GLOBSYN FINISHING SCHOOL ML WITH PYTHON TRAINING PROJECT

Project Name - Prediction of Claims Management using Python

Group Members:

Aarati Shah, Kalyani Govt. Engineering College,10200119051

Arghya Paul, Kalyani Govt. Engineering College,10200319015

Dibyanshu Shekhar Dey, Kalyani Govt. Engineering College,10201619054

Sahin Khatun, Kalyani Govt. Engineering College,10200119053

Subhankar Bose, Kalyani Govt Engineering. College, 10200319021

Table of Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty **Sir Titas Roy Chowdhury** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him from time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Aarati Shah

Arghya Paul

Dibyanshu Shekhar Dey

Sahin Khatun

Subhankar Bose

Project Objective

Description of problem:-

We are provided with an anonymized dataset containing both categorical and numeric variables available when the claims were received by a renowned insurance company. All string type variables are categorical. There are no ordinal variables.

The "target" column in the train set is the variable to predict. It is equal to 1 for claims suitable for an accelerated approval.

claims for which approval could be accelerated leading to faster payments 1

claims for which additional information is required before approval 0

What is our objective:

The primary objective of the Machine Learning model is to build a model in order to determine if a person's insurance claim could be accelerated leading to faster payments or if it requires approval.

Here precision is the correct metric.

A false positive is when a person determines something is true when it is actually false (also called a type I error). A false positive is a "false alarm." A false negative is saying something is false when it is actually true (also called a type II error). So, false positives take less time and false negatives take much time.But false positives are more dangerous and it must be avoided.

The false positive should be lower and so the precision will increase.So better the precision better the model. Precision= True Positive/True Positive+False Positive. If we decrease false positive recall becomes

Project Description

How are you planning to solve the project?

First we need to analyze the data and try to understand how the columns are related to the result of the data.

In case of doing so, we also need to take care of the facts related to the categorical and numerical data and if necessary then we may need to change or encode those with the relevant encoding.

We will do a detailed Exploratory Data Analysis (EDA) of the data, to get a better understanding related to the relations between the columns as well as with the final output column.

After analysing the EDA we will just fit a generalized ML algo and try to get the accuracy in that , without doing any kind of feature selection . This will help us to understand the base value of our model metric to analyze further.

Then we will consider the columns which are highly correlated to the final output column and perform the fitting of those ML algos and will compare the metrics between them.

After that we will implement GridSearchCV and RFE techniques to better optimize our models and get better results by selecting the most relevant features for our model implementation.

Missing Value Handling: -

The missing numerical values have been replaced by median while the categorical values after being converted to numerical values using Label Encoding have been replaced by mode.

Outlier Handling: -

The data consists of many outliers specific to certain columns , but analyzing further with the problem objective and column attributes we have come to the conclusion that the columns which primarily showcase some data as outliers can not be removed as they are huge in number so we are

not concerned about outliers. And as there are a large number of missing values trying to find outliers is predominately giving us the median value thus causing a concern of overfitting.

Outliers detected:

Number of outliers in v1 = 16119

Number of outliers in v2 = 14609

Number of outliers in v3 = 50000

Number of outliers in v4 = 16160

Number of outliers in v5 = 14303

Number of outliers in v6 = 16479

Number of outliers in v7 = 16038

Number of outliers in v8 = 9694

Number of outliers in v9 = 16081

Number of outliers in v10 = 6382

Number of outliers in v11 = 16033

Number of outliers in v12 = 5339

Number of outliers in v13 = 15742

Number of outliers in v14 = 2796

Number of outliers in v15 = 15728

Number of outliers in v16 = 16279

Number of outliers in v17 = 16626

Number of outliers in v18 = 16256

Number of outliers in v19 = 15372

Number of outliers in v20 = 15670

Number of outliers in v21 = 1385

Number of outliers in v22 = 0

Number of outliers in v23 = 16641

Number of outliers in v24 = 0

Number of outliers in v25 = 8982

Number of outliers in v26 = 15836

Number of outliers in v27 = 15859

Number of outliers in v28 = 16607

Number of outliers in v29 = 15220

Number of outliers in v30 = 50000

Number of outliers in v31 = 50000

Number of outliers in v32 = 16165

Number of outliers in v33 = 15463

Number of outliers in v34 = 61

Number of outliers in v35 = 15975

Number of outliers in v36 = 7801

Number of outliers in v37 = 16125

Number of outliers in v38 = 50000

Number of outliers in v39 = 13440

Number of outliers in v40 = 79

Number of outliers in v41 = 15945

Number of outliers in v42 = 16206

Number of outliers in v43 = 15491

Number of outliers in v44 = 16606

Number of outliers in v45 = 16212

Number of outliers in v46 = 9293

Number of outliers in v47 = 0

Number of outliers in v48 = 16027

Number of outliers in v49 = 16375

Number of outliers in v50 = 2289

Number of outliers in v51 = 16899

Number of outliers in v52 = 0

Number of outliers in v53 = 15445

Number of outliers in v54 = 9416

Number of outliers in v55 = 15671

Number of outliers in v56 = 0

Number of outliers in v57 = 15938

Number of outliers in v58 = 9150

Number of outliers in v59 = 16047

Number of outliers in v60 = 16108

Number of outliers in v61 = 15817

Number of outliers in v62 = 50000

Number of outliers in v63 = 9210

Number of outliers in v64 = 16379

Number of outliers in v65 = 15848

Number of outliers in v66 = 0

Number of outliers in v67 = 16035

Number of outliers in v68 = 15298

Number of outliers in v69 = 15386

Number of outliers in v70 = 13412

Number of outliers in v71 = 1

Number of outliers in v72 = 1540

Number of outliers in v73 = 15792

Number of outliers in v74 = 50000

Number of outliers in v75 = 0

Number of outliers in v76 = 16537

Number of outliers in v77 = 15672

Number of outliers in v78 = 16533

Number of outliers in v79 = 9840

Number of outliers in v80 = 16094

Number of outliers in v81 = 15296

Number of outliers in v82 = 13239

Number of outliers in v83 = 15581

Number of outliers in v84 = 16284

Number of outliers in v85 = 17090

Number of outliers in v86 = 15787

Number of outliers in v87 = 16222

Number of outliers in v88 = 15424

Number of outliers in v89 = 9684

Number of outliers in v90 = 16116

Number of outliers in v91 = 0

Number of outliers in v92 = 16199

Number of outliers in v93 = 16100

Number of outliers in v94 = 16223

Number of outliers in v95 = 16641

Number of outliers in v96 = 16257

Number of outliers in v97 = 15735

Number of outliers in v98 = 14366

Number of outliers in v99 = 16165

Number of outliers in v100 = 9977

Number of outliers in v101 = 16284

Number of outliers in v102 = 18032

Number of outliers in v103 = 16424

Number of outliers in v104 = 16163

Number of outliers in v105 = 9692

Number of outliers in v106 = 16145

Number of outliers in v107 = 0

Number of outliers in v108 = 13670

Number of outliers in v109 = 14083

Number of outliers in v110 = 0

Number of outliers in v111 = 15453

Number of outliers in v112 = 0

Number of outliers in v113 = 50000

Number of outliers in v114 = 650

Number of outliers in v115 = 15521

Number of outliers in v116 = 15569

Number of outliers in v117 = 15423

Number of outliers in v118 = 15484

Number of outliers in v119 = 15633

Number of outliers in v120 = 15384

Number of outliers in v121 = 15695

Number of outliers in v122 = 16171

Number of outliers in v123 = 16644

Number of outliers in v124 = 10176

Number of outliers in v125 = 0

Number of outliers in v126 = 15767

Number of outliers in v127 = 15678

Number of outliers in v128 = 15248

Number of outliers in v129 = 50000

Number of outliers in v130 = 15512

Number of outliers in v131 = 16358

Data Cleaning:

As the dataset is code unfriendly we can not drop columns as we don't know which columns contain relevant information. So we are not dropping any columns.

Project Scope

The project can be used by insurance companies in order to detect which person's insurance claim is ready to be accepted and whose insurance claims need to be processed further. By doing this they prevent potential frauds.

Data Description

1.Source of data:-

The data has been provided by Globsyn Finishing School.

2.Each feature/column description:-

a.Data set description:-

Number of rows - 114321

Number of columns- 133

The data looks like:-

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v130	v131
0	3	1	1.335739	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	AU	1.804126	3.113719	2.024285	0	0.636365	2.857144
1	4	1	NaN	NaN	C	NaN	9.191265	NaN	NaN	2.301630	...	NaN	NaN	0.598896	AF	NaN	NaN	1.957825	0	NaN	NaN
2	5	1	0.943877	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	AE	1.773709	3.922193	1.120468	2	0.883118	1.176472
3	6	1	0.797415	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	CJ	1.415230	2.954381	1.990847	1	1.677108	1.034483
4	8	1	NaN	NaN	C	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Z	NaN	NaN	NaN	0	NaN	NaN

5 rows × 133 columns

(1)Who will use/how this prediction will add value ?

Ans: Dataset may be used by both insurance companies and insurance buyers. An insurance company wants to bring more revenue in premiums than it should pay on conditional payouts. This dataset and its analysis will help the company to assess the risks involved. For a buyer it is equally important to assess the performance of an insurance company and take a risk for his/her future. This dataset will provide such an assessment.

(2)Define +ve case/-ve case(classification)

Ans: +ve case :claims for which approval could be accelerated leading to faster payments -> 1

-ve case:claims for which additional information is required before approval -> 0

(3)What metric is appropriate(accuracy,recall,precision etc)

Ans: A false positive is when a person determines something is true when it is actually false (also called a type I error). A false positive is a “false alarm.” A false negative is saying something is false when it is actually true (also called a type II error). So, false positives take less time and false negatives take much time.But false positives are more dangerous and it must be avoided.

The false positive should be lower and so the precision will increase.

(4)Is the datafile in csv format ?What is a separator ?(can "read_csv" be used to open a file?)

Ans: Yes,the datafile is in csv format.Here ',' is the separator. Yes, "read_csv" can be used to open files.

(5)Header present ? Are column names code friendly?

Ans:Header is present and column names are not code friendly.

(6) How many rows and columns are present in the file ?

Ans: rows - 114321

columns- 133

(7) Check which column has null(na)(percentage)

Ans:

v1 43.59

v2 43.56

v3 3.02

v4 43.56

v5 42.53

v6 43.59

v7 43.59

v8 42.53

v9 43.61

v10 0.07

v11 43.59

v12 0.08

v13 43.59

v15 43.59

v16 43.64

v17 43.56

v18 43.59

v19 43.6

v20 43.6

v21 0.53

v22 0.44

v23 44.33

v25 42.53

v26 43.59

v27 43.59

v28 43.59

v29 43.59

v30 52.58

v31 3.02

v32 43.59

v33 43.59

v34 0.1

v35 43.59

v36 42.53

v37 43.6

v39 43.59

v40 0.1

v41 43.59

v42 43.59

v43 43.59

v44 43.56

v45 43.59

v46 42.53

v48 43.56

v49 43.59

v50 0.08

v51 44.33

v53 43.59

v54 42.53

v55 43.59

v56 6.02

v57 43.59

v58 43.59

v59 43.56

v60 43.59

v61 43.56

v63 42.53

v64 43.56

v65 43.6

v67 43.59

v68 43.59

v69 43.64

v70 42.54

v73 43.59

v76 43.56

v77 43.59

v78 43.64

v80 43.61

v81 42.53

v82 42.53

v83 43.59

v84 43.59

v85 44.33

v86 43.59

v87 42.57

v88 43.59

v89 42.53

v90 43.59

v92 43.6

v93 43.59

v94 43.59

v95 43.6

v96 43.59

v97 43.6

v98 42.56

v99 43.59

v100 43.59

v101 43.56

v102 44.89

v103 43.59

v104 43.59

v105 42.56

v106 43.56

v108 42.53

v109 42.53

v111 43.59

v112 0.33

v113 48.38

v114 0.03

v115 43.64

v116 43.59

v117 42.53

v118 43.6

v119 44.33

v120 43.59

v121 43.6

v122 43.61

v123 44.33

v124 42.53

v125 0.07

v126 43.59

v127 43.59

v128 42.53

v130 43.6

v131 43.64

(8) Find your X and y(target)

Ans: y : 'target'

X: rest of the columns

(9) how many numeric and categorical(string)

-in all numeric which are discrete / continuous

Ans: Part 1:

numeric:114

categorical:19

Part 2:

continuous - 108

discrete - 6

(10) For all categoricals , find their labels?

Ans:['v3',

'v22',

'v24',

'v30',

'v31',

'v47',

'v52',

'v56',

'v66',

'v71',

'v74',

'v75',

'v79',

'v91',

'v107',

'v110',

'v112',

'v113',

'v125']

(11) in classification : distribution of y (how many 1 and 0)

Ans: 1 87021 (76.11%) 0 27300 (23.89%)Name: target, dtype: int64

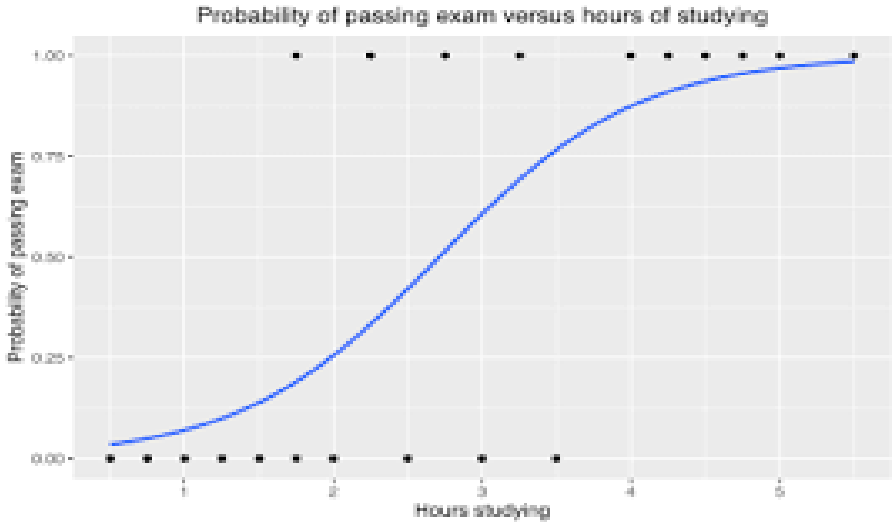
(12) Do all continuous columns have the same scale ?

Ans: We can infer from the description shown in the df.describe() snippet that there is a considerable difference of scale between the minimum and the maximum value. So the columns don't have the same scale

Model building

1.Logistic Regression:-

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.



[68]:
```
X = df1.drop('target',axis = 1)
y = df1.target # Target variable
```

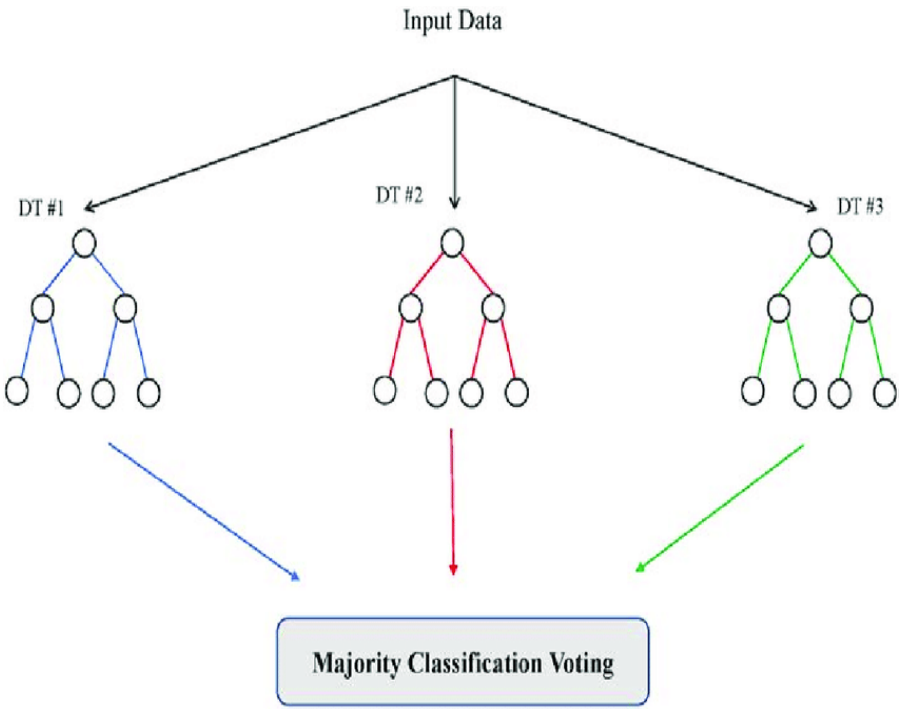
[69]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
```

[70]:
```
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

Because of low recall Logistic regression will not work properly.

2.Decision Tree:-

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter..



decision tree

[74]:
```
from sklearn.tree import DecisionTreeClassifier
dct = DecisionTreeClassifier()
```

[75]:
```
dct.fit(X_train,y_train)
```

[75]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

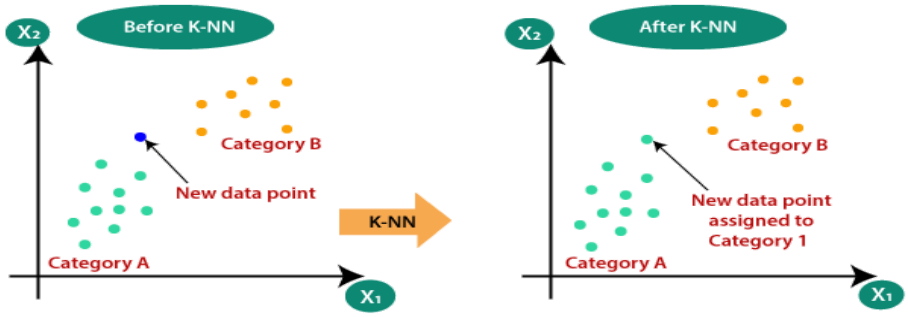
[76]:
```
y_pred = dct.predict(X_test)
```

[77]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.68384
Precision: 0.798681651177869
Recall: 0.7796413502109705
f1 Score: 0.7890466531440162
AUC: 0.5813769664962137
```

3.KNN:-

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.



In [78]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

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

In [79]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Out[79]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

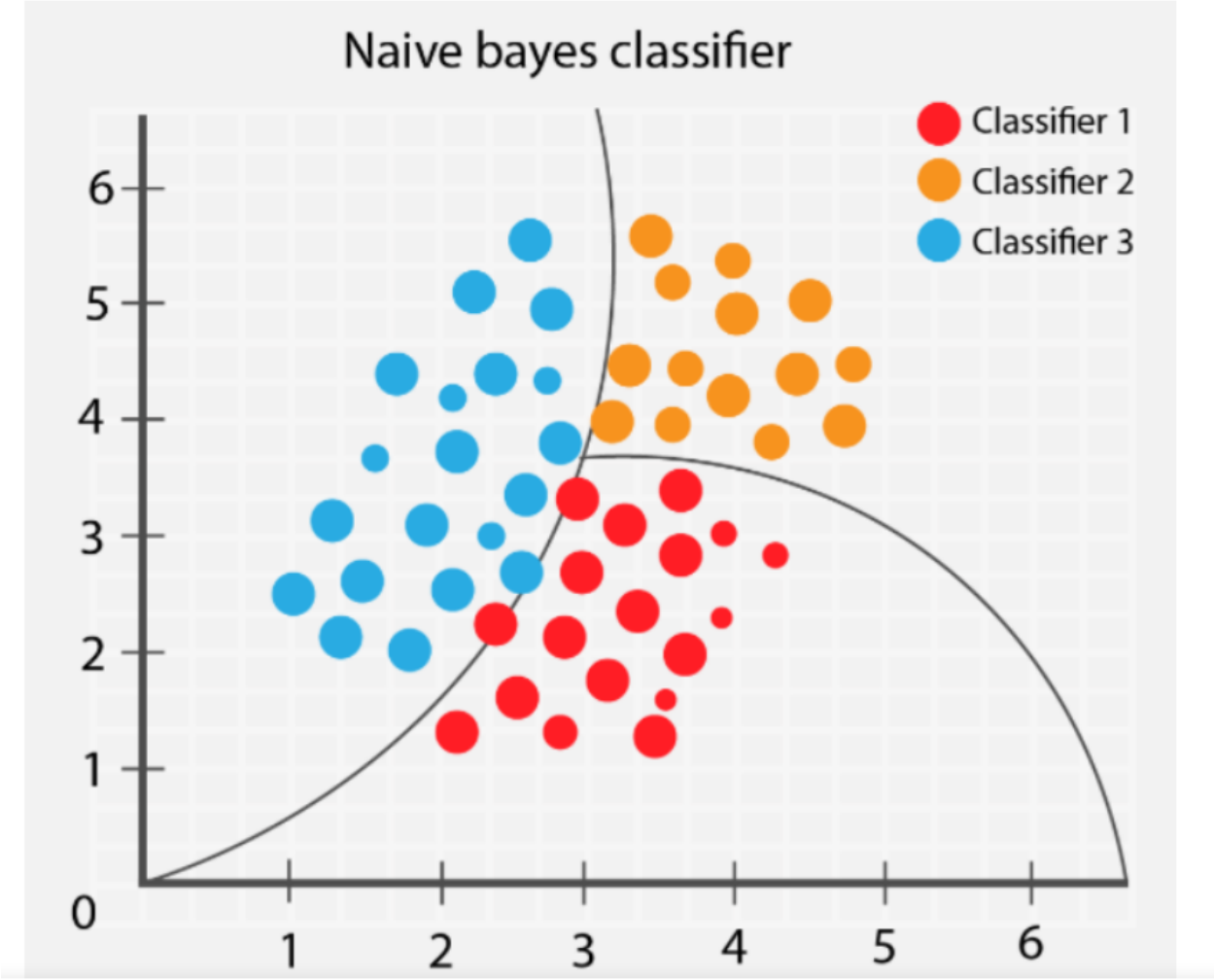
In [80]:
```
y_pred = classifier.predict(X_test)
```

In [82]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.72928
Precision: 0.7770405380839848
Recall: 0.9017932489451477
f1 Score: 0.8347817595937896
AUC: 0.5447707966580043
```

4.Naive Bayes:-

Naive Bayes uses a similar method to predict the probability of different classes based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.



naive bayes

In [83]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train,y_train)
```

Out[83]:
```
GaussianNB()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [84]:
```
y_pred =naive_bayes.predict(X_test)
```

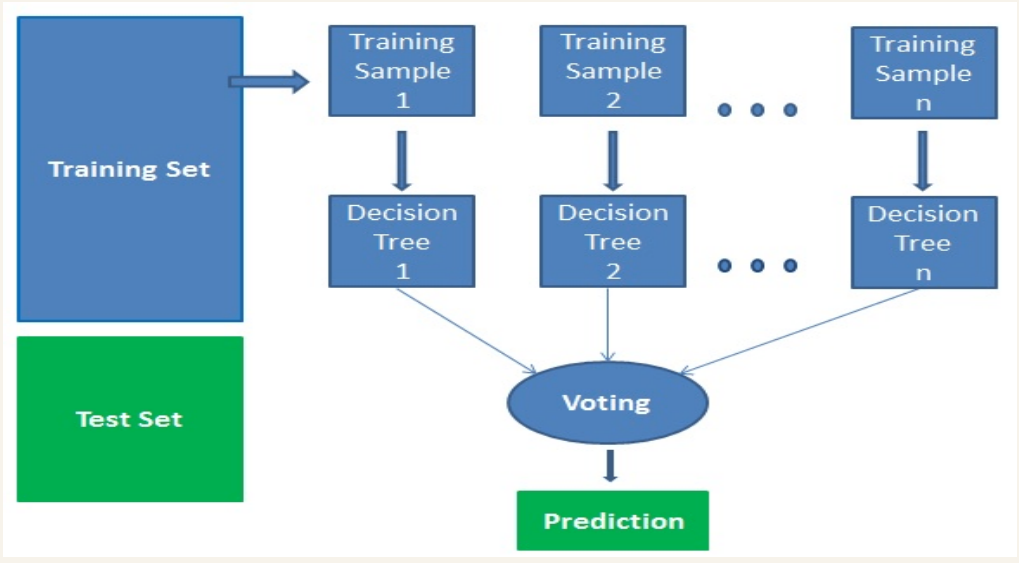
In [85]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.55976
Precision: 0.85451952219647
Recall: 0.505590717299578
f1 Score: 0.6352972363973756
AUC: 0.6176960209014446
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV , cross_val_score
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score,roc_curve,auc
```

5. Random Forest

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 consists of trees. It is said that the more trees there are, the more robust a forest is. Random forests create decision trees on randomly selected data samples, get predictions from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance.



Random forest

df1

In [72]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

In [73]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.77272
Precision: 0.7856466741244299
Recall: 0.9630801687763713
f1 Score: 0.8653618311928345
AUC: 0.5691228658451393
```

Code

1.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
from sklearn import metrics
```

2.

In [3]:
```
df=pd.read_csv("train.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v130	
0	3	1	1.335739	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	AU	1.804126	3.113719	2.024285	0	0.636365	2.8
1	4	1	NaN	NaN	C	NaN	9.191265	NaN	NaN	2.301630	...	NaN	NaN	0.598896	AF	NaN	NaN	1.957825	0	NaN	
2	5	1	0.943877	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	AE	1.773709	3.922193	1.120468	2	0.883118	1.1
3	6	1	0.797415	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	CJ	1.415230	2.954381	1.990847	1	1.677108	1.0
4	8	1	NaN	NaN	C	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Z	NaN	NaN	NaN	0	NaN	

5 rows × 133 columns

3.

In [5]:
```
df.info
```

Out[5]:
```
<bound method DataFrame.info of            ID  target        v1        v2 v3        v4         v5        v6  \
0           3       1  1.335739  8.727474  C  3.921026   7.915266  2.599278
1           4       1       NaN       NaN  C       NaN   9.191265       NaN
2           5       1  0.943877  5.310079  C  4.410969   5.326159  3.979592
3           6       1  0.797415  8.304757  C  4.225930  11.627438  2.097700
4           8       1       NaN       NaN  C       NaN        NaN       NaN
...       ...     ...       ...       ... ..       ...        ...       ...
114316  228708       1       NaN       NaN  C       NaN        NaN       NaN
114317  228710       1       NaN       NaN  C       NaN        NaN       NaN
114318  228711       1       NaN       NaN  C       NaN  10.069277       NaN
114319  228712       1       NaN       NaN  C       NaN  10.106144       NaN
114320  228713       1  1.619763  7.932978  C  4.640085   8.473141  2.351470

             v7        v8  ...      v122      v123      v124  v125      v126  \
0      3.176895  0.012941  ...  8.000000  1.989780  0.035754    AU  1.804126
1           NaN  2.301630  ...       NaN       NaN  0.598896    AF       NaN
2      3.928571  0.019645  ...  9.333333  2.477596  0.013452    AE  1.773709
3      1.987549  0.171947  ...  7.018256  1.812795  0.002267    CJ  1.415230
4           NaN       NaN  ...       NaN       NaN       NaN     Z       NaN
...         ...       ...  ...       ...       ...       ...   ...       ...
114316      NaN       NaN  ...       NaN       NaN       NaN    AL       NaN
114317      NaN       NaN  ...       NaN       NaN       NaN     E       NaN
114318      NaN  0.323324  ...       NaN       NaN  0.156764     Q       NaN
114319      NaN  0.309226  ...       NaN       NaN  0.490658    BW       NaN
114320  2.826766  3.479754  ...  7.936508  2.944285  3.135205     V  1.943149

           v127      v128  v129      v130      v131
0      3.113719  2.024285     0  0.636365  2.857144
1           NaN  1.957825     0       NaN       NaN
2      3.922193  1.120468     2  0.883118  1.176472
3      2.954381  1.990847     1  1.677108  1.034483
4           NaN       NaN     0       NaN       NaN
...         ...       ...   ...       ...       ...
114316      NaN       NaN     0       NaN       NaN
114317      NaN       NaN     1       NaN       NaN
114318      NaN  2.417606     2       NaN       NaN
114319      NaN  3.526650     0       NaN       NaN
114320  4.385553  1.604493     0  1.787610  1.386138

[114321 rows x 133 columns]>
```

4.

6. How many rows and columns present in the file ?

In [6]:
```
df.shape
```

Out[6]: (114321, 133)

5.

7. check which column has null(na)(percentage)[IMPORTANT]

In [6]:
```
percent_missing = dict(round(df.isnull().sum() * 100 / len(df), 2))
```

6.

In [8]:
```
percent_missing
```

Out[8]:
```
{'ID': 0.0,
 'target': 0.0,
 'v1': 43.59,
 'v2': 43.56,
 'v3': 3.02,
 'v4': 43.56,
 'v5': 42.53,
 'v6': 43.59,
 'v7': 43.59,
 'v8': 42.53,
 'v9': 43.61,
 'v10': 0.07,
 'v11': 43.59,
 'v12': 0.08,
 'v13': 43.59,
 'v14': 0.0,
 'v15': 43.59,
 'v16': 43.64,
 'v17': 43.56,
 'v18': 43.59,
 'v19': 43.6,
 'v20': 43.6,
 'v21': 0.53,
 'v22': 0.44,
 'v23': 44.33,
 'v24': 0.0,
 'v25': 42.53,
 'v26': 43.59,
 'v27': 43.59,
 'v28': 43.59,
 'v29': 43.59,
 'v30': 52.58,
 'v31': 3.02,
 'v32': 43.59,
```

7.

```
'v33': 43.59,
'v34': 0.1,
'v35': 43.59,
'v36': 42.53,
'v37': 43.6,
'v38': 0.0,
'v39': 43.59,
'v40': 0.1,
'v41': 43.59,
'v42': 43.59,
'v43': 43.59,
'v44': 43.56,
'v45': 43.59,
'v46': 42.53,
'v47': 0.0,
'v48': 43.56,
'v49': 43.59,
'v50': 0.08,
'v51': 44.33,
'v52': 0.0,
'v53': 43.59,
'v54': 42.53,
'v55': 43.59,
'v56': 6.02,
'v57': 43.59,
'v58': 43.59,
'v59': 43.56,
'v60': 43.59,
'v61': 43.56,
'v62': 0.0,
'v63': 42.53,
'v64': 43.56,
'v65': 43.6,
'v66': 0.0,
'v67': 43.59,
'v68': 43.59,
'v69': 43.64,
'v70': 42.54,
'v71': 0.0,
'v72': 0.0,
'v73': 43.59,
'v74': 0.0,
'v75': 0.0,
'v76': 43.56,
'v77': 43.59,
'v78': 43.64,
'v79': 0.0,
'v80': 43.61,
'v81': 42.53,
'v82': 42.53,
'v83': 43.59,
'v84': 43.59,
'v85': 44.33,
'v86': 43.59,
'v87': 42.57,
'v88': 43.59,
'v89': 42.53,
'v90': 43.59,
'v91': 0.0,
'v92': 43.6,
'v93': 43.59,
'v94': 43.59,
'v95': 43.6,
'v96': 43.59,
'v97': 43.6,
'v98': 42.56,
'v99': 43.59,
'v100': 43.59,
'v101': 43.56,
'v102': 44.89,
'v103': 43.59,
'v104': 43.59,
'v105': 42.56,
'v106': 43.56,
'v106': 43.56,
'v107': 0.0,
'v108': 42.53,
'v109': 42.53,
'v110': 0.0,
'v111': 43.59,
'v112': 0.33,
'v113': 48.38,
'v114': 0.03,
'v115': 43.64,
'v116': 43.59,
'v117': 42.53,
'v118': 43.6,
'v119': 44.33,
'v120': 43.59,
'v121': 43.6,
'v122': 43.61,
'v123': 44.33,
'v124': 42.53,
'v125': 0.07,
'v126': 43.59,
'v127': 43.59,
'v128': 42.53,
'v129': 0.0,
'v130': 43.6,
'v131': 43.64}
```

8.

8 part

- y : 'target'
- X: rest of the columns

9. how many numeric and categorical(string)

In [10]:
```
numeric_data=df.select_dtypes(include=[np.number])
numeric_data.shape[1]
```

Out[10]: 114

In [11]:
```
categorical_data=df.select_dtypes(exclude=[np.number])
categorical_data.shape[1]
```

Out[11]: 19

9.

In [12]:
```
numeric_data
```

Out[12]:

	ID	target	v1	v2	v4	v5	v6	v7	v8	v9	...	v121	v122	v123	v124	v126	v127	v128
0	3	1	1.335739	8.727474	3.921026	7.915266	2.599278	3.176895	0.012941	9.999999	...	0.803572	8.000000	1.989780	0.035754	1.804126	3.113719	2.024285
1	4	1	NaN	NaN	NaN	9.191265	NaN	NaN	2.301630	NaN	...	NaN	NaN	NaN	0.598896	NaN	NaN	1.957825
2	5	1	0.943877	5.310079	4.410969	5.326159	3.979592	3.928571	0.019645	12.666667	...	2.238806	9.333333	2.477596	0.013452	1.773709	3.922193	1.120468
3	6	1	0.797415	8.304757	4.225930	11.627438	2.097700	1.987549	0.171947	8.965516	...	1.956521	7.018256	1.812795	0.002267	1.415230	2.954381	1.990847
4	8	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
114316	228708	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
114317	228710	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
114318	228711	1	NaN	NaN	NaN	10.069277	NaN	NaN	0.323324	NaN	...	NaN	NaN	NaN	0.156764	NaN	NaN	2.417606
114319	228712	1	NaN	NaN	NaN	10.106144	NaN	NaN	0.309226	NaN	...	NaN	NaN	NaN	0.490658	NaN	NaN	3.526650
114320	228713	1	1.619763	7.932978	4.640085	8.473141	2.351470	2.826766	3.479754	9.629630	...	4.016948	7.936508	2.944285	3.135205	1.943149	4.385553	1.604493

114321 rows × 114 columns

10.

in all numeric which are discrete / continuous

In [13]:
```
# continuous variable as float, discrete variable as int
```

9 part 2

In [14]:
```
# 9 part 2
num=dict(numeric_data.select_dtypes(include=['float64']).dtypes)
```

In [15]:
```
num
```

Out[15]:
```
{'v1': dtype('float64'),
 'v2': dtype('float64'),
 'v4': dtype('float64'),
 'v5': dtype('float64'),
 'v6': dtype('float64'),
 'v7': dtype('float64'),
 'v8': dtype('float64'),
 'v9': dtype('float64'),
 'v10': dtype('float64'),
 'v11': dtype('float64'),
 'v12': dtype('float64'),
 'v13': dtype('float64'),
 'v14': dtype('float64'),
 'v15': dtype('float64'),
 'v16': dtype('float64'),
 'v17': dtype('float64'),
 'v18': dtype('float64'),
 'v19': dtype('float64'),
 'v20': dtype('float64'),
```

11.

```
                        'v67': dtype('float64'),
                        'v68': dtype('float64'),
                        'v69': dtype('float64'),
                        'v70': dtype('float64'),
                        'v73': dtype('float64'),
 'v21': dtype('float64'),'v76': dtype('float64'),
 'v23': dtype('float64'),'v77': dtype('float64'),
 'v25': dtype('float64'),'v78': dtype('float64'),
 'v26': dtype('float64'),'v80': dtype('float64'),
 'v27': dtype('float64'),'v81': dtype('float64'),
 'v28': dtype('float64'),'v82': dtype('float64'),
 'v29': dtype('float64'),'v83': dtype('float64'),
 'v32': dtype('float64'),'v84': dtype('float64'),
 'v33': dtype('float64'),'v85': dtype('float64'),
 'v34': dtype('float64'),'v86': dtype('float64'),
 'v35': dtype('float64'),'v87': dtype('float64'),
 'v36': dtype('float64'),'v88': dtype('float64'),
 'v37': dtype('float64'),'v89': dtype('float64'),
 'v39': dtype('float64'),'v90': dtype('float64'),
 'v40': dtype('float64'),'v92': dtype('float64'),
 'v41': dtype('float64'),'v93': dtype('float64'),
 'v42': dtype('float64'),'v94': dtype('float64'),
 'v43': dtype('float64'),'v95': dtype('float64'),
 'v44': dtype('float64'),'v96': dtype('float64'),
 'v45': dtype('float64'),'v97': dtype('float64'),
 'v46': dtype('float64'),'v98': dtype('float64'),
 'v48': dtype('float64'),'v99': dtype('float64'),
 'v49': dtype('float64'),'v100': dtype('float64'),
 'v50': dtype('float64'),'v101': dtype('float64'),
 'v51': dtype('float64'),'v102': dtype('float64'),'v118': dtype('float64'),
 'v53': dtype('float64'),'v103': dtype('float64'),'v119': dtype('float64'),
 'v54': dtype('float64'),'v104': dtype('float64'),'v120': dtype('float64'),
 'v55': dtype('float64'),'v105': dtype('float64'),'v121': dtype('float64'),
 'v57': dtype('float64'),'v106': dtype('float64'),'v122': dtype('float64'),
 'v58': dtype('float64'),'v108': dtype('float64'),'v123': dtype('float64'),
 'v59': dtype('float64'),'v109': dtype('float64'),'v124': dtype('float64'),
 'v60': dtype('float64'),'v111': dtype('float64'),'v126': dtype('float64'),
 'v61': dtype('float64'),'v114': dtype('float64'),'v127': dtype('float64'),
 'v63': dtype('float64'),'v115': dtype('float64'),'v128': dtype('float64'),
 'v64': dtype('float64'),'v116': dtype('float64'),'v130': dtype('float64'),
 'v65': dtype('float64'),'v117': dtype('float64'),'v131': dtype('float64')}
```

12.

```
In [16]: len(num)

Out[16]: 108

In [17]: num1=dict(numeric_data.select_dtypes(include=['int64']).dtypes)

In [18]: len(num1)

Out[18]: 6
```

13.

11.in classification : distribution of y (how many 1 and 0)

In [20]:
```
df['target'].value_counts()
```

Out[20]:
```
1    87021
0    27300
Name: target, dtype: int64
```

14.

12. are all continuous columns have same scale ?

In [21]:
```
df.describe()
```

Out[21]:

	ID	target	v1	v2	v4	v5	v6	v7	v8	v9	...	v121	
count	114321.000000	114321.000000	6.448900e+04	6.452500e+04	6.452500e+04	6.569700e+04	6.448900e+04	6.448900e+04	6.570200e+04	6.447000e+04	...	6.448100e+04	6.44700
mean	114228.928228	0.761199	1.630686e+00	7.464411e+00	4.145098e+00	8.742359e+00	2.436402e+00	2.483921e+00	1.496569e+00	9.031859e+00	...	2.737596e+00	6.82243
std	65934.487362	0.426353	1.082813e+00	2.961676e+00	1.148263e+00	2.036018e+00	5.999653e-01	5.894485e-01	2.783003e+00	1.930262e+00	...	1.356294e+00	1.79597
min	3.000000	0.000000	-9.996497e-07	-9.817614e-07	-6.475929e-07	-5.287068e-07	-9.055091e-07	-9.468765e-07	-7.783778e-07	-9.828757e-07	...	-9.820642e-07	-9.97
25%	57280.000000	1.000000	9.135798e-01	5.316428e+00	3.487398e+00	7.605918e+00	2.065064e+00	2.101477e+00	8.658986e-02	7.853659e+00	...	1.786965e+00	5.64771
50%	114189.000000	1.000000	1.469550e+00	7.023803e+00	4.205991e+00	8.670867e+00	2.412790e+00	2.452166e+00	3.860317e-01	9.059582e+00	...	2.436195e+00	6.74911
75%	171206.000000	1.000000	2.136128e+00	9.465497e+00	4.833250e+00	9.771353e+00	2.775285e+00	2.834285e+00	1.625246e+00	1.023256e+01	...	3.379175e+00	7.91139
max	228713.000000	1.000000	2.000000e+01	2.000000e+01	2.000000e+01	2.000000e+01	2.000000e+01	2.000000e+01	2.000000e+01	2.000000e+01	...	2.000000e+01	2.00000

8 rows × 114 columns

16.

Taking 50000 rows in a new dataframe

In [22]:
```
df1=pd.read_csv('train.csv',nrows=50000)
```

analysing and visualising data

In [23]:
```
df1['target'].value_counts()
```

Out[23]:
```
1    38064
0    11936
Name: target, dtype: int64
```

17.

filling null values with the median of the column

In [24]:
```
df1=df1.fillna(df1.median())
df1
```

Out[24]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v
0	3	1	1.335739	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	AU	1.804126	3.113719	2.024285	0	0.636
1	4	1	1.463753	7.027493	C	4.198941	9.191265	2.408972	2.447977	2.301630	...	6.728972	2.742854	0.598896	AF	1.614076	2.967717	1.957825	0	1.560
2	5	1	0.943877	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	AE	1.773709	3.922193	1.120468	2	0.883
3	6	1	0.797415	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	CJ	1.415230	2.954381	1.990847	1	1.677
4	8	1	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	Z	1.614076	2.967717	1.799131	0	1.560
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
49995	99892	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	BM	1.614076	2.967717	1.799131	0	1.560
49996	99894	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	BM	1.614076	2.967717	1.799131	0	1.560
49997	99896	0	1.868688	5.046488	C	3.561714	7.889016	2.548563	2.191142	2.089548	...	5.868946	3.964544	0.215784	AA	1.730469	4.006411	1.461821	0	1.914
49998	99899	1	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	E	1.614076	2.967717	1.799131	1	1.560
49999	99900	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	W	1.614076	2.967717	1.799131	0	1.560

50000 rows × 133 columns

18.

In [25]:
```
df1.isna().sum()
```

Out[25]:
```
ID          0
target      0
v1          0
v2          0
v3       1499
         ...
v127        0
v128        0
v129        0
v130        0
v131        0
Length: 133, dtype: int64
```

filling the value of null values in categorical columns with the mode

19.

filling the value of null values in categorical columns with the mode

In [26]:
```
df1= df1.fillna(df1.mode().iloc[0])
df1
```

Out[26]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v
0	3	1	1.335739	8.727474	C	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	AU	1.804126	3.113719	2.024285	0	0.636
1	4	1	1.463753	7.027493	C	4.198941	9.191265	2.408972	2.447977	2.301630	...	6.728972	2.742854	0.598896	AF	1.614076	2.967717	1.957825	0	1.560
2	5	1	0.943877	5.310079	C	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	AE	1.773709	3.922193	1.120468	2	0.883
3	6	1	0.797415	8.304757	C	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	CJ	1.415230	2.954381	1.990847	1	1.677
4	8	1	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	Z	1.614076	2.967717	1.799131	0	1.560
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
49995	99892	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	BM	1.614076	2.967717	1.799131	0	1.560
49996	99894	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	BM	1.614076	2.967717	1.799131	0	1.560
49997	99896	0	1.868688	5.046488	C	3.561714	7.889016	2.548563	2.191142	2.089548	...	5.868946	3.964544	0.215784	AA	1.730469	4.006411	1.461821	0	1.914
49998	99899	1	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	E	1.614076	2.967717	1.799131	1	1.560
49999	99900	0	1.463753	7.027493	C	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	W	1.614076	2.967717	1.799131	0	1.560

50000 rows × 133 columns

20.

In [27]:
```
df1.isna().sum()
```

Out[27]:
```
ID        0
target    0
v1        0
v2        0
v3        0
         ..
v127      0
v128      0
v129      0
v130      0
v131      0
Length: 133, dtype: int64
```

In [28]:
```
numeric_data=df1.select_dtypes(include=[np.number])
numeric_data.shape[1]
```

Out[28]: 114

In [29]:
```
categorical_data=df1.select_dtypes(exclude=[np.number])
categorical_data.shape[1]
```

Out[29]: 19

21.

In [30]:
```
corrM=df1.corr()
corrM
```

Out[30]:

	ID	target	v1	v2	v4	v5	v6	v7	v8	v9	...	v121	v122	v123	v124	v126	v1
ID	1.000000	0.006807	-0.002018	-0.001385	0.003470	0.006563	0.002696	-0.002326	-0.003843	-0.002129	...	-0.002502	0.001542	-0.003651	-0.003114	-0.003854	0.0042
target	0.006807	1.000000	-0.011501	0.026289	0.043072	0.014099	0.029269	0.021006	-0.012733	-0.008484	...	-0.038030	0.002768	-0.047435	-0.011173	0.003332	0.0026
v1	-0.002018	-0.011501	1.000000	-0.204085	-0.147494	-0.058706	-0.016822	0.100214	0.178239	-0.023139	...	0.363630	-0.131335	0.310246	0.170700	0.035538	-0.0015
v2	-0.001385	0.026289	-0.204085	1.000000	0.531646	0.197791	0.026105	0.350913	-0.325145	-0.010288	...	-0.548028	0.173387	-0.554359	-0.224850	-0.097756	0.0185
v4	0.003470	0.043072	-0.147494	0.531646	1.000000	0.254731	0.383322	0.477593	-0.169870	-0.207135	...	-0.613752	0.045372	-0.696352	-0.059035	-0.124894	0.2890
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
v127	0.004283	0.002694	-0.001512	0.018576	0.289043	-0.226143	0.161627	0.245021	0.016172	0.129556	...	0.038947	0.103359	-0.126392	-0.024320	-0.071240	1.0000
v128	0.006809	0.020449	-0.012639	0.266430	0.471369	0.590957	0.235555	0.092573	0.052344	-0.277443	...	-0.341189	-0.051541	-0.219111	0.296014	-0.125475	-0.1445
v129	0.002622	0.141490	-0.016862	0.063915	0.086051	0.038439	0.038904	0.029504	-0.008212	-0.027816	...	-0.064671	0.000747	-0.074240	-0.001090	-0.012756	0.0116
v130	-0.001879	-0.037538	0.279149	-0.548215	-0.698891	-0.239639	-0.296732	-0.600244	0.348151	0.079801	...	0.815506	-0.188129	0.735913	0.254271	0.089523	-0.2122
v131	0.001007	0.015308	0.702566	0.180334	0.305775	0.149536	0.160915	0.322907	-0.008131	-0.152780	...	-0.228294	-0.012898	-0.154424	0.047562	-0.024583	0.0710

114 rows × 114 columns

22.

converting categorical to numeric using labelEncoder

In [31]:
```
l=df1.select_dtypes(include=['object']).columns.tolist()
l
```

Out[31]:
```
['v3',
 'v22',
 'v24',
 'v30',
 'v31',
 'v47',
 'v52',
 'v56',
 'v66',
 'v71',
 'v74',
 'v75',
 'v79',
 'v91',
 'v107',
 'v110',
 'v112',
 'v113',
 'v125']
```

23.

In [32]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

def labelEncode(data, col):
    for i in col:
        data[i] = le.fit_transform(data[i])

labelEncode(df1, l)
```

In [33]:
```
df1
```

Out[33]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v
0	3	1	1.335739	8.727474	2	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	21	1.804126	3.113719	2.024285	0	0.636
1	4	1	1.463753	7.027493	2	4.198941	9.191265	2.408972	2.447977	2.301630	...	6.728972	2.742854	0.598896	6	1.614076	2.967717	1.957825	0	1.560
2	5	1	0.943877	5.310079	2	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	5	1.773709	3.922193	1.120468	2	0.883
3	6	1	0.797415	8.304757	2	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	64	1.415230	2.954381	1.990847	1	1.677
4	8	1	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	89	1.614076	2.967717	1.799131	0	1.560
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
49995	99892	0	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	40	1.614076	2.967717	1.799131	0	1.560
49996	99894	0	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	40	1.614076	2.967717	1.799131	0	1.560
49997	99896	0	1.868688	5.046488	2	3.561714	7.889016	2.548563	2.191142	2.089548	...	5.868946	3.964544	0.215784	1	1.730469	4.006411	1.461821	0	1.914
49998	99899	1	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	68	1.614076	2.967717	1.799131	1	1.560
49999	99900	0	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	86	1.614076	2.967717	1.799131	0	1.560

50000 rows × 133 columns

24.

In [34]:
```
categorical_data=df1.select_dtypes(exclude=[np.number])
categorical_data.shape[1]
```

Out[34]:
```
0
```

In [35]:
```
numeric_data=df1.select_dtypes(include=[np.number])
numeric_data.shape[1]
```

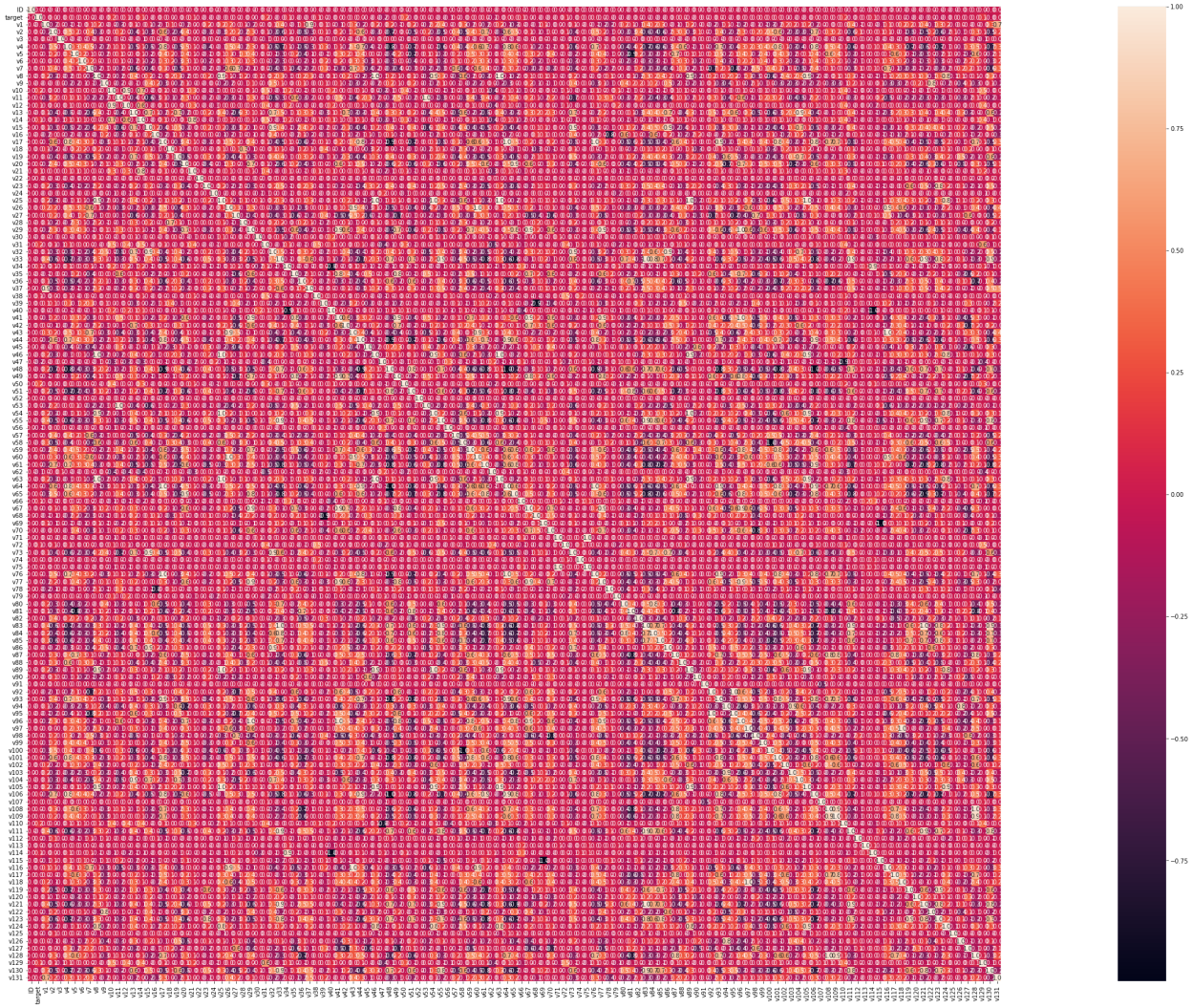
Out[35]:
```
133
```

25.

heatmp and correlation

In [227…

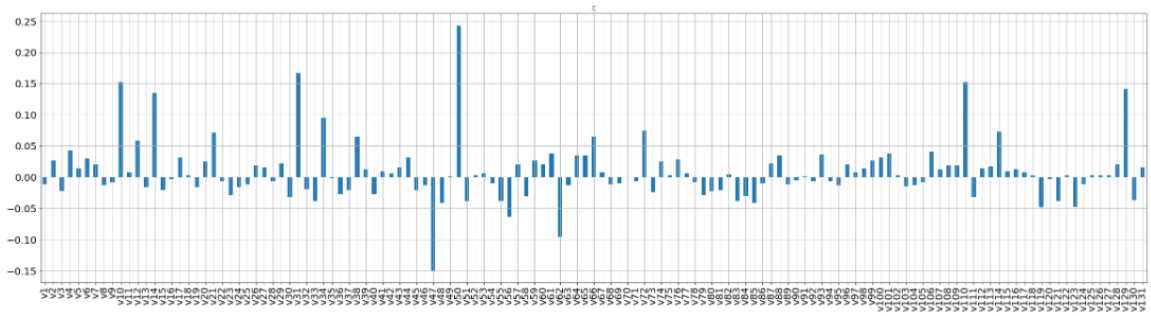
```
plt.figure(figsize=(70, 30))
sns.heatmap(df1.corr(),annot=True, square = True, fmt = '0.1f')
```



26.

In [20]:
```
a=df1.drop(['ID', 'target'],axis=1)
a.corrwith(df1['target']).plot.bar(figsize=(40,10),title="C",fontsize=20,rot=90,grid=True)
```

Out[20]:
```
<AxesSubplot:title={'center':'C'}>
```



27.

In [21]:
```
a
```

Out[21]:

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	...	v122	v123	v124	v125	v126	v127	v128	v129
0	1.335739	8.727474	2	3.921026	7.915266	2.599278	3.176895	0.012941	9.999999	0.503281	...	8.000000	1.989780	0.035754	21	1.804126	3.113719	2.024285	0
1	1.463753	7.027493	2	4.198941	9.191265	2.408972	2.447977	2.301630	9.037901	1.312910	...	6.728972	2.742854	0.598896	6	1.614076	2.967717	1.957825	0
2	0.943877	5.310079	2	4.410969	5.326159	3.979592	3.928571	0.019645	12.666667	0.765864	...	9.333333	2.477596	0.013452	5	1.773709	3.922193	1.120468	2
3	0.797415	8.304757	2	4.225930	11.627438	2.097700	1.987549	0.171947	8.965516	6.542669	...	7.018256	1.812795	0.002267	64	1.415230	2.954381	1.990847	1
4	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	9.037901	1.050328	...	6.728972	2.742854	0.139864	89	1.614076	2.967717	1.799131	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
49995	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	9.037901	1.028446	...	6.728972	2.742854	0.139864	40	1.614076	2.967717	1.799131	0
49996	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	9.037901	2.625821	...	6.728972	2.742854	0.139864	40	1.614076	2.967717	1.799131	0
49997	1.868688	5.046488	2	3.561714	7.889016	2.548563	2.191142	2.089548	8.148149	1.312910	...	5.868946	3.964544	0.215784	1	1.730469	4.006411	1.461821	0
49998	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	9.037901	1.291028	...	6.728972	2.742854	0.139864	68	1.614076	2.967717	1.799131	1
49999	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	9.037901	1.050328	...	6.728972	2.742854	0.139864	86	1.614076	2.967717	1.799131	0

50000 rows × 131 columns

28.

In [39]:
```
df1.head()
```

Out[39]:

	ID	target	v1	v2	v3	v4	v5	v6	v7	v8	...	v122	v123	v124	v125	v126	v127	v128	v129	v130	
0	3	1	1.335739	8.727474	2	3.921026	7.915266	2.599278	3.176895	0.012941	...	8.000000	1.989780	0.035754	21	1.804126	3.113719	2.024285	0	0.636365	2.8
1	4	1	1.463753	7.027493	2	4.198941	9.191265	2.408972	2.447977	2.301630	...	6.728972	2.742854	0.598896	6	1.614076	2.967717	1.957825	0	1.560976	1.5
2	5	1	0.943877	5.310079	2	4.410969	5.326159	3.979592	3.928571	0.019645	...	9.333333	2.477596	0.013452	5	1.773709	3.922193	1.120468	2	0.883118	1.1
3	6	1	0.797415	8.304757	2	4.225930	11.627438	2.097700	1.987549	0.171947	...	7.018256	1.812795	0.002267	64	1.415230	2.954381	1.990847	1	1.677108	1.0
4	8	1	1.463753	7.027493	2	4.198941	8.673604	2.408972	2.447977	0.389266	...	6.728972	2.742854	0.139864	89	1.614076	2.967717	1.799131	0	1.560976	1.5

5 rows × 133 columns

29.

In [40]:
```
colnames = df1.columns[2:]
```

In [41]:
```
colnames
```

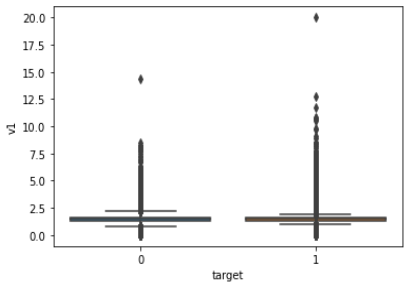
Out[41]:
```
Index(['v1', 'v2', 'v3', 'v4', 'v5', 'v6', 'v7', 'v8', 'v9', 'v10',
       ...
       'v122', 'v123', 'v124', 'v125', 'v126', 'v127', 'v128', 'v129', 'v130',
       'v131'],
      dtype='object', length=131)
```

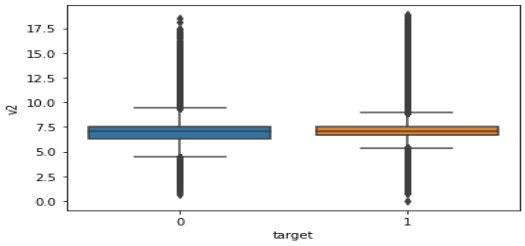
30. Outlier detection (only a few boxplots shown here out of 131 boxplots)

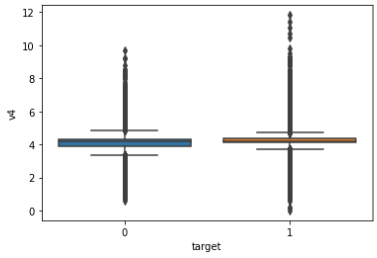
outlier detection

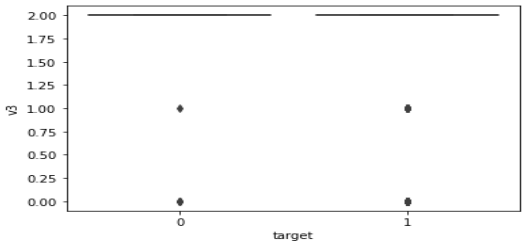
In [42]:
```
def detect(m, n, dt):
    sns.boxplot(x=m, y=n, data=dt)
    plt.show()
```

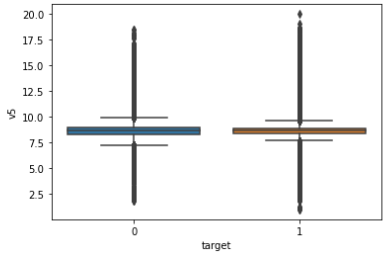
In [43]:
```
#plt.figure(figsize=(20,8))
for i in range(len(colnames)):
    detect(df1['target'], colnames[i], df1)
```

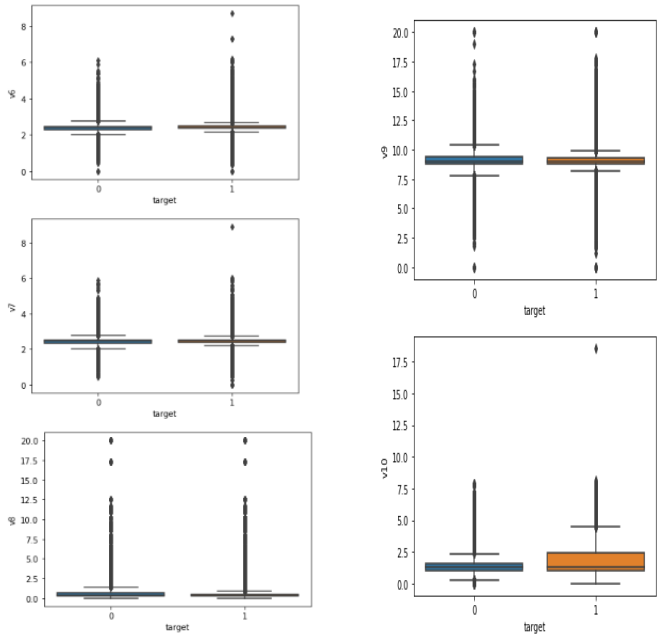










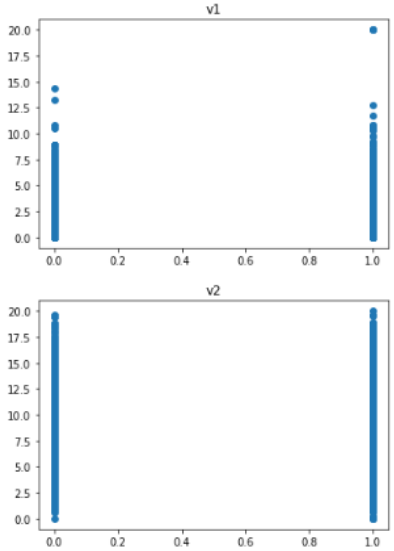


31. Scatterplot (shown here for first two plots under observation)

target

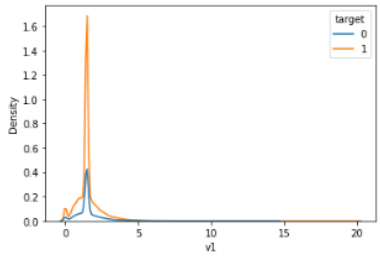
In [21]:
```
# scatter plot
def scatter(m, n, dt):
    plt.scatter(x=m, y=n, data=dt)
    plt.show()
```

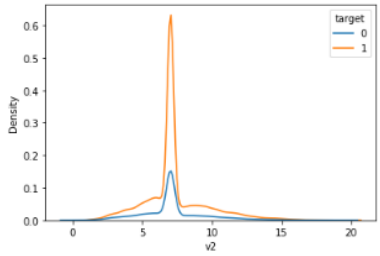
In [22]:
```
for i in range(len(colnames)):
    plt.title("{}".format(colnames[i]))
    scatter(df['target'], colnames[i], df)
```



32. KDEplot (shown here for first two columns under observation)

In [28]:
```
for i in range(len(colnames)):
    sns.kdeplot(data=df, x=colnames[i], hue="target")
    plt.show()
```





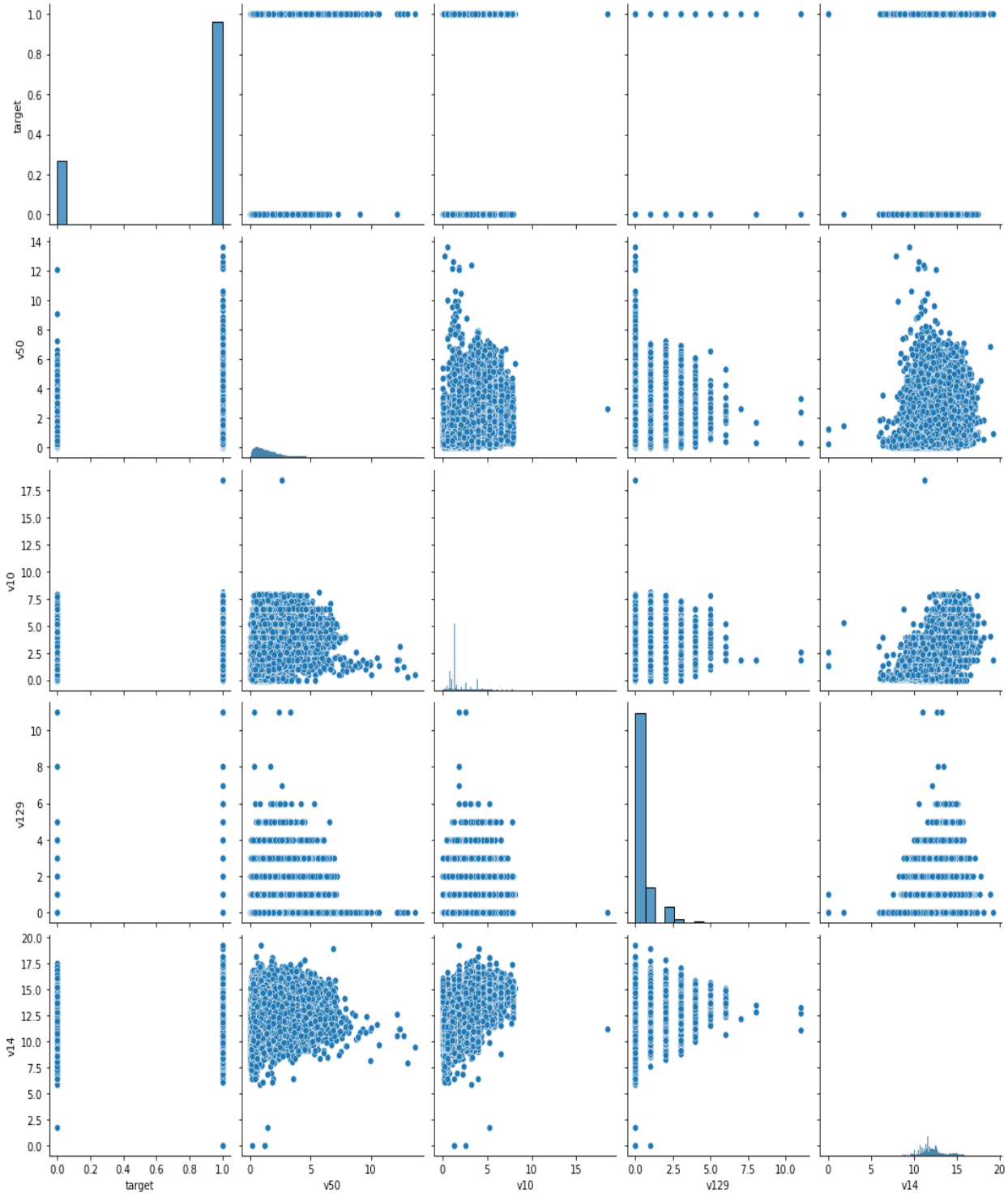
33.

In [44]:
```
k=50
cols=corrM.nlargest(k,'target')['target'].index
cols
```

Out[44]:
```
Index(['target', 'v50', 'v10', 'v129', 'v14', 'v34', 'v72', 'v114', 'v21',
       'v38', 'v12', 'v4', 'v106', 'v61', 'v101', 'v93', 'v64', 'v88', 'v65',
       'v44', 'v17', 'v100', 'v6', 'v76', 'v59', 'v2', 'v99', 'v20', 'v87',
       'v29', 'v60', 'v7', 'v128', 'v96', 'v57', 'v108', 'v26', 'v109', 'v43',
       'v27', 'v131', 'v5', 'v98', 'v116', 'v39', 'v115', 'v41', 'v67', 'v117',
       'v97'],
      dtype='object')
```

In [45]:
```
pplot=cols[:5]
sns.pairplot(df1[pplot],height=3)
plt.show() # take 5 columns
```

34.



35. Modeling Logistic regression

In [66]:
```
X = df1.drop('target',axis = 1)
y = df1.target # Target variable
```

In [67]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
```

In [68]:
```
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

In [69]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.75992
Precision: 0.763786541420358
Recall: 0.9911727616645649
```

36. Random forest

Random forest

df1

In [72]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

In [73]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.77272
Precision: 0.7856466741244299
Recall: 0.9630801687763713
f1 Score: 0.8653618311928345
AUC: 0.5691228658451393
```

37. Decision tree

decision tree

In [74]:
```
from sklearn.tree import DecisionTreeClassifier
dct = DecisionTreeClassifier()
```

In [75]:
```
dct.fit(X_train,y_train)
```

Out[75]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [76]:
```
y_pred = dct.predict(X_test)
```

In [77]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.68384
Precision: 0.798681651177869
Recall: 0.7796413502109705
f1 Score: 0.7890466531440162
AUC: 0.5813769664962137
```

38. kNN

kNN

In [78]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

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

In [79]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Out[79]:
```
KNeighborsClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [80]:
```
y_pred = classifier.predict(X_test)
```

In [82]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.72928
Precision: 0.7770405380839848
Recall: 0.9017932489451477
f1 Score: 0.8347817595937896
AUC: 0.5447707966580043
```

39. Naive Bayes

naive bayes

In [83]:
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train,y_train)
```

Out[83]:
```
GaussianNB()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [84]:
```
y_pred =naive_bayes.predict(X_test)
```

In [85]:
```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1 Score:", metrics.f1_score(y_test, y_pred))
print("AUC:", metrics.roc_auc_score(y_test, y_pred))
```

```
Accuracy: 0.55976
Precision: 0.85451952219647
Recall: 0.505590717299578
f1 Score: 0.6352972363973756
AUC: 0.6176960209014446
```

40. Grid Search

In [169…
```
from sklearn.model_selection import GridSearchCV , cross_val_score
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_score, roc_auc_score,roc_curve,auc,make_scorer
```

In [170…
```
precision_scorer=make_scorer(precision_score,zero_division=0)
```

41.

on logreg

In [171…
```
logmodel_params = {'C': [0.001, 0.01, 0.1, 1, 10], 'class_weight': [None, 'balanced'], 'penalty': ['l1', 'l2']}
```

In [172…
```
grid_search_log = GridSearchCV(estimator=logreg,param_grid=logmodel_params,scoring=precision_scorer,cv=5,n_jobs=-1,return_train_score=True)
```

In [173…
```
grid_search_log = grid_search_log.fit(X_train,y_train)
```

42.

In [174…
```
best_precision = grid_search_log.best_score_
```

In [175…
```
print('Precision on Cross Validation set :',best_precision)
```

```
Recall on Cross Validation set : 0.8266108366215359
```

In [176…
```
best_parameters = grid_search_log.best_params_
best_parameters
```

Out[176…
```
{'C': 1, 'class_weight': 'balanced', 'penalty': 'l2'}
```

In [177…
```
bestmodel=grid_search_log.best_estimator_
```

43.

In [178…
```
train_pred=bestmodel.predict(X_train)
test_pred=bestmodel.predict(X_test)
```

In [179…
```
# on train data
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [180…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.6000355076685671
Accuracy: 0.58448
Precison: 0.8308665918138206
Recall: 0.5702676194479473
F1 Score: 0.6763325163059284
```

In [181…
```
# on test data
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

44.

In [182…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5884338864303045
Accuracy: 0.57528
Precison: 0.8229694457239367
Recall: 0.5632618747372846
F1 Score: 0.6687878220724937
```

45.

grid search on decision tree

In [197…
```
parameters = {'max_depth':[10,20,30,40,50,60,70,80,90,100,500,1000]}
grid_search_dt = GridSearchCV(estimator=dct,param_grid=parameters,scoring = precision_scorer,cv=5,n_jobs=-1)
grid_search_dt = grid_search_dt.fit(X_train,y_train)
```

In [198…
```
best_precision = grid_search_dt.best_score_
```

In [200…
```
print('Precision on Cross Validation set :',best_precision)
```

```
Recall on Cross Validation set : 0.8266108366215359
```

In [201…
```
best_parameters = grid_search_dt.best_params_
best_parameters
```

Out[201…
```
{'max_depth': 90}
```

In [202…
```
bestmodel=grid_search_dt.best_estimator_
```

46.

In [204…
```
train_pred=bestmodel.predict(X_train)
test_pred=bestmodel.predict(X_test)
```

In [205…
```
## train data
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [140…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.6305279410886364
Accuracy: 0.8125066666666667
Precison: 0.8130474001222102
Recall: 0.9787725935266919
F1 Score: 0.8882460462528807
```

In [206…
```
# test data
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

47.

In [207…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5767619260125743
Accuracy: 0.68272
Precison: 0.7988369588628043
Recall: 0.7795292139554435
F1 Score: 0.789064993085842
```

48.

grid search on random forest

In [209…
```
param_rf = {'n_estimators': [10,50,100,200,500]}
grid_search_rf = GridSearchCV(estimator=clf,param_grid=param_rf,scoring=precision_scorer,cv=5,n_jobs=-1)
grid_search_rf = grid_search_rf.fit(X_train,y_train)
```

In [210…
```
best_precision = grid_search_rf.best_score_
```

In [212…
```
print('Precision on Cross Validation set :',best_recall)
```

```
Precision on Cross Validation set : 0.8005613611853223
```

In [213…
```
best_parameters = grid_search_rf.best_params_
best_parameters
```

Out[213…
```
{'n_estimators': 10}
```

In [214…
```
bestmodel=grid_search_rf.best_estimator_
```

In [215…
```
train_pred=bestmodel.predict(X_train)
test_pred=bestmodel.predict(X_test)
```

In [216…
```
# test data
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

49.

In [217…

```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5870781903090829
Accuracy: 0.73888
Precison: 0.7991387559808613
Recall: 0.8775746111811685
F1 Score: 0.8365220875488331
```

In [218…

```
# train data
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [219…

```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.9851963730902772
Accuracy: 0.99112
Precison: 0.9918418575462817
Recall: 0.9965321563682219
F1 Score: 0.9941814750746972
```

50.

RFE

RFE

on logistic regression

In [159…

```
X = df1.drop('target',axis = 1)
y = df1.target # Target variable
```

In [160…

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
```

51.

In [183…
```
list(X_train.columns)
```

Out[183…
```
['ID',
 'v1',
 'v2',
 'v3',
 'v4',
 'v5',
 'v6',
 'v7',
 'v8',
 'v9',
 'v10',
 'v11',
 'v12',
 'v13',
 'v14',
 'v15',
 'v16',
 'v17',
 'v18',
 'v19',
 'v20',
 'v21',
 'v22',
 'v23',
 'v24',
 'v25',
 'v26',
 'v27',
 'v28',
 'v29',
 'v30',
 'v31',
 'v32',
```

52.

In [184…
```
from sklearn import feature_selection
```

In [185…
```
log_model_rfe = LogisticRegression()
```

In [186…
```
rfeobj=feature_selection.RFE(estimator=log_model_rfe,n_features_to_select=20)
```

53.

In [187…
```
rfeobj.fit(X_train,y_train)
```

54.

In [188…
```
rfeobj.support_
```

Out[188…
```
array([False, False, False, False, False, False,  True, False, False,
       False,  True, False,  True, False, False, False, False, False,
       False, False, False,  True, False, False,  True, False, False,
       False, False, False, False,  True, False, False,  True, False,
       False, False,  True, False, False, False, False, False, False,
       False, False,  True, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False,  True, False, False, False, False, False,
        True,  True,  True, False, False, False, False, False,  True,
       False, False, False, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False,  True, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False,  True, False, False])
```

In [189…
```
X_train.columns[rfeobj.support_]
```

Out[189…
```
Index(['v6', 'v10', 'v12', 'v21', 'v24', 'v31', 'v34', 'v38', 'v47', 'v50',
       'v62', 'v66', 'v72', 'v73', 'v74', 'v80', 'v85', 'v110', 'v126',
       'v129'],
      dtype='object')
```

In [190…
```
selected=list(X_train.columns[rfeobj.support_])
```

In [191…
```
X_train1=X_train[(X_train.columns[rfeobj.support_])]
X_test1 =X_test[(X_train.columns[rfeobj.support_])]
```

55.

In [192…
```
log_model_rfe = LogisticRegression()
log_model_rfe.fit(X_train1,y_train)

train_pred=log_model_rfe.predict(X_train1)
test_pred=log_model_rfe.predict(X_test1)
```

In [193…
```
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [194…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5296184057254025
Accuracy: 0.7665066666666667
Precison: 0.772391962565373
Recall: 0.9829410116295362
F1 Score: 0.865038996269922
```

56.

F1 Score: 0.8659589902699322

In [195…
```
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

In [196…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5313475146134575
Accuracy: 0.76768
Precison: 0.7730426164519326
Recall: 0.9836065573770492
F1 Score: 0.8657047724750278
```

57. On random forest

In [104…
```
rf_model_rfe = RandomForestClassifier()
```

In [105…
```
rfeobj=feature_selection.RFE(estimator=rf_model_rfe,n_features_to_select=20)
```

In [106…
```
rfeobj.fit(X_train,y_train)
```

Out[106…
```
RFE(estimator=RandomForestClassifier(), n_features_to_select=20)
```

In [107…
```
rfeobj.support_
```

Out[107…
```
array([ True, False, False, False, False, False,  True, False, False,
       False,  True,  True,  True, False,  True, False, False, False,
       False, False, False,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
        True, False, False, False,  True, False, False, False, False,
       False, False, False, False, False,  True, False,  True, False,
        True, False,  True, False, False, False, False, False, False,
       False, False, False, False, False,  True, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False])
```

In [108…
```
X_train.columns[rfeobj.support_]
```

Out[108…
```
Index(['ID', 'v6', 'v10', 'v11', 'v12', 'v14', 'v21', 'v22', 'v34', 'v36',
       'v40', 'v50', 'v52', 'v54', 'v56', 'v68', 'v98', 'v112', 'v114',
       'v125'],
      dtype='object')
```

In [110…
```
selected=list(X_train.columns[rfeobj.support_])
```

In [111…
```
X_train1=X_train[ list(X_train.columns[rfeobj.support_]) ]
X_test1=X_test[list(X_train.columns[rfeobj.support_])]
```

58. On Decision Tree

In [112…
```
rf_model_rfe = DecisionTreeClassifier()
rf_model_rfe.fit(X_train1,y_train)

train_pred=rf_model_rfe.predict(X_train1)
test_pred=rf_model_rfe.predict(X_test1)
```

In [113…
```
# train data
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [116…
```
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

In [117…
```
# test data
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

In [118…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5582024545650222
Accuracy: 0.6732
Precison: 0.789468073766123
Recall: 0.7782681799075242
F1 Score: 0.7838281208657459
```

59.

on decision tree

In [88]:
```
list(X_train.columns)
```

Out[88]:
```
['ID',
 'v1',
 'v2',
 'v3',
 'v4',
 'v5',
 'v6',
 'v7',
 'v8',
 'v9',
 'v10',
```

60.

In [89]:
```
dt_model_rfe = DecisionTreeClassifier()
```

In [90]:
```
rfeobj=feature_selection.RFE(estimator=dt_model_rfe,n_features_to_select=20)
```

In [91]:
```
rfeobj.fit(X_train,y_train)
```

Out[91]:
```
RFE(estimator=DecisionTreeClassifier(), n_features_to_select=20)
```

In [92]:
```
rfeobj.support_
```

Out[92]:
```
array([ True, False, False, False, False, False,  True, False, False,
       False, False, False,  True, False,  True, False,  True, False,
       False, False, False,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False,  True, False, False, False, False,
       False, False,  True, False, False,  True, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False,  True, False,  True,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False, False,
       False, False, False, False,  True, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False])
```

61.

In [93]:
```
X_train.columns[rfeobj.support_]
```

Out[93]:
```
Index(['ID', 'v6', 'v12', 'v14', 'v16', 'v21', 'v22', 'v34', 'v40', 'v47',
       'v50', 'v52', 'v55', 'v66', 'v87', 'v89', 'v99', 'v112', 'v114',
       'v125'],
      dtype='object')
```

In [94]:
```
selected=list(X_train.columns[rfeobj.support_])
```

In [95]:
```
X_train1=X_train[list(X_train.columns[rfeobj.support_])]
X_test1=X_test[list(X_train.columns[rfeobj.support_])]
```

In [96]:
```
dt_model_rfe = DecisionTreeClassifier()
dt_model_rfe.fit(X_train1,y_train)

train_pred=dt_model_rfe.predict(X_train1)
test_pred=dt_model_rfe.predict(X_test1)
```

In [97]:
```
# train data
roc=roc_auc_score(y_train, train_pred)
acc = accuracy_score(y_train, train_pred)
prec = precision_score(y_train, train_pred)
rec = recall_score(y_train, train_pred)
f1 = f1_score(y_train, train_pred)
```

62.

In [98]:
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  1.0
Accuracy: 1.0
Precison: 1.0
Recall: 1.0
F1 Score: 1.0
```

In [99]:
```
# test data
roc=roc_auc_score(y_test, test_pred)
acc = accuracy_score(y_test, test_pred)
prec = precision_score(y_test, test_pred)
rec = recall_score(y_test, test_pred)
f1 = f1_score(y_test, test_pred)
```

In [100…
```
print('ROC: ',roc)
print('Accuracy:',acc)
print('Precison:',prec)
print('Recall:',rec)
print('F1 Score:',f1)
```

```
ROC:  0.5682647371380725
Accuracy: 0.68064
Precison: 0.7943307757885762
Recall: 0.7833123160992014
F1 Score: 0.7887830687830688
```

After RFE

Decision Tree

ROC: 0.5682647371380725

Accuracy: 0.68064

Precision: 0.7943307757885762

Recall: 0.7833123160992014

F1 Score: 0.7887830687830688

Random Forest

ROC: 0.5582024545650222

Accuracy: 0.6732

Precision: 0.789468073766123

Recall: 0.7782681799075242

F1 Score: 0.7838281208657459

Final Model -

After recursive feature elimination we find that the **decision tree** gives the **best precision and higher AUC score**.

Future scope of improvement:-

This model would have been better if it was built on reinforcement learning or Neural Networks. The data could have been balanced by generation of data by ‘smot’. A larger dataset might have provided a better result if some of the outliers could have been removed.

9 Project Certificates

9.1 Certificate

This is to certify that **Mr Subhankar Bose** of Kalyani Government Engineering College, roll number: 10200319021, has successfully completed a project on Prediction of Claims Management using Python under the guidance of **Mr. Titas Roy Chowdhury.**

_____

Mr. Titas Roy Chowdhury

Globsyn Finishing School

9 Project Certificates

9.2 Certificate

This is to certify that **Mr Dibyanshu Shekhar Dey** of Kalyani Government Engineering College, roll number:10201619054, has successfully completed a project on Prediction of Claims Management using Python under the guidance of **Mr. Titas Roy Chowdhury**.

_____

Mr. Titas Roy Chowdhury

Globsyn Finishing School

9 Project Certificates

9.3 Certificate

This is to certify that **Mr Arghya Paul** of Kalyani Government Engineering College, roll number: 10200319015 , has successfully completed a project on Prediction of Claims Management using Python under the guidance of **Mr. Titas Roy Chowdhury**.

_____

Mr. Titas Roy Chowdhury

Globsyn Finishing School

9 Project Certificates

9.4 Certificate

This is to certify that **Ms Sahin Khatun** of Kalyani Government Engineering College, roll number:10200119053, has successfully completed a project on Prediction of Claims Management using Python under the guidance of **Mr. Titas Roy Chowdhury**.

_____

Mr. Titas Roy Chowdhury

Globsyn Finishing School

9 Project Certificates

9.5 Certificate

This is to certify that **Ms Aarati Shah** of Kalyani Government Engineering College, roll number: 10200119051, has successfully completed a project on Prediction of Claims Management using Python under the guidance of **Mr. Titas Roy Chowdhury**.

_____

Mr. Titas Roy Chowdhury

Globsyn Finishing School