In [40]:
```
import numpy as np
import pandas as pd
```

In [41]:
```
df = pd.read_csv("/Users/dev/Personal/DS & AI Class Notes/Data Sets/Adaboos
```

Checking Balance of DF

In [42]:
```
df["Class"].value_counts()
```

Out[42]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [43]:
```
dfone = df[df["Class"] ==1]
```

In [44]:
```
dfzero = df[df["Class"] == 0].sample(492)
```

In [45]:
```
dfzero.shape,dfone.shape
```

Out[45]:
```
((492, 31), (492, 31))
```

In [49]:
```
df1 = pd.concat([dfone,dfzero])
```

In [55]:
```
df1 = df1.sample(frac=1)
```

In [64]:
```
df1["Class"].value_counts()
```

Out[64]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [65]:
```
def checkz(df):
    return df[df == 0].value_counts()
```

In [66]:
```
for i in df1.columns:
    print(checkz(df[i]))
```

```
0.0    2
Name: Time, dtype: int64
Series([], Name: V1, dtype: int64)
Series([], Name: V2, dtype: int64)
Series([], Name: V3, dtype: int64)
Series([], Name: V4, dtype: int64)
Series([], Name: V5, dtype: int64)
Series([], Name: V6, dtype: int64)
Series([], Name: V7, dtype: int64)
Series([], Name: V8, dtype: int64)
Series([], Name: V9, dtype: int64)
Series([], Name: V10, dtype: int64)
Series([], Name: V11, dtype: int64)
Series([], Name: V12, dtype: int64)
Series([], Name: V13, dtype: int64)
Series([], Name: V14, dtype: int64)
Series([], Name: V15, dtype: int64)
Series([], Name: V16, dtype: int64)
Series([], Name: V17, dtype: int64)
Series([], Name: V18, dtype: int64)
Series([], Name: V19, dtype: int64)
Series([], Name: V20, dtype: int64)
Series([], Name: V21, dtype: int64)
Series([], Name: V22, dtype: int64)
Series([], Name: V23, dtype: int64)
Series([], Name: V24, dtype: int64)
Series([], Name: V25, dtype: int64)
Series([], Name: V26, dtype: int64)
Series([], Name: V27, dtype: int64)
Series([], Name: V28, dtype: int64)
0.0    1825
Name: Amount, dtype: int64
0    284315
Name: Class, dtype: int64
```

In [68]:

```
df.columns
```

Out[68]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20'
,
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [69]:

```
clist = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V1(
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20'
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']
```

In [61]:

```
def zeroremove(df):
    m = round(df.mean(),2)
    df.replace(0,m,inplace = True)
```

In [62]:
```
for i in clist:
    zeroremove(df[i])
```

In [63]:
```
for i in df.columns:
    print(checkz(df[i]))
```

```
Series([], Name: Time, dtype: int64)
Series([], Name: V1, dtype: int64)
Series([], Name: V2, dtype: int64)
Series([], Name: V3, dtype: int64)
Series([], Name: V4, dtype: int64)
Series([], Name: V5, dtype: int64)
Series([], Name: V6, dtype: int64)
Series([], Name: V7, dtype: int64)
Series([], Name: V8, dtype: int64)
Series([], Name: V9, dtype: int64)
Series([], Name: V10, dtype: int64)
Series([], Name: V11, dtype: int64)
Series([], Name: V12, dtype: int64)
Series([], Name: V13, dtype: int64)
Series([], Name: V14, dtype: int64)
Series([], Name: V15, dtype: int64)
Series([], Name: V16, dtype: int64)
Series([], Name: V17, dtype: int64)
Series([], Name: V18, dtype: int64)
Series([], Name: V19, dtype: int64)
Series([], Name: V20, dtype: int64)
Series([], Name: V21, dtype: int64)
Series([], Name: V22, dtype: int64)
Series([], Name: V23, dtype: int64)
Series([], Name: V24, dtype: int64)
Series([], Name: V25, dtype: int64)
Series([], Name: V26, dtype: int64)
Series([], Name: V27, dtype: int64)
Series([], Name: V28, dtype: int64)
Series([], Name: Amount, dtype: int64)
0    284315
Name: Class, dtype: int64
```

In [70]:
```
def odiqr(df):
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    iqr = q3 - q1
    low = q1 - (1.5 * iqr)
    high = q3 + (1.5 * iqr)
    m = df.mean()
    df = df.apply(lambda x : m if x < low else (m if x > high else x ) )
    return df
```

In [75]:
```
def odmsd(df):
    m = round(df.mean(),2)
    s = round(df.std(),2)
    low = round(m-(3*s),2)
    high = round(m+(3*s),2)
    ft1 = df[df<low]
    ft2 = df[df>high]
    df = df.map(lambda x : low if x < low else (high if x > high else x ))
```

In [76]:
```
for i in clist:
    print(f'{i} is {df1[i].skew()}')
```

```
Time is 0.07432341681272475
V1 is -2.6085603012232563
V2 is 1.795575448827865
V3 is -2.205329032930736
V4 is 0.8331263321632946
V5 is -2.266576127242939
V6 is 0.46307224760809645
V7 is -2.720576416278144
V8 is -3.650602841072597
V9 is -1.241290929614122
V10 is -1.6644772673459114
V11 is 1.0356462258545986
V12 is -1.3542217291030367
V13 is -0.002139172322972022
V14 is -0.9992109456055122
V15 is -0.5061684523317908
V16 is -1.3976963866402168
V17 is -1.5146368931966372
V18 is -1.3967406999682332
V19 is 0.4569547868693889
V20 is 2.6496288833049473
V21 is 3.8687825594559504
V22 is -1.5511344378832106
V23 is -6.626672076470413
V24 is -0.4198966762428677
V25 is -0.6464599374837476
V26 is 0.6042073895389056
V27 is -2.7159334283414918
V28 is -0.04083312854155775
Amount is 5.343839360229817
```

In [77]:
```
for i in df1.columns:
    if df[i].skew() >= 0.5:
        odmsd(df[i])
    else:
        df[i] = odiqr(df[i])
```

In [78]:
```
for i in clist:
    print(f'{i} is {df[i].skew()}')
```

```
Time is -0.0355676180063216
V1 is -0.37913693857688197
V2 is -0.05728061207137957
V3 is -0.26369343036312826
V4 is 0.676292097985747
V5 is 0.09409192340543848
V6 is 1.826580664998085
V7 is 2.553907417429514
V8 is 0.3636041497844991
V9 is 0.5546797719063509
V10 is 1.1871405899625278
V11 is 0.07415694830329334
V12 is -0.13988846437496277
V13 is -0.015437466125834208
V14 is -0.030474047058365247
V15 is -0.18289543364369218
V16 is -0.08353512476885572
V17 is 0.26081261454438887
V18 is 0.03988352840383541
V19 is -0.04189898726108976
V20 is 0.02690746913916284
V21 is 3.5929911930778453
V22 is 0.0014802093677941728
V23 is 0.02945252844586551
V24 is -0.4308098546898199
V25 is -0.08182636098075079
V26 is 0.5766926172084218
V27 is 0.13318294605704664
V28 is 11.19209119221281
Amount is 16.977724453761024
```

In [80]:
```
df1
```

Out[80]:

	Time	V1	V2	V3	V4	V5	V6	
26382	34033.0	-1.508404	-0.182394	0.664991	-0.716993	-1.565775	0.515343	-3.
231541	146803.0	1.837506	-0.460695	-0.429255	0.282574	-0.391341	0.012993	-0.
220634	142251.0	-0.691135	0.890081	-0.230734	-0.494018	0.396162	-1.139473	0.
93420	64410.0	-0.635049	0.984151	1.597940	-0.162115	0.028175	-0.225490	0.
9487	14073.0	-4.153014	8.204797	-15.031714	10.330100	-3.994426	-3.250013	-10.
...	...	...	...	...	...	...	...	
255934	157467.0	-1.198221	-0.994297	-0.014804	0.160180	2.266832	-2.037787	0.
154694	102622.0	-2.877176	4.569649	-9.553069	4.441079	-3.653961	-1.877981	-3.
278058	168017.0	1.957778	-0.229079	-0.896589	0.124183	-0.274162	-0.228216	-0.
163861	116258.0	-1.258358	-1.146954	1.518207	-3.264535	0.602702	1.196127	-0
183215	125703.0	1.999926	-0.368815	-0.460516	0.364296	-0.401536	-0.066027	-0

984 rows × 31 columns

In [81]:
```
X = df1.drop("Class",axis=1)
```

In [82]:
```
X.sample()
```

Out[82]:

	Time	V1	V2	V3	V4	V5	V6	V
272521	165132.0	-7.503926	-0.360628	-3.830952	2.486103	2.497367	1.332437	-6.78396

1 rows × 30 columns

In [83]:
```
y = df1["Class"]
```

In [84]:
```
y.sample()
```

Out[84]:
```
222201    0
Name: Class, dtype: int64
```

In [85]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold , cross_val_score
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier , AdaBoostClassifier
```

In [86]:
```
Xtrain,Xtest,ytrain,ytest = train_test_split(X,y,test_size=.30)
```

In [87]:
```
X.shape , Xtrain.shape , Xtest.shape
```

Out[87]:
```
((984, 30), (688, 30), (296, 30))
```

In [88]:
```
y.shape , ytrain.shape , ytest.shape
```

Out[88]:
```
((984,), (688,), (296,))
```

In [89]:
```
kf = KFold(n_splits=11)
```

In [90]:
```
dct = DecisionTreeClassifier()
```

In [91]:
```
algo = [ dct ]
```

Without Feature Scaling

In [92]:
```
for i in algo:
    i.fit(Xtrain,ytrain)
    s = i.score(Xtest,ytest)
    print(f'{i} = {s}')
```

```
DecisionTreeClassifier() = 0.8885135135135135
```

With Feature Scaling

In [93]:
```
ss = StandardScaler()
```

In [94]:
```
ss.fit(Xtrain)
```

Out[94]:
```
StandardScaler()
```

In [95]:
```
Xtrain_ss = ss.transform(Xtrain)
```

In [96]:
```
Xtest_ss = ss.transform(Xtest)
```

In [97]:
```
for i in algo:
    i.fit(Xtrain_ss,ytrain)
    s = i.score(Xtest_ss,ytest)
    print(f'{i} = {s}')
```

```
DecisionTreeClassifier() = 0.8851351351351351
```

With Cross Validation

In [98]:
```
for i in algo:
    s = cross_val_score(i,X,y,cv = kf)
    print(f'{i} = {s.mean()}')
```

```
DecisionTreeClassifier() = 0.8983770287141073
```

Boosting

In [99]:
```
rfc = RandomForestClassifier(n_estimators=150,max_depth=4,max_leaf_nodes=6)
```

In [100…
```
abc = AdaBoostClassifier()
```

In [101…
```
algo1 = [rfc  , abc ]
```

In [102…
```
for i in algo1:
    print(i)
```

```
RandomForestClassifier(max_depth=4, max_leaf_nodes=6, n_estimators=150)
AdaBoostClassifier()
```

Without Feature Scaling

In [103…
```
for i in algo1:
    i.fit(Xtrain,ytrain)
    s = i.score(Xtest,ytest)
    print(f'{i} = {s}')
```

```
RandomForestClassifier(max_depth=4, max_leaf_nodes=6, n_estimators=150) = 0
.9324324324324325
AdaBoostClassifier() = 0.9121621621621622
```

With Feature Scaling

In [104…
```
for i in algo1:
    i.fit(Xtrain_ss,ytrain)
    s = i.score(Xtest_ss,ytest)
    print(f'{i} = {s}')
```

```
RandomForestClassifier(max_depth=4, max_leaf_nodes=6, n_estimators=150) = 0
.9324324324324325
AdaBoostClassifier() = 0.9121621621621622
```

With Cross Validation (Boosting)

In [105…
```
%%time
for i in algo1:
    s = cross_val_score(i,X,y,cv = kf)
    print(f'{i} = {s.mean()}')
```

```
RandomForestClassifier(max_depth=4, max_leaf_nodes=6, n_estimators=150) = 0
.924809896720009
AdaBoostClassifier() = 0.926796050391556
CPU times: user 3.81 s, sys: 18.3 ms, total: 3.83 s
Wall time: 3.83 s
```

With GridSearch CV

In [126…
```
from sklearn.model_selection import GridSearchCV
```

In [127…
```
dic = { 'n_estimators' : [100,125],'criterion': ['gini', 'entropy'], 'max_d
       ,'min_samples_leaf' : [ 1,3]  }
```

In [128…
```
kf1 = KFold(n_splits=20)
```

In [129…
```
gvc = GridSearchCV(RandomForestClassifier(),param_grid=dic,cv = kf1)
```

In [130…
```
gvc.fit(X,y)
```

Out[130…
```
GridSearchCV(cv=KFold(n_splits=20, random_state=None, shuffle=False),
             estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [2,
3],
                         'min_samples_leaf': [1, 3],
                         'n_estimators': [100, 125]})
```

In [131…
```
gvc.best_params_
```

Out[131…
```
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 1,
 'n_estimators': 125}
```

In [132…
```
gvc.best_estimator_
```

Out[132…
```
RandomForestClassifier(max_depth=3, n_estimators=125)
```

In [133…
```
gvc.best_score_
```

Out[133…
```
0.9248571428571429
```

In [ ]: