Credit Card Fraud Detection

In [74]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_csv("creditcard.csv")
```

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

Out[3]:

	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

5 rows × 31 columns

In [4]:
```
pd.options.display.max_columns = None
```

In [5]:
```
data.head()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0

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

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.

In [7]:
```
data.shape
```

Out[7]:
```
(284807, 31)
```

Columns and Rows in the DataFrame

In [9]:
```
print(f'No of Columns:{data.shape[1]}')
print(f'No of Rows:{data.shape[0]}')
```

```
No of Columns:31
No of Rows:284807
```

Information of dataset

In [10]:
```
data.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
```

Check for null values

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
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
```

Scale the values of 'Amount' col to [0-1]

In [14]:
```
from sklearn.preprocessing import StandardScaler
```

In [15]:
```
s_c= StandardScaler()
data['Amount'] = s_c.fit_transform(pd.DataFrame(data['Amount']))
```

In [17]:
```
data.sample(n=10)
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
141874	84513.0	1.306180	-0.348481	0.565490	-0.925251	-0.773318	-0.337359	-0.569415	0.002861	1.831886	-1.087731	-1.
86180	61119.0	-0.223393	0.809788	2.108311	0.864301	-0.171786	-0.290985	0.755621	-0.293363	-0.378321	-0.144528	-0.
204897	135493.0	-0.429910	1.182096	0.902845	1.098094	0.204689	0.483165	0.226346	0.509890	-0.214428	-0.747681	-1.
59974	49093.0	1.070016	-0.697632	0.917743	-0.722627	-1.430901	-0.595634	-0.702648	0.146596	1.739327	-0.898363	1.
195829	131238.0	-1.991851	-1.245559	0.592136	-3.108989	0.069898	0.385119	-0.634879	0.594820	-2.379455	0.587191	-0.
57692	48019.0	1.179829	0.259013	0.898674	1.191075	-0.743264	-0.946985	-0.081440	-0.115762	0.198919	-0.033225	-0.
21617	31768.0	1.485932	-1.582470	-0.368462	-2.276561	0.388209	3.903950	-2.126739	1.016423	-1.176493	1.305314	-0.
275698	166681.0	2.050831	-0.000680	-1.047995	0.414249	-0.070078	-1.194519	0.256780	-0.390695	0.397496	0.042819	-0.
154875	103447.0	-0.419564	-0.132591	1.729644	-2.226663	-0.105440	0.393884	-0.318073	0.086491	0.654292	-0.721938	-0.
144236	85960.0	-1.360350	0.546257	1.691625	-0.888857	1.128357	-0.381815	0.925430	-0.111397	-0.209736	-0.403776	0.

In [18]:
```
data.head()
```

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0

Remove columns which are not required for analysis Ex:'Time'

In [20]:
```
data=data.drop(['Time'], axis =1)
data.head()
```

Out[20]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196

check for duplicates

In [22]:
```
data.duplicated().any()
```

Out[22]:
```
True
```

In [23]:
```
#shape of dataframe with duplicates
data.shape
```

Out[23]:
```
(284807, 30)
```

In [24]:
```
data=data.drop_duplicates()
```

In [26]:
```
#shape of dataframe after removing duplicates
data.shape
```

Out[26]:
```
(275663, 30)
```

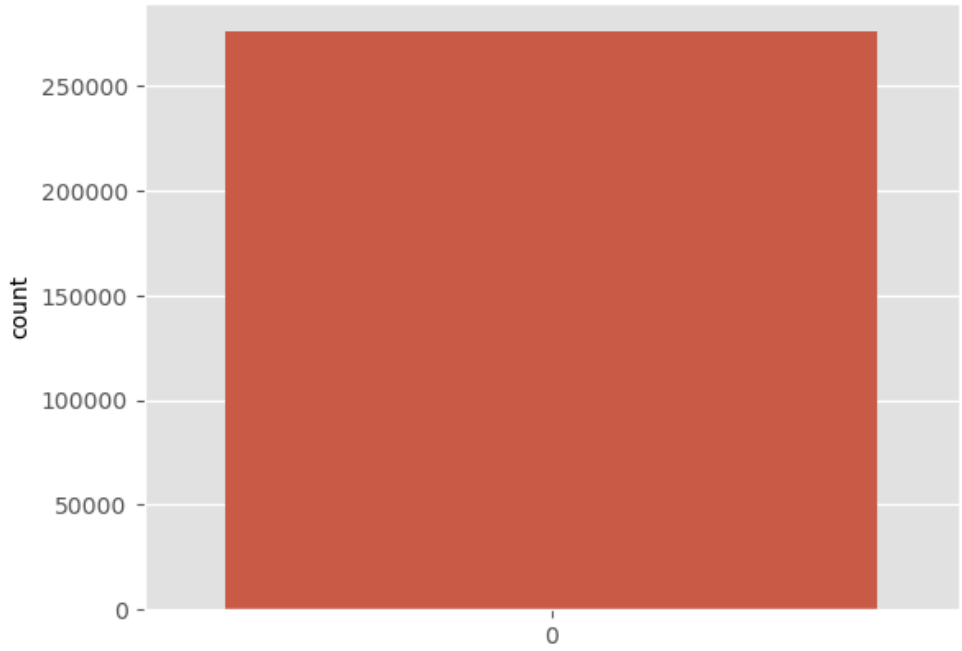
Classify legit and fraud transactions

In [28]:
```
data['Class'].value_counts()
```

Out[28]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [29]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [30]:
```
sns.countplot(data['Class'])
plt.show()
```



In [31]:
```
X = data.drop('Class', axis = 1)
Y = data['Class']
```

split the dataset into testing and training data

In [34]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

In [35]:
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
```

Considering two classifiers

1.Logistic Regression

2.Decision Tree Classifier

In [38]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier()
}

for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    model=clf
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    print(f"\n Accuaracy: {accuracy_score(Y_test, Y_pred)}")
    print(f"\n Precision: {precision_score(Y_test, Y_pred)}")
    print(f"\n Recall: {recall_score(Y_test, Y_pred)}")
    print(f"\n F1 Score: {f1_score(Y_test, Y_pred)}")
```

```
==========Logistic Regression===========

 Accuaracy: 0.9992200678359603

 Precision: 0.8870967741935484

 Recall: 0.6043956043956044

 F1 Score: 0.718954248366013

==========Decision Tree Classifier===========

 Accuaracy: 0.9989479984764116

 Precision: 0.6701030927835051

 Recall: 0.7142857142857143

 F1 Score: 0.6914893617021276
```

Data set is imbalanced which may lead to underfit

use Undersampling

In [40]:
```
legit = data[data['Class']==0]
fraud = data[data['Class']==1]
```

In [41]:
```
legit.shape
```

Out[41]:
```
(275190, 30)
```

In [42]:
```
fraud.shape
```

Out[42]:
```
(473, 30)
```

sample legit transactions

In [44]:
```
legit_sample=legit.sample(n=473)
```

In [45]:
```
legit_sample.shape
```

Out[45]:
```
(473, 30)
```

In [46]:
```
new_data= pd.concat([legit_sample,fraud], ignore_index=True)
```

In [47]:
```
new_data.head()
```

Out[47]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
0	-0.337446	0.552381	-0.141146	-1.035277	1.716910	-1.105765	1.640183	-0.529329	-0.967204	-0.220305	0.528163	0.318813
1	-0.575464	1.129638	1.967203	1.076055	0.194090	0.358161	0.439568	0.145235	-0.253237	-0.239420	-1.411023	0.089358
2	-1.682716	-0.537660	-0.109151	-1.047832	3.659474	-1.605816	1.059066	-0.330072	-0.916610	-0.568464	0.334857	0.121530
3	-4.444284	-1.810202	0.274576	1.018935	1.181810	-2.093054	-1.159731	0.594231	-0.413277	-0.195297	1.058898	1.137068
4	0.038361	0.739336	0.259281	-0.577311	0.289271	-1.085231	0.873154	-0.082839	0.216715	-0.366653	-1.171021	-0.761320

In [48]:
```
new_data.shape
```

Out[48]:
```
(946, 30)
```

In [49]:
```
new_data['Class'].value_counts()
```

Out[49]:
```
0    473
1    473
Name: Class, dtype: int64
```

In [50]:
```
X1 = new_data.drop('Class', axis = 1)
Y1 = new_data['Class']
```

In [51]:
```
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size = 0.2, random_state = 42)
```

In [52]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier()
}

for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    model=clf
    model.fit(X1_train, Y1_train)
    Y1_pred = clf.predict(X1_test)
    print(f"\n Accuaracy: {accuracy_score(Y1_test, Y1_pred)}")
    print(f"\n Precision: {precision_score(Y1_test, Y1_pred)}")
    print(f"\n Recall: {recall_score(Y1_test, Y1_pred)}")
    print(f"\n F1 Score: {f1_score(Y1_test, Y1_pred)}")
```

```
==========Logistic Regression==========

 Accuaracy: 0.9368421052631579

 Precision: 0.9591836734693877

 Recall: 0.9215686274509803

 F1 Score: 0.9400000000000001

==========Decision Tree Classifier==========

 Accuaracy: 0.8631578947368421

 Precision: 0.88

 Recall: 0.8627450980392157

 F1 Score: 0.8712871287128714
```

use Oversampling

In [55]:
```
X2 = data.drop('Class', axis = 1)
Y2 = data['Class']
```

In [56]:
```
X2.shape
```

Out[56]:
```
(275663, 29)
```

In [57]:
```
Y2.shape
```

Out[57]:
```
(275663,)
```

In [58]:
```
from imblearn.over_sampling import SMOTE
```

In [59]:
```
X_res, Y_res = SMOTE().fit_resample(X2,Y2)
```

In [60]:
```
Y_res.value_counts()
```

Out[60]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [61]:
```
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X_res, Y_res, test_size = 0.2, random_state = 42)
```

In [62]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier()
}

for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    clf.fit(X2_train, Y2_train)
    Y2_pred = clf.predict(X2_test)
    print(f"\n Accuaracy: {accuracy_score(Y2_test, Y2_pred)}")
    print(f"\n Precision: {precision_score(Y2_test, Y2_pred)}")
    print(f"\n Recall: {recall_score(Y2_test, Y2_pred)}")
    print(f"\n F1 Score: {f1_score(Y2_test, Y2_pred)}")
```

```
==========Logistic Regression===========

 Accuaracy: 0.9452378356771685

 Precision: 0.973325601623659

 Recall: 0.9154955184262676

 F1 Score: 0.9435252674773745

==========Decision Tree Classifier===========

 Accuaracy: 0.9983647661615611

 Precision: 0.9975676607794377

 Recall: 0.9991636819809828

 F1 Score: 0.9983650335168128
```

In [64]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_res, Y_res)
```

Out[64]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [65]:
```
import joblib
```

In [66]:
```
joblib.dump(dtc, "credit_card_model.pkl")
```

Out[66]:
```
['credit_card_model.pkl']
```

In [67]:
```
model = joblib.load("credit_card_model.pkl")
```

Test with a random transaction

In [68]:
```
pred = model.predict([[-0.425965884412454,0.960523044882985,1.14110934232219,-0.168252079760302,0.42098688077
```

```
C:\Users\kisha\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have valid feature n
ames, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

In [71]:
```
pred
```

Out[71]:
```
array([0], dtype=int64)
```

In [73]:
```
if pred[0]==0:
    print("Normal Transaction")
else:
    print("Fraud Transaction")
```

```
Normal Transaction
```

In [ ]: