In [3]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
```

In [7]:

```python
data = pd.read_csv("kaggle_credit.csv.zip")
```

In [8]:

```python
data
```

Out[8]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078

284807 rows × 31 columns

In [9]:

```python
data.info
```

Out[9]:

```
<bound method DataFrame.info of             Time         V1         V2        V3        V4        V
\
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
```

```
1       -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2        1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3        1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4        0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
...           ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...       ...       ...       ...       ...       ...     ...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]>
```

In [10]:

```
data.shape
```

Out[10]:

```
(284807, 31)
```

In [11]:

```
data.describe()
```

Out[11]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+

8 rows × 31 columns

In [13]:

```
########## determine the number of fraud case in the data set #############

fraud = data[data['Class']==1]
valid = data[data['Class']==0]
outlierfraction  = len(fraud)/float(len(valid))
print(outlierfraction)
print('Fraud Cases:{}'.format(len(data[data['Class']==1])))
print('Valid Transactions:{}'.format(len(data[data['Class']==0])))
```

```
0.0017304750013189597
Fraud Cases:492
Valid Transactions:284315
```

In [ ]:

```
# Only 0.17% of all transactions are fraudulent. The data is highly imbalanced.
# Let's apply our unbalanced models first, and if we don't get good accuracy, we can find a way to
balance this dataset.
```

In [14]:

```
print('Amount details of Fraud Transaction')
fraud.Amount.describe()    ######## we can see in this avarage monetary transaction there is more f
raudulent transactions ########
```

```
Amount details of Fraud Transaction
```

Out[14]:

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

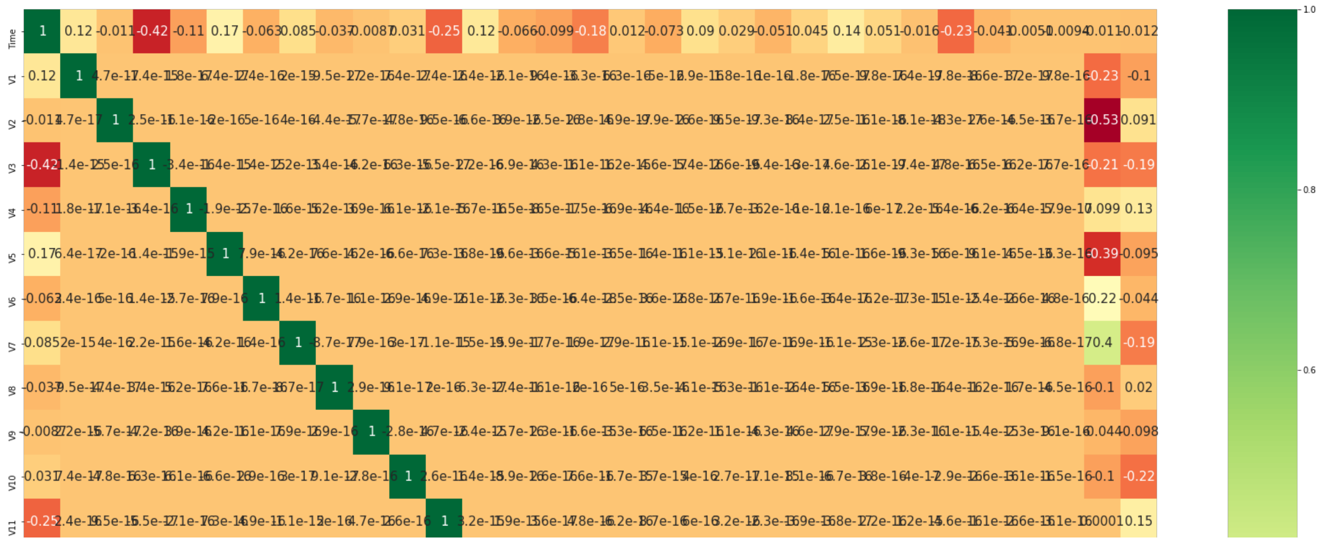
In [16]:

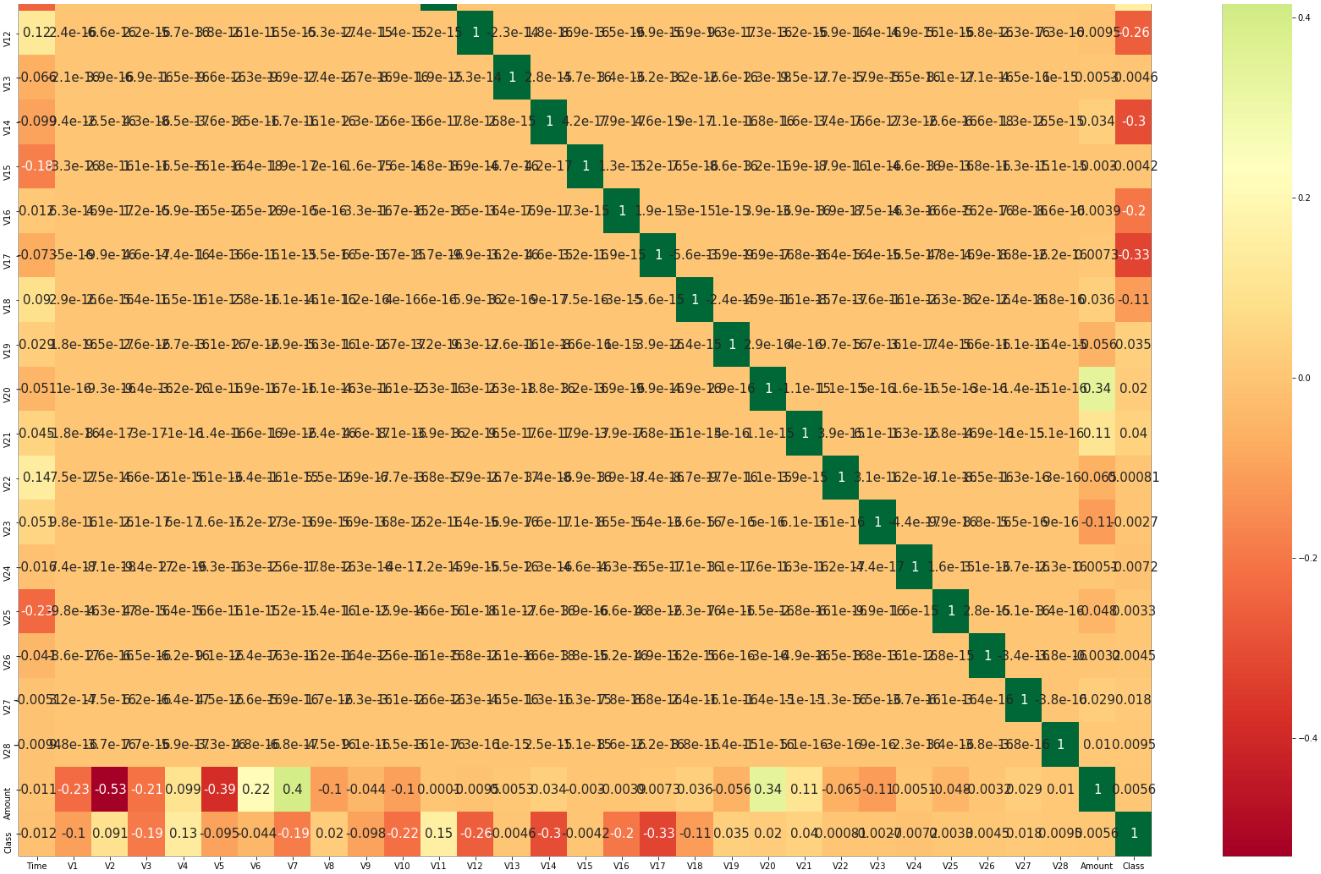
```
######## plotting correlation matrix ###########

plt.figure(figsize=(30,30))
sns.heatmap(data.corr(),annot=True,cmap="RdYlGn",annot_kws={"size":15})
```

Out[16]:

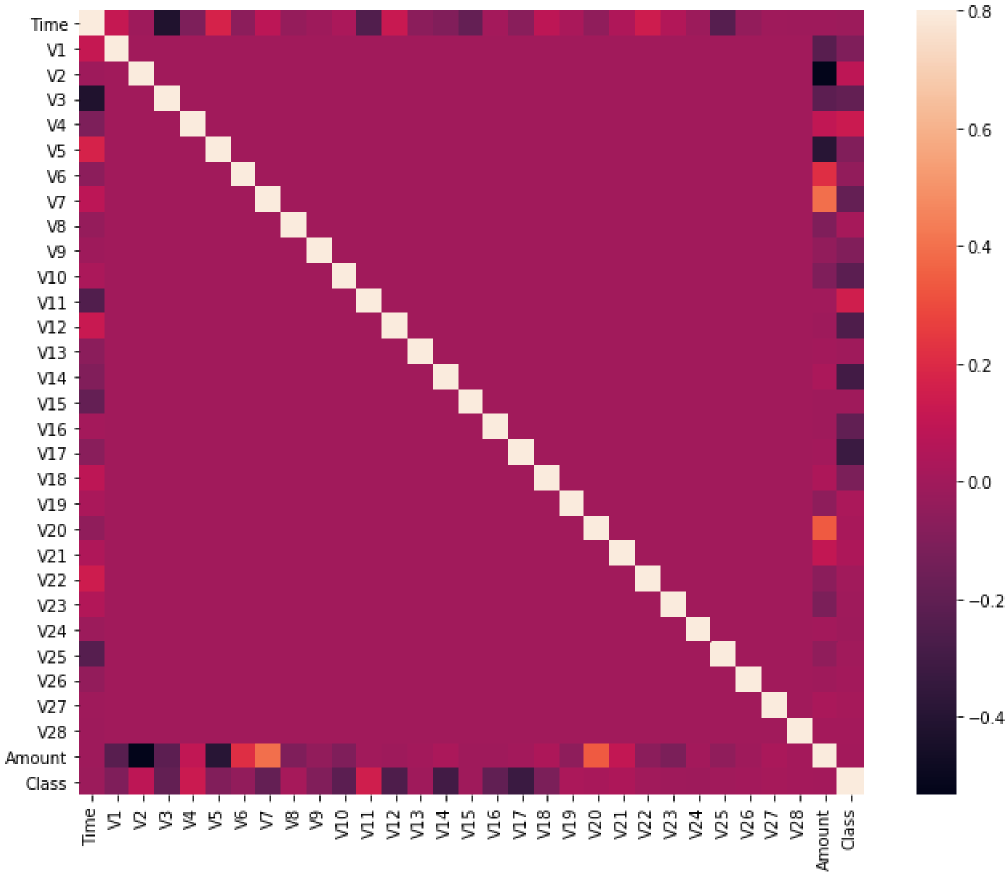
```
<matplotlib.axes._subplots.AxesSubplot at 0x288ed486148>
```





In [17]:

```
corrmat = data.corr()
fig = plt.figure(figsize=(12,9))
sns.heatmap(corrmat,vmax = .8, square = True)
plt.show()
```



In [ ]:

In [ ]:

```
# In the heat map, we can clearly see that most of the features are not correlated with other feat
ures, but there are some features that are positively or negatively correlated with each other. Fo
r example, V2 and V5 are strongly negatively correlated with the Amount function.
# We also see some correlation with the V20 and Amount.
```

In [23]:

```
######### dividing X and Y from the dataset ###########

X= data.drop(['Class'],axis = 1)
Y =data['Class']
print(X.shape)
print(Y.shape)
xData = X.values
ydata = Y.values
```

```
(284807, 30)
(284807,)
```

In [26]:

```
####### using skitlearn to split data into training and testing ##############

from sklearn.model_selection import train_test_split
xTrain,xTest,yTrain,yTest = train_test_split(xData,ydata,test_size = 0.2, random_state = 42)
```

In [28]:

```
######## using Random Forest classifier ##############

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(xTrain,yTrain)
ypred = rfc.predict(xTest)
```

In [32]:

```
########## evaluating each classifier ############

from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score,matthews_corrcoef
from sklearn.metrics import confusion_matrix
```

In [43]:

```
n_outliers = len(fraud)
n_errors = (ypred != yTest).sum()
print('The model used in Random Forest Classifier')

acc = accuracy_score(yTest,ypred)
print('accuracy_score {}'.format(acc))

prec = precision_score(yTest,ypred)     ##### Precision quantifies the number of positive class
predictions that actually belong to the positive class(correctly predicted positive observation to
the total predicted positive observation.
print('precision_score {}'.format(prec))

rec = recall_score(yTest, yPred)    ##### Recall quantifies the number of positive class
predictions made out of all positive examples in the dataset. true positives divided by the total
number of true positives and false negatives.
print('recall_score {}'.format(rec))

f1 = f1_score(yTest,ypred)          ##### F-Measure provides a single score that balances both the
concerns of precision and recall in one number.
print('f1_score {}'.format(f1))

MCC = matthews_corrcoef(yTest,ypred)    ######## as a measure of the quality of binary (two-class)
classifications,
print('matthews_corrcoef {}'.format(MCC))
```

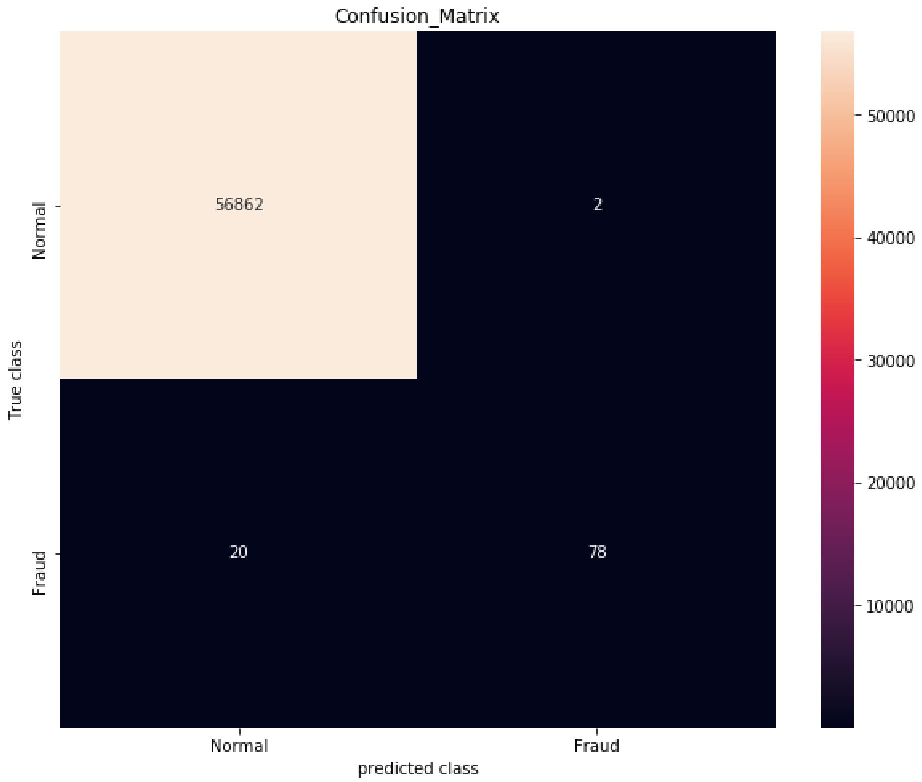
```
The model used in Random Forest Classifier
accuracy_score 0.9996137776061234
precision_score 0.975
recall_score 0.7959183673469388
f1_score 0.8764044943820225
matthews_corrcoef 0.8807418913871203
```

In [37]:

```python
########### confusion matrix ##########

LABELS = ['Normal','Fraud']
conf_matrix = confusion_matrix(yTest,ypred)

plt.figure(figsize = (10,8))
sns.heatmap(conf_matrix,xticklabels=LABELS,yticklabels=LABELS, annot=True, fmt='d');
plt.title('Confusion_Matrix')
plt.xlabel('predicted class')
plt.ylabel('True class')
plt.show()
```



In [38]:

```python
# Let's check the rest of the algorithms
# Importing Libraries
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn import metrics
```

In [42]:

```python
###### decision tree #######

# Decision Tree
```

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(xTrain, yTrain)
# predictions
yPred = decision_tree.predict(xTest)
acc_decision_tree = round(decision_tree.score(xTrain,yTrain)*100,2)
acc_decision_tree
```

Out[42]:

```
100.0
```

In [44]:

```
n_outliers2 = len(fraud)
n_errors2 = (ypred != yTest).sum()
print('Decision Tree')

acc2 = accuracy_score(yTest,ypred)
print('accuracy_score {}'.format(acc))

prec2 = precision_score(yTest,ypred)    ##### Precision quantifies the number of positive class predictions that actually belong to the positive class(correctly predicted positive observation to the total predicted positive observation.
print('precision_score {}'.format(prec))

rec2 = recall_score(yTest, yPred)   ##### Recall quantifies the number of positive class predictions made out of all positive examples in the dataset. true positives divided by the total number of true positives and false negatives.
print('recall_score {}'.format(rec))

f1 = f1_score(yTest,ypred)         ##### F-Measure provides a single score that balances both the concerns of precision and recall in one number.
print('f1_score {}'.format(f1))

MCC2 = matthews_corrcoef(yTest,ypred)   ######## as a measure of the quality of binary (two-class) classifications,
print('matthews_corrcoef {}'.format(MCC))
```

```
Decision Tree
accuracy_score 0.9996137776061234
precision_score 0.975
recall_score 0.7959183673469388
f1_score 0.8764044943820225
matthews_corrcoef 0.8807418913871203
```

In [46]:

```
########### Random Forest ############

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(xTrain, yTrain)
y_pred = random_forest.predict(xTest)
random_forest.score(xTrain, yTrain)
acc_random_forest = round(random_forest.score(xTrain, yTrain) * 100, 2)
acc_random_forest
```

Out[46]:

```
100.0
```

In [47]:

```
# Gaussian Naive Bayes
gaussian = GaussianNB()
gaussian.fit(xTrain, yTrain)
y_pred = gaussian.predict(xTest)
acc_gaussian = round(gaussian.score(xTrain, yTrain) * 100, 2)
acc_gaussian
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

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

```
           0       1.00      0.99      1.00     56864
           1       0.15      0.63      0.24        98

    accuracy                           0.99     56962
   macro avg       0.57      0.81      0.62     56962
weighted avg       1.00      0.99      1.00     56962

accuracy: 0.9930128857835048
```

In [48]:

```
# Perceptron
perceptron = Perceptron()
perceptron.fit(xTrain, yTrain)
y_pred = perceptron.predict(xTest)
acc_perceptron = round(perceptron.score(xTrain, yTrain) * 100, 2)
acc_perceptron
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9981917769741231
```

In [49]:

```
# Linear SVC
linear_svc = LinearSVC()
linear_svc.fit(xTrain, yTrain)
y_pred = linear_svc.predict(xTest)
acc_linear_svc = round(linear_svc.score(xTrain, yTrain) * 100, 2)
acc_linear_svc
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

```
C:\Users\Dipsikha\anaconda3\lib\site-packages\sklearn\svm\_base.py:977: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.73      0.19      0.31        98

    accuracy                           1.00     56962
   macro avg       0.86      0.60      0.65     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9984902215512096
```

In [50]:

```
# Stochastic Gradient Descent
sgd = SGDClassifier()
sgd.fit(xTrain, yTrain)
y_pred = sgd.predict(xTest)
acc_sgd = round(sgd.score(xTrain, yTrain) * 100, 2)
acc_sgd
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

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

```
           0       1.00      1.00      1.00     56864
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9981917769741231
```

In [51]:

```python
# K Nearest Neighbor:
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(xTrain, yTrain)
y_pred = knn.predict(xTest)
acc_knn = round(knn.score(xTrain, yTrain) * 100, 2)
acc_knn
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       1.00      0.09      0.17        98

    accuracy                           1.00     56962
   macro avg       1.00      0.55      0.58     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9984375548611355
```

In [52]:

```python
# Support Vector Machines
svc = SVC()
svc.fit(xTrain, yTrain)
y_pred = svc.predict(xTest)
acc_svc = round(svc.score(xTrain, yTrain) * 100, 2)
acc_svc
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9982795547909132
```

```
C:\Users\Dipsikha\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [53]:

```python
# AdaBoost
from sklearn.ensemble import AdaBoostClassifier
Ada = AdaBoostClassifier(random_state=1)
Ada.fit(xTrain, yTrain)
y_pred = Ada.predict(xTest)
acc_add = round(Ada.score(xTrain, yTrain) * 100, 2)
acc_add
print(classification_report(yTest, y_pred))
print("accuracy:",metrics.accuracy_score(yTest, y_pred))
```

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

```
           0       1.00      1.00      1.00     56864
           1       0.86      0.72      0.78        98

    accuracy                           1.00     56962
   macro avg       0.93      0.86      0.89     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9993153330290369
```

In [54]:

```
import xgboost as xgb
XGB = xgb.XGBClassifier(random_state = 1)
XGB.fit(xTrain,yTrain)
y_pred = XGB.predict(xTest)
acc_XGB = round(XGB.score(xTrain,yTrain)*100,2)
acc_XGB
print(classification_report(yTest,y_pred))
print("accuracy:",metrics.accuracy_score(yTest,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.99      0.80      0.88        98

    accuracy                           1.00     56962
   macro avg       0.99      0.90      0.94     56962
weighted avg       1.00      1.00      1.00     56962

accuracy: 0.9996313331694814
```

In [55]:

```
####### collecting all the model in a table ##########

Result  = pd.DataFrame({
    'MODEL':['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest','Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent','Decision Tree', 'AdaBoost', 'XGBoost'],
    'SCORE':[acc_linear_svc, acc_knn, acc_sgd,acc_random_forest,
              acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree, acc_XGB, acc_add]})
Result = Result.sort_values(by = 'SCORE', ascending = False)
Result = Result.set_index('SCORE')
Result.head(10)
```

Out[55]:

	MODEL
SCORE	
100.00	Random Forest
100.00	Decision Tree
100.00	AdaBoost
99.93	XGBoost
99.86	KNN
99.85	Support Vector Machines
99.82	Logistic Regression
99.82	Perceptron
99.82	Stochastic Gradient Decent
99.35	Naive Bayes

In [ ]: