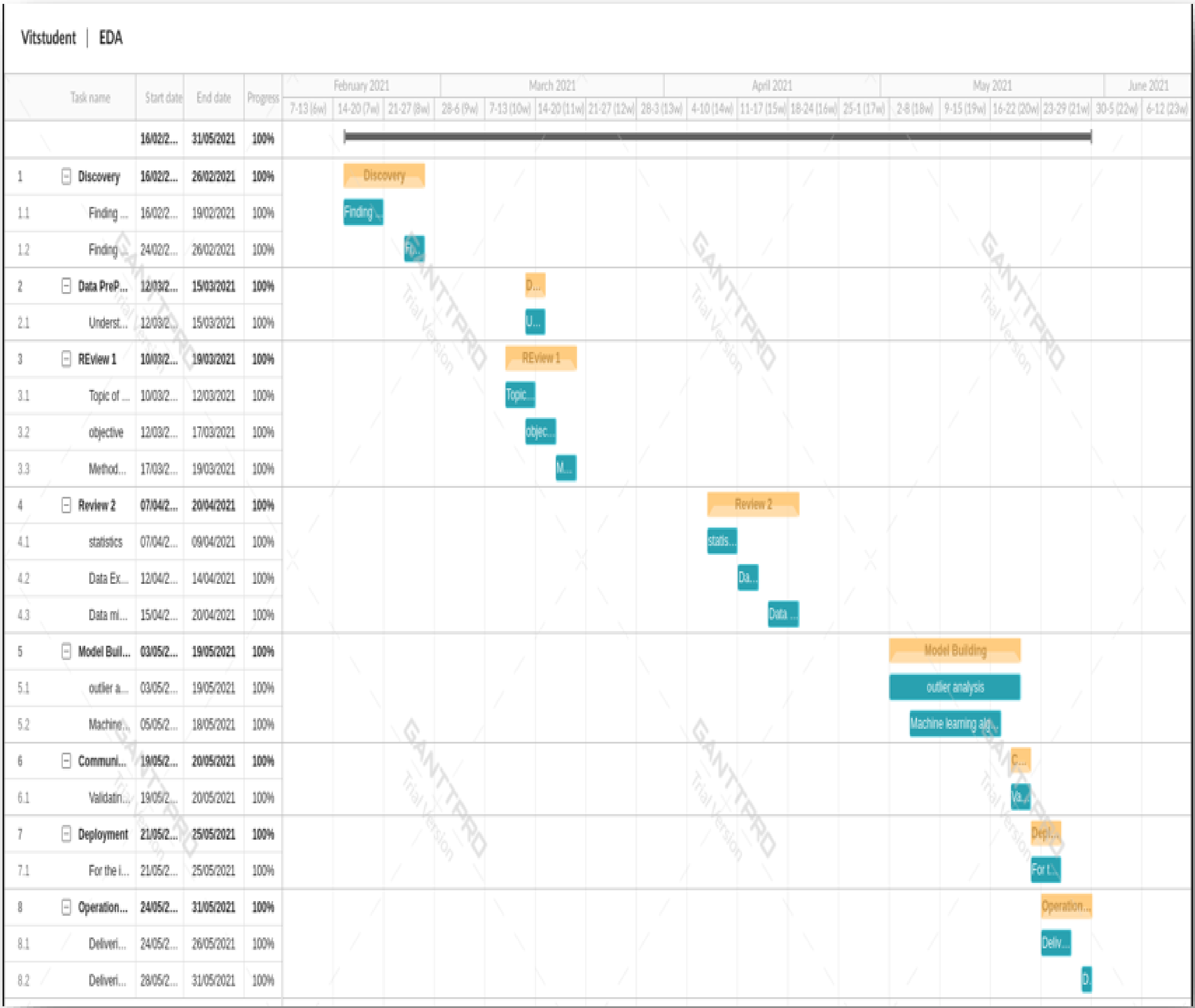
OUTLIER IDENTIFICATION AND CORRECTION

WORKING MODEL

- K-NEAREST NEIGHBORS (KNN) CLASSIFICATION
- SUPPORT VECTOR MACHINE(SVM) CLASSIFICATION

- DECISION TREE CLASSIFICATION
- RANDOM FOREST CLASSIFICATION

FLOW CHART



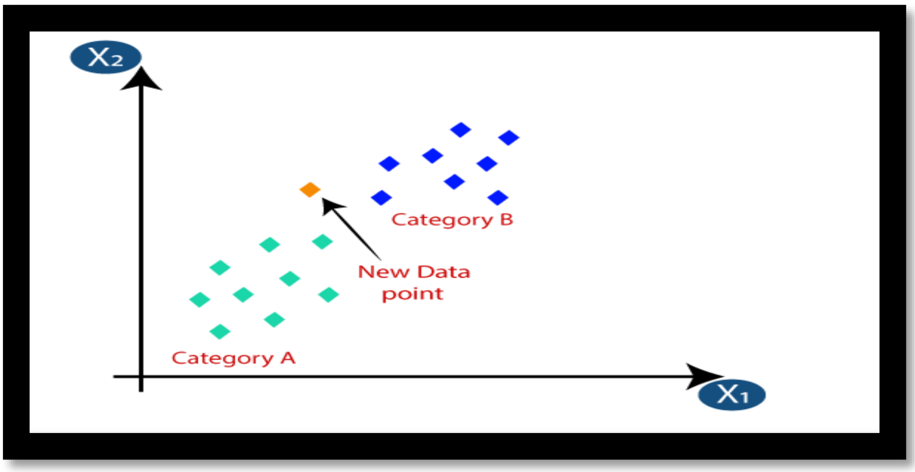
DESCRIPTION OF EACH MODULES

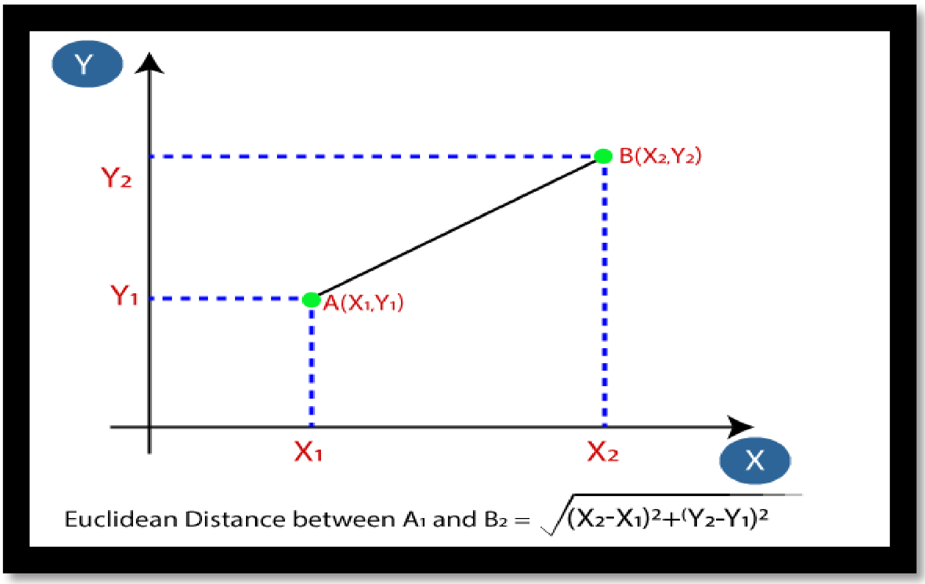
K-NEAREST NEIGHBOUR MODULE

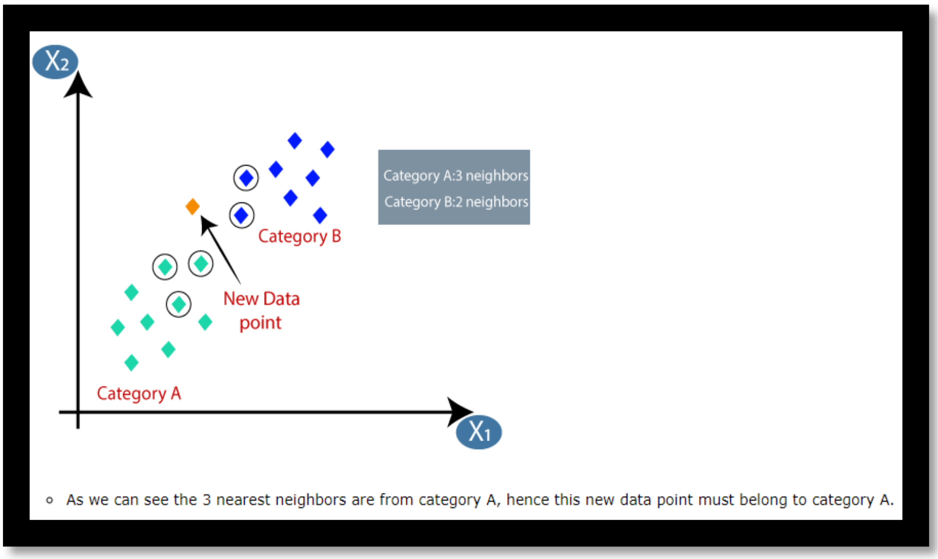
- K-NEAREST NEIGHBOUR IS ONE OF THE SIMPLEST MACHINE LEARNING ALGORITHMS BASED ON SUPERVISED LEARNING TECHNIQUE.
- K-NN ALGORITHM ASSUMES THE SIMILARITY BETWEEN THE NEW CASE/DATA AND AVAILABLE CASES AND PUT THE NEW CASE INTO THE CATEGORY THAT IS MOST SIMILAR TO THE AVAILABLE CATEGORIES.
- K-NN ALGORITHM 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.

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.


X_2
Category B
New Data
point
Category A
X_1


Y
Y_2
$B(X_2,Y_2)$
Y_1
$A(X_1,Y_1)$
X_1
X_2
X
Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

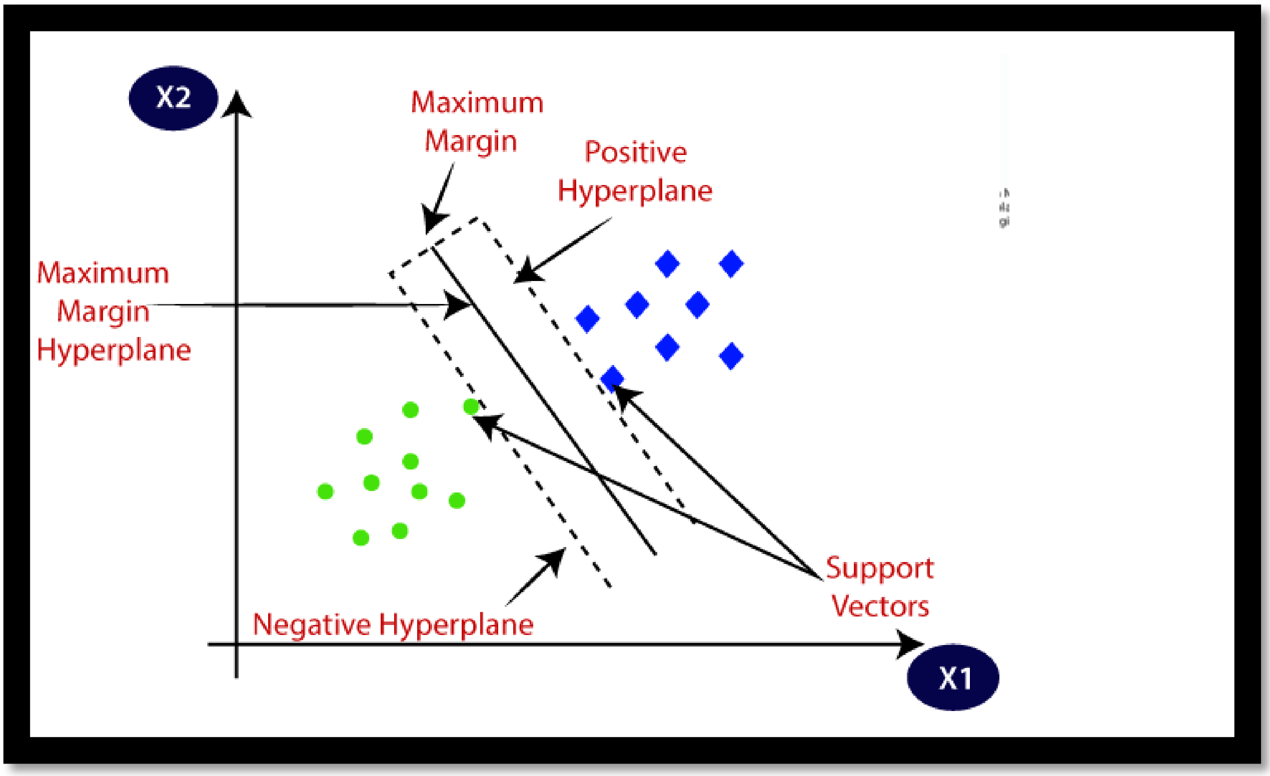

X_2
Category A:3 neighbors
Category B:2 neighbors
Category B
New Data
point
Category A
X_1
As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

SUPPORT VECTOR MACHINE ALGORITHM

SUPPORT VECTOR MACHINE OR SVM IS ONE OF THE MOST POPULAR SUPERVISED LEARNING ALGORITHMS, WHICH IS USED FOR CLASSIFICATION AS WELL AS REGRESSION PROBLEMS. HOWEVER, PRIMARILY, IT IS USED FOR CLASSIFICATION PROBLEMS IN MACHINE LEARNING.

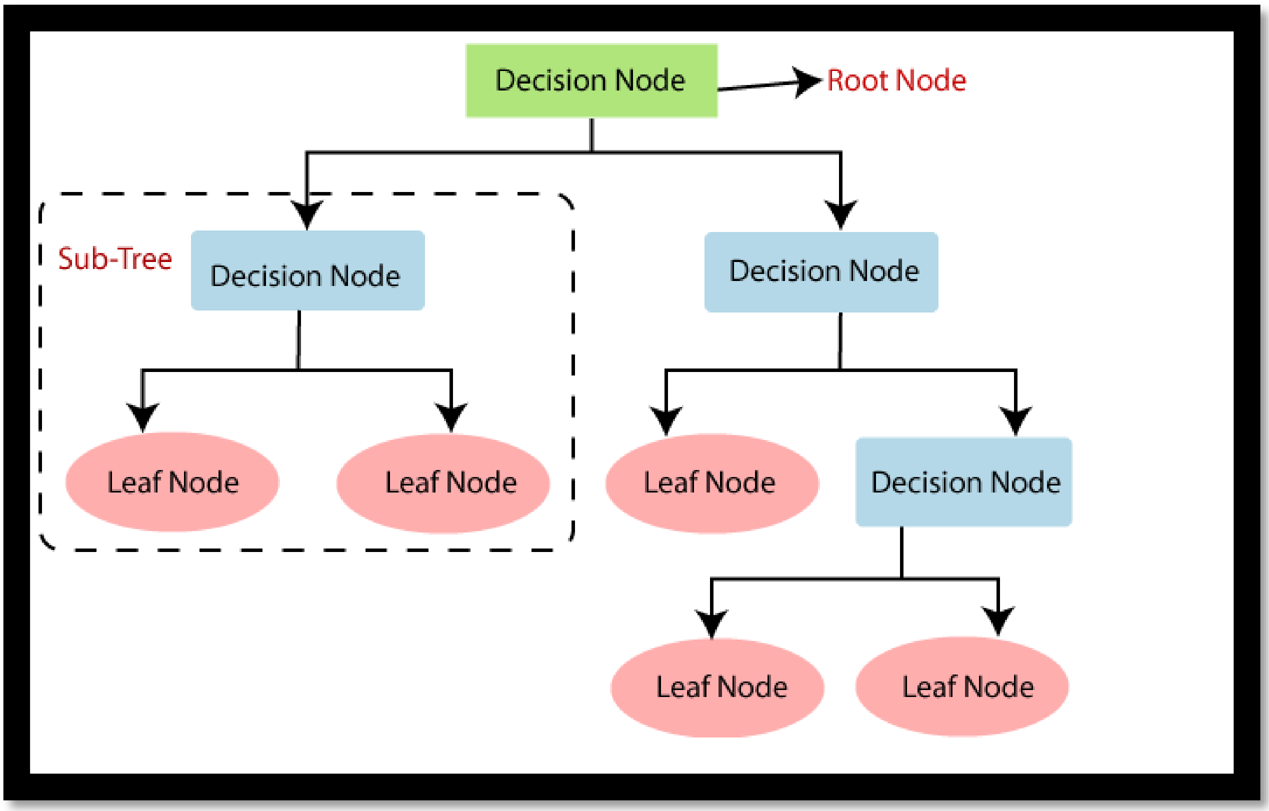
THE GOAL OF THE SVM ALGORITHM IS TO CREATE THE BEST LINE OR DECISION BOUNDARY THAT CAN SEGREGATE N-DIMENSIONAL SPACE INTO CLASSES SO THAT WE CAN EASILY PUT THE NEW DATA POINT IN THE CORRECT CATEGORY IN THE FUTURE. THIS BEST DECISION BOUNDARY IS CALLED A HYPERPLANE.

SVM CHOOSES THE EXTREME POINTS/VECTORS THAT HELP IN CREATING THE HYPERPLANE. THESE EXTREME CASES ARE CALLED AS SUPPORT VECTORS, AND HENCE ALGORITHM IS TERMED AS SUPPORT VECTOR MACHINE.



DECISION TREE CLASSIFICATION ALGORITHM

- DECISION TREE IS A SUPERVISED LEARNING TECHNIQUE THAT CAN BE USED FOR BOTH CLASSIFICATION AND REGRESSION PROBLEMS, BUT MOSTLY IT IS PREFERRED FOR SOLVING CLASSIFICATION PROBLEMS. IT IS A TREE-STRUCTURED CLASSIFIER, WHERE INTERNAL NODES REPRESENT THE FEATURES OF A DATASET, BRANCHES REPRESENT THE DECISION RULES AND EACH LEAF NODE REPRESENTS THE OUTCOME.
- IN A DECISION TREE, THERE ARE TWO NODES, WHICH ARE THE DECISION NODE AND LEAF NODE. DECISION NODES ARE USED TO MAKE ANY DECISION AND HAVE MULTIPLE BRANCHES, WHEREAS LEAF NODES ARE THE OUTPUT OF THOSE DECISIONS AND DO NOT CONTAIN ANY FURTHER BRANCHES.
- THE DECISIONS OR THE TEST ARE PERFORMED ON THE BASIS OF FEATURES OF THE GIVEN DATASET.



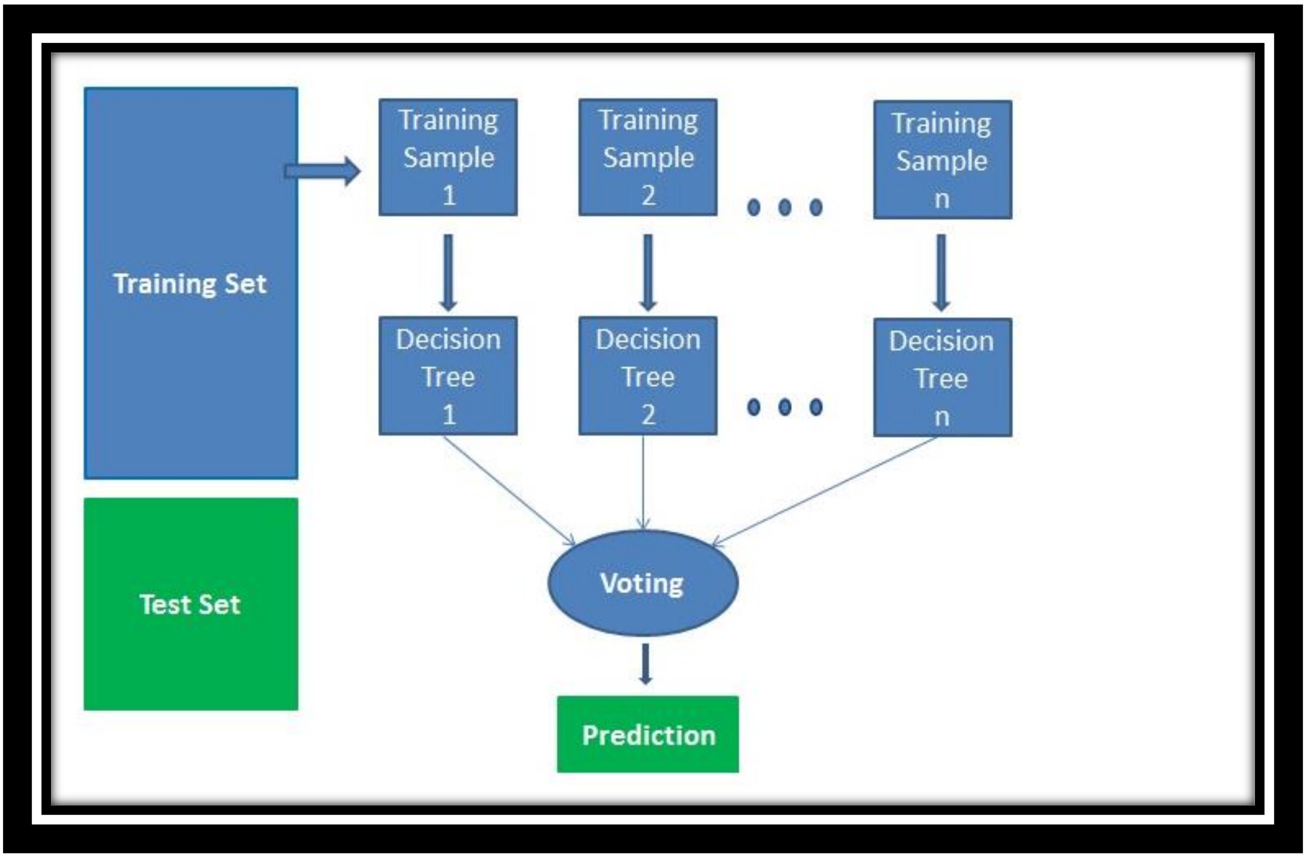
- STEP-1: BEGIN THE TREE WITH THE ROOT NODE, SAYS S, WHICH CONTAINS THE COMPLETE DATASET.
- STEP-2: FIND THE BEST ATTRIBUTE IN THE DATASET USING ATTRIBUTE SELECTION MEASURE (ASM).

- STEP-3: DIVIDE THE S INTO SUBSETS THAT CONTAINS POSSIBLE VALUES FOR THE BEST ATTRIBUTES.
- STEP-4: GENERATE THE DECISION TREE NODE, WHICH CONTAINS THE BEST ATTRIBUTE.
- STEP-5: RECURSIVELY MAKE NEW DECISION TREES USING THE SUBSETS OF THE DATASET CREATED IN STEP -3. CONTINUE THIS PROCESS UNTIL A STAGE IS REACHED WHERE YOU CANNOT FURTHER CLASSIFY THE NODES AND CALLED THE FINAL NODE AS A LEAF NODE.

RANDOM FORESTS CLASSIFIERS

- RANDOM FORESTS IS A SUPERVISED LEARNING ALGORITHM. IT CAN BE USED BOTH FOR CLASSIFICATION AND REGRESSION. IT IS ALSO THE MOST FLEXIBLE AND EASY TO USE ALGORITHM. A FOREST IS COMPRISED OF TREES. IT IS SAID THAT THE MORE TREES IT HAS, THE MORE ROBUST A FOREST IS. RANDOM FORESTS CREATES DECISION TREES ON RANDOMLY SELECTED DATA SAMPLES, GETS PREDICTION FROM EACH TREE AND SELECTS THE BEST SOLUTION BY MEANS OF VOTING. IT ALSO PROVIDES A PRETTY GOOD INDICATOR OF THE FEATURE IMPORTANCE.

- RANDOM FORESTS HAS A VARIETY OF APPLICATIONS, SUCH AS RECOMMENDATION ENGINES, IMAGE CLASSIFICATION AND FEATURE SELECTION. IT CAN BE USED TO CLASSIFY LOYAL LOAN APPLICANTS, IDENTIFY FRAUDULENT ACTIVITY AND PREDICT DISEASES. IT LIES AT THE BASE OF THE BORUTA ALGORITHM, WHICH SELECTS IMPORTANT FEATURES IN A DATASET.



RESULTS AND DISCUSSION:

DATA EXPLORATION AND STATISTICS

DATA EXPLORATION REFERS TO THE INITIAL STEP IN DATA ANALYSIS IN WHICH DATA ANALYSTS USE DATA VISUALIZATION AND STATISTICAL TECHNIQUES TO DESCRIBE DATASET CHARACTERIZATIONS, SUCH AS SIZE, QUANTITY, AND ACCURACY, IN ORDER TO BETTER UNDERSTAND THE NATURE OF THE DATA.

WE USED MANY CHART LIKE BAR, HEAT, BOXPLOT , SCATTER, PIE CHART ETC.. TO VISUALIZE AND UNDERSTAND OUR DATA.

MISSING DATA HANDLING

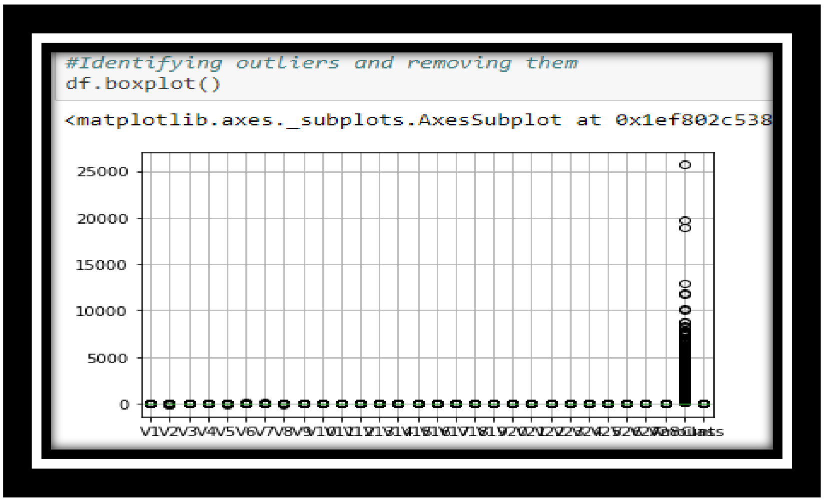
WE TRY TO FIND MISSING VALUES IN OUR DATA SET, USING MISSINGNO FUNCTION AND THERE IS NO MISSING DATA IN OUR FUNCTION.

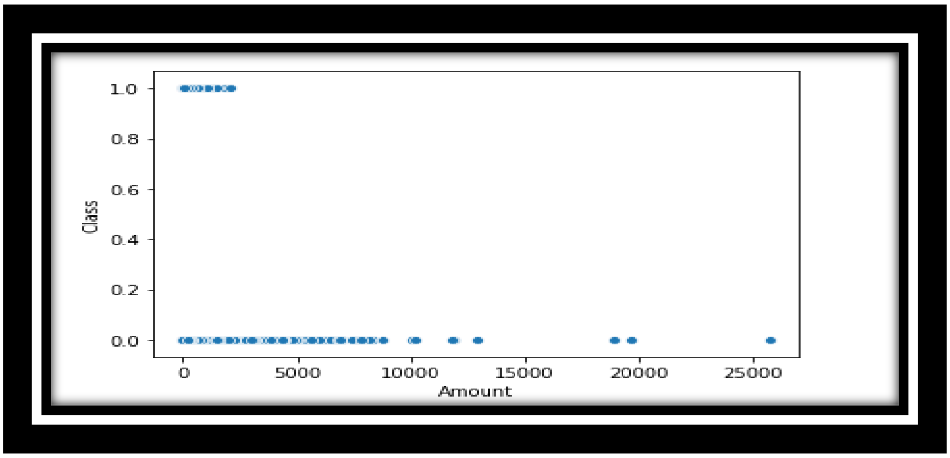
WE USED DIFFERENT TYPES OF HANDLING TECHNIQUES I.E, DELETION TECHNIQUES, IMPUTATION TECHNIQUES.

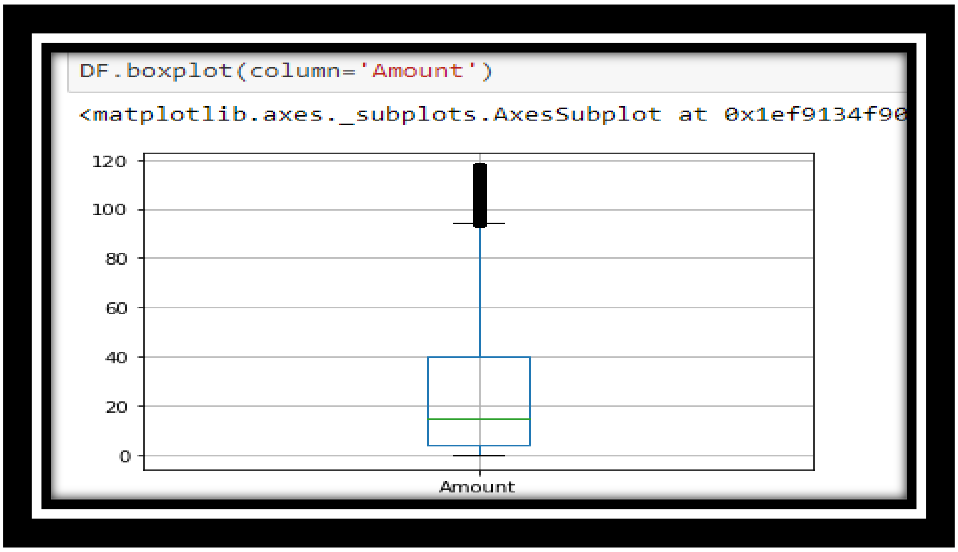
FINALLY, WE DECIDED TO USE MISSING DATA HANDLING USING MEAN. BECAUSE IT GIVES APPROXIMATE VALUES.

OUTLIER IDENTIFICATION

OUR GOAL OF OUTLIER IDENTIFICATION IS TO PROPERLY ANALYSE THE DATA TO DETERMINE WHICH OUTLIERS ARE REPRESENTATIVE OF VALID DATA POINTS (AND SHOULD BE KEPT), AND WHICH OUTLIERS LIKELY REPRESENT ERRORS, AND SHOULD BE REMOVED FROM THE DATA SET. DATA SHOULD NOT BE EXCLUDED SIMPLY BECAUSE THEY ARE IDENTIFIED AS OUTLIERS. WE USED BOX AND SCATTER PLOT FOR IDENTIFYING OUTLIERS. USING OUTLIER FORMULA WE FIND OUTLIERS AND WE REMOVED THAT. WE USED DATA WITHOUT OUTLIERS FOR NEXT WORKING MODULES.

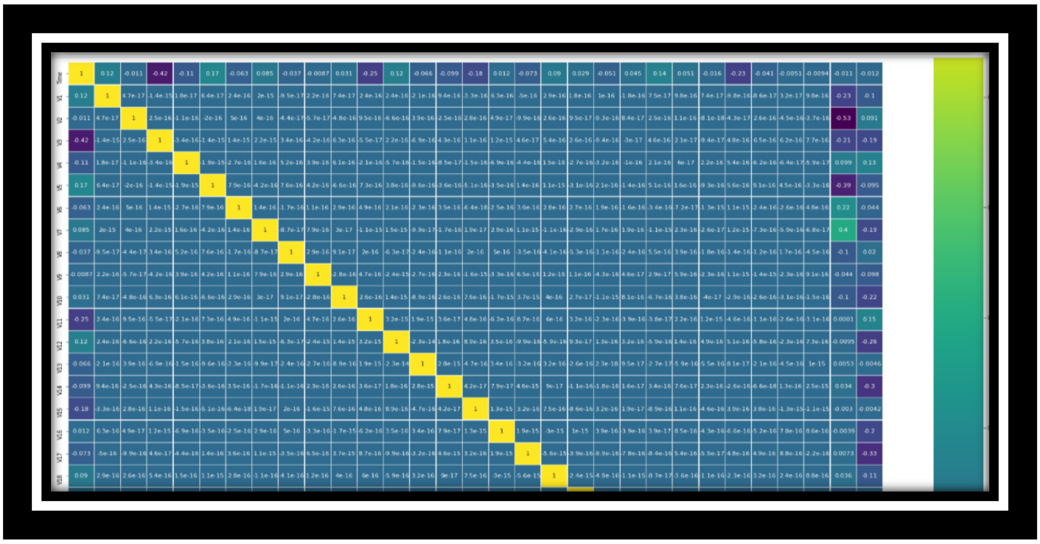






SVM

IN SVM CLASSIFICATION WE USED HEAT MAP TO FIND CORRELATION AMONG VARIABLES



HEAT MAP SHOWS THAT NONE OF THE VARIABLES ARE NOT AUTO-CORRELATED WITH EACH OTHER(AS NONE OF THEM AS SHOWN ARE HAVING DEEP GREEN OR DEEP BLUE)

DEEP GREEN INDICATES AND ABOVE INDICATES POSITIVE AUTO-CORRELATION AND DEEP BLUE INDICATES NEGATIVE AUTO-CORRELATION. ALTHOUGH DEEP GREEN AND BEYOND IS SHOWN IN DIAGONAL LINE. IT IS ONLY THE AUTO-CORRELATION BETWEEN SAME VARIABLES. HENCE IT CAN BE IGNORED

THIS DATASET IS HIGHLY UNBALANCED AND WE FOUND 2 CLASSES IN DATA SET

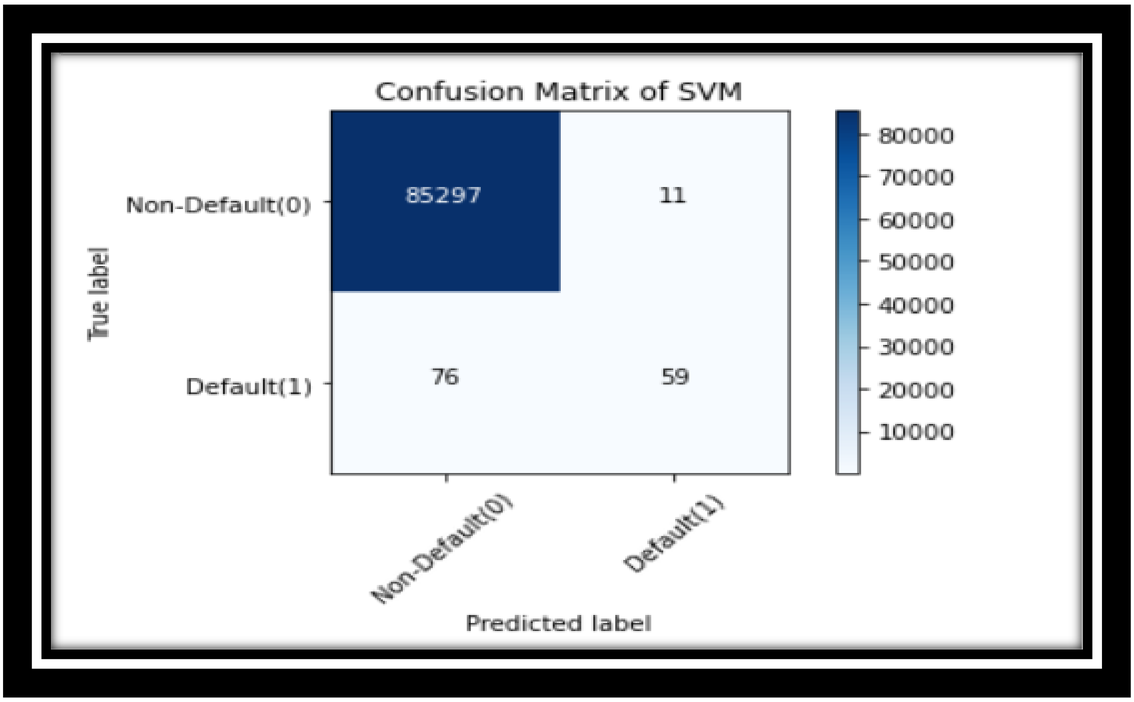
0 --> NORMAL TRANSACTION

1 --> FRAUDULENT TRANSACTION

WE DIVIDED COLUMNS INTO DEPENDENT AND INDEPENDENT . NEXT WE SPLIT TRAINING AND TESTING DATA SET.

WE IMPORT SVC FUNCTION FROM SKLEARN PACKAGE AND USING THIS FUNCTION WE PREDICTED SVM CLASSIFICATION. WE GOT 0.999 ACCURACY OF OUR PREDICTION

IT MATCHES WITH ORIGINAL VALUE.



KNN

FOR KNN CLASSIFICATION WE DIVIDE COLUMNS INTO   DEPENDENT AND INDEPENDENT COLUMNS AS Y AND X

NEXT, WE SPLIT 80% DATA -> TRAINING DATA SET AND 20% DATA INTO -> TESTING DATA SET

We used KNeighborClassifier function that already available from sklearn. neighbours' packages and we predict value.

Next, we found confusion matrix and misclassification and we get almost same value.

```
[[55030     4]
 [   29    70]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55034
           1       0.95      0.71      0.81        99

    accuracy                           1.00     55133
   macro avg       0.97      0.85      0.90     55133
weighted avg       1.00      1.00      1.00     55133
```

DECISION TREE

We divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

To understand model performance, divide the dataset into a training set and a test set. For finding decision tree we used Decision Tree function() Available in packages. We predict data set and we got good accuracy value and we visualize the decision tree.

RANDOM FOREST

We select random samples from a given dataset. We divide given columns into two types of variables dependent (or target variable) and independent variable (or feature variables).To understand model performance, divide the dataset into a training set and a test set

We Construct a decision tree for each sample and get a prediction result from each decision tree.

Performed a vote for each predicted result.

We selected the prediction result with the most votes as the final prediction.

We predict data set and we got good accuracy value.

COMPLETE PROGRAM/CODE AND OUTPUT:

In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
from warnings import filterwarnings
filterwarnings('ignore')
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [2]:
```python
data = pd.read_csv("D:\\4th sem works\\Projects\\Credit card fraud detection\\creditcard.csv")
```

In [3]:
```python
data.head(10)
```

Out[3]:

	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	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.2327
6	4.0	1.229658	NaN	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.7501
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.4152
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.3732
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.0697

10 rows × 31 columns

In [4]:
```python
data.dtypes
```

Out[4]:
```
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     float64
dtype: object
```

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
Time      18
V1        57
V2        62
V3        79
V4        76
V5        77
V6        67
V7        54
V8        57
V9        69
V10       83
V11       71
V12       79
V13       91
V14      106
V15      105
V16      111
V17      108
V18      105
V19       92
V20       80
V21       75
V22       56
V23       70
V24       82
V25       80
V26       65
V27       57
V28       27
Amount    31
Class     18
dtype: int64
```

In [6]:
```
data.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
count	284789.000000	284750.000000	284745.000000	284728.000000	284731.000000	284730.000000	284740.000000	284753.000000	284750.000000	284738.00000
mean	94813.761353	0.000001	-0.000107	0.000054	0.000066	-0.000028	-0.000016	-0.000007	-0.000012	-0.0001
std	47485.002629	1.958765	1.651383	1.516219	1.415878	1.380317	1.332239	1.237157	1.194406	1.0985
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.4340
25%	54203.000000	-0.920409	-0.598655	-0.890308	-0.848636	-0.691643	-0.768234	-0.554067	-0.208610	-0.6431
50%	84691.000000	0.018234	0.065413	0.179865	-0.019816	-0.054358	-0.274180	0.040079	0.022356	-0.0515
75%	139319.000000	1.315634	0.803679	1.027210	0.743403	0.611920	0.398550	0.570384	0.327345	0.5970
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.5949

8 rows × 31 columns

Data processing

```
#Here there is no need of Time column so we are removing the Time Column
data = data.drop("Time", axis=1)
```

```
data.head(10)
```

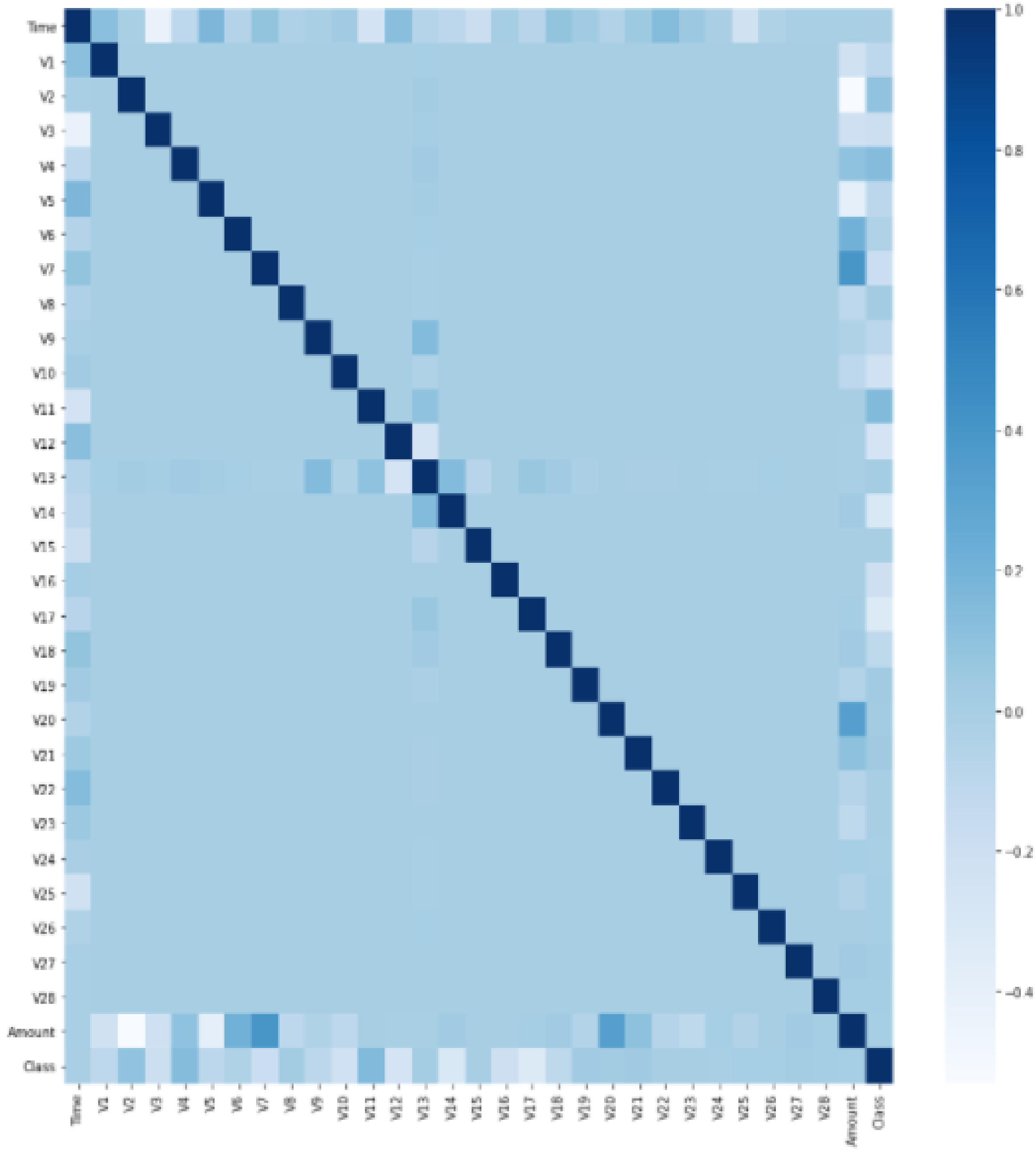
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.
5	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	-0.371407	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.
6	1.229658	NaN	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	-0.099254	...	-0.167716	-0.270710	-0.154104	-0.780055	0.
7	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	1.249376	...	1.943465	-1.015455	0.057504	-0.649709	-0.
8	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	-0.410430	...	-0.073425	-0.268092	-0.204233	1.011592	0.
9	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	-0.366846	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.

10 rows × 30 columns

DATA EXPLORATION

Correlation Matrix

```
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(15,15))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),cmap="Blues")
```



Plotting Data For Normal And Fraud Transactions

```
fraud = data[data['Class']==1]
normal = data[data['Class']==0]

outlierFraction = len(fraud)/float(len(normal))
print("outlier fraction is ",outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
print("There is only 0.17% fraud transactions out all the transactions. The data is highly Unbalanced.")
```

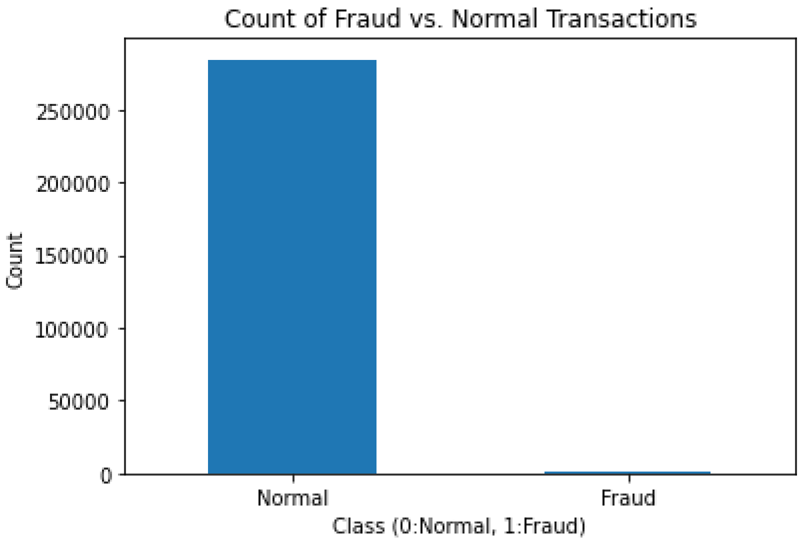
```
outlier fraction is  0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
There is only 0.17% fraud transactions out all the transactions. The data is highly Unbalanced.
```

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
```

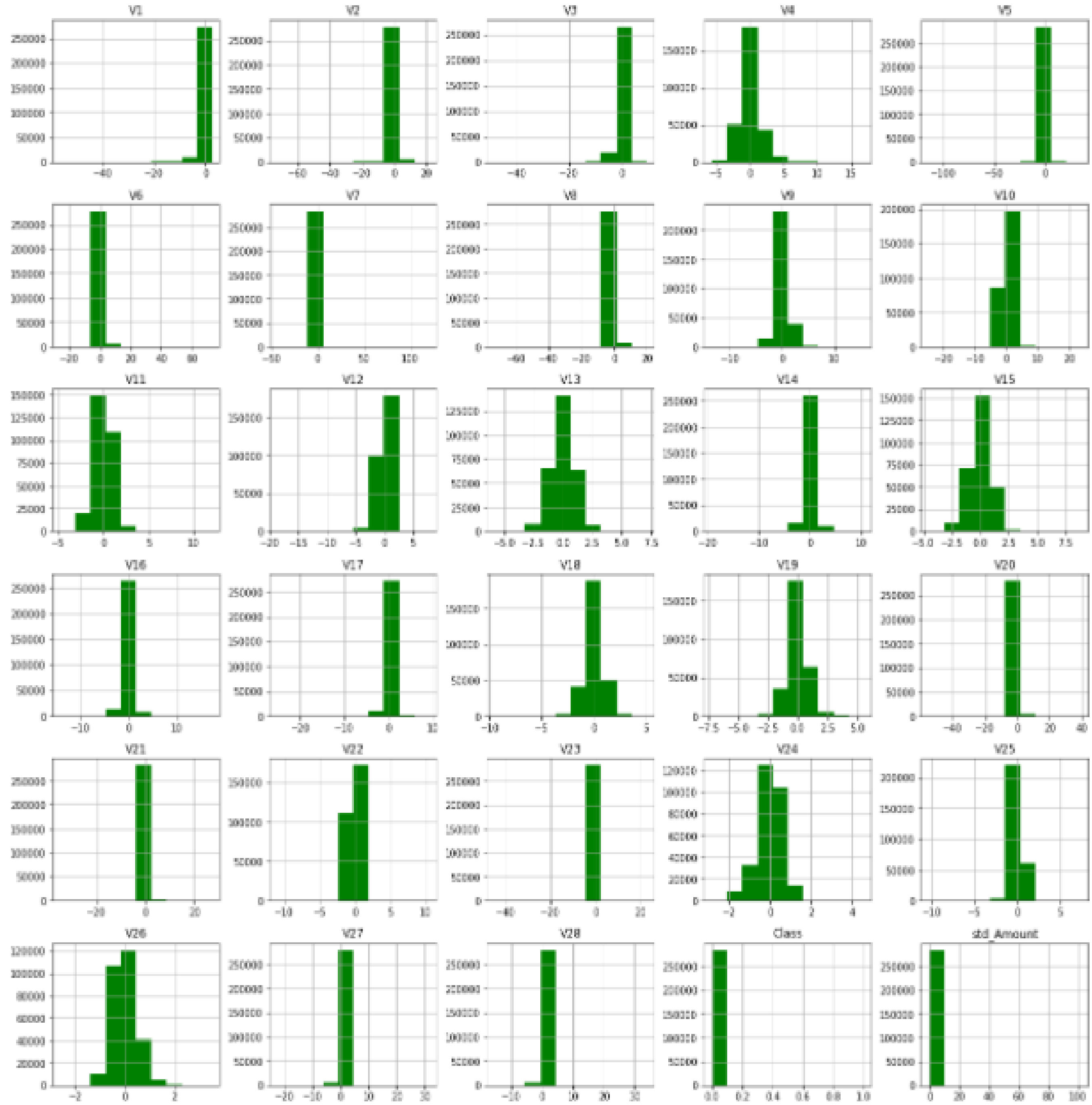
```
#standard scaling
data['std_Amount'] = scaler.fit_transform(data['Amount'].values.reshape (-1,1))

#removing Amount
data = data.drop("Amount", axis=1)
```

```
import matplotlib.pyplot as plt

LABELS = ["Normal", "Fraud"]
count_classes = pd.value_counts(data['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(2), LABELS)
plt.title('Count of Fraud vs. Normal Transactions')
plt.ylabel('Count')
plt.xlabel('Class (0:Normal, 1:Fraud)')
```

```
Text(0.5, 0, 'Class (0:Normal, 1:Fraud)')
```

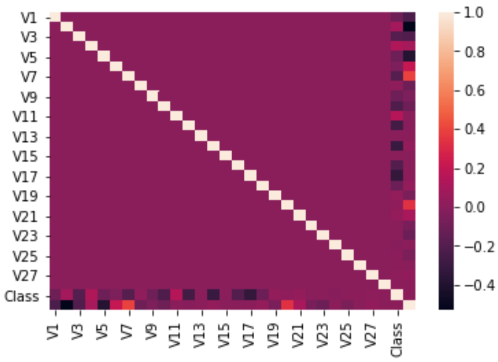


```
data.hist(linewidth=1, histtype='stepfilled', facecolor='g', figsize=(20, 20));
```



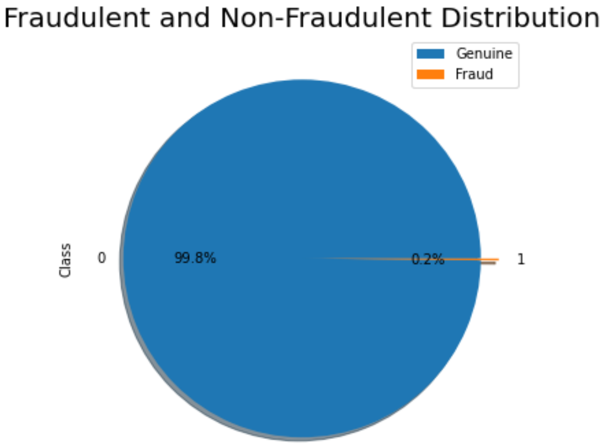
```python
import seaborn as sn
sn.heatmap(data.corr())
```

<AxesSubplot:>

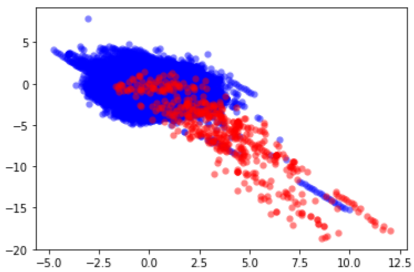


Scatter Plot

```python
import matplotlib.pyplot as plt
data["Class"].value_counts().plot(kind = 'pie',explode=[0, 0.1],figsize=(6, 6),autopct='%1.1f%%',shadow=1)
plt.title("Fraudulent and Non-Fraudulent Distribution",fontsize=20)
plt.legend(["Genuine","Fraud"])
plt.show()
```



In [18]:
```python
# Comparison between fraud and non-fraud cases
plt.scatter(data.loc[data['Class'] == 0]['V11'], data.loc[data['Class'] == 0]['V12'],label='Class #0', alpha=0.5, linewidth=0.15,
plt.scatter(data.loc[data['Class'] == 1]['V11'], data.loc[data['Class'] == 1]['V12'],label='Class #1', alpha=0.5, linewidth=0.15,
plt.show()
```

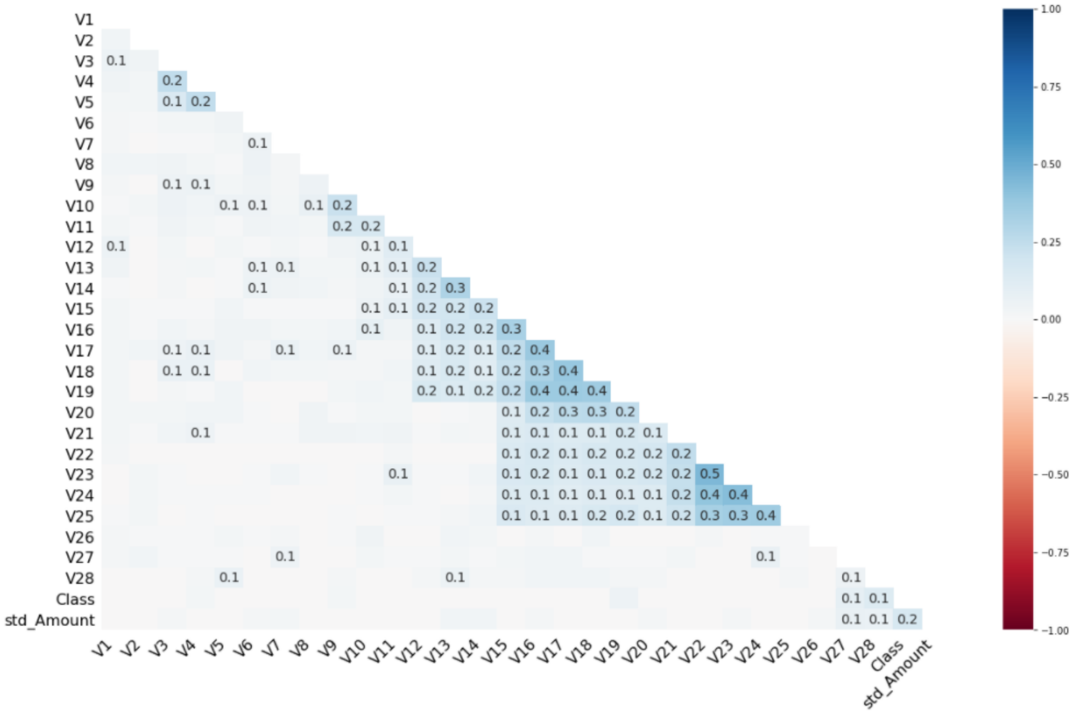


Visualization Of the Missing Values

In [19]:
```
import missingno as msno
%matplotlib inline
```

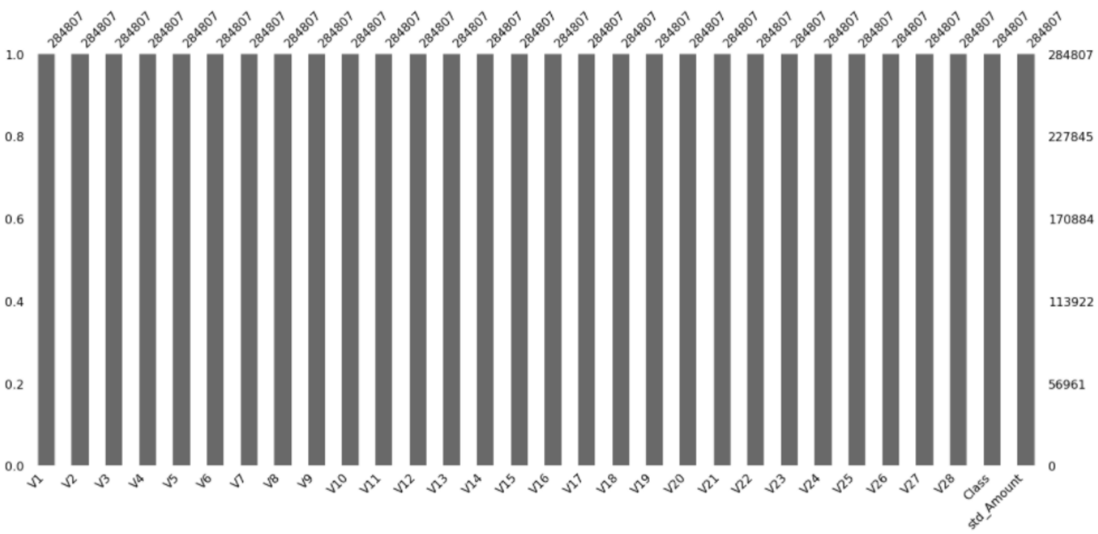
In [19]:
```
msno.heatmap(data)
```

Out[19]: <AxesSubplot:>



In [23]:
```
msno.bar(data)
```

Out[23]: <AxesSubplot:>



STATISTICS

In [30]:
```
df=df.drop_duplicates(keep='first')
df
```

Out[30]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284397 rows × 30 columns

In [31]:
```
#shape of dataset
df.shape
```

Out[31]: (284397, 30)

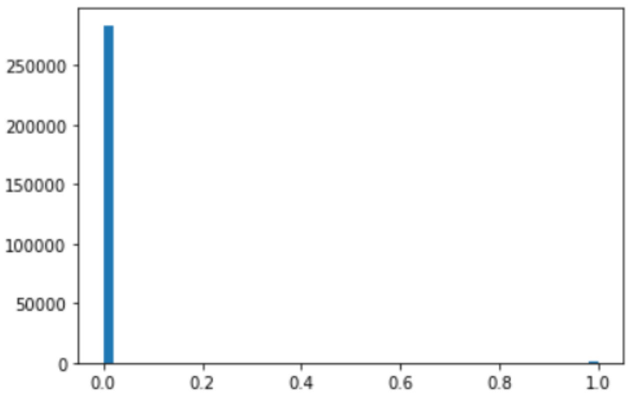
In [32]:
```
#summary statistics
df.describe()
```

Out[32]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V1
count	284340.000000	284335.000000	284318.000000	284321.000000	284320.000000	284330.000000	284343.000000	284340.000000	284328.000000	284315.0000
mean	0.001665	-0.001668	0.000419	-0.000735	-0.000114	-0.001117	0.000586	0.000162	-0.000902	-0.0009
std	1.955267	1.649997	1.513217	1.414647	1.378926	1.331406	1.232236	1.184861	1.096948	1.0829
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.434066	-24.5882
25%	-0.919305	-0.599351	-0.890351	-0.848805	-0.691257	-0.768580	-0.553638	-0.208639	-0.643711	-0.5354
50%	0.018822	0.064672	0.179956	-0.020474	-0.054499	-0.274738	0.040079	0.022244	-0.052190	-0.0931
75%	1.315706	0.802934	1.026846	0.741789	0.611324	0.397241	0.570071	0.326782	0.596467	0.4537
max	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.594995	23.7451

In [35]:
```
num_bins=50
plt.hist(df['Class'],num_bins)
```

Out[35]:
```
(array([283893.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
             0.,      0.,      0.,      0.,      0.,      0.,      0.,
           486.]),
 array([0.  , 0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 ,
        0.22, 0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42,
        0.44, 0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64,
        0.66, 0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86,
        0.88, 0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ]),
 <BarContainer object of 50 artists>)
```



In [33]:
```python
df_sort=df.sort_values(by='Amount',ascending=False).head()
df_sort.head()
```

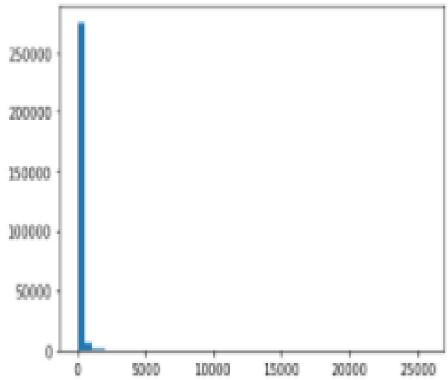
Out[33]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	
274771	-35.548539	-31.850484	-48.325589	15.304184	-113.743307	73.301626	120.589494	-27.347360	-3.872425	-12.005487	...	-21.620120	5.712303	-1.5
58465	-36.802320	-63.344698	-20.645794	16.715537	-20.672064	7.694002	24.956587	-4.730111	-2.687312	-8.423404	...	11.455313	-10.933144	-17.1
151296	-34.549296	-60.464618	-21.340854	16.875344	-19.229075	6.335259	24.422716	-4.964566	0.188912	-8.908182	...	11.502580	-9.499423	-16.5
46841	-23.712839	-42.172688	-13.320825	9.925019	-13.945538	5.564891	15.710844	-2.844253	-1.580725	-5.533256	...	7.921600	-6.320710	-11.3
54018	-21.780665	-38.305310	-12.122469	9.752791	-12.880794	4.256017	14.785051	-2.818253	-0.667338	-5.545590	...	7.437478	-5.619439	-10.5

5 rows × 30 columns

In [34]:
```python
#histogram
num_bins=50
plt.hist(df['Amount'],num_bins)
```

Out[34]:
```
(array([2.75553e+05, 6.00800e+03, 1.53900e+03, 6.27000e+02, 2.27000e+02,
        1.44000e+02, 9.20000e+01, 6.90000e+01, 3.30000e+01, 2.20000e+01,
        1.00000e+01, 1.10000e+01, 5.00000e+00, 5.00000e+00, 5.00000e+00,
        5.00000e+00, 1.00000e+00, 2.00000e+00, 0.00000e+00, 2.00000e+00,
        0.00000e+00, 0.00000e+00, 1.00000e+00, 1.00000e+00, 0.00000e+00,
        1.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 1.00000e+00, 0.00000e+00, 1.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
        0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00, 1.00000e+00]),
 array([-7.30367500e-02,  5.13751624e+02,  1.02757628e+03,  1.54140095e+03,
         2.05522561e+03,  2.56905027e+03,  3.08287493e+03,  3.59669959e+03,
         4.11052425e+03,  4.62434891e+03,  5.13817357e+03,  5.65199823e+03,
         6.16582289e+03,  6.67964755e+03,  7.19347221e+03,  7.70729687e+03,
         8.22112154e+03,  8.73494620e+03,  9.24877086e+03,  9.76259552e+03,
         1.02764202e+04,  1.07902448e+04,  1.13040695e+04,  1.18178942e+04,
         1.23317188e+04,  1.28455435e+04,  1.33593681e+04,  1.38731928e+04,
         1.43870175e+04,  1.49008421e+04,  1.54146668e+04,  1.59284914e+04,
         1.64423161e+04,  1.69561408e+04,  1.74699654e+04,  1.79837901e+04,
         1.84976147e+04,  1.90114394e+04,  1.95252641e+04,  2.00390887e+04,
         2.05529134e+04,  2.10667381e+04,  2.15805627e+04,  2.20943874e+04,
         2.26082120e+04,  2.31220367e+04,  2.36358614e+04,  2.41496860e+04,
         2.46635107e+04,  2.51773353e+04,  2.56911600e+04]),
 <BarContainer object of 50 artists>)
```



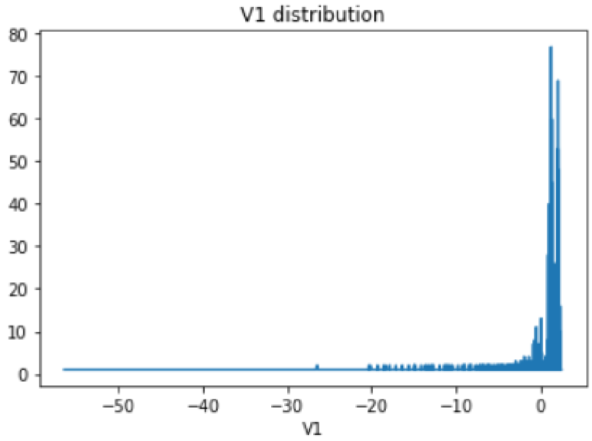
In [35]:
```python
num_bins=50
```

In [36]:
```
#count by category-cross tabulate
make_dist=df.groupby('V1').size()
make_dist
```

Out[36]:
```
V1
-56.407510    1
-46.855047    1
-41.928738    1
-40.470142    1
-40.042538    1
             ..
 2.430507     1
 2.439207     1
 2.446505     1
 2.451888     1
 2.454930     1
Length: 275597, dtype: int64
```

In [37]:
```
#distribution of categorial distribution
make_dist.plot(title='V1 distribution')
```

Out[37]:
```
<AxesSubplot:title={'center':'V1 distribution'}, xlabel='V1'>
```



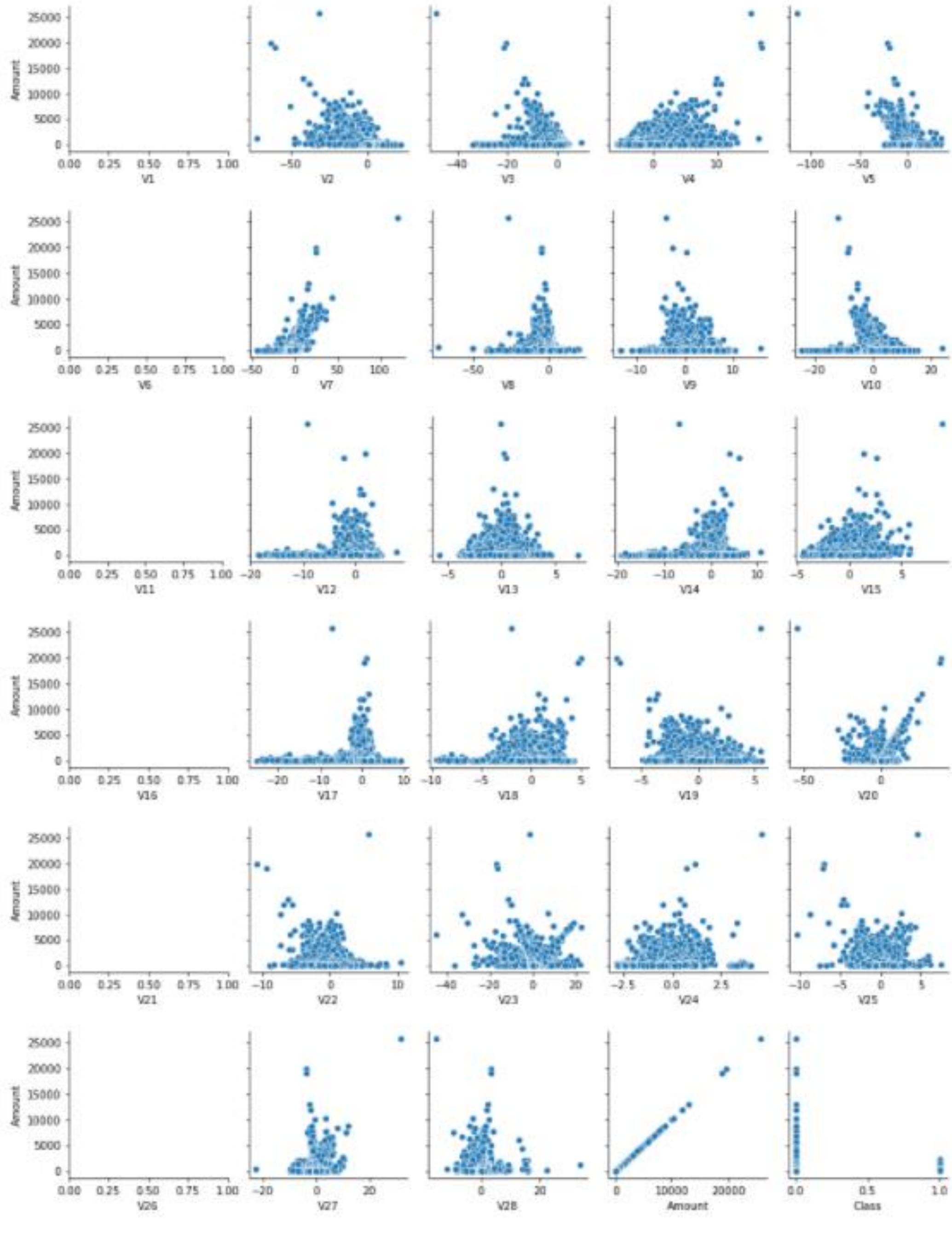
In [38]:
```
#select  all numerical variables
df_num=df.select_dtypes(include=['float64','int64'])
df_num.head()
```

Out[38]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.

5 rows × 30 columns

In [39]:
```
#correlation plots using 'pairplot'
for i in range(0,len(df_num.columns),5):
    sns.pairplot(df_num,y_vars=['Amount'],x_vars=df_num.columns[i:i+5])
```



MISSING DATA HANDLING

In [40]:
```
data = pd.read_csv("D:\\4th sem works\\Projects\\Credit card fraud detection\\creditcard.csv")
```

In [41]:
```
data = data.drop("Time", axis=1)
data
```

Out[41]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.0669
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.3398
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.6892
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.1755
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.1412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.5093
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.0162
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.6401
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.1232
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.0087

284807 rows × 30 columns

checking missing value

In [42]:
```
data.isnull()
```

Out[42]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284803	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284804	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284805	False	False	False	False	False	True	True	False	False	False	...	False	False	False	False	False	False	False	False	False	False
284806	False	False	False	False	False	True	True	False	False	False	...	False	False	False	False	False	False	False	False	True	False

284807 rows × 30 columns

To find the count of null values in the data frame

In [43]:
```python
def null_table(data):
    print(pd.isnull(data).sum())

null_table(data)
```

```
V1         57
V2         62
V3         79
V4         76
V5         77
V6         67
V7         54
V8         57
V9         69
V10        83
V11        71
V12        79
V13        91
V14       106
V15       105
V16       111
V17       108
V18       105
V19        92
V20        80
V21        75
V22        56
V23        70
V24        82
V25        80
V26        65
V27        57
V28        27
Amount     31
Class      18
dtype: int64
```

Deletion method

In [44]:
```python
data1=data.copy()
```

In [45]:
```python
data1
```

Out[45]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.0669
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.3398
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.6892
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.1755
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.1412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.5093
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.0162
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.6401
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.1232
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.0087

284807 rows × 30 columns

In [46]:
```
data1.head()
```

Out[46]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.

5 rows × 30 columns

In [47]:
```
len(data1)
```

Out[47]: 284807

list wise deletion

In [48]:
```
data1.isnull().sum()
```

Out[48]:
```
V1         57
V2         62
V3         79
V4         76
V5         77
V6         67
V7         54
V8         57
V9         69
V10        83
V11        71
V12        79
V13        91
V14       106
V15       105
V16       111
V17       108
V18       105
V19        92
V20        80
V21        75
V22        56
V23        70
V24        82
V25        80
V26        65
V27        57
V28        27
Amount     31
Class      18
dtype: int64
```

In [49]:
```python
data1.dropna(inplace=True)
```

In [50]:
```python
len(data1)
```

Out[50]:
```
283366
```

In [51]:
```python
data1
```

Out[51]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.0669
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.3398
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.6892
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.1755
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.1412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284800	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-2.606837	-4.918215	-0.118228	0.435402	0.267772	...	-0.268048	-0.717211	0.297930	-0.3597
284801	0.120316	0.931005	-0.546012	-0.745097	1.130314	1.058415	0.024330	0.115093	-0.204064	-0.657422	...	-0.314205	-0.808520	0.050343	0.1028
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.5093
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.0162
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.6401

283366 rows × 30 columns

In [52]:
```python
data1.isnull().sum()
```

Out[52]:
```
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
```

row wise Deletion

In [53]:
```python
data2=data.copy()
```

In [54]:
```python
data2
```

Out[54]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.0669
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.3398
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.6892
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.1755
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.1412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.5093
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.0162
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.6401
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.1232
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.0087

284807 rows × 30 columns

In [55]:
```python
data2.isnull().sum(axis=1).value_counts()
```

Out[55]:
```
0     283366
1       1102
2        195
3         62
5         26
4         25
11        12
8          6
7          5
9          4
6          3
13         1
dtype: int64
```

```
In [56]: data2.isnull().sum()
```

```
Out[56]: V1          57
V2          62
V3          79
V4          76
V5          77
V6          67
V7          54
V8          57
V9          69
V10         83
V11         71
V12         79
V13         91
V14        106
V15        105
V16        111
V17        108
V18        105
V19         92
V20         80
V21         75
V22         56
V23         70
V24         82
V25         80
V26         65
V27         57
V28         27
Amount      31
Class       18
dtype: int64
```

Imputation Methods

1.FILL NULL VALUE WITH SCALAR VALUE

```
In [57]: datascalar=data.fillna(0)
```

```
In [58]: datascalar
```

Out[58]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.0669
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.3398
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.6892
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.1755
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.1412
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.5093
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.0162
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.6401
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.000000	0.000000	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.1232
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	0.000000	0.000000	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.0087

284807 rows × 30 columns

In [59]:
```
#count of null value in data set after fill null value using scalar method
```

In [60]:
```
def null_table(datascalar):
    print(pd.isnull(datascalar).sum())

null_table(datascalar)
```

```
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
```

2.random sample from existing value

In [42]:
```
datarandom=data.replace(to_replace = np.nan,value=0.232045)
```

In [43]:
```
datarandom
```

Out[43]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.232045	0.232045	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	0.232045	0.232045	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008

284807 rows × 30 columns

In [44]:
```
# count of null values after random method
```

In [45]:
```
def null_table(datarandom):
    print(pd.isnull(datarandom).sum())

null_table(datarandom)
```

```
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
```

3.1 using mean method

In [46]:
```
data0=data.copy()
```

In [47]:
```
data0
```

Out[47]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008

284807 rows × 30 columns

In [48]:
```
data0.isnull().sum()
```
Out[48]:
```
V1         57
V2         62
V3         79
V4         76
V5         77
V6         67
V7         54
V8         57
V9         69
V10        83
V11        71
V12        79
V13        91
V14       106
V15       105
V16       111
V17       108
V18       105
V19        92
V20        80
V21        75
V22        56
V23        70
V24        82
V25        80
V26        65
V27        57
V28        27
Amount     31
Class      18
dtype: int64
```
In [49]:
```
np.mean(data0.V1)
```
Out[49]:
```
1.0031341106060737e-06
```
In [50]:
```
data0['V1'].fillna(np.mean(data0.V1),inplace=True)
```
In [51]:
```
data0.V1.isnull().sum()
```
Out[51]:
```
0
```
In [52]:
```
data0['V1'].fillna(np.mean(data0.V1),inplace=True)
data0['V2'].fillna(np.mean(data0.V2),inplace=True)
data0['V3'].fillna(np.mean(data0.V3),inplace=True)
data0['V4'].fillna(np.mean(data0.V4),inplace=True)
data0['V5'].fillna(np.mean(data0.V5),inplace=True)
data0['V6'].fillna(np.mean(data0.V6),inplace=True)
data0['V7'].fillna(np.mean(data0.V7),inplace=True)
data0['V8'].fillna(np.mean(data0.V8),inplace=True)
data0['V9'].fillna(np.mean(data0.V9),inplace=True)
data0['V10'].fillna(np.mean(data0.V10),inplace=True)
data0['V11'].fillna(np.mean(data0.V11),inplace=True)
data0['V12'].fillna(np.mean(data0.V12),inplace=True)
data0['V13'].fillna(np.mean(data0.V13),inplace=True)
data0['V14'].fillna(np.mean(data0.V14),inplace=True)
data0['V15'].fillna(np.mean(data0.V15),inplace=True)
data0['V16'].fillna(np.mean(data0.V16),inplace=True)
data0['V17'].fillna(np.mean(data0.V17),inplace=True)
data0['V18'].fillna(np.mean(data0.V18),inplace=True)
data0['V19'].fillna(np.mean(data0.V19),inplace=True)
data0['V20'].fillna(np.mean(data0.V20),inplace=True)
data0['V21'].fillna(np.mean(data0.V21),inplace=True)
data0['V22'].fillna(np.mean(data0.V22),inplace=True)
data0['V23'].fillna(np.mean(data0.V23),inplace=True)
data0['V24'].fillna(np.mean(data0.V24),inplace=True)
data0['V25'].fillna(np.mean(data0.V25),inplace=True)
data0['V26'].fillna(np.mean(data0.V26),inplace=True)
data0['V27'].fillna(np.mean(data0.V27),inplace=True)
data0['V28'].fillna(np.mean(data0.V28),inplace=True)
data0['Amount'].fillna(np.mean(data0.Amount),inplace=True)
data0['Class'].fillna(np.mean(data0.Class),inplace=True)
```

In [53]:
```
# data set after using mean method
```

In [54]:
```
data0
```

Out[54]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.000016	-0.000007	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.000016	-0.000007	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008

284807 rows × 30 columns

In [55]:
```
#count of null values
data0.isnull().sum()
```

Out[55]:
```
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
```

3.2 median method

In [56]:
```
data1=data.copy()
```

In [57]:
```
data1
```

Out[57]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

In [58]:
```
data1.isnull().sum()
```

Out[58]:
```
V1          57
V2          62
V3          79
V4          76
V5          77
V6          67
V7          54
V8          57
V9          69
V10         83
V11         71
V12         79
V13         91
V14        106
V15        105
V16        111
V17        108
V18        105
V19         92
```

In [58]:
```
data1.isnull().sum()
```

Out[58]:
```
V1          57
V2          62
V3          79
V4          76
V5          77
V6          67
V7          54
V8          57
V9          69
V10         83
V11         71
V12         79
V13         91
V14        106
V15        105
V16        111
V17        108
V18        105
V19         92
V20         80
V21         75
V22         56
V23         70
V24         82
V25         80
V26         65
V27         57
V28         27
Amount      31
Class       18
dtype: int64
```

In [59]:
```
data1['V1'].fillna(data1['V1'].median(),inplace=True)
```

In [60]:
```
data1['V1'].median()
```

Out[60]:
```
0.018234050499999998
```

In [61]:
```
data1.V1.isnull().sum()
```

Out[61]:
```
0
```

```
data1['V2'].fillna(data1['V2'].median(),inplace=True)
data1['V3'].fillna(data1['V3'].median(),inplace=True)
data1['V4'].fillna(data1['V4'].median(),inplace=True)
data1['V5'].fillna(data1['V5'].median(),inplace=True)
data1['V6'].fillna(data1['V6'].median(),inplace=True)
data1['V7'].fillna(data1['V7'].median(),inplace=True)
data1['V8'].fillna(data1['V8'].median(),inplace=True)
data1['V9'].fillna(data1['V9'].median(),inplace=True)
data1['V10'].fillna(data1['V10'].median(),inplace=True)
data1['V11'].fillna(data1['V11'].median(),inplace=True)
data1['V12'].fillna(data1['V12'].median(),inplace=True)
data1['V13'].fillna(data1['V13'].median(),inplace=True)
data1['V14'].fillna(data1['V14'].median(),inplace=True)
data1['V15'].fillna(data1['V15'].median(),inplace=True)
data1['V16'].fillna(data1['V16'].median(),inplace=True)
data1['V17'].fillna(data1['V17'].median(),inplace=True)
data1['V18'].fillna(data1['V18'].median(),inplace=True)
data1['V19'].fillna(data1['V19'].median(),inplace=True)
data1['V20'].fillna(data1['V20'].median(),inplace=True)
data1['V21'].fillna(data1['V21'].median(),inplace=True)
data1['V22'].fillna(data1['V22'].median(),inplace=True)
data1['V23'].fillna(data1['V23'].median(),inplace=True)
data1['V24'].fillna(data1['V24'].median(),inplace=True)
data1['V25'].fillna(data1['V25'].median(),inplace=True)
data1['V26'].fillna(data1['V26'].median(),inplace=True)
data1['V27'].fillna(data1['V27'].median(),inplace=True)
data1['V28'].fillna(data1['V28'].median(),inplace=True)
data1['Amount'].fillna(data1['Amount'].median(),inplace=True)
data1['Class'].fillna(data1['Class'].median(),inplace=True)
```

In [63]:

```
data1
```

Out[63]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.274180	0.040079	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.274180	0.040079	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

In [64]:

```
#count of null values
data1.isnull().sum()
```

Out[64]:

```
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
```

3.3 mode

In [65]:
```
data2=data.copy()
```

In [66]:
```
data2
```

Out[66]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	NaN	NaN	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	NaN	NaN	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

In [67]:
```
data2.isnull().sum()
```

Out[67]:
```
V1       57
V2       62
V3       79
V4       76
V5       77
V6       67
V7       54
V8       57
V9       69
V10      83
V11      71
V12      79
V13      91
V14     106
V15     105
V16     111
V17     108
```

In [68]:

```
data2['V1'].value_counts()
data2['V1'].fillna(1.245674,inplace=True)
data2['V2'].value_counts()
data2['V2'].fillna(-0.326668,inplace=True)
data2['V4'].value_counts()
data2['V4'].fillna(-0.842316,inplace=True)
data2['V5'].value_counts()
data2['V5'].fillna( 2.463072,inplace=True)
data2['V6'].value_counts()
data2['V6'].fillna(-1.011073,inplace=True)
data2['V7'].value_counts()
data2['V7'].fillna( 0.014953,inplace=True)
data2['V8'].value_counts()
data2['V8'].fillna(-0.160211,inplace=True)
data2['V9'].value_counts()
data2['V9'].fillna( 0.608606,inplace=True)
data2['V10'].value_counts()
data2['V10'].fillna(-0.044575,inplace=True)
data2['V11'].value_counts()
data2['V11'].fillna(-0.356749 ,inplace=True)
data2['V12'].value_counts()
data2['V12'].fillna(0.350564,inplace=True)
data2['V13'].value_counts()
data2['V13'].fillna(-0.141238 ,inplace=True)
data2['V14'].value_counts()
data2['V14'].fillna(0.690972,inplace=True)
data2['V15'].value_counts()
data2['V15'].fillna( 1.124147 ,inplace=True)
data2['V16'].value_counts()
data2['V16'].fillna( 0.342470,inplace=True)
data2['V17'].value_counts()
data2['V17'].fillna(-0.374656 ,inplace=True)
data2['V18'].value_counts()
data2['V18'].fillna(-0.052640 ,inplace=True)
data2['V19'].value_counts()
data2['V19'].fillna(-0.330590,inplace=True)
data2['V20'].value_counts()
data2['V20'].fillna(-0.180370,inplace=True)
data2['V21'].value_counts()
data2['V21'].fillna(-0.262581 ,inplace=True)
data2['V22'].value_counts()
data2['V22'].fillna(-0.816264,inplace=True)
data2['V23'].value_counts()
data2['V23'].fillna(0.020675 ,inplace=True)
data2['V24'].value_counts()
data2['V24'].fillna(0.357827 ,inplace=True)
data2['V25'].value_counts()
data2['V25'].fillna(0.186423 ,inplace=True)
data2['V26'].value_counts()
data2['V26'].fillna( 0.096544 ,inplace=True)
data2['V27'].value_counts()
data2['V27'].fillna(-0.035866,inplace=True)
data2['V28'].value_counts()
data2['V28'].fillna(-0.060282,inplace=True)
data2['Amount'].value_counts()
data2['Amount'].fillna(1,inplace=True)
data2['Class'].value_counts()
data2['Class'].fillna(0.0 ,inplace=True)
```

In [69]:
```
data2
```

Out[69]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-1.011073	0.014953	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-1.011073	0.014953	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

In [70]:
```
#count of null values after using mode method
data2.isnull().sum()
```

Out[70]:
```
V1         0
V2         0
V3        79
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
```

LAST OBSERVATION CARRIED FORWARD

In [71]:
```
data3 = data.copy()
```

In [72]:
```
#Filling null values with the previous ones
data3.fillna(method = 'pad')
```

Out[72]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.649617	1.577006	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

REGRESSION IMPUTATION

In [73]:
```python
from sklearn.experimental import enable_iterative_imputer
```

In [74]:
```python
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
import pandas as pd
```

In [75]:
```python
itr=IterativeImputer(estimator = LinearRegression())
```

In [76]:
```python
data = pd.read_csv("D:\\4th sem works\\Projects\\Credit card fraud detection\\creditcard.csv")
```

In [77]:
```python
data = data.drop("Time", axis=1)
```

In [78]:
```python
data[['V1','V2','V3','V4','V5','V6','V7','V8','V9','V10','V11','V12','V13','V14','V15','V16','V17','V18','V19','V20','V21','V22'_
```

In [79]:
```python
data[['V1','V2','V3','V4','V5','V6','V7','V8','V9','V10','V11','V12','V13','V14','V15','V16','V17','V18','V19','V20','V21','V22'_
```

Out[79]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.06
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.68
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.17
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	3.031260	-0.296827	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.623708	-0.686180	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.649617	1.577006	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.154135	-0.260011	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.001685	0.049597	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.00

284807 rows × 30 columns

In [80]:
```python
data.isnull().sum()
```

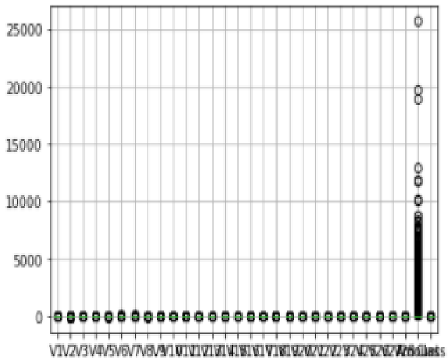
Out[80]:
```
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

OUTLIER ANALYSIS

In [25]:
```
df = dataset
```

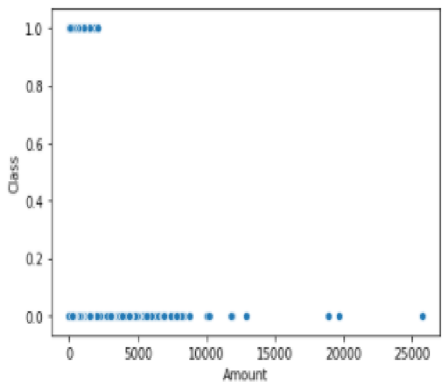
In [26]:
```
#Identifying outliers and removing them
df.boxplot()
```

Out[26]:
```
<AxesSubplot:>
```



In [11]:
```
sns.scatterplot(data=df,x='Amount',y='Class')
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef8cd04748>
```



In [12]:
```
Q1=df['Amount'].quantile(0.25)
Q3=df['Amount'].quantile(0.75)
IQR=Q3-Q1
print(Q1,Q3,IQR)
```

```
6.3 79.91 73.61
```

In [13]:
```
df.describe()
```

Out[13]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V·
count	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.000000	275663.0000(
mean	-0.037460	-0.002430	0.025520	-0.004359	-0.010660	-0.014206	0.008586	-0.005698	-0.012363	0.0031'
std	1.952522	1.667260	1.507538	1.424323	1.378117	1.313213	1.240348	1.191596	1.100108	1.0870:
min	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.434066	-24.5882(
25%	-0.941105	-0.614040	-0.843168	-0.862847	-0.700192	-0.765861	-0.552047	-0.209618	-0.659904	-0.5389(
50%	-0.059659	0.070249	0.200736	-0.035098	-0.060556	-0.270931	0.044848	0.022980	-0.064724	-0.0917!
75%	1.294471	0.819067	1.048461	0.753943	0.604521	0.387704	0.583885	0.322319	0.593098	0.4707(
max	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.594995	23.7451:

8 rows × 30 columns

In [14]:
```
lower_outlier=df.Amount<(Q1-1.5*IQR)
upper_outlier=df.Amount<(Q1+1.5*IQR)
```

In [15]:
```
df[lower_outlier|upper_outlier]
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
5	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	-0.371407	...	-0.208254	-0.559825	-0.026398	-0.37
6	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	-0.099254	...	-0.167716	-0.270710	-0.154104	-0.78
7	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	1.249376	...	1.943465	-1.015455	0.057504	-0.64
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284801	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.115093	-0.204064	-0.657422	...	-0.314205	-0.808520	0.050343	0.10
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12

225551 rows × 30 columns

In [16]:
```
df.Amount.count()
```

Out[16]: 275663

In [17]:
```
df.Amount.mean()
```

Out[17]: 90.5783797244607

In [18]:
```
df.Amount.mode()
```

Out[18]:
```
0    1.0
dtype: float64
```

In [19]:
```
df.Amount.median()
```

Out[19]:
```
23.74
```

In [20]:
```
#dataset(df) without outliers=DF
DF=df[lower_outlier|upper_outlier]
```

In [21]:
```
DF
```

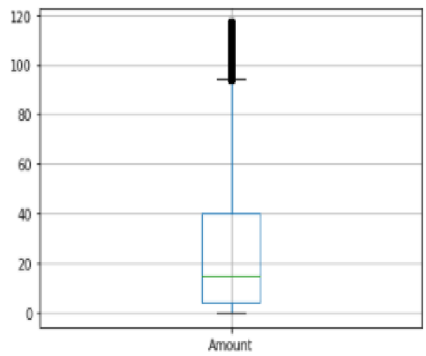
Out[21]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.33
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.14
5	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	-0.371407	...	-0.208254	-0.559825	-0.026398	-0.37
6	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	-0.099254	...	-0.167716	-0.270710	-0.154104	-0.78
7	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	1.249376	...	1.943465	-1.015455	0.057504	-0.64
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284801	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.115093	-0.204064	-0.657422	...	-0.314205	-0.808520	0.050343	0.10
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.50
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.01
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.64
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.12

225551 rows × 30 columns

In [22]:
```
DF.boxplot(column='Amount')
```

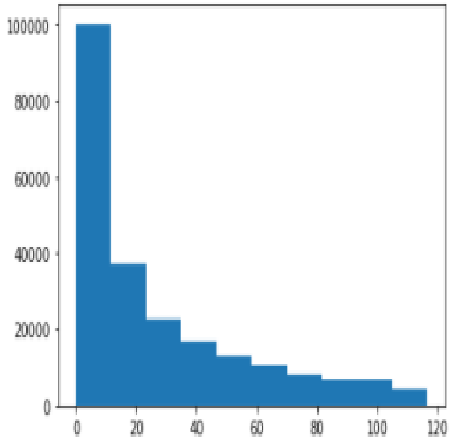
Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef9134f908>
```



In [23]:
```
DF.Amount.mean()
```

Out[23]:
```
26.85431569802672
```

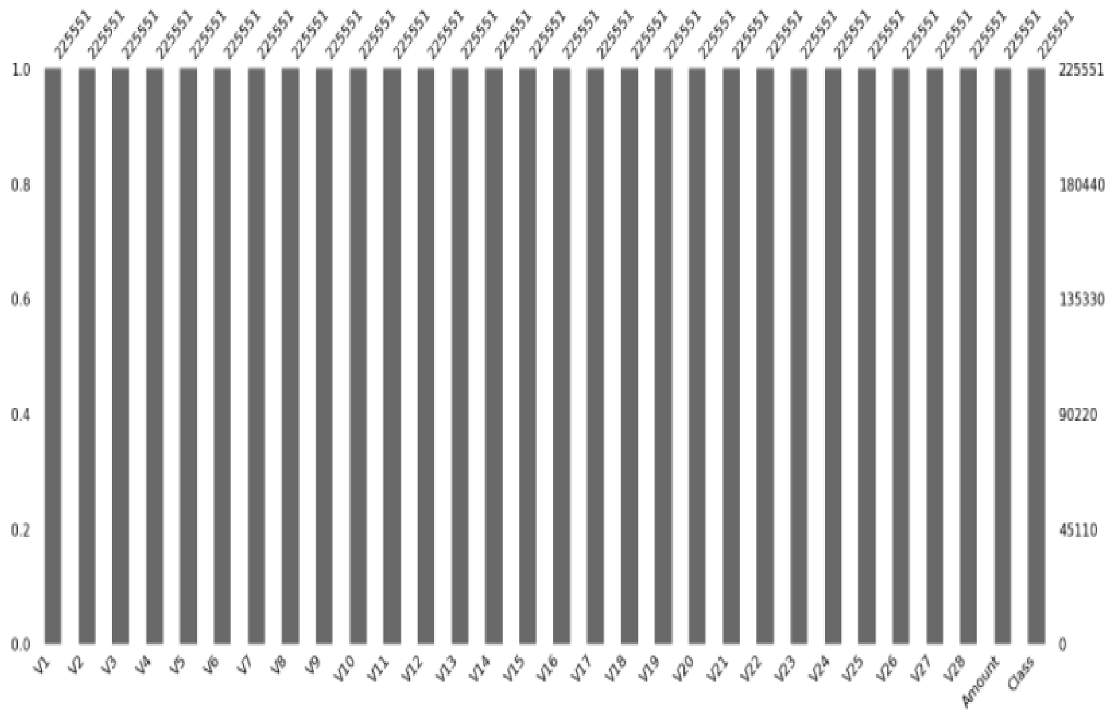
```
In [24]: #Understansing relationships and insights through data
plt.hist(DF.Amount)
plt.show()
```




```
In [25]: #identifying missing values
import missingno as msno
%matplotlib inline
```

```
In [27]: msno.bar(DF)
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1ef8d3109c8>
```



Hence no missing data

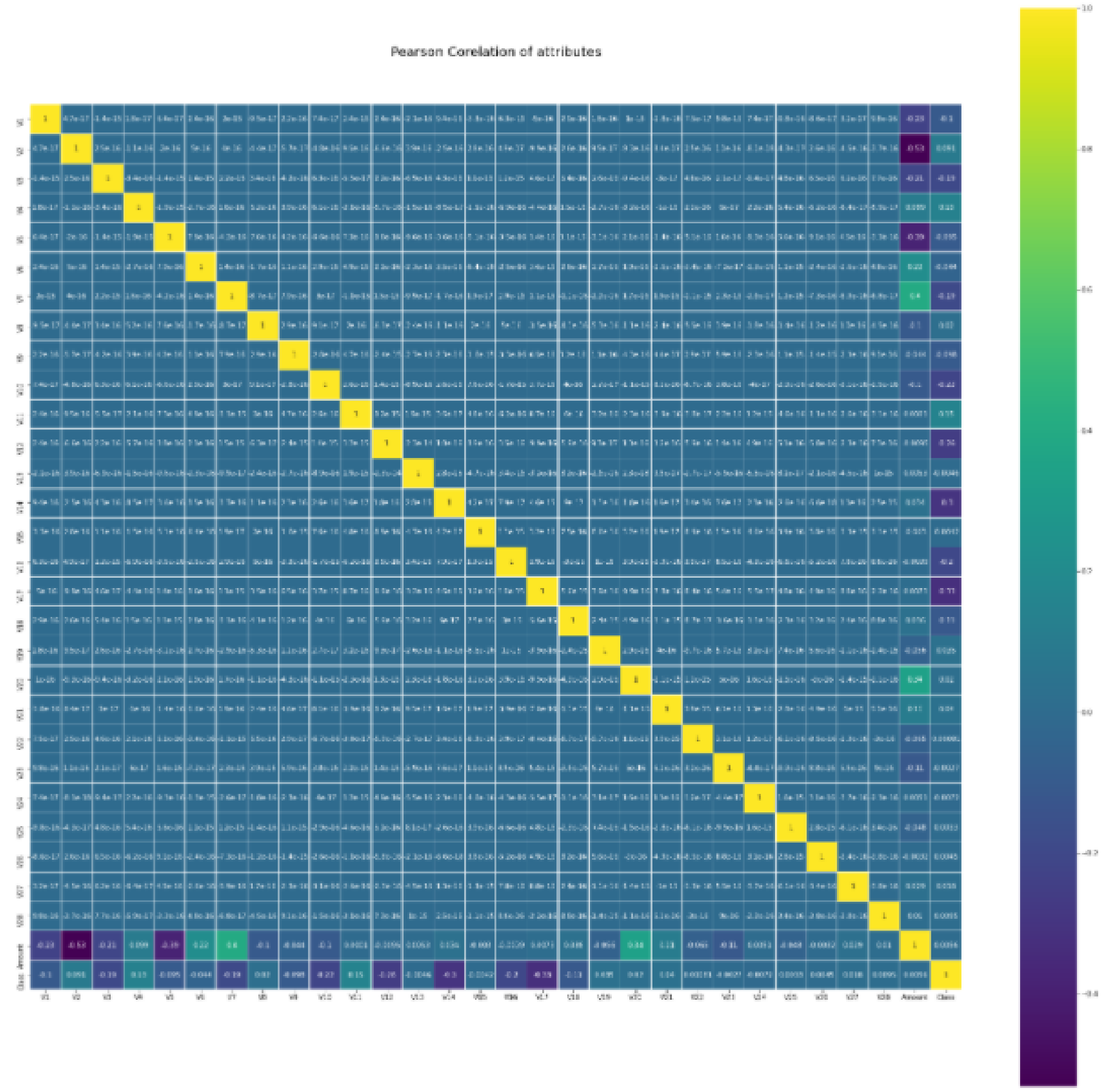
SVM CLASSIFICATION

Heat map shows that none of the variables are not auto-correlated with each other(as none of them as shown are having deep green or deep blue)

Deep green indicates and above indicates positive auto-correlation and deep blue indicates negative auto-correlation. Although deep green and beyond is shown in diagonal line. It is only the auto-correlation between same variables.hence it can be ignored

In [46]:
```
#creating heat map to identify auto correlation between the variables
colormap = plt.cm.viridis #color range to be used in heatmap
plt.figure(figsize=(30,30))
plt.title('Pearson Corelation of attributes',y=1.05,size = 19)
sns.heatmap(dataset.corr(),linewidth = 0.1,vmax = 1.0,
            square = True,cmap = colormap,linecolor = 'white',annot = True)
```

Out[46]: `<AxesSubplot:title={'center':'Pearson Corelation of attributes'}>`



Classes though imbalanced(non-fraud"0" vs fraud"1"). This is only natural representation of the actual system. Hence we may go with the same dataset or prefer to do an upampling

In [48]:
```
dataset.groupby('Class').count()
```

Out[48]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	V22	V23	V24	V25	V26	V
Class																			
0	284315	284315	284315	284315	284315	284315	284315	284315	284315	284315	...	284315	284315	284315	284315	284315	284315	284315	2843
1	492	492	492	492	492	492	492	492	492	492	...	492	492	492	492	492	492	492	4

2 rows × 29 columns

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [49]:
```
from sklearn import metrics
```

In [50]:
```
SVM_df_class = dataset['Class']
```

In [51]:
```
SVM_df_class
```

Out[51]:
```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

In [52]:
```
y = SVM_df_class
```

In [53]:
```
#Extracting first 30 columns which are independent variables and adding them into another dataframe
dataset_x = dataset.iloc[:,0:30]
```

In [54]:
```
dataset_x
```

Out[54]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	...	0.213454	0.111864	1.014480	-0.509
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	...	0.214205	0.924384	0.012463	-1.016
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	...	0.232045	0.578229	-0.037501	0.640
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	...	0.265245	0.800049	-0.163298	0.123
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	...	0.261057	0.643078	0.376777	0.008

284807 rows × 30 columns

In [55]:
```
#Converting dataframe into array
x = np.array(dataset_x)
```

In [56]:
```
x
```

Out[56]:
```
array([[-1.35980713e+00, -7.27811733e-02,  2.53634674e+00, ...,
        -2.10530535e-02,  1.49620000e+02,  0.00000000e+00],
       [ 1.19185711e+00,  2.66150712e-01,  1.66480113e-01, ...,
         1.47241692e-02,  2.69000000e+00,  0.00000000e+00],
       [-1.35835406e+00, -1.34016307e+00,  1.77320934e+00, ...,
        -5.97518406e-02,  3.78660000e+02,  0.00000000e+00],
       ...,
       [ 1.91956501e+00, -3.01253846e-01, -3.24963981e+00, ...,
        -2.65608286e-02,  6.78800000e+01,  0.00000000e+00],
       [-2.40440050e-01,  5.30482513e-01,  7.02510230e-01, ...,
         1.04532821e-01,  1.00000000e+01,  0.00000000e+00],
       [-5.33412522e-01, -1.89733337e-01,  7.03337367e-01, ...,
         1.36489143e-02,  2.17000000e+02,  0.00000000e+00]])
```

In [57]:
```
from sklearn.model_selection import train_test_split
```

In [58]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1)
```

In [59]:
```
y_test
```

Out[59]:
```
169876    0
127467    0
137900    0
21513     0
134700    0
         ..
128956    0
177494    0
26287     0
198160    0
25893     0
Name: Class, Length: 85443, dtype: int64
```

In [60]:
```
#Since we have to transform an array into zscore we are using the below statement
from scipy import stats
```

In [61]:
```
y_train
```

Out[61]:
```
191125    0
153710    0
261216    0
190724    0
127492    0
         ..
21440     0
117583    0
73349     0
267336    0
128037    0
Name: Class, Length: 199364, dtype: int64
```

In [62]:
```
#Best Practice to transform train and test dataset separately
#x_train = stats.zscore(x_train,axis=1,ddof=1)#transforming trainingset to z transformation,to ensure all the data are in same sc
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)
```

In [63]:
```
#Best Practice to transform train and test dataset separately
#x_train = stats.zscore(x_train,axis=1,ddof=1)#transforming trainingset to z transformation,to ensure all the data are in same sc
x_train_scaled
```

Out[63]:
```
array([[9.55064631e-01, 7.69403723e-01, 8.20150337e-01, ...,
        3.13574354e-01, 5.59336363e-04, 0.00000000e+00],
       [9.58288455e-01, 7.78249137e-01, 8.31888435e-01, ...,
        3.13891233e-01, 3.50315050e-04, 0.00000000e+00],
       [9.93385813e-01, 7.68013187e-01, 8.04627258e-01, ...,
        3.12186758e-01, 7.70693110e-05, 0.00000000e+00],
       ...,
       [9.38757325e-01, 7.85158545e-01, 8.65782476e-01, ...,
        3.13756862e-01, 2.95043120e-04, 0.00000000e+00],
       [9.88101473e-01, 7.59878682e-01, 8.36090900e-01, ...,
        3.12725673e-01, 3.68998519e-03, 0.00000000e+00],
       [9.46271935e-01, 7.72463435e-01, 8.83301232e-01, ...,
```

IMPORTING SUPPORT VECTOR CLASSIFIER

In [65]:
```
from sklearn.svm import SVC
```

In [66]:
```
svc = SVC()
```

Fitting the model using classifier

In [67]:
```
svc.fit(x_train_scaled,y_train)
```

Out[67]:
```
SVC()
```

Determining accuracy of the training and test set

In [68]:
```
print("Accuracy on training set:{:.3f}".format(svc.score(x_train_scaled,y_train)))
print("Accuracy on test set:{:.3f}".format(svc.score(x_test_scaled,y_test)))
```

```
Accuracy on training set:1.000
Accuracy on test set:1.000
```

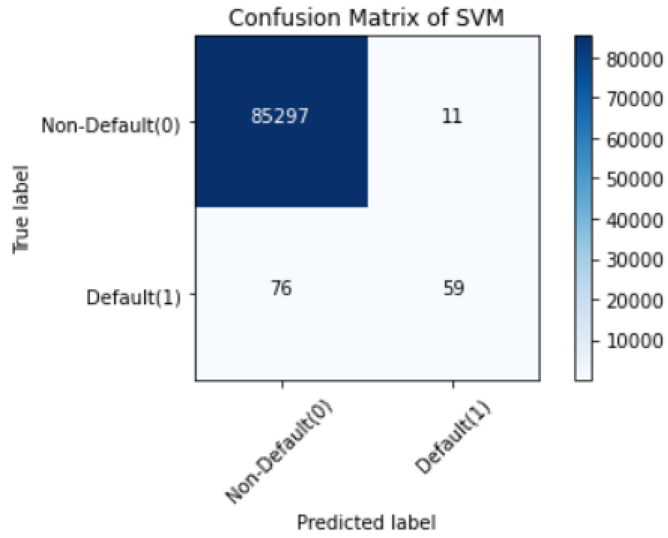
In [69]:
```
m = svc.predict(x_test_scaled)
print(m)
```

```
[0 0 0 ... 0 0 0]
```

In [70]:
```
print(metrics.confusion_matrix(y_test,m))
```

```
[[85308     0]
 [    0   135]]
```

```python
# Compute confusion matrix for the SVM
svm_matrix = confusion_matrix(y_test, m, labels = [0, 1])

#SVM
svm_cm_plot = plot_confusion_matrix(svm_matrix,
                                    classes = ['Non-Default(0)','Default(1)'],
                                    normalize = False, title = 'SVM')
plt.savefig('svm_cm_plot.png')
plt.show()
```



KNN CLASIFICATION

In [7]:
```
df = dataset
```

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [9]:
```
df.head()
```

Out[9]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.

5 rows × 30 columns

In [10]:
```
#Preprocessing
X = df.iloc[:, 0:29].values
y = df.iloc[:, 29].values
```

In [11]:
```
X
```

Out[11]:
```
array([[-1.35980713e+00, -7.27811733e-02,  2.53634674e+00, ...,
         1.33558377e-01, -2.10530535e-02,  1.49620000e+02],
       [ 1.19185711e+00,  2.66150712e-01,  1.66480113e-01, ...,
        -8.98309914e-03,  1.47241692e-02,  2.69000000e+00],
       [-1.35835406e+00, -1.34016307e+00,  1.77320934e+00, ...,
        -5.53527940e-02, -5.97518406e-02,  3.78660000e+02],
       ...,
       [ 1.91956501e+00, -3.01253846e-01, -3.24963981e+00, ...,
         4.45477214e-03, -2.65608286e-02,  6.78800000e+01],
       [-2.40440050e-01,  5.30482513e-01,  7.02510230e-01, ...,
         1.08820735e-01,  1.04532821e-01,  1.00000000e+01],
```

In [12]:
```
y
```

Out[12]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [14]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [15]:
```
#Training and Predictions
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Out[15]:
```
KNeighborsClassifier()
```

In [37]:
```
y_pred = classifier.predict(X_test)
```

In [38]:
```
#Evaluating the Algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[55030     4]
 [   29    70]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55034
           1       0.95      0.71      0.81        99

    accuracy                           1.00     55133
   macro avg       0.97      0.85      0.90     55133
weighted avg       1.00      1.00      1.00     55133
```

In [39]:
```
y_pred
```

Out[39]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [40]:
```
#Evaluating the Algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[55030     4]
 [   29    70]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55034
           1       0.95      0.71      0.81        99

    accuracy                           1.00     55133
   macro avg       0.97      0.85      0.90     55133
weighted avg       1.00      1.00      1.00     55133
```

DECISION TREE CLASSIFICATION

In [21]:
```
df = dataset
```

In [22]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [23]:
```
df.head()
```

Out[23]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.277838	-0.110474	0.066928	0.128539	-0
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.638672	0.101288	-0.339846	0.167170	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.771679	0.909412	-0.689281	-0.327642	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	0.005274	-0.190321	-1.175575	0.647376	-0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.798278	-0.137458	0.141267	-0.206010	0.

5 rows × 31 columns

In [24]:
```
#Preprocessing
X = df.iloc[:, 0:29].values
y = df.iloc[:, 29].values
```

In [25]:
```
X
```

Out[25]:
```
array([[-1.35980713e+00, -7.27811733e-02,  2.53634674e+00, ...,
         1.33558377e-01, -2.10530535e-02,  1.49620000e+02],
       [ 1.19185711e+00,  2.66150712e-01,  1.66480113e-01, ...,
        -8.98309914e-03,  1.47241692e-02,  2.69000000e+00],
       [-1.35835406e+00, -1.34016307e+00,  1.77320934e+00, ...,
        -5.53527940e-02, -5.97518406e-02,  3.78660000e+02],
       ...,
       [ 1.91956501e+00, -3.01253846e-01, -3.24963981e+00, ...,
         4.45477214e-03, -2.65608286e-02,  6.78800000e+01],
       [-2.40440050e-01,  5.30482513e-01,  7.02510230e-01, ...,
         1.08820735e-01,  1.04532821e-01,  1.00000000e+01],
       [-5.33412522e-01, -1.89733337e-01,  7.03337367e-01, ...,
        -2.41530880e-03,  1.36489143e-02,  2.17000000e+02]])
```

In [26]:
```
y
```

Out[26]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

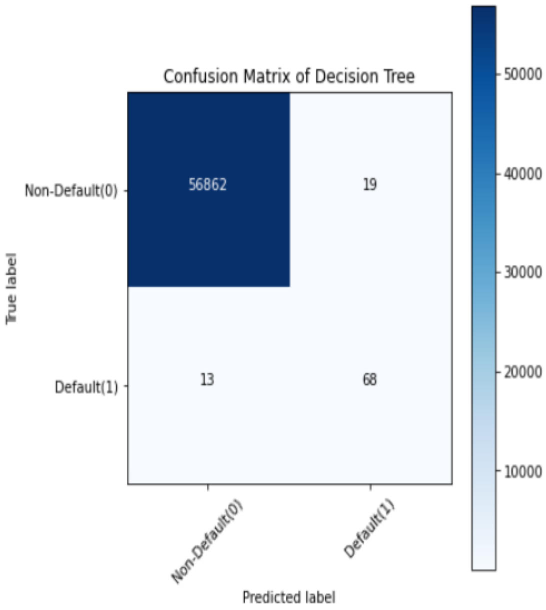
In [35]:
```
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)
```

In [36]:
```
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, tree_yhat)))
```

```
Accuracy score of the Decision Tree model is 0.9994382219725431
```

In [40]:
```
# 3. Confusion Matrix

# defining the plot function
import itertools
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix for the models

tree_matrix = confusion_matrix(y_test, tree_yhat, labels = [0, 1]) # Decision Tree
```

In [41]:
```python
# Plot the confusion matrix
plt.rcParams['figure.figsize'] = (6, 6)
tree_cm_plot = plot_confusion_matrix(tree_matrix,classes = ['Non-Default(0)','Default(1)'], normalize = False,
                                     title = 'Decision Tree')
plt.savefig('tree_cm_plot.png')
plt.show()
```



RANDOM FOREST CLASSIFICATION

In [42]:
```python
df = dataset
```

In [43]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [44]:
```python
df.head()
```

Out[44]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.277838	-0.110474	0.066928	0.128539	-0
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.638672	0.101288	-0.339846	0.167170	0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.771679	0.909412	-0.689281	-0.327642	-0.
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	0.005274	-0.190321	-1.175575	0.647376	-0.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.798278	-0.137458	0.141267	-0.206010	0.

5 rows × 31 columns

In [45]:
```python
#Preprocessing
X = df.iloc[:, 0:29].values
y = df.iloc[:, 29].values
```

In [46]:
```python
X
```

Out[46]:
```
array([[-1.35980713e+00, -7.27811733e-02,  2.53634674e+00, ...,
         1.33558377e-01, -2.10530535e-02,  1.49620000e+02],
       [ 1.19185711e+00,  2.66150712e-01,  1.66480113e-01, ...,
        -8.98309914e-03,  1.47241692e-02,  2.69000000e+00],
       [-1.35835406e+00, -1.34016307e+00,  1.77320934e+00, ...,
        -5.53527940e-02, -5.97518406e-02,  3.78660000e+02],
       ...,
       [ 1.91956501e+00, -3.01253846e-01, -3.24963981e+00, ...,
         4.45477214e-03, -2.65608286e-02,  6.78800000e+01],
       [-2.40440050e-01,  5.30482513e-01,  7.02510230e-01, ...,
         1.08820735e-01,  1.04532821e-01,  1.00000000e+01],
```

In [47]:
```
y
```

Out[47]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

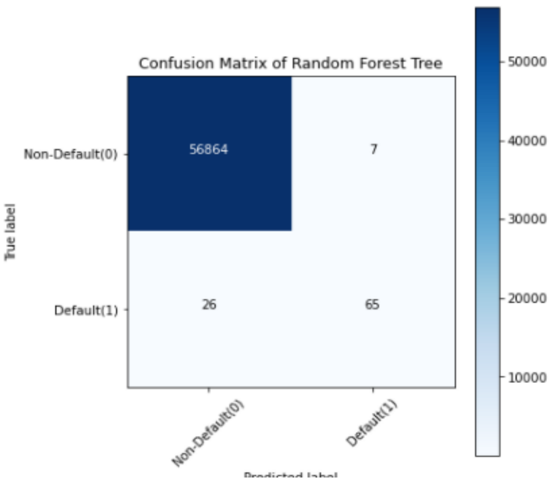
In [49]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [50]:
```
rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
```

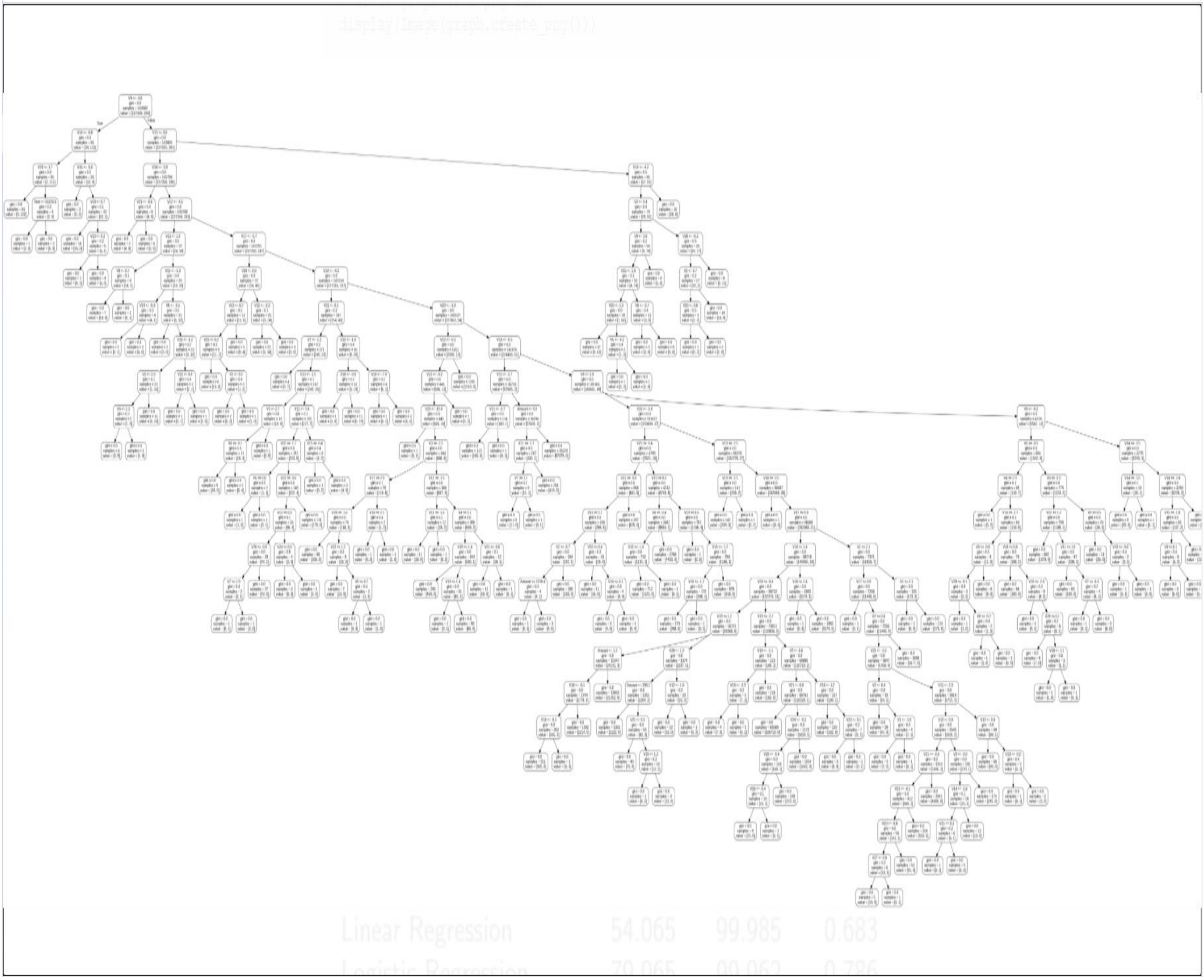
In [51]:
```
print('Accuracy score of the Random Forest Tree model is {}'.format(accuracy_score(y_test, rf_yhat)))
```

```
Accuracy score of the Random Forest Tree model is 0.999420666409185
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix for the random forest tree model
rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1]) # Random Forest Tree

#Random forest tree
rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                classes = ['Non-Default(0)','Default(1)'],
                                normalize = False, title = 'Random Forest Tree')
plt.savefig('rf_cm_plot.png')
plt.show()
```



In [53]:
```python
#visualizing the random tree
feature_list = list(X.columns)
# Import tools needed for visualization
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydot
#pulling out one tree from the forest
tree = rfc.estimators_[5]
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
display(Image(graph.create_png()))
```



CONCLUSION:

We analysed the past data to our knowledge of understanding,and modelled the data as we saw fit for four Machine Learning(ML) Algorithm,

1)SVM

2)KNN

3)Decision tree

4)Random forest

All 4 models provided good(0.9998 and 1.0) accuracy. When used for prediction provided the desired results.Hence we conclude all the 4 models can be used in the future to identify whether a new transaction is fraudulent or not. We were also able to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

FUTURE WORK:

We plan on using the same dataset to train a model on a few more ML algorithms that are suitable for this dataset such as

Logistic Regression

Linear Discriminant Analysis

Classification Trees

XGBoost Classifier

And will also include a comprehensive tuning of the previous done models. Having a data set with non anonymized features would make this particularly interesting as outputting the feature importance would enable one to see what specific factors are important for detecting fraudulent.We also plan on using neural networks approach helps automatically identify the characteristics most often found in fraudulent transactions; this method is most effective if you have a lot of transaction samples.

REFERENCES:

- https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba
- https://www.youtube.com/watch?v=5NcbVYhQJvw&t=8s

- https://towardsdatascience.com/detecting-credit-card-fraud-using-machine-learning-a3d83423d3b8#:~:text=popular%20classification%20algorithms%3A-,Logistic%20Regression,Classification%20Trees
- Machine Learning Group — ULB, Credit Card Fraud Detection (2018), Kaggle
- https://www.sciencedirect.com/science/article/pii/S1877050915007103

- https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/
- https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/
- https://www.geeksforgeeks.org/decision-tree-implementation-python/
- https://www.w3schools.com/python/python_ml_decision_tree.asp
- https://medium.com/analytics-vidhya/credit-card-fraud-detection-in-python-using-scikit-learn-f9046a030f50