Credit Card Fraud Detection-Internship T1

In [31]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import warnings;
warnings.filterwarnings('ignore');


```

Importing dataset

In [32]:

```
data = pd.read_csv('creditcard.csv')
```

3 - Dataset Exploration

In [33]:

```
data.shape
```

Out[33]:

```
(284807, 31)
```

In [34]:

```
data.head()
```

Out[34]:

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

5 rows × 31 columns

In [35]:
```
data.describe()
```

Out[35]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.4873
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.3322
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.6160
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.6829
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.7418
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.9856
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.3301

8 rows × 31 columns

In [36]:
```
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 [37]:
```
data.isnull().sum()
```

Out[37]:
```
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
```

4 - Data Preprocessing

4.1 - Data Cleaning

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

Out[38]:
```
(284807, 31)
```

In [39]:
```
data.drop_duplicates(inplace=True)
```

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

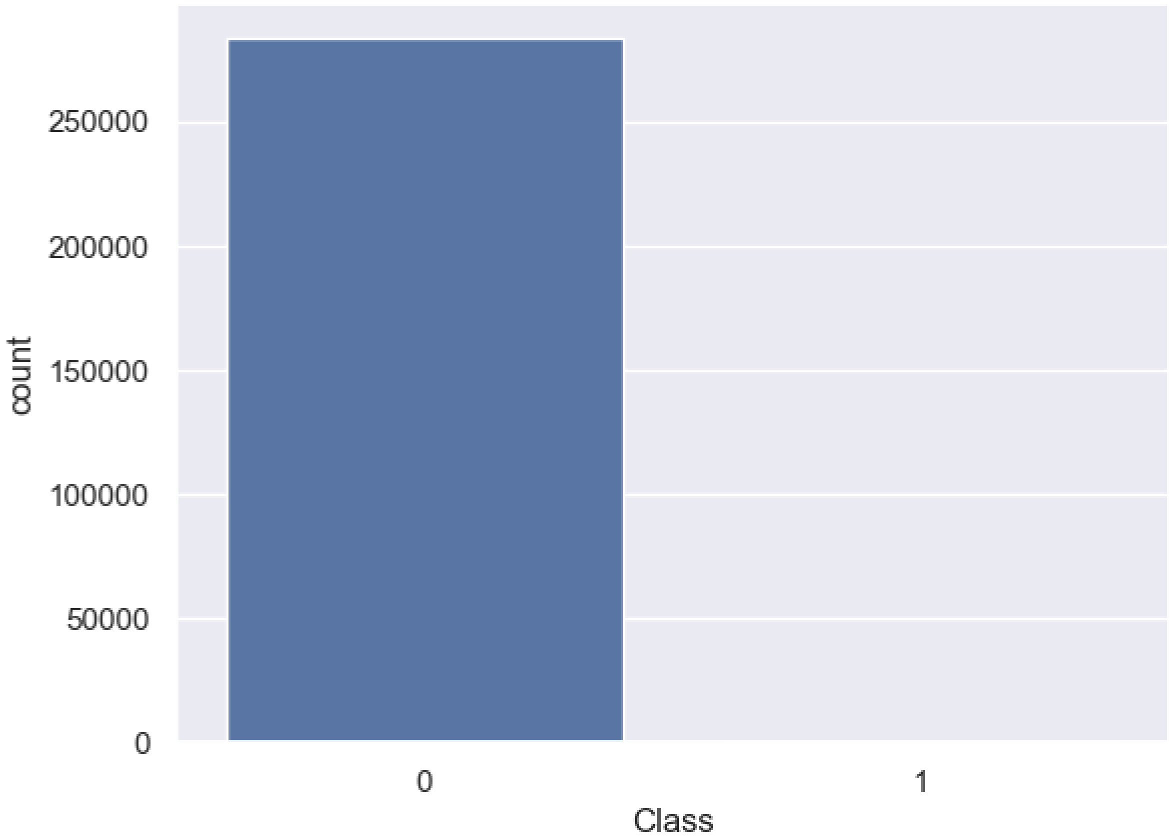
Out[40]:
```
(283726, 31)
```

4.2 - Identifying Data Imbalance

In [41]:
```
sns.set_theme()
```

In [42]:
```
sns.countplot(x='Class', data=data)
print(data['Class'].value_counts())
```

```
0    283253
1       473
Name: Class, dtype: int64
```



0 -> Genuine Transaction

1 -> Fraud Transaction

From above countplot, it seems dataset is highly imbalanced.

4.3 - Data Standardization

In [43]:
```
scaler = StandardScaler()
```

In [44]:
```
# Standardizing 'Time' and 'Amount' column
data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1,1))
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1,1))
```

In [45]:

```
data.head()
```

Out[45]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.996823	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3637
1	-1.996823	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2554
2	-1.996802	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514
3	-1.996802	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387
4	-1.996781	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8177

5 rows × 31 columns

4.4 - Split the data into Features (X) and the Target (y)

In [46]:

```
x = data.drop(['Class'], axis = 1)
y = data['Class']
```

In [47]:

```
print(x)
```

```
            Time         V1         V2        V3        V4        V5  \
0      -1.996823  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1      -1.996823   1.191857   0.266151  0.166480  0.448154  0.060018
2      -1.996802  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3      -1.996802  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4      -1.996781  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  1.642235 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  1.642257  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  1.642278   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  1.642278  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  1.642362  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V20       V21  \
0       0.462388  0.239599  0.098698  0.363787  ...  0.251412 -0.018307
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775
2       1.800499  0.791461  0.247676 -1.514654  ...  0.524980  0.247998
3       1.247203  0.237609  0.377436 -1.387024  ... -0.208038 -0.108300
4       0.095921  0.592941 -0.270533  0.817739  ...  0.408542 -0.009431
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  1.475829  0.213454
284803  1.058415  0.024330  0.294869  0.584800  ...  0.059616  0.214205
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.001396  0.232045
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.127434  0.265245
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.382948  0.261057

             V22       V23       V24       V25       V26       V27       V28  \
0       0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1      -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2       0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3       0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4       0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
...          ...       ...       ...       ...       ...       ...       ...
284802  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
284803  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
284804  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
284805  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
284806  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649

          Amount
0       0.244200
1      -0.342584
2       1.158900
3       0.139886
4      -0.073813
...          ...
284802 -0.350252
284803 -0.254325
284804 -0.082239
284805 -0.313391
284806  0.513290

[283726 rows x 30 columns]
```

In [48]:
```
print(y)
```

```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 283726, dtype: int64
```

4.5 - Split the dataset into Training and Testing sets

In [49]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_s
print(x.shape, x_train.shape, x_test.shape)
```

```
(283726, 30) (226980, 30) (56746, 30)
```

5. Handling Data Imbalanace

In [50]:
```
oversampler = RandomOverSampler(sampling_strategy=0.5, random_state=42)
```

In [51]:
```
x_resampled, y_resampled = oversampler.fit_resample(x_train, y_train)
```

In [52]:
```
print(x_resampled.shape, y_resampled.shape)
```

```
(339888, 30) (339888,)
```

6 - Building & Training Model

For this Classification task, we'll use machine learning models like Logstic Regression or Random Forest Classification.

In [53]:
```
model = LogisticRegression()
```

In [54]:
```
model.fit(x_resampled, y_resampled)
```

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

7 - Model Evaluation

In [55]:

```
# Accuracy of training data
x_resampled_predictions = model.predict(x_resampled)

accuracy_score = accuracy_score(x_resampled_predictions, y_resampled)
print("Accuracy Score of training data:",accuracy_score)
```

```
Accuracy Score of training data: 0.9603869509956221
```

In [ ]:

In [ ]:

In [ ]: