# Credit Card Fraud Detection

Using the Machine Learning Classification Algorithms to detect Credit Card Fraudulent Activities

# Credit Card Fraud Detection

Dataset :-- https://drive.google.com/drive/folders/14hpoXhDQgP2x5gGAcYwunA6tMlNPZEKG

Most of the approaches involve building model on such imbalanced data, and thus fails to produce results on real-time new data because of overfitting on training data and a bias towards the majoritarian class of legitimate transactions. Thus, we can see this as an anomaly detection problem.

1) What time does the Credit Card Frauds usually take place?

2) What are the general trends of amounts for Credit Card Fraud Transactions?

3) How do we balance the data to not let the model overfit on legitimate transactions?

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier

from mlxtend.plotting import plot_learning_curves
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.metrics import precision_score, recall_score, f1_score,
roc_auc_score, accuracy_score, classification_report
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, matthews_corrcoef
from sklearn.feature_selection import mutual_info_classif

import warnings
warnings.filterwarnings("ignore")
```

# Data Understanding

```
# read the dataset using pandas library
credit_card_data = pd.read_csv(r"C:\Users\user\Documents\
creditcard.csv")
```

```
credit_card_data

              Time         V1         V2        V3        V4        
V5  \
0              0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321   

1              0.0   1.191857   0.266151  0.166480  0.448154  0.060018   

2              1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198   

3              1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309   

4              2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193   

...            ...        ...        ...       ...       ...       ...   

284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473   

284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229   

284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515   

284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961   

284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546   


              V6        V7        V8        V9  ...       V21       
V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  
0.277838   
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -
0.638672   
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  
0.771679   
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  
0.005274   
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  
0.798278   
...          ...       ...       ...       ...  ...       ...       ..
.   
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  
0.111864   
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  
0.924384   
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  
0.578229   
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  
0.800049   
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  
0.643078   
```

```
              V23       V24       V25       V26       V27       V28
Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
69.99
...          ...       ...       ...       ...       ...       ...
...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649
217.00

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

[284807 rows x 31 columns]
```

```
# getting the first 5 row's values
credit_card_data.head()
```

```
   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
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -
0.078803
```

```
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
0.592941

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

```
# getting the last 5 row's values
credit_card_data.tail()
```

```
            Time         V1         V2        V3        V4
V5  \
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473

284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229

284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515

284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961

284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546


              V6        V7        V8        V9  ...       V21
V22  \
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205
0.924384
```

```
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057
0.643078

             V23       V24       V25       V26       V27       V28
Amount  \
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649
217.00

        Class
284802      0
284803      0
284804      0
284805      0
284806      0

[5 rows x 31 columns]
```

```
#getting the information about the whole dataset
credit_card_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
```

```python
#to check the null value in each column
credit_card_data.isnull().sum()
```

```
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
```

# Data Preparation

The Data does not have any missing values and hence, need not be handled.

The Data has only Target Variable Class as the categorical variable.

Remaining Features are numerical and need to be only standardized for comparison after balancing the dataset

```
credit_card_data.describe()
```

```
                Time            V1            V2            V3
V4  \
count  284807.000000  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
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00
1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -
5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -
8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -
1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00
7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00
1.687534e+01

                 V5            V6            V7            V8
V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean  -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -
3.147640e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
```

```
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01
1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.473120e-16  8.042109e-16  5.282512e-16  4.456271e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean   1.426896e-15  1.701640e-15 -3.662252e-16 -1.217809e-16
88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

```python
# checking how many transactions are fraudulent and legitimate
credit_card_data['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```python
#seperating the values of fraud and legit data
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

```python
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```python
legit.Amount.describe()
```

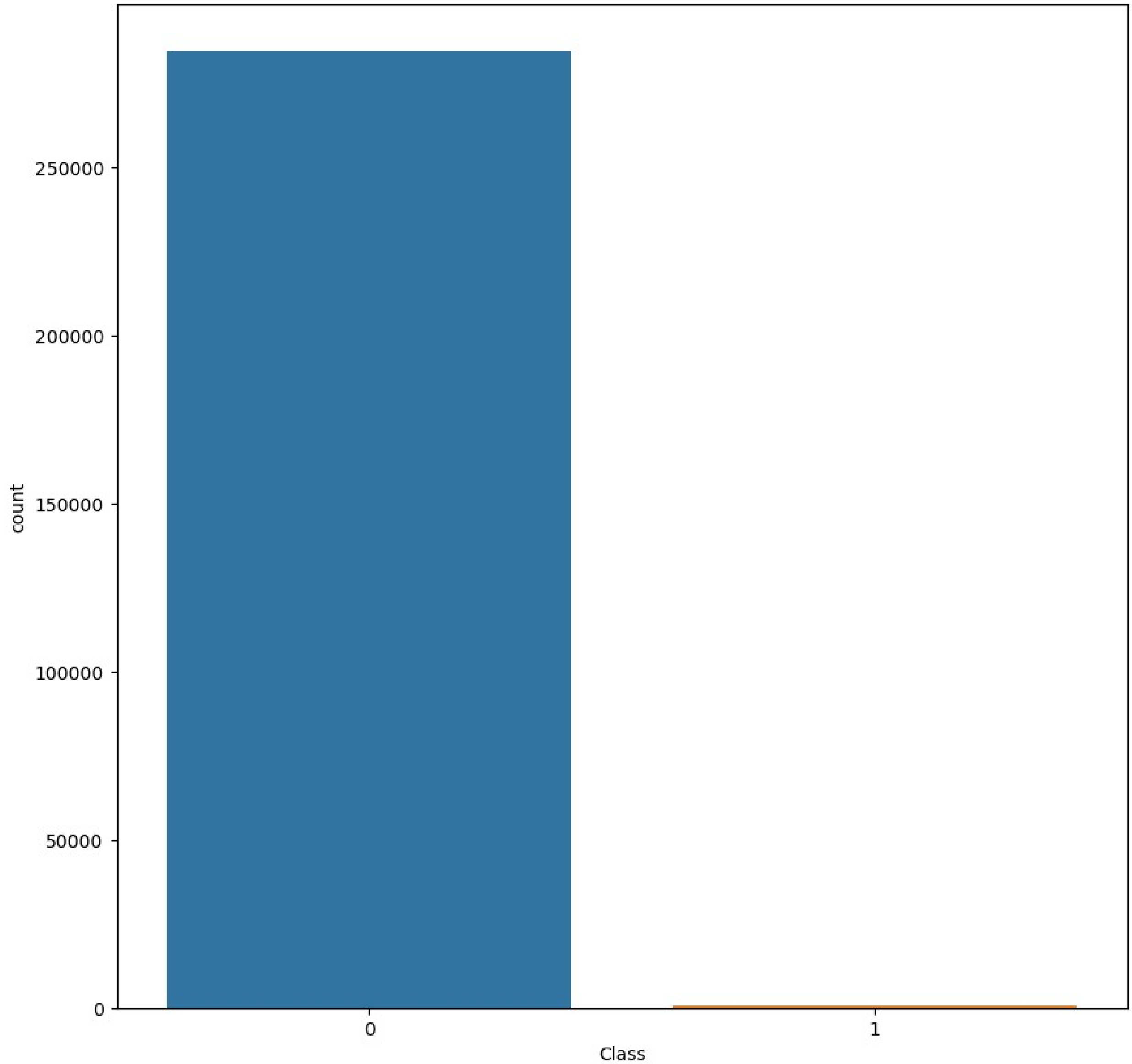
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```python
fraud.Amount.describe()
```

```
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
```

```python
def countplot_data(data, feature):
    plt.figure(figsize = (10, 10))
    sns.countplot(x = feature, data = data)
    plt.show
```

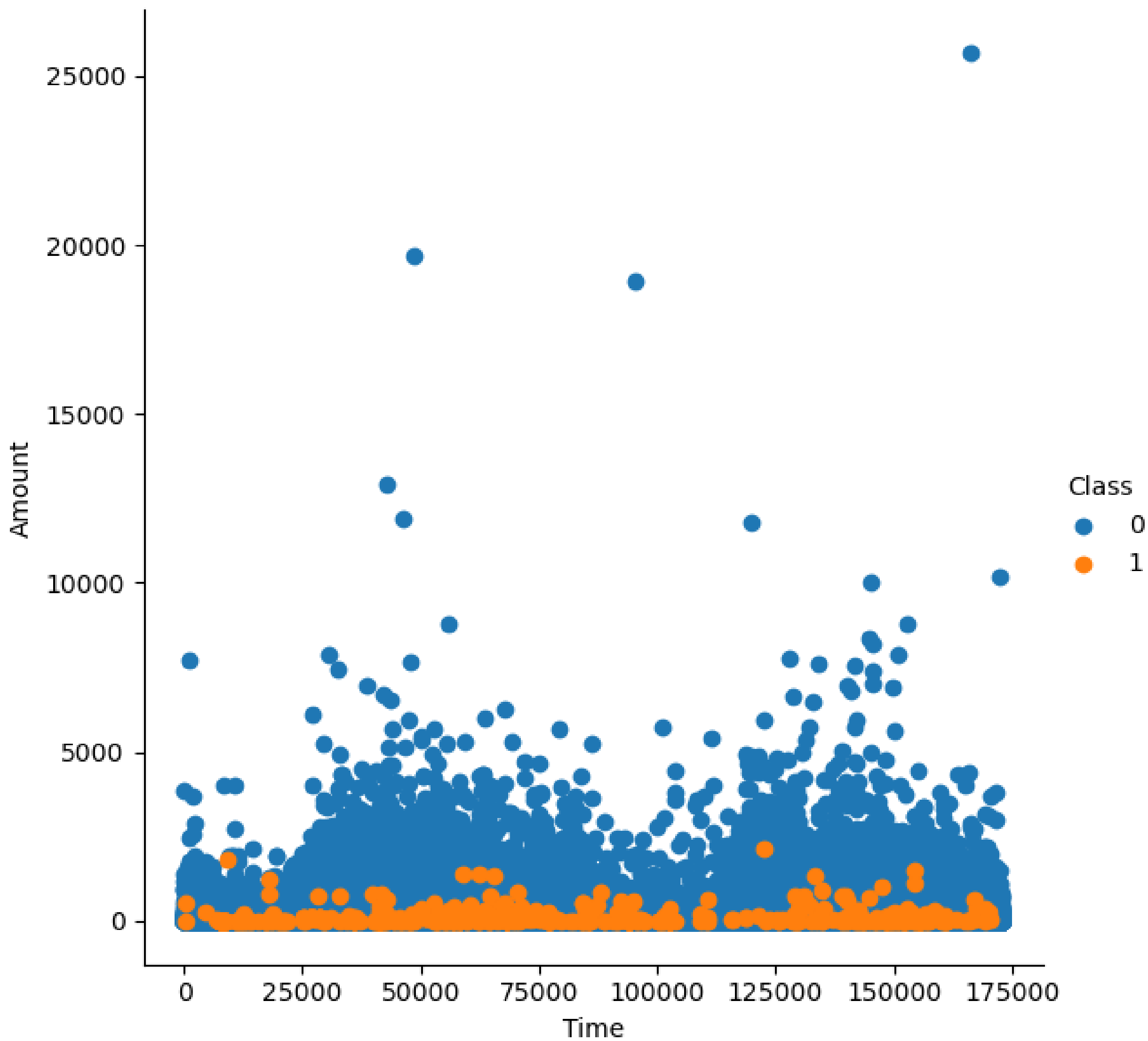
```python
countplot_data(credit_card_data, credit_card_data.Class)
```



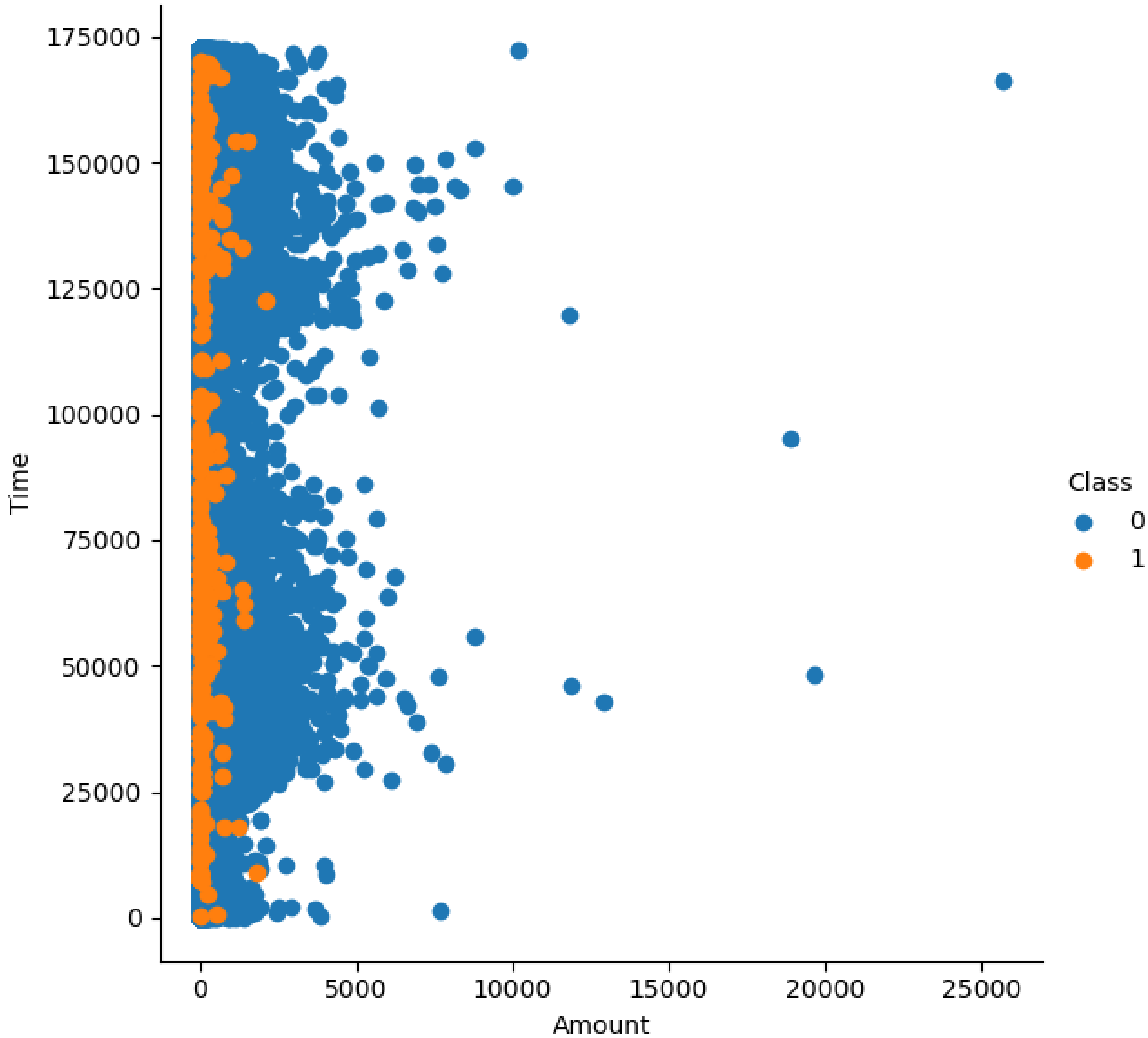

```
def pairplot_data_grid(data, feature1, feature2, target):
    sns.FacetGrid(data, hue=target, size=6).map(plt.scatter, feature1, 
feature2).add_legend()
    plt.show()
```

# Relationship of fraud transactions with amount of money

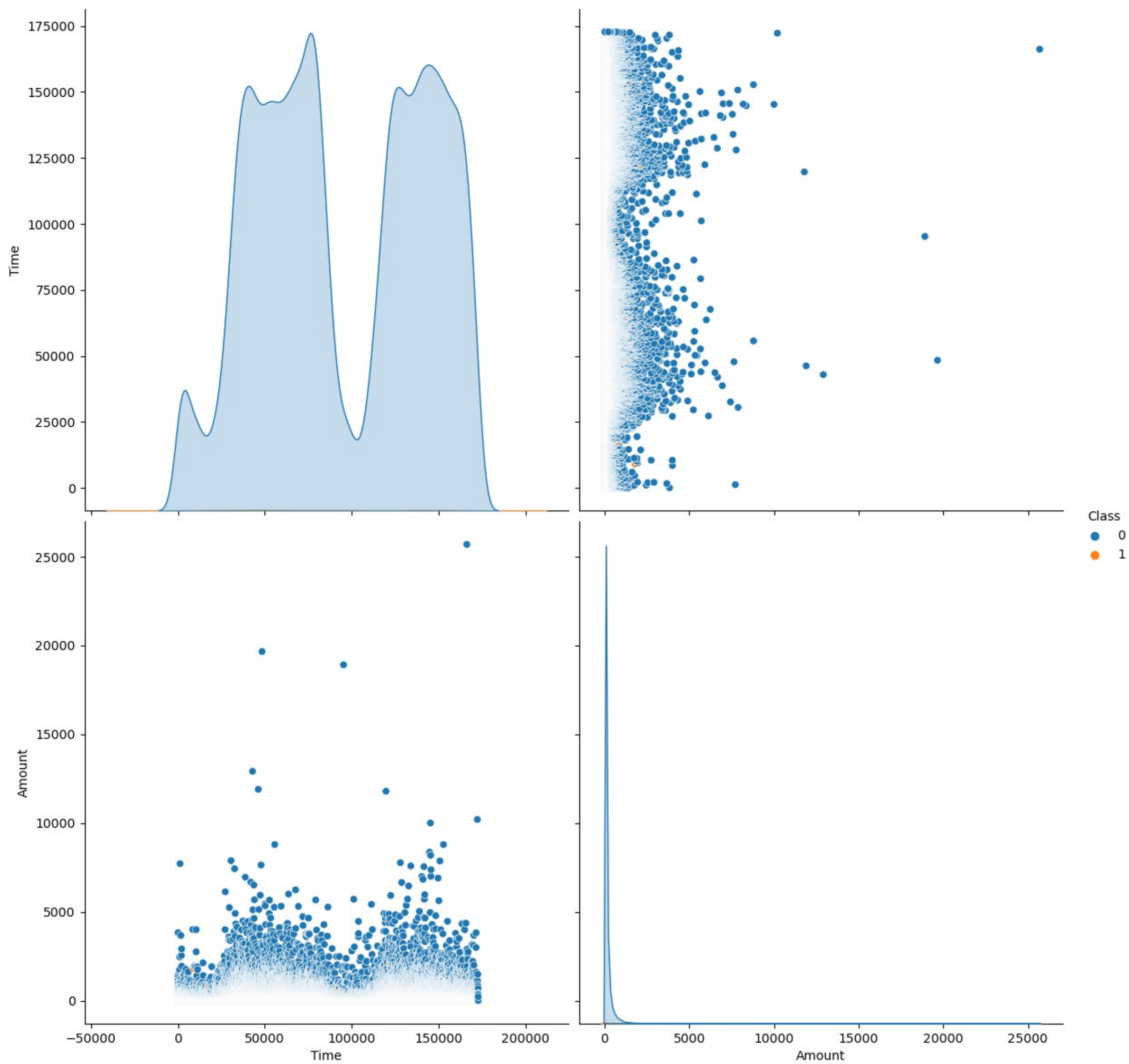
```
pairplot_data_grid(credit_card_data, "Time", "Amount", "Class")
```




```
pairplot_data_grid(credit_card_data, "Amount", "Time", "Class")
```

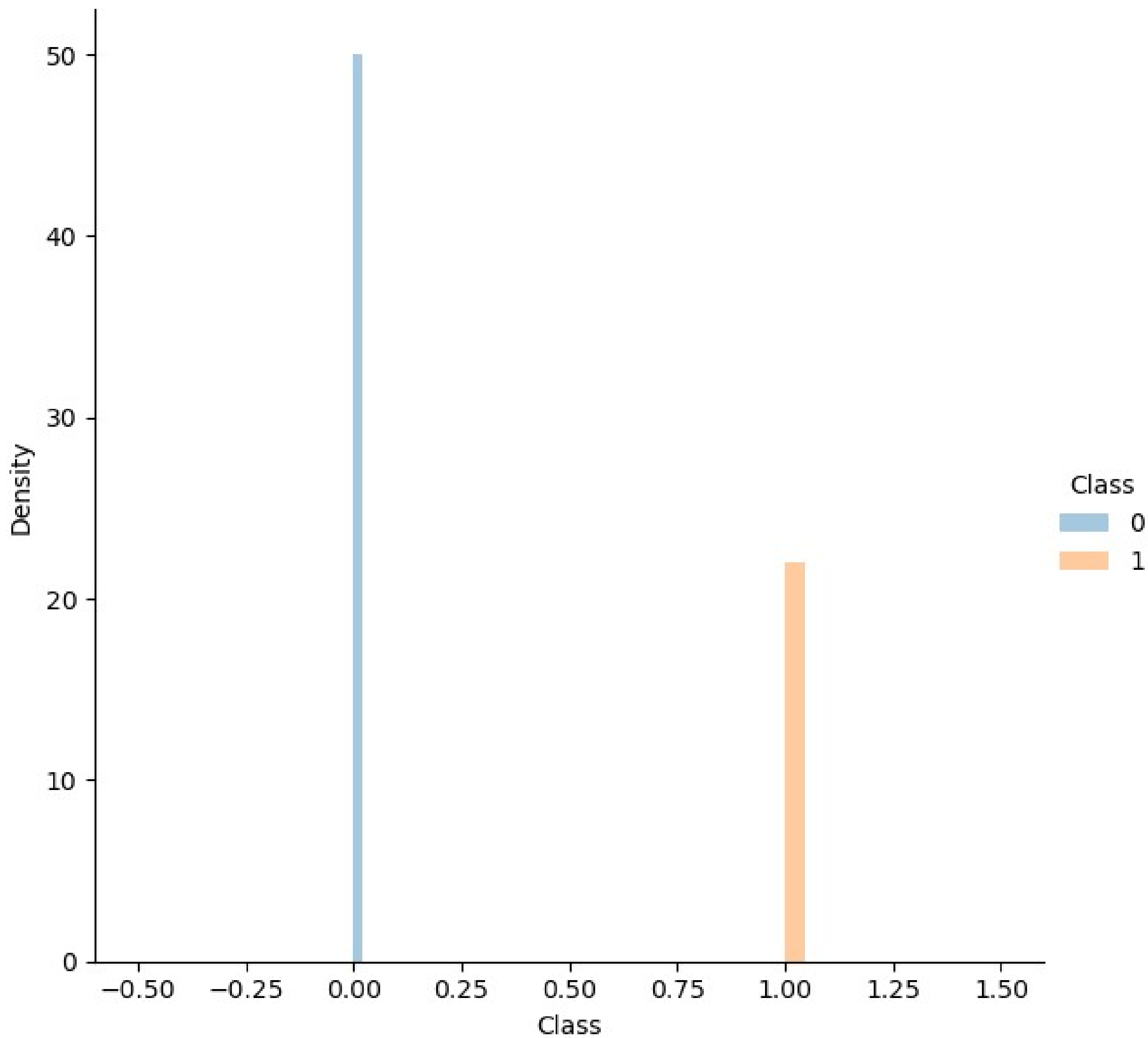



```
credit_card_data_refine = credit_card_data[["Time", "Amount",
"Class"]]
sns.pairplot(credit_card_data_refine, hue="Class", size=6)
plt.show()
```




# The relationship between Time and Transaction.

```
sns.FacetGrid(credit_card_data_refine, hue="Class",
size=6).map(sns.distplot, "Class").add_legend()
plt.show()
```




# Modelling

Study the Feature Correlations of the given data

Plot a Heatmap

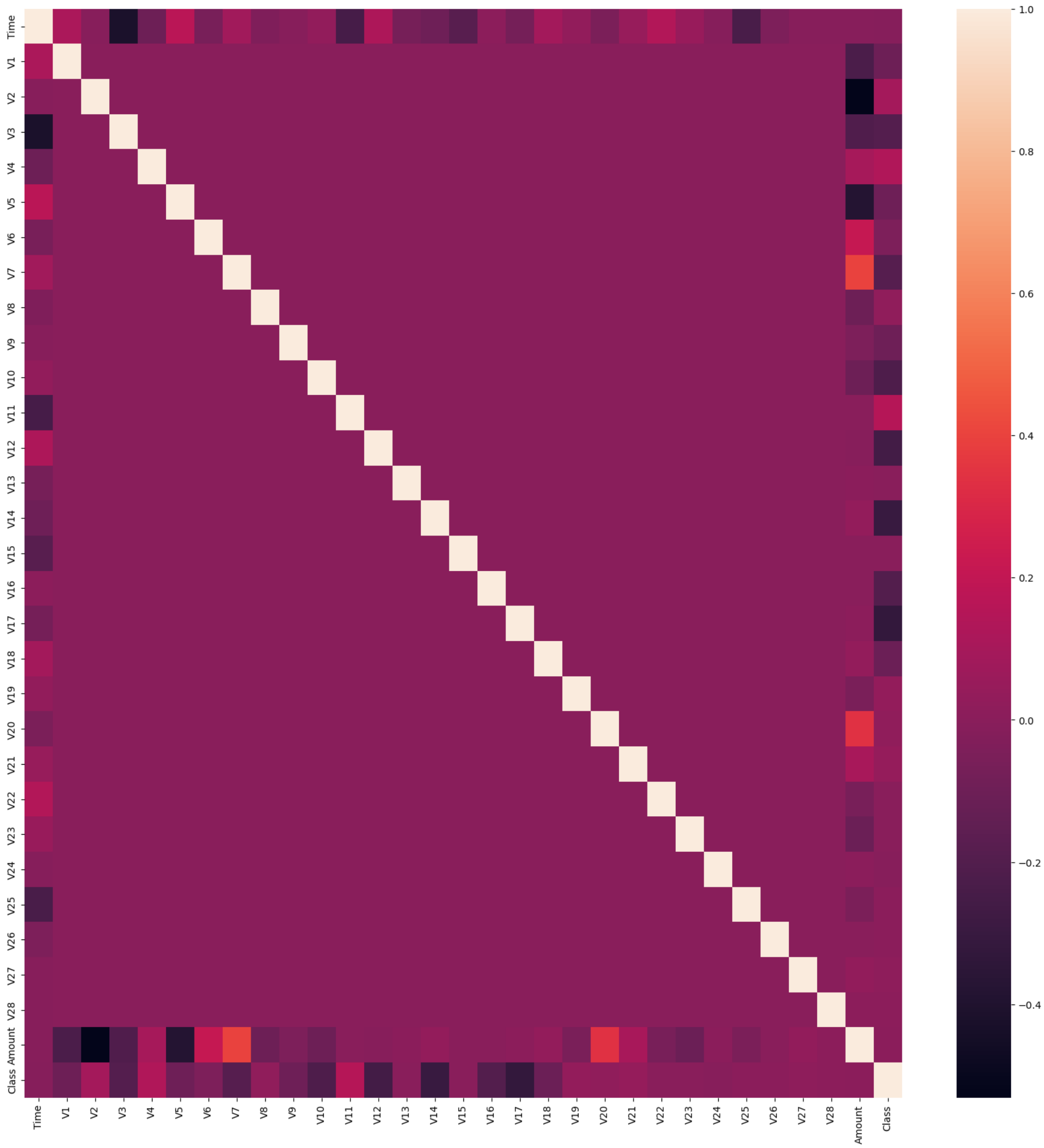
Run GridSearch on the Data

Fine Tune the Classifiers

Create Pipelines for evaluation

```
plt.figure(figsize = (20, 20))
credit_card_data_corr = credit_card_data.corr()
sns.heatmap(credit_card_data_corr)
```

```
<AxesSubplot:>
```



# Now spliting the dataset into Features and Targets

```
X = credit_card_data.drop(labels='Class', axis=1) # Features
Y = credit_card_data.loc[:,'Class']               # Target Variable
```

# Now spliting the dataset into Train data and Test data

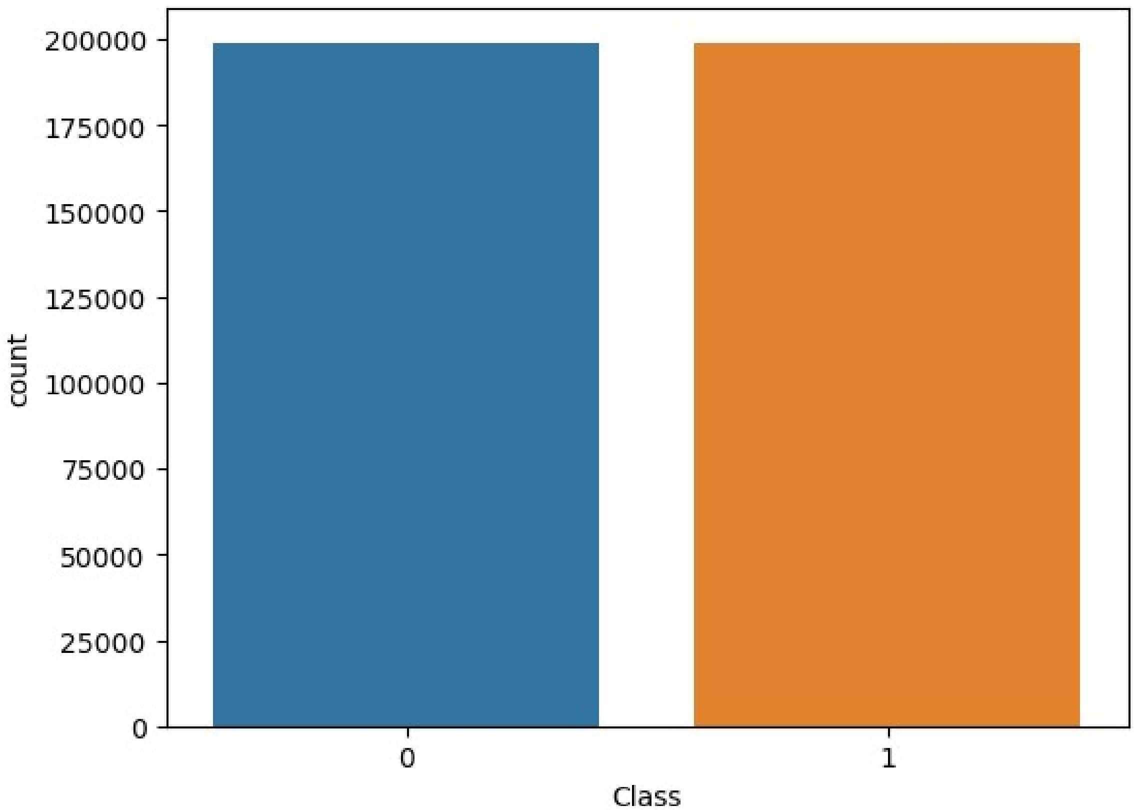
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=1, stratify=Y)
```

# Balancing the fraud and legitimate transactions in data

```
# Use Synthetic Minority Oversampling Techniques.
sm = SMOTE(random_state = 42)
X_res, Y_res = sm.fit_resample(X_train, Y_train)

sns.countplot(Y_res)

<AxesSubplot:xlabel='Class', ylabel='count'>
```




# Evaluation

We make use of AUC-ROC Score, Classification Report, Accuracy and F1-Score to evaluate the performance of the classifiers

```
# Evaluation of Classifiers
def grid_eval(grid_clf):
    """
        Method to Compute the best score and parameters computed by
grid search
        Parameter:
            grid_clf: The Grid Search Classifier
    """
    print("Best Score", grid_clf.best_score_)
    print("Best Parameter", grid_clf.best_params_)

def evaluation(Y_test, grid_clf, X_test):
    """
        Method to compute the following:
            1. Classification Report
            2. F1-score
            3. AUC-ROC score
```

```
            4. Accuracy
        Parameters:
            y_test: The target variable test set
            grid_clf: Grid classifier selected
            X_test: Input Feature Test Set
    """
    Y_pred = grid_clf.predict(X_test)
    print('CLASSIFICATION REPORT')
    print(classification_report(Y_test, Y_pred))

    print('AUC-ROC')
    print(roc_auc_score(Y_test, Y_pred))

    print('F1-Score')
    print(f1_score(Y_test, Y_pred))

    print('Accuracy')
    print(accuracy_score(Y_test, Y_pred))

# The parameters of each classifier are different
# Hence, we do not make use of a single method and this is not to
violate DRY Principles
# We set pipelines for each classifier unique with parameters
param_grid_sgd = [{
    'model__loss': ['log'],
    'model__penalty': ['l1', 'l2'],
    'model__alpha': np.logspace(start=-3, stop=3, num=20)
}, {
    'model__loss': ['hinge'],
    'model__alpha': np.logspace(start=-3, stop=3, num=20),
    'model__class_weight': [None, 'balanced']
}]

pipeline_sgd = Pipeline([
    ('scaler', StandardScaler(copy=False)),
    ('model', SGDClassifier(max_iter=1000, tol=1e-3, random_state=1,
warm_start=True))
])

MCC_scorer = make_scorer(matthews_corrcoef)
grid_sgd = GridSearchCV(estimator=pipeline_sgd,
param_grid=param_grid_sgd,
                        scoring=MCC_scorer, n_jobs=-1,
pre_dispatch='2*n_jobs', cv=5, verbose=1, return_train_score=False)


grid_sgd.fit(X_res, Y_res)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('scaler',
StandardScaler(copy=False)),
                                       ('model',
                                        SGDClassifier(random_state=1,

warm_start=True))]),
             n_jobs=-1,
             param_grid=[{'model__alpha': array([1.00000000e-03,
2.06913808e-03, 4.28133240e-03, 8.85866790e-03,
       1.83298071e-02, 3.79269019e-02, 7.84759970e-02, 1.62377674e-01,
       3.35981829e-01, 6.95192796e-01, 1.43844989e+00,...
       1.83298071e-02, 3.79269019e-02, 7.84759970e-02, 1.62377674e-01,
       3.35981829e-01, 6.95192796e-01, 1.43844989e+00, 2.97635144e+00,
       6.15848211e+00, 1.27427499e+01, 2.63665090e+01, 5.45559478e+01,
       1.12883789e+02, 2.33572147e+02, 4.83293024e+02,
1.00000000e+03]),
                          'model__class_weight': [None, 'balanced'],
                          'model__loss': ['hinge']}],
             scoring=make_scorer(matthews_corrcoef), verbose=1)
```

```
grid_eval(grid_sgd)
```

```
Best Score 0.9560162686072134
Best Parameter {'model__alpha': 0.001, 'model__loss': 'log',
'model__penalty': 'l1'}
```

```
evaluation(Y_test, grid_sgd, X_test)
```

```
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     85295
           1       0.14      0.91      0.25       148

    accuracy                           0.99     85443
   macro avg       0.57      0.95      0.62     85443
weighted avg       1.00      0.99      0.99     85443

AUC-ROC
0.9479720619851928
F1-Score
0.2460973370064279
Accuracy
0.990391254988706
```

```
pipeline_rf = Pipeline([
    ('model', RandomForestClassifier(n_jobs=-1, random_state=1))])
param_grid_rf = {'model__n_estimators': [75]}
grid_rf = GridSearchCV(estimator=pipeline_rf,
param_grid=param_grid_rf,
```

```
                       scoring=MCC_scorer, n_jobs=-1,
pre_dispatch='2*n_jobs', cv=5, verbose=1, return_train_score=False)
grid_rf.fit(X_res, Y_res)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('model',

RandomForestClassifier(n_jobs=-1,

random_state=1))]),
             n_jobs=-1, param_grid={'model__n_estimators': [75]},
             scoring=make_scorer(matthews_corrcoef), verbose=1)
```

```
grid_eval(grid_rf)
```

```
Best Score 0.9997538267139271
Best Parameter {'model__n_estimators': 75}
```

```
evaluation(Y_test, grid_rf, X_test)
```

```
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.90      0.86      0.88       148

    accuracy                           1.00     85443
   macro avg       0.95      0.93      0.94     85443
weighted avg       1.00      1.00      1.00     85443

AUC-ROC
0.9323445023075716
F1-Score
0.879725085910653
Accuracy
0.9995903701883126
```

```
pipeline_lr = Pipeline([
    ('model', LogisticRegression(random_state=1))
])
param_grid_lr = {'model__penalty': ['l2'],
                 'model__class_weight': [None, 'balanced']}
grid_lr = GridSearchCV(estimator=pipeline_lr,
param_grid=param_grid_lr,
                       scoring=MCC_scorer, n_jobs=-1,
pre_dispatch='2*n_jobs', cv=5, verbose=1, return_train_score=False)
grid_lr.fit(X_res, Y_res)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('model',

LogisticRegression(random_state=1))]),
             n_jobs=-1,
             param_grid={'model__class_weight': [None, 'balanced'],
                         'model__penalty': ['l2']},
             scoring=make_scorer(matthews_corrcoef), verbose=1)
```

```
grid_eval(grid_lr)
```

```
Best Score 0.959816277887179
Best Parameter {'model__class_weight': None, 'model__penalty': 'l2'}
```

```
evaluation(Y_test, grid_lr, X_test)
```

```
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     85295
           1       0.15      0.91      0.26       148

    accuracy                           0.99     85443
   macro avg       0.57      0.95      0.63     85443
weighted avg       1.00      0.99      0.99     85443

AUC-ROC
0.948212404326479
F1-Score
0.2557251908396946
Accuracy
0.9908711070538253
```

# Conclusion

The K-Nearest Neighbors Classifier tuned with Grid Search with the best parameter and its counterparts to give a test accuracy of nearly 99.9% and a perfect F1-Score with minimal overfitting