In [1]:
```
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	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2

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	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2

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

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

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

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

In [8]:
```
print("Number of columns: {}".format(data.shape[1]))
print("Number of rows: {}".format(data.shape[0]))
```

```
Number of columns: 31
Number of rows: 284807
```

In [9]:
```
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
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
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 [11]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [13]:
```
data.head()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2

In [14]:
```
data = data.drop(['Time'], axis =1)
```

In [15]:
```
data.head()
```

Out[15]:

	V1	V2	V3	V4	V5	V6	V7	V8
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

In [16]:
```
data.duplicated().any()
```

Out[16]:
```
True
```

In [17]:
```
data = data.drop_duplicates()
```

In [18]:
```
data.shape
```

Out[18]:
```
(275663, 30)
```

In [19]:
```
data['Class'].value_counts()
```

Out[19]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [20]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [21]:
```
X = data.drop('Class', axis = 1)
y=data['Class']
```

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, randc
```

In [24]:
```
import numpy as np
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, recall_s
```

In [25]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier()
}
```

```python
for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    clf.fit(X_train, y_train)
    y_pred = clf.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)}")
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:\n{conf_matrix}\n")
```

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

 Accuaracy: 0.9992563437505668

 Precision: 0.890625

 Recall: 0.6263736263736264

 F1 Score: 0.7354838709677419
Confusion Matrix for Logistic Regression:
[[55035     7]
 [   34    57]]


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

 Accuaracy: 0.998911722561805

 Precision: 0.6631578947368421

 Recall: 0.6923076923076923

 F1 Score: 0.6774193548387096
Confusion Matrix for Decision Tree Classifier:
[[55010    32]
 [   28    63]]


==========SVM==========

 Accuaracy: 0.9993288955797798

 Precision: 0.9354838709677419

 Recall: 0.6373626373626373

 F1 Score: 0.7581699346405228
Confusion Matrix for SVM:
[[55038     4]
 [   33    58]]


==========Random Forest==========

 Accuaracy: 0.9994377233235993

 Precision: 0.9054054054054054

 Recall: 0.7362637362637363

 F1 Score: 0.8121212121212121
Confusion Matrix for Random Forest:
[[55035     7]
 [   24    67]]


==========Naive Bayes==========

 Accuaracy: 0.9781618994068888
```

```
 Precision: 0.057279236276849645

 Recall: 0.7912087912087912

 F1 Score: 0.10682492581602374
Confusion Matrix for Naive Bayes:
[[53857  1185]
 [   19    72]]


==========KNN===========

 Accuaracy: 0.999419585366296

 Precision: 0.8831168831168831

 Recall: 0.7472527472527473

 F1 Score: 0.8095238095238095
Confusion Matrix for KNN:
[[55033     9]
 [   23    68]]
```

In [26]:
```
#undersampling
```

In [27]:
```
normal = data[data['Class']==0]
fraud = data[data['Class']==1]
```

In [28]:
```
normal.shape
```

Out[28]:
```
(275190, 30)
```

In [29]:
```
fraud.shape
```

Out[29]:
```
(473, 30)
```

In [30]:
```
normal_sample = normal.sample(n=473)
```

In [31]:
```
normal_sample.shape
```

Out[31]:
```
(473, 30)
```

In [32]:
```
new_data = pd.concat([normal_sample,fraud], ignore_index=True)
```

In [33]:
```
new_data.head()
```

Out[33]:

	V1	V2	V3	V4	V5	V6	V7	V8
0	1.786403	-0.418618	-2.828184	0.375693	0.726157	-1.412580	1.222661	-0.600657
1	1.145502	-0.538674	0.960678	-0.830265	-1.183146	-0.177999	-0.839344	0.315109
2	2.244636	-1.499404	-1.052182	-1.651386	-1.218889	-0.411978	-1.227834	0.008199
3	-1.043354	0.771407	0.997782	-0.753236	1.158731	-0.339265	0.457603	0.138741
4	-1.064859	1.226340	1.454448	0.763100	-0.173218	0.473856	0.299699	0.644049

In [34]:
```
new_data['Class'].value_counts()
```

Out[34]:
```
0    473
1    473
Name: Class, dtype: int64
```

In [35]:
```
X = new_data.drop('Class', axis = 1)
y= new_data['Class']
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, rando
```

In [37]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier()
}

for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    clf.fit(X_train, y_train)
    y_pred = clf.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)}")
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:\n{conf_matrix}\n")
```

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

 Accuaracy: 0.9473684210526315

 Precision: 0.9893617021276596

 Recall: 0.9117647058823529

 F1 Score: 0.9489795918367347
Confusion Matrix for Logistic Regression:
[[87  1]
 [ 9 93]]


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

 Accuaracy: 0.9210526315789473

 Precision: 0.9065420560747663

 Recall: 0.9509803921568627

 F1 Score: 0.9282296650717703
Confusion Matrix for Decision Tree Classifier:
[[78 10]
 [ 5 97]]


==========SVM==========

 Accuaracy: 0.9210526315789473

 Precision: 0.978021978021978

 Recall: 0.8725490196078431

 F1 Score: 0.9222797927461139
Confusion Matrix for SVM:
[[86  2]
 [13 89]]


==========Random Forest==========

 Accuaracy: 0.9421052631578948

 Precision: 0.989247311827957

 Recall: 0.9019607843137255

 F1 Score: 0.9435897435897436
Confusion Matrix for Random Forest:
[[87  1]
 [10 92]]


==========Naive Bayes==========

 Accuaracy: 0.9052631578947369
```

```
 Precision: 0.9565217391304348

 Recall: 0.8627450980392157

 F1 Score: 0.9072164948453608
Confusion Matrix for Naive Bayes:
[[84  4]
 [14 88]]


==========KNN===========

 Accuaracy: 0.9421052631578948

 Precision: 0.989247311827957

 Recall: 0.9019607843137255

 F1 Score: 0.9435897435897436
Confusion Matrix for KNN:
[[87  1]
 [10 92]]
```

In [38]:
```
# OVERSAMPLING
```

In [39]:
```
X = data.drop('Class', axis = 1)
y= data['Class']
```

In [40]:
```
X.shape
```

Out[40]:
```
(275663, 29)
```

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

Out[41]:
```
(275663,)
```

In [42]:
```
from imblearn.over_sampling import SMOTE
```

In [43]:
```
X_res, y_res = SMOTE().fit_resample(X,y)
```

In [44]:
```
y_res.value_counts()
```

Out[44]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [45]:
```
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size = 0.
```

In [46]:
```
classifier = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree Classifier": DecisionTreeClassifier(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'KNN': KNeighborsClassifier()
}
```

```python
for name, clf in classifier.items():
    print(f"\n==========={name}===========")
    clf.fit(X_train, y_train)
    y_pred = clf.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)}")
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix for {name}:\n{conf_matrix}\n")
```

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

 Accuaracy: 0.9457193211962645

 Precision: 0.9732249720088028

 Recall: 0.91658636801629

 F1 Score: 0.9440569261738683
Confusion Matrix for Logistic Regression:
[[53686  1387]
 [ 4588 50415]]


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

 Accuaracy: 0.9979196191722083

 Precision: 0.9971680644809934

 Recall: 0.9986727996654728

 F1 Score: 0.9979198648366322
Confusion Matrix for Decision Tree Classifier:
[[54917   156]
 [   73 54930]]


==========SVM==========

 Accuaracy: 0.9808405101929576

 Precision: 0.9836863089359523

 Recall: 0.9778739341490464

 F1 Score: 0.9807715101065818
Confusion Matrix for SVM:
[[54181   892]
 [ 1217 53786]]


==========Random Forest==========

 Accuaracy: 0.999918238308078

 Precision: 0.9998363993310551

 Recall: 1.0

 F1 Score: 0.9999181929736854
Confusion Matrix for Random Forest:
[[55064     9]
 [    0 55003]]


==========Naive Bayes==========

 Accuaracy: 0.9121788582433955
```

```
 Precision: 0.9723484059178996

 Recall: 0.8483719069868916

 F1 Score: 0.9061392521821872
Confusion Matrix for Naive Bayes:
[[53746  1327]
 [ 8340 46663]]


==========KNN===========

 Accuaracy: 0.9990551982266798

 Precision: 0.9981127624439726

 Recall: 1.0

 F1 Score: 0.9990554899645808
Confusion Matrix for KNN:
[[54969   104]
 [    0 55003]]
```

In [47]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_res, y_res)
```

Out[47]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [48]:
```
import joblib
```

In [49]:
```
joblib.dump(dtc, "credit_card_model.pkl")
```

Out[49]:
```
['credit_card_model.pkl']
```

In [50]:
```
model = joblib.load("credit_card_model.pkl")
```

In [51]:
```
pred = model.predict([[-1.3598071336738,-0.0727811733098497,2.53634673796914,1.3
```

```
c:\Users\KEVIN\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn
\base.py:493: UserWarning: X does not have valid feature names, but DecisionTreeC
lassifier was fitted with feature names
  warnings.warn(
```

In [52]:
```
pred[0]
```

Out[52]:
```
0
```

In [53]:
```
if pred[0] == 0:
    print("Normal Transcation")
else:
    print("Fraud Transcation")
```

```
Normal Transcation
```