In [3]:
```
import pandas as pd
```

In [4]:
```
data = pd.read_csv("Credit Card Capstone.csv ")
```

In [7]:
```
data.head()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
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	-0.617801	-0.9
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	1.065235	0.4
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	0.066084	0.7
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	0.178228	0.5
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	0.538196	1.3

In [8]:
```
pd.options.display.max_columns = None
```

In [9]:
```
data.head()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
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	-0.617801	-0.9
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	1.065235	0.4
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	0.066084	0.7
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	0.178228	0.5
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	0.538196	1.3

In [10]:
```
data.tail()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.7
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.9
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.9
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.0

In [11]:
```
data.shape
```

Out[11]:
```
(284807, 31)
```

In [12]:
```
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 [13]:
```
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 [14]:
```
data.isnull().sum()
```

Out[14]:
```
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 [15]:
```
from sklearn.preprocessing import StandardScaler
```

In [16]:
```
sc = StandardScaler()
data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [17]:
```
data.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
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	-0.617801	-0.9
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	1.065235	0.4
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	0.066084	0.7
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	0.178228	0.5
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	0.538196	1.3

In [18]:
```
data = data.drop(['Time'],axis=1)
```

In [19]:
```
data.head()
```

Out[19]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
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	-0.991390
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	0.489095
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	0.717293
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	0.507757
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	1.345852

In [20]:
```
data.duplicated().any()
```

Out[20]:
```
True
```

In [21]:
```
data = data.drop_duplicates()
```

In [22]:
```
data.shape
```

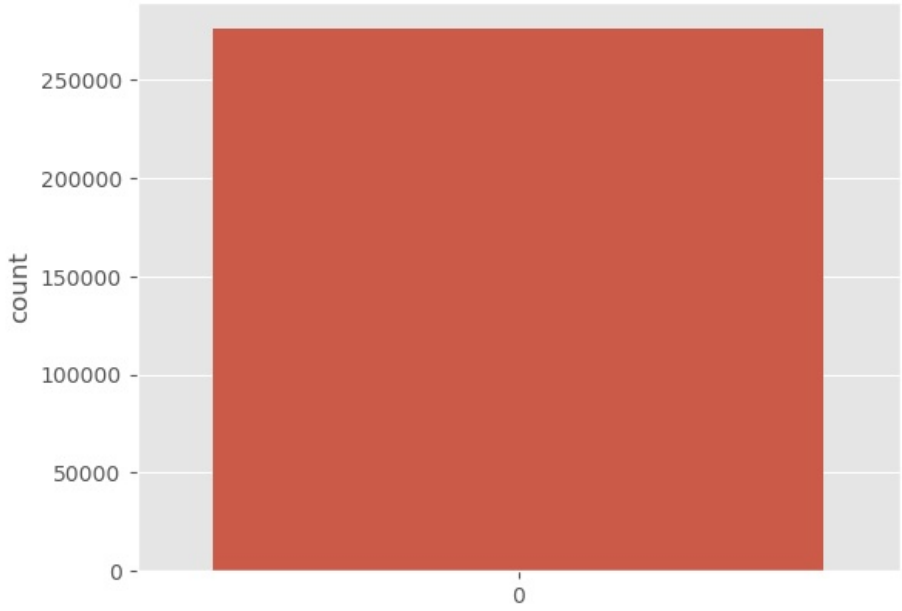
Out[22]:
```
(275663, 30)
```

In [23]:
```
data['Class'].value_counts()
```

Out[23]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [28]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

In [29]:
```
sns.countplot(data['Class'])
plt.show()
```



In [27]:
```
X = data.drop('Class',axis=1)
y = data['Class']
```

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [31]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [ ]:
```
import numpy as np
```

In [32]:
```
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
```

In [40]:
```
classifier ={
    "Logistic Regression":LogisticRegression(),
    "Decision Tree Classifier":DecisionTreeClassifier(),
}
for name,clf in classifier.items():
    print(f"\n=============={name}==========")
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print(f"\n Accuracy:{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==========

 Accuracy:0.9992200678359603

 Precision:0.8870967741935484

 Recall:0.6043956043956044

 F1 Score:0.718954248366013

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

 Accuracy:0.9990568262202311

 Precision:0.7096774193548387

 Recall:0.7252747252747253

 F1 Score:0.7173913043478262
```

In [33]:
```
# Undersampling
```

In [34]:
```
normal = data[data['Class']==0]
fraud = data[data['Class']==1]
```

In [35]:
```
normal.shape
```

Out[35]:
```
(275190, 30)
```

In [36]:
```
fraud.shape
```

Out[36]:
```
(473, 30)
```

In [37]:
```
normal_sample = normal.sample(n=473)
```

In [38]:
```
normal_sample.shape
```

Out[38]:
```
(473, 30)
```

In [39]:
```
new_data = pd.concat([normal_sample,fraud],ignore_index=True)
```

In [41]:
```
new_data.head()
```

Out[41]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
0	0.956263	-1.096976	-0.149957	-0.334294	-0.851049	-0.497953	0.034073	-0.258625	-0.951725	0.457016	-0.374379	0.497543	0.991806
1	0.708308	-3.795859	-2.549069	-1.032161	-1.454594	-0.628932	0.538933	-0.582555	-1.867507	1.256376	-0.797848	-0.809263	0.723141
2	-2.923767	1.930753	0.941767	0.688651	-0.018782	0.839237	-1.042656	-2.974310	0.811210	1.438429	1.073748	0.920150	-0.839574
3	1.926063	0.096148	-1.775737	0.551743	0.283471	-1.342968	0.444069	-0.464836	0.369193	-0.553308	-0.116003	0.826922	1.452665
4	-1.859420	-1.407000	1.020623	-2.789918	-0.828784	-0.499326	-0.537072	0.574959	-2.350720	0.437779	-0.029979	-0.661732	-0.039377

In [42]:
```
new_data['Class'].value_counts()
```

Out[42]:
```
0    473
1    473
Name: Class, dtype: int64
```

In [43]:
```
X = new_data.drop('Class',axis=1)
y = new_data['Class']
```

In [44]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [44]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

In [57]:
```
classifier ={
    "Logistic Regression":LogisticRegression(),
    "Decision Tree Classifier":DecisionTreeClassifier(),
}
for name,clf in classifier.items():
    print(f"\n=============={name}==========")
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print(f"\n Accuracy:{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==========

 Accuracy:0.9315789473684211

 Precision:0.9587628865979382

 Recall:0.9117647058823529

 F1 Score:0.9346733668341709

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

 Accuracy:0.8842105263157894

 Precision:0.8636363636363636

 Recall:0.9313725490196079

 F1 Score:0.8962264150943398
```

In [45]:
```
# OVERSAMPLING
```

In [46]:
```
X = data.drop('Class',axis=1)
y = data['Class']
```

In [47]:
```
X.shape
```

Out[47]:
```
(275663, 29)
```

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

Out[48]:
```
(275663,)
```

In [49]:
```
from imblearn.over_sampling import SMOTE
```

In [50]:
```
X_res,y_res = SMOTE().fit_resample(X,y)
```

In [51]:
```
y_res.value_counts()
```

Out[51]:
```
0    275190
1    275190
Name: Class, dtype: int64
```

In [52]:
```
X_train,X_test,y_train,y_test = train_test_split(X_res,y_res,test_size=0.2,random_state=42)
```

In [53]:
```
classifier ={
    "Logistic Regression":LogisticRegression(),
    "Decision Tree Classifier":DecisionTreeClassifier(),
}
for name,clf in classifier.items():
    print(f"\n=============={name}==========")
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print(f"\n Accuracy:{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==========

 Accuracy:0.9448290272175588

 Precision:0.9733023795705166

 Recall:0.9146773812337509

 F1 Score:0.9430796772046901

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

 Accuracy:0.9979559577019513

 Precision:0.9969157081171305

 Recall:0.9990000545424795

 F1 Score:0.9979567929822649
```

In [54]:
```
dtc=DecisionTreeClassifier()
dtc.fit(X_res,y_res)
```

Out[54]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [55]:
```
import joblib
```

In [57]:
```
joblib.dump(dtc,"credit_card_model.pkl")
```

Out[57]:
```
['credit_card_model.pkl']
```

In [58]:
```
model=joblib.load("credit_card_model.pkl")
```

In [59]:
```
pred=model.predict([[-1.359807134,-0.072781173,2.536346738,1.378155224,-0.33832077,0.462387778,0.239598554,0.09
```

```
C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names
, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

In [60]:
```
pred[0]
```

Out[60]:
```
0
```

In [66]:
```
if pred[0] == 0:
    print("Normal Transaction")

else:
          print("Fraud Transaction")
```

```
Normal Transaction
```

In [ ]:

In [ ]: