Importing packages

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB #naive bayes
from sklearn import tree #decision tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score # for checking gini score
import catboost
from catboost import CatBoostClassifier
```

Loading the data

In [2]:
```
df = pd.read_csv(r'C:/Users/aksha_0iafvb0/OneDrive/Desktop/creditcardclg.csv')
```

In [3]:
```
# prints first 5 rows of the table
df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206

5 rows × 31 columns

In [4]:
```
# prints last 5 rows of the table
df.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V2
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	1.014480	-0.50934
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	0.012463	-1.01622
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	-0.037501	0.64013
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	-0.163298	0.12320
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	0.376777	0.00879

5 rows × 31 columns

Checking the shape of data

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

Out[5]: (284807, 31)

Checking for null values

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [7]:
```
# print full summary
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [8]:
```python
# describe the data
df.describe()
```

Out[8]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
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+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15
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+00	1.098632e+00
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+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
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-02	-5.142873e-02
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-01	5.971390e-01
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+01	1.559499e+01

8 rows × 31 columns

Checking number of records of each kind of transaction class (Fraud and Non-Fraud)

In [9]:
```python
count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction class distribution")
plt.xticks(range(2))
plt.xlabel("Class")
plt.ylabel("Frequency")
```

Out[9]: `Text(0, 0.5, 'Frequency')`

In [10]:
```python
df['Class'].nunique()
```

Out[10]: `2`

In [11]:
```python
df.groupby(['Class'])['Class'].count()
```

Out[11]:
```
Class
0    284315
1       492
Name: Class, dtype: int64
```

In [12]:
```python
(492/(492+284315))*100
```

Out[12]: `0.1727485630620034`

In [13]:
```python
# The data set is highly imbalanced. Looking at each of the fraud(1) and non-fraud(0) transactions.

frauds = df[df['Class']==1]
normal = df[df['Class']==0]
```

In [14]:
```python
frauds.shape
```

Out[14]: `(492, 31)`

In [15]:
```
normal.shape
```

Out[15]:
```
(284315, 31)
```

Checking the amount of money involved in each kind of transaction

In [16]:
```
# Fraud transactions
frauds.Amount.describe()
```

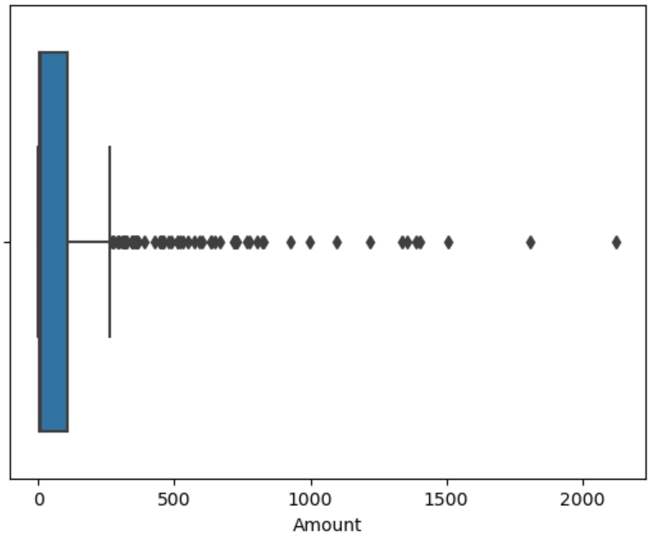
Out[16]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [17]:
```
# Non-fraud transactions
normal.Amount.describe()
```

Out[17]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

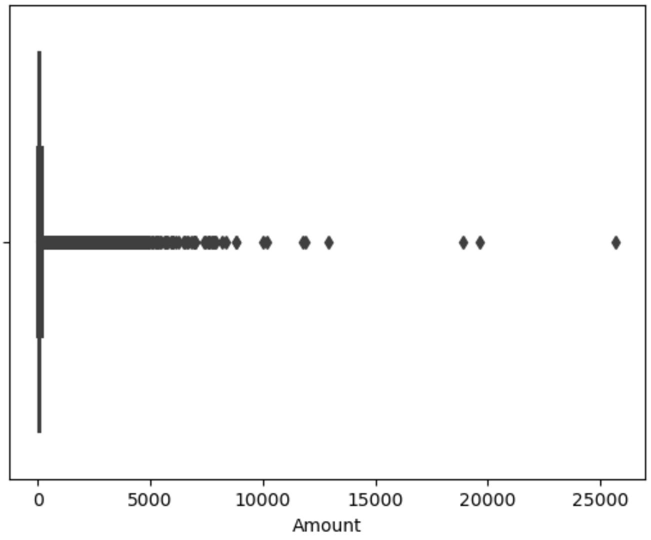
In [18]:
```
# Box plot of frauds transactions
sns.boxplot(x=frauds['Amount'])
```

Out[18]:
```
<AxesSubplot:xlabel='Amount'>
```



In [19]:
```
# box plot of normal transactions
sns.boxplot(x = normal['Amount'])
```

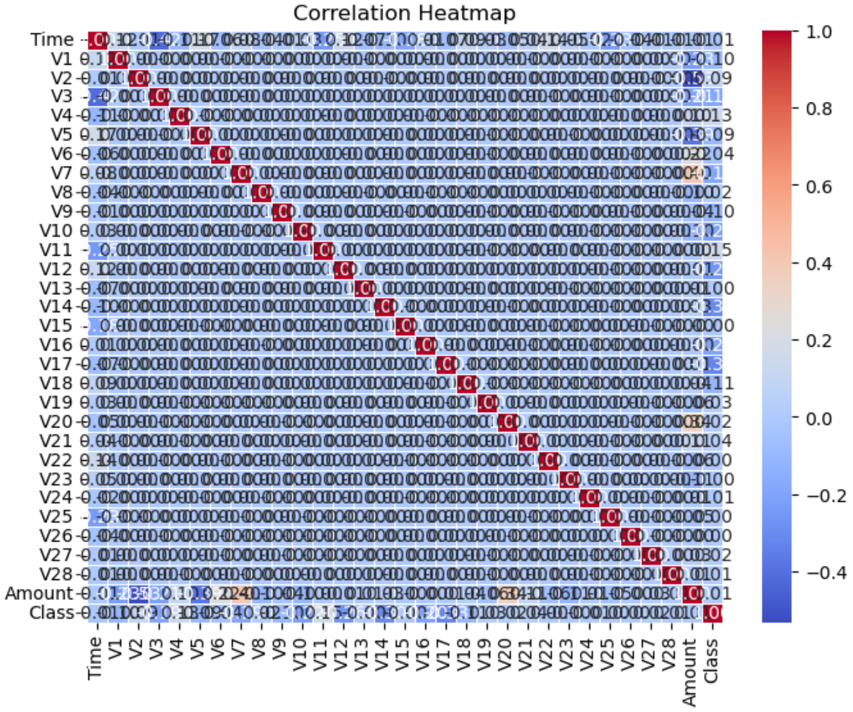
Out[19]: <AxesSubplot:xlabel='Amount'>



In [20]:
```
# heat map of correlation between variables
#sns.heatmap(df,cmap="RdYlGn",annot=True)

correlation_matrix = df.corr()

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [21]:
```
# separating the dependent and independent variables and stores in x and y variables
x=df[['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']]
y=df[['Class']]
```

In [22]:
```
y
```

Out[22]:

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

284807 rows × 1 columns

In [23]:
```
x
```

Out[23]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.251412	-0.018307	0.277838	-0.110
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.069083	-0.225775	-0.638672	0.101
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.524980	0.247998	0.771679	0.909
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.208038	-0.108300	0.005274	-0.190
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.408542	-0.009431	0.798278	-0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	1.475829	0.213454	0.111864	1.014
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.059616	0.214205	0.924384	0.012
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.001396	0.232045	0.578229	-0.037
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.127434	0.265245	0.800049	-0.163
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.382948	0.261057	0.643078	0.376

284807 rows × 30 columns

Spiting the data into 80% training and 20% testing

In [24]:
```python
# train test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

In [25]:
```python
print(x_train.shape,y_train.shape,x_test.shape,y_test.shape)
```

```
(227845, 30) (227845, 1) (56962, 30) (56962, 1)
```

In [26]:
```python
# different model functions
y_train.groupby(['Class'])['Class'].count()
```

Out[26]:
```
Class
0    227437
1       408
Name: Class, dtype: int64
```

In [27]:
```python
y_test.groupby(['Class'])['Class'].count()
```

Out[27]:
```
Class
0    56878
1       84
Name: Class, dtype: int64
```

Training the credit card fraud detection model

1. Logistic Regression Model

In [28]:
```
model = LogisticRegression()
# Fit the model to the data
model.fit(x_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/lin
ear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[28]: LogisticRegression()

In [29]:
```
# Make predictions
y_pred = model.predict(x_test)
```

In [30]:
```
set(y_pred)
```

Out[30]: {0, 1}

In [31]:
```
# convert array into pandas dataframe
d=pd.DataFrame(y_pred)
```

In [32]:
```
d
```

Out[32]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

56962 rows × 1 columns

In [33]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1 # formula to predict gini score
```

In [34]:
```
gini # prints the gini score
```

Out[34]: 0.7492439959210944

In [35]:
```
# checking the accuracy score predicted by the logistic regression model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9988764439450862
```

2. Random Forest

In [36]:
```
rf_classifier = RandomForestClassifier()
# Train the classifier
rf_classifier.fit(x_train, y_train)
```

```
C:\Users\aksha_0iafvb0\AppData\Local\Temp\ipykernel_27384\662079852.py:3: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  rf_classifier.fit(x_train, y_train)
```

Out[36]: RandomForestClassifier()

In [37]:
```
# Make predictions on the test set
y_pred = rf_classifier.predict(x_test)
```

In [38]:
```
# convert array into pandas dataframe
d=pd.DataFrame(y_pred)
```

In [39]:
```
d # prints the dataframe after converting into pandas from array
```

Out[39]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

56962 rows × 1 columns

In [40]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1 # formula to predict gini score
```

In [41]:
```
gini #prints the gini score
```

Out[41]:
```
0.8213406639775358
```

In [42]:
```
# Calculate accuracy by Random Forest Classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9996488887328394
```

3. Support vector Machine (SVC)

In [43]:
```
svm_classifier = SVC()

# Train the classifier
svm_classifier.fit(x_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[43]:
```
SVC()
```

In [44]:
```
# Make predictions on the test set
y_pred = svm_classifier.predict(x_test)
```

In [49]:
```
set(y_pred)
```

Out[49]:
```
{0}
```

In [50]:
```
# convert array into pandas dataframe
d=pd.DataFrame(y_pred)
```

In [51]:
```
d # prints the dataframe after converting into pandas from array
```

Out[51]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

56962 rows × 1 columns

In [52]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1 # formula to predict gini score
```

In [53]:
```
gini #prints the gini score
```

Out[53]:
```
0.0
```

In [54]:
```
# Calculate accuracy by svc model
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

```
Accuracy: 0.9985253326779256
```

4. Naive's bayes

In [55]:
```
# Build a Gaussian Classifier
model = GaussianNB()

# Model training
model.fit(x_train, y_train)

# Predict Output
predicted = model.predict(x_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [56]:
```
# convert array into pandas dataframe
d=pd.DataFrame(predicted)
```

In [57]:
```
d # prints the dataframe after converting into pandas from array
```

Out[57]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

56962 rows × 1 columns

In [58]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1 # formula to predict gini score
```

In [59]:
```
gini #prints the gini score
```

Out[59]:
```
0.7073410256539059
```

In [60]:
```
#checking the accuracy by naive bayes model
accuracy1 = accuracy_score(y_test, predicted)

print("Accuracy:", accuracy1)
```

```
Accuracy: 0.9926442189529862
```

5. Decision Tree Classifier

In [61]:
```
# defining decision tree classifier

model = tree.DecisionTreeClassifier(min_samples_split=10)

# train data on new data and new target
model=model.fit(x_train, y_train)
```

In [62]:
```
prediction = model.predict(x_test) #  assign removed data as input
```

In [64]:
```
# convert array into pandas dataframe
d=pd.DataFrame(prediction)
```

In [65]:
```
d # prints the dataframe after converting into pandas from array
```

Out[65]:

	0
0	0
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

56962 rows × 1 columns

In [66]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1 # formula to predict gini score
```

In [67]:
```
gini #prints the gini score
```

Out[67]: 0.7496835331762719

In [68]:
```
#accuracy score by decision tree classifier
accuracy1 = accuracy_score(y_test, prediction)

print("Accuracy:", accuracy1)
```

```
Accuracy: 0.9993153330290369
```

ploting a decision tree of the dataset

In [69]:
```
tree.plot_tree(model)
```

Out[69]:
```
[Text(0.21912379148827624, 0.9814814814814815, 'X[17] <= -2.769\ngini = 0.004\nsamples = 227845\nvalue = [227437, 408]'),
 Text(0.1357234314980794, 0.9444444444444444, 'X[10] <= -1.981\ngini = 0.348\nsamples = 361\nvalue = [81, 280]'),
 Text(0.08706786171574904, 0.9074074074074074, 'X[26] <= -0.225\ngini = 0.264\nsamples = 320\nvalue = [50, 270]'),
 Text(0.061459667093469l, 0.8703703703703703, 'X[27] <= 1.175\ngini = 0.465\nsamples = 106\nvalue = [39, 67]'),
 Text(0.040973111395646605, 0.8333333333333334, 'X[14] <= -2.87\ngini = 0.284\nsamples = 76\nvalue = [13, 63]'),
 Text(0.030729833546734954, 0.7962962962962963, 'X[29] <= 1.05\ngini = 0.138\nsamples = 67\nvalue = [5, 62]'),
 Text(0.020486555697823303, 0.7592592592592593, 'X[28] <= 0.165\ngini = 0.415\nsamples = 17\nvalue = [5, 12]'),
 Text(0.010243277848911651, 0.7222222222222222, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.030729833546734954, 0.7222222222222222, 'gini = 0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.040973111395646605, 0.7592592592592593, 'gini = 0.0\nsamples = 50\nvalue = [0, 50]'),
 Text(0.05121638924455826, 0.7962962962962963, 'gini = 0.198\nsamples = 9\nvalue = [8, 1]'),
 Text(0.08194622279129321, 0.8333333333333334, 'X[29] <= 94.99\ngini = 0.231\nsamples = 30\nvalue = [26, 4]'),
 Text(0.07170294494238157, 0.7962962962962963, 'gini = 0.0\nsamples = 25\nvalue = [25, 0]'),
 Text(0.09218950064020487, 0.7962962962962963, 'gini = 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.11267605633802817, 0.8703703703703703, 'X[11] <= 1.135\ngini = 0.098\nsamples = 214\nvalue = [11, 203]'),
 Text(0.10243277848911651, 0.8333333333333334, 'gini = 0.444\nsamples = 6\nvalue = [4, 2]'),
 Text(0.12291933418693982, 0.8333333333333334, 'X[5] <= -22.585\ngini = 0.065\nsamples = 208\nvalue = [7, 201]'),
 Text(0.11267605633802817, 0.7962962962962963, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.13316261203585147, 0.7962962962962963, 'X[4] <= -0.304\ngini = 0.047\nsamples = 206\nvalue = [5, 201]'),
```

6. Catboost Model

In [70]:
```
df
```

Out[70]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	1.014480	-0.509
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	0.012463	-1.016
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	-0.037501	0.640
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	-0.163298	0.123
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	0.376777	0.008

284807 rows × 31 columns

In [71]:
```
x=df[['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']]
y=df[['Class']]
```

In [72]:
```
# train test split
x_temp, x_test, y_temp, y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

In [73]:
```
# train test split
x_train, x_val, y_train, y_val = train_test_split(x_temp, y_temp, test_size=0.25, random_state=2)
```

In [74]:
```
# Create CatBoost classifier
cat_model = CatBoostClassifier( depth=5,
                                learning_rate=0.01,
                                iterations=1500,
                              random_seed=2,
                              loss_function='Logloss',
                              eval_metric='AUC',
                              od_type='Iter',
                              boosting_type='Plain',
                              use_best_model=True,
                              one_hot_max_size=2)
```

In [75]:
```
# Train the model
model=cat_model.fit(x_train, y_train,early_stopping_rounds=100,eval_set=(x_val,y_val))
```

```
0:      test: 0.8568622 best: 0.8568622 (0)     total: 231ms    remaining: 5m 46s
1:      test: 0.9022036 best: 0.9022036 (1)     total: 284ms    remaining: 3m 32s
2:      test: 0.9549925 best: 0.9549925 (2)     total: 334ms    remaining: 2m 46s
3:      test: 0.9639464 best: 0.9639464 (3)     total: 383ms    remaining: 2m 23s
4:      test: 0.9649631 best: 0.9649631 (4)     total: 438ms    remaining: 2m 10s
5:      test: 0.9680959 best: 0.9680959 (5)     total: 493ms    remaining: 2m 2s
6:      test: 0.9677843 best: 0.9680959 (5)     total: 546ms    remaining: 1m 56s
7:      test: 0.9709125 best: 0.9709125 (7)     total: 594ms    remaining: 1m 50s
8:      test: 0.9682731 best: 0.9709125 (7)     total: 644ms    remaining: 1m 46s
9:      test: 0.9662186 best: 0.9709125 (7)     total: 697ms    remaining: 1m 43s
10:     test: 0.9655670 best: 0.9709125 (7)     total: 746ms    remaining: 1m 41s
11:     test: 0.9660417 best: 0.9709125 (7)     total: 798ms    remaining: 1m 38s
12:     test: 0.9621671 best: 0.9709125 (7)     total: 862ms    remaining: 1m 38s
13:     test: 0.9620320 best: 0.9709125 (7)     total: 918ms    remaining: 1m 37s
14:     test: 0.9621649 best: 0.9709125 (7)     total: 978ms    remaining: 1m 36s
15:     test: 0.9605312 best: 0.9709125 (7)     total: 1.04s    remaining: 1m 36s
16:     test: 0.9592117 best: 0.9709125 (7)     total: 1.09s    remaining: 1m 35s
17:     test: 0.9610342 best: 0.9709125 (7)     total: 1.15s    remaining: 1m 34s
18:     test: 0.9624094 best: 0.9709125 (7)     total: 1.2s     remaining: 1m 33s
```

In [76]:
```
# Make predictions
y_pred = model.predict(x_test)
```

In [77]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9995611109160493
```

In [78]:
```
d=pd.DataFrame(y_pred)  # converting the array into pandas dataframe
```

In [79]:
```
d.groupby([0])[0].count()
```

Out[79]:
```
0
0    56893
1       69
Name: 0, dtype: int64
```

In [80]:
```
y_test.groupby(['Class'])['Class'].count()
```

Out[80]:
```
Class
0    56878
1       84
Name: Class, dtype: int64
```

In [81]:
```
gini=2*roc_auc_score(y_test['Class'],d[0])-1  # formula to predict gini score
```

In [82]:
```
gini
```

Out[82]:
```
0.761816854453726
```

In [ ]: