In [1]:
```
pip install mlxtend
```

```
Requirement already satisfied: mlxtend in c:\users\varsha\anaconda3\lib\site-packages (0.23.1)Note: you may need to res
tart the kernel to use updated packages.

Requirement already satisfied: scipy>=1.2.1 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (1.9.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (1.21.5)
Requirement already satisfied: pandas>=0.24.2 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (1.4.4)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (1.0.
2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (3.5.2)
Requirement already satisfied: joblib>=0.13.2 in c:\users\varsha\anaconda3\lib\site-packages (from mlxtend) (1.3.2)
Requirement already satisfied: cycler>=0.10 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlx
tend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0
->mlxtend) (1.4.2)
Requirement already satisfied: packaging>=20.0 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0->
mlxtend) (21.3)
Requirement already satisfied: pillow>=6.2.0 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0->ml
xtend) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.0.0-
>mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\varsha\anaconda3\lib\site-packages (from matplotlib>=3.
0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\varsha\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxten
d) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\varsha\anaconda3\lib\site-packages (from scikit-learn>=
1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\varsha\anaconda3\lib\site-packages (from python-dateutil>=2.7->matp
lotlib>=3.0.0->mlxtend) (1.16.0)
[notice] A new release of pip is available: 23.3.1 -> 24.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [2]:
```
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\varsha\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\varsha\anaconda3\lib\site-packages (from imblearn) (0.12.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\varsha\anaconda3\lib\site-packages (from imbalanced-learn->imb
learn) (1.21.5)
Requirement already satisfied: scipy>=1.5.0 in c:\users\varsha\anaconda3\lib\site-packages (from imbalanced-learn->imbl
earn) (1.9.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\varsha\anaconda3\lib\site-packages (from imbalanced-lear
n->imblearn) (1.0.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\varsha\anaconda3\lib\site-packages (from imbalanced-learn->imb
learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\varsha\anaconda3\lib\site-packages (from imbalanced-lea
rn->imblearn) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 23.3.1 -> 24.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [50]:
```
# Importing Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.svm import SVC

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_repo
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import make_scorer, matthews_corrcoef

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

In [4]:
```
# Read Data into a Dataframe
df = pd.read_csv('creditcard.csv')
```

In [5]:
```
df
```

Out[5]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078

284807 rows × 31 columns

In [6]:
```
# Describe Data
df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-0
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+0

8 rows × 31 columns

In [7]:
```
df.columns
```

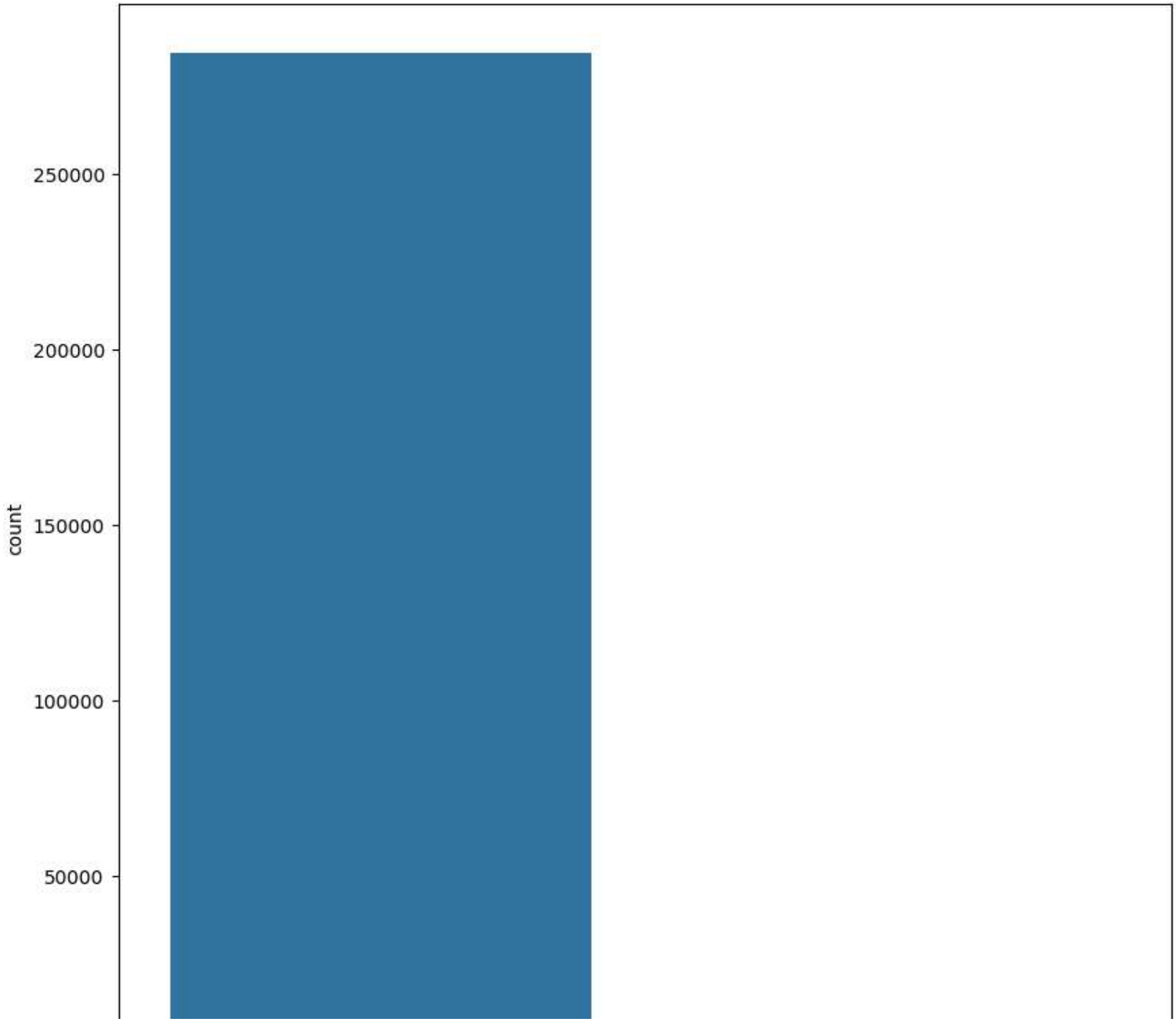
Out[7]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [8]:
```
df.isna().sum()
```

Out[8]:
```
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
```

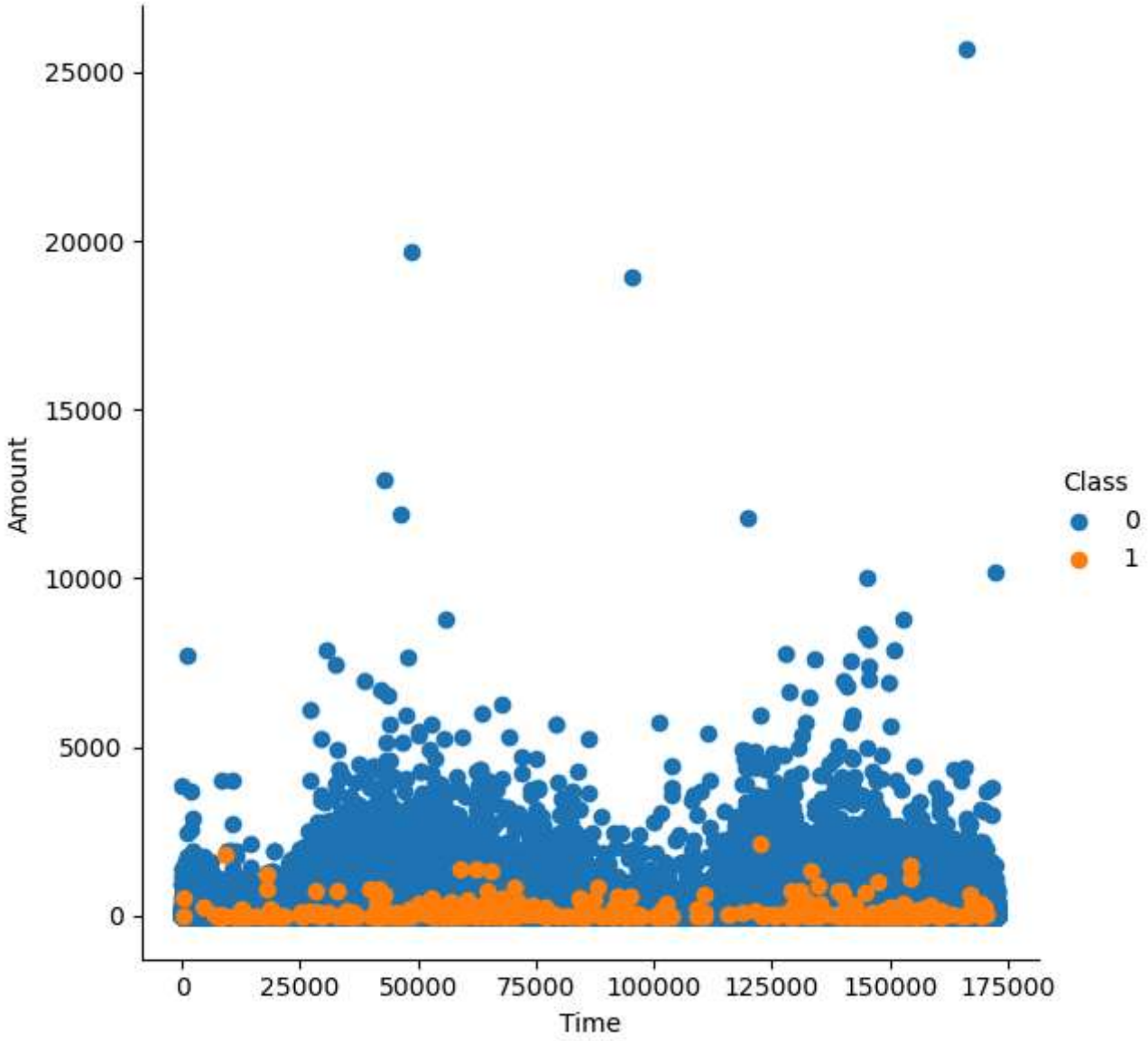
In [9]:
```
def countplot_data(data, feature):
    '''
        Method to compute countplot of given dataframe
        Parameters:
            data(pd.Dataframe): Input Dataframe
            feature(str): Feature in Dataframe
    '''
    plt.figure(figsize=(10,10))
    sns.countplot(x=feature, data=data)
    plt.show()
```

In [10]:
```
countplot_data(df, df.Class)
```

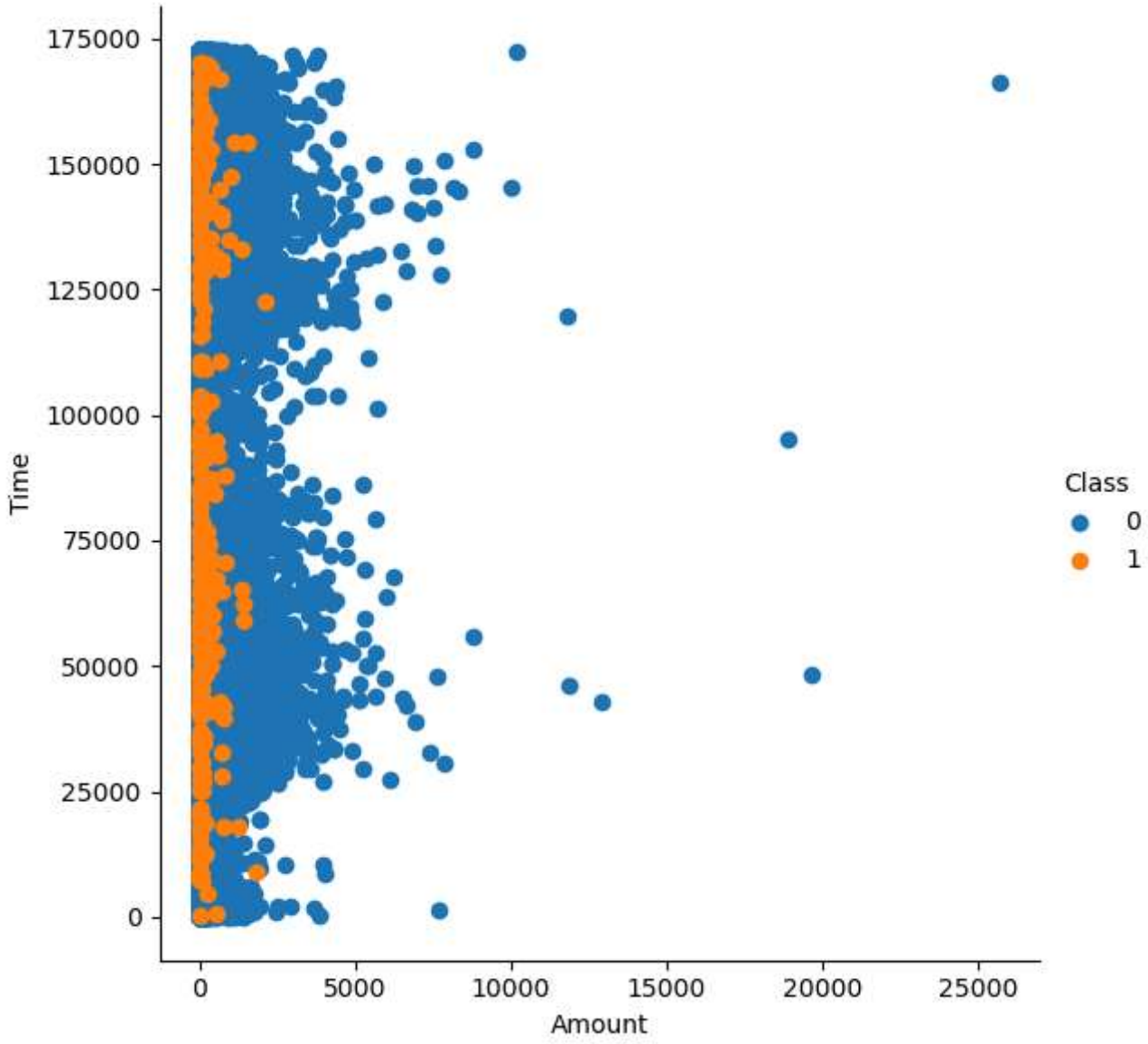

250000
200000
150000
100000
50000
count

In [11]:
```
def pairplot_data_grid(data, feature1, feature2, target):
    '''
        Method to construct pairplot of the given feature wrt data
        Parameters:
            data(pd.DataFrame): Input Dataframe
            feature1(str): First Feature for Pair Plot
            feature2(str): Second Feature for Pair Plot
            target: Target or Label (y)
    '''

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

In [12]:
```
pairplot_data_grid(df, "Time", "Amount", "Class")
```



In [13]:
```
pairplot_data_grid(df, "Amount", "Time", "Class")
```



In [14]:
```
df_refine = df.copy() # Creating a copy of df for refinement

amount_more = 0
amount_less = 0
for i in range(df_refine.shape[0]):
    if df_refine.iloc[i]["Amount"] < 2500:
        amount_less += 1
    else:
```

```
        amount_more += 1

 print(amount_more)
print(amount_less)
```

```
449
284358
```

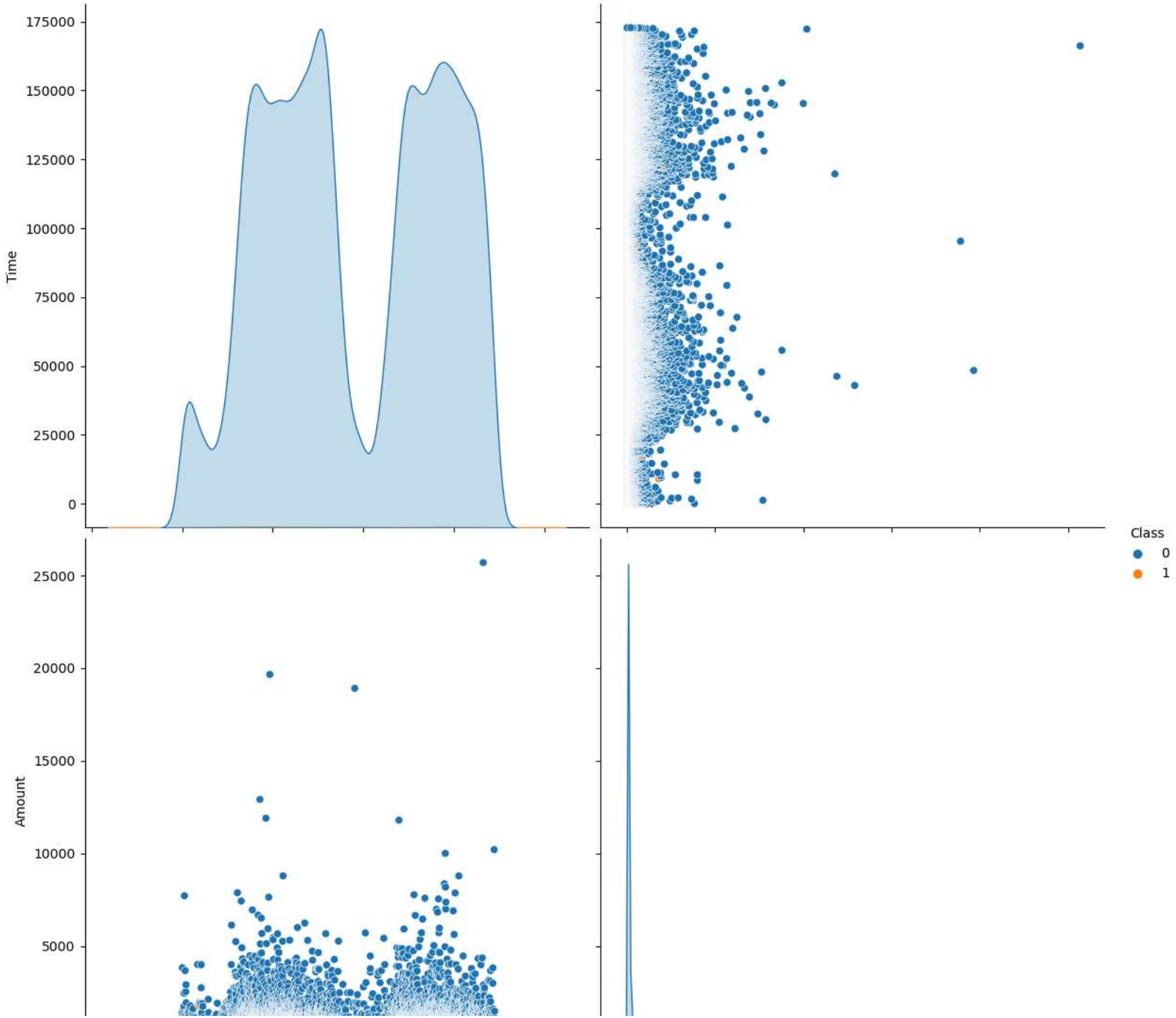
In [15]:
```
percentage_less = (amount_less/df.shape[0])*100
percentage_less
```

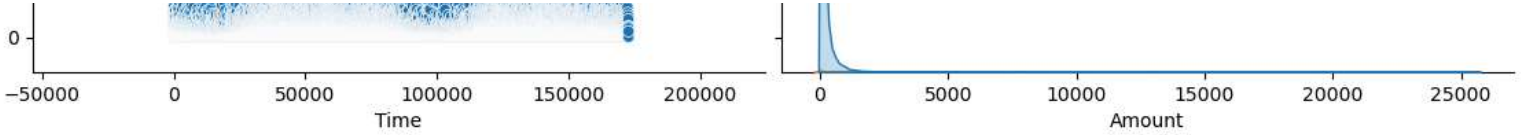
Out[15]:
```
99.84234938045763
```

In [16]:
```
fraud = 0
legitimate = 1
for i in range(df_refine.shape[0]):
    if(df_refine.iloc[i]["Amount"]<2500):
        if(df_refine.iloc[i]["Class"] == 0):
            legitimate += 1
        else:
            fraud+=1
print(fraud)
print(legitimate)
```

```
492
283867
```

In [17]:
```
df_refine = df[["Time", "Amount", "Class"]]
sns.pairplot(df_refine, hue="Class", size=6)
plt.show()
```

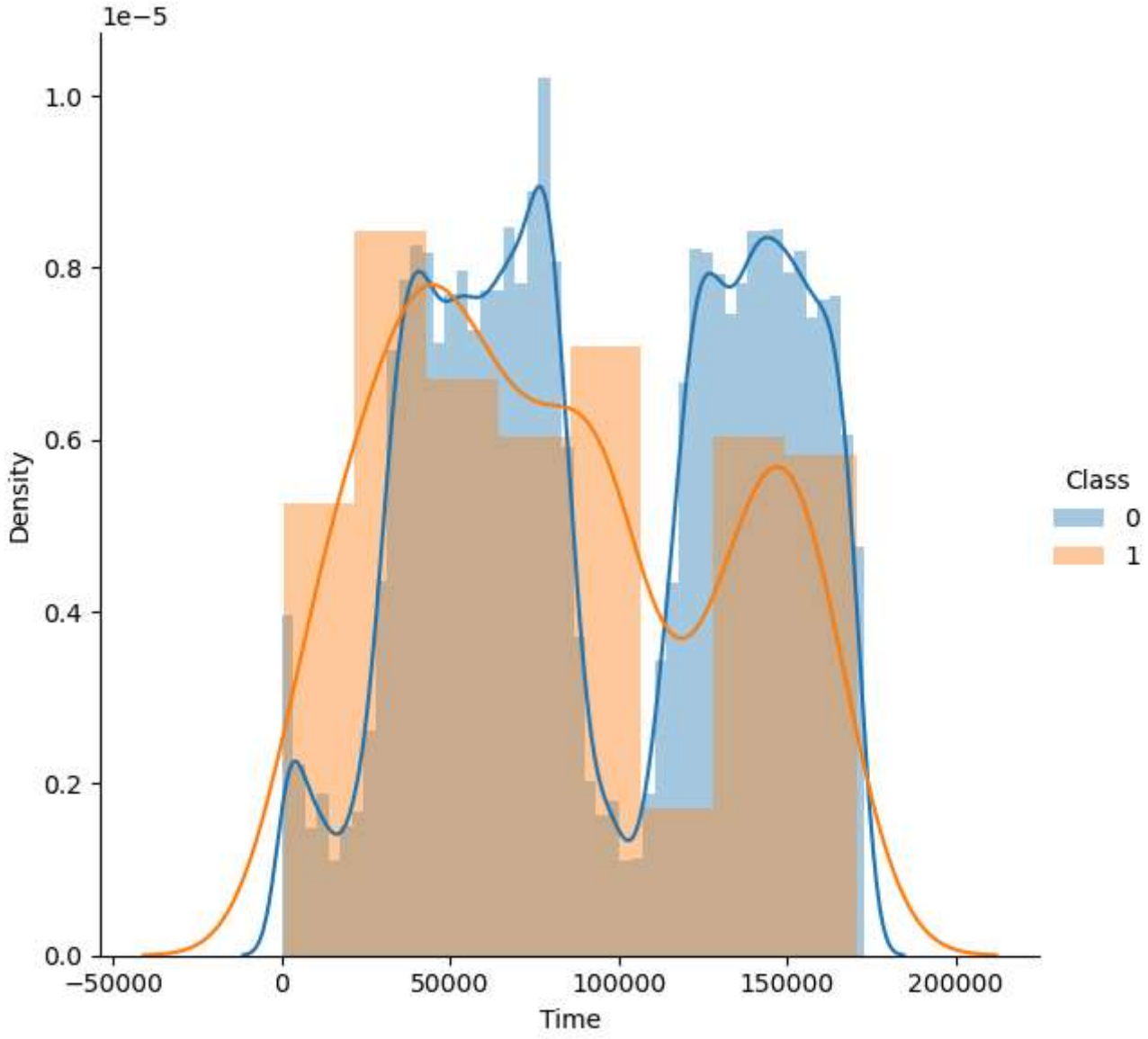

Time
175000
150000
125000
100000
75000
50000
25000
0
Amount
25000
20000
15000
10000
5000
Class
0
1



In [18]:
```
df.Class.value_counts()
```

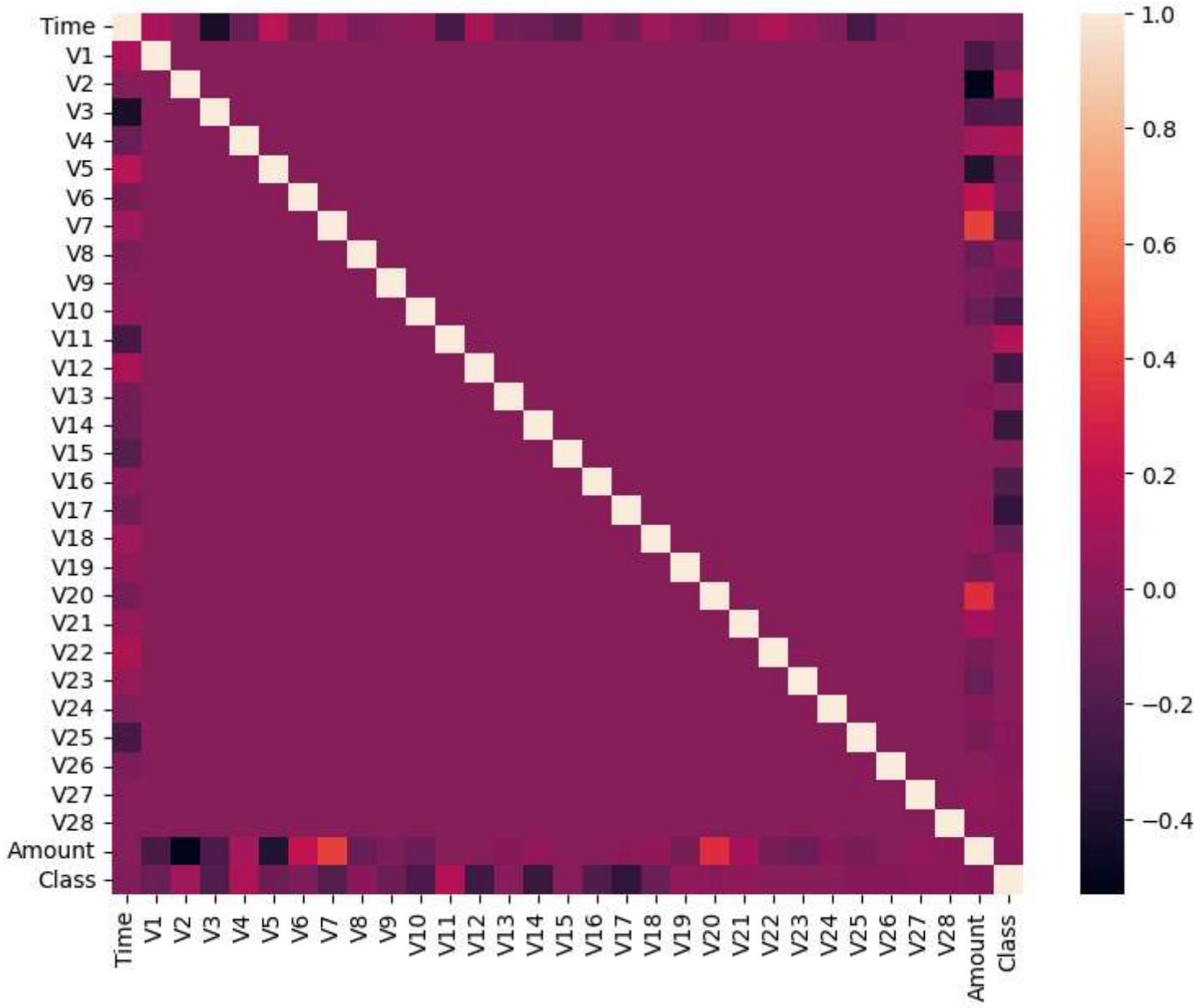
Out[18]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [19]:
```
sns.FacetGrid(df_refine, hue="Class", size=6).map(sns.distplot,"Time").add_legend()
plt.show()
```



In [51]:
```
plt.figure(figsize=(9,7))
df_corr = df.corr()
sns.heatmap(df_corr)
```

Out[51]:
```
<AxesSubplot:>
```



In [21]:
```
# Create Train and Test Data in ratio 70:30
X = df.drop(labels='Class', axis=1) # Features
y = df.loc[:,'Class']               # Target Variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

In [22]:
```
# Use Synthetic Minority Oversampling
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

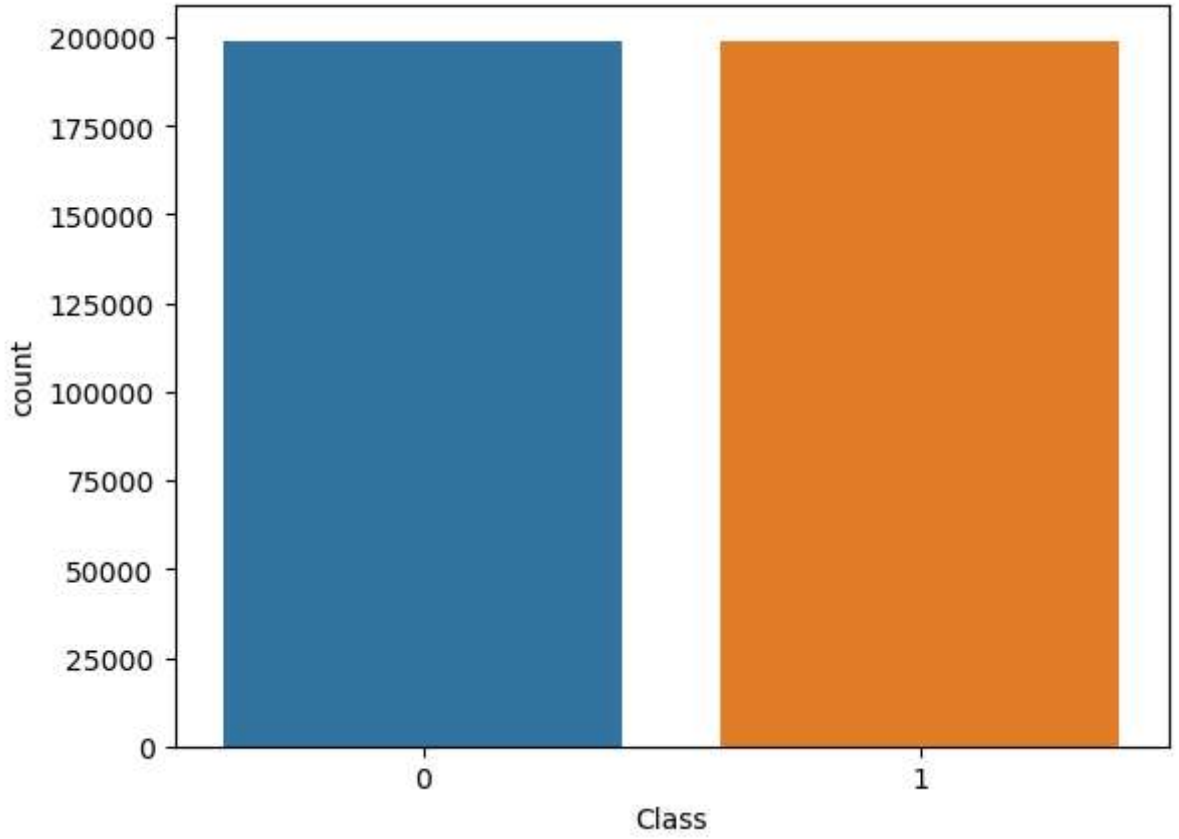
In [23]:
```
from sklearn.feature_selection import mutual_info_classif
mutual_infos = pd.Series(data=mutual_info_classif(X_res, y_res, discrete_features=False, random_state=1), index=X_train
```

In [24]:
```
mutual_infos.sort_values(ascending=False)
```

Out[24]:
```
V14       0.535037
V10       0.464777
V12       0.456051
V17       0.438193
V4        0.427426
V11       0.404044
Amount    0.392941
V3        0.387191
V16       0.335318
V7        0.304175
V2        0.291492
V9        0.256679
Time      0.247989
V21       0.235031
V27       0.229915
V1        0.220743
V18       0.198264
V8        0.174393
V6        0.171974
V28       0.170493
V5        0.157362
V20       0.107488
V19       0.099837
V23       0.067332
V24       0.063567
V26       0.046973
V25       0.031607
V22       0.031539
V13       0.024931
V15       0.022442
dtype: float64
```

In [25]:
```
sns.countplot(y_res)
```

Out[25]:
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```

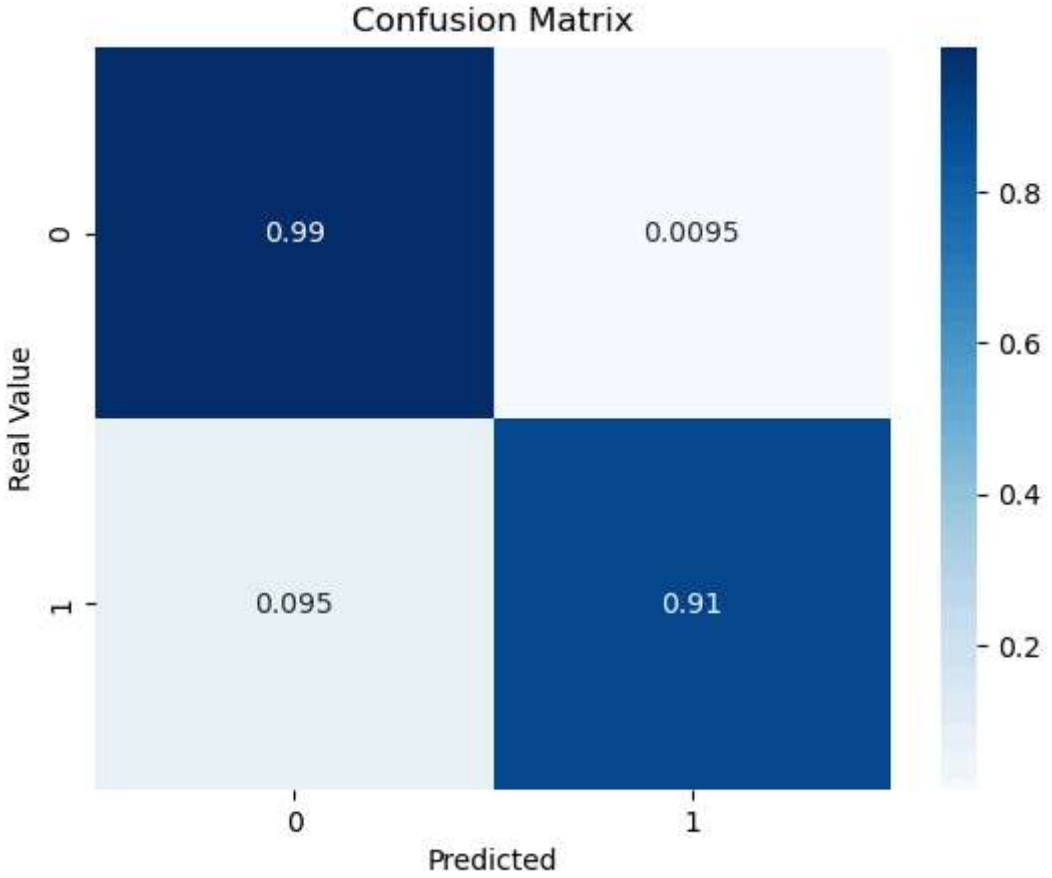


In [49]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    f1_score,
    accuracy_score,
    make_scorer,
    matthews_corrcoef,
    confusion_matrix
)
```

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

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 Parameters", 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))

    return y_pred

# Define parameters for SGDClassifier
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=

# Assuming X_res and y_res are defined and available
grid_sgd.fit(X_res, y_res)

grid_eval(grid_sgd)
y_pred = evaluation(y_test, grid_sgd, X_test)

# Confusion matrix
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred, normalize='true'), annot=True, ax=ax, cmap='Blues')

ax.set_title("Confusion Matrix")
ax.set_ylabel("Real Value")
ax.set_xlabel("Predicted")

plt.show()
```

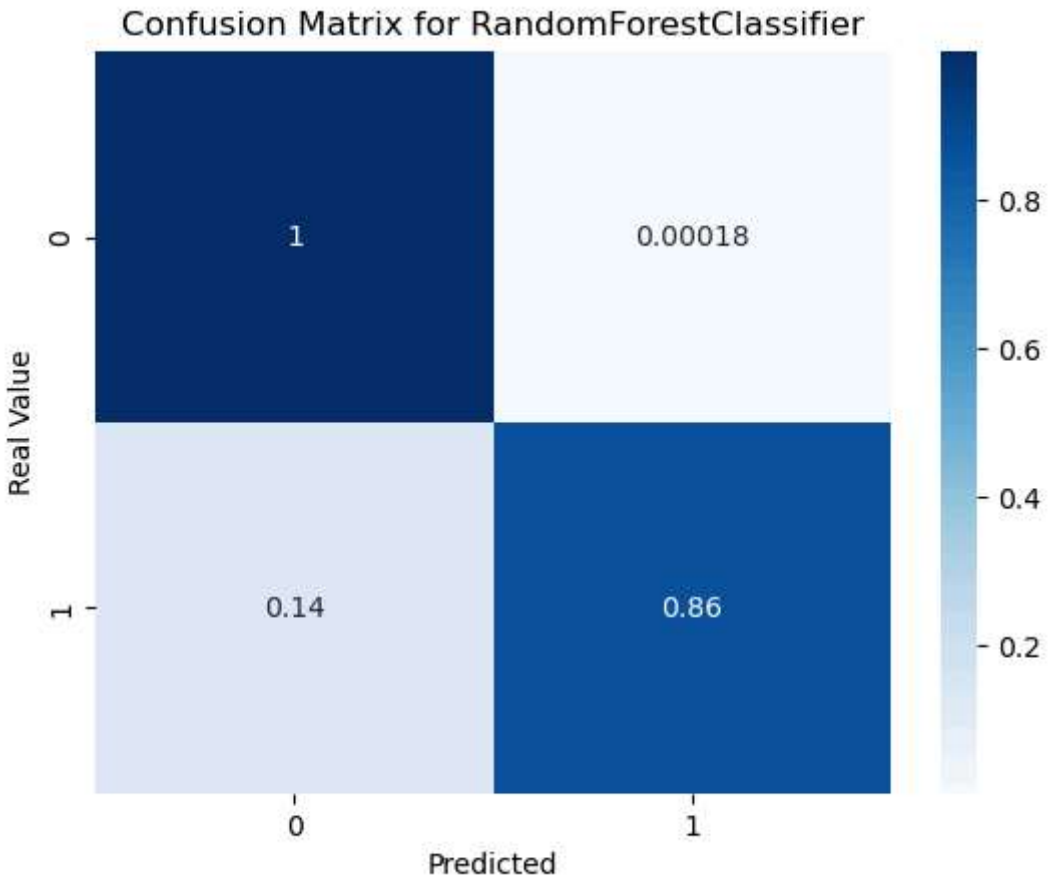
```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
Best Score 0.9560162686072134
Best Parameters {'model__alpha': 0.001, 'model__loss': 'log', 'model__penalty': 'l1'}
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           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
```



In [46]:
```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    f1_score,
    accuracy_score,
    make_scorer,
    matthews_corrcoef,
    confusion_matrix
)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Define the pipeline and parameter grid for RandomForestClassifier
pipeline_rf = Pipeline([
    ('model', RandomForestClassifier(n_jobs=-1, random_state=1))
])
param_grid_rf = {'model__n_estimators': [75]}

MCC_scorer = make_scorer(matthews_corrcoef)
grid_rf = GridSearchCV(estimator=pipeline_rf, param_grid=param_grid_rf, scoring=MCC_scorer, n_jobs=-1, pre_dispatch='2*

# Assuming X_res and y_res are defined and available
grid_rf.fit(X_res, y_res)

# Evaluate the grid search results
grid_eval(grid_rf)
y_pred_rf = evaluation(y_test, grid_rf, X_test)

# Confusion matrix for RandomForestClassifier
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred_rf, normalize='true'), annot=True, ax=ax, cmap='Blues')

ax.set_title("Confusion Matrix for RandomForestClassifier")
ax.set_ylabel("Real Value")
ax.set_xlabel("Predicted")

plt.show()
```

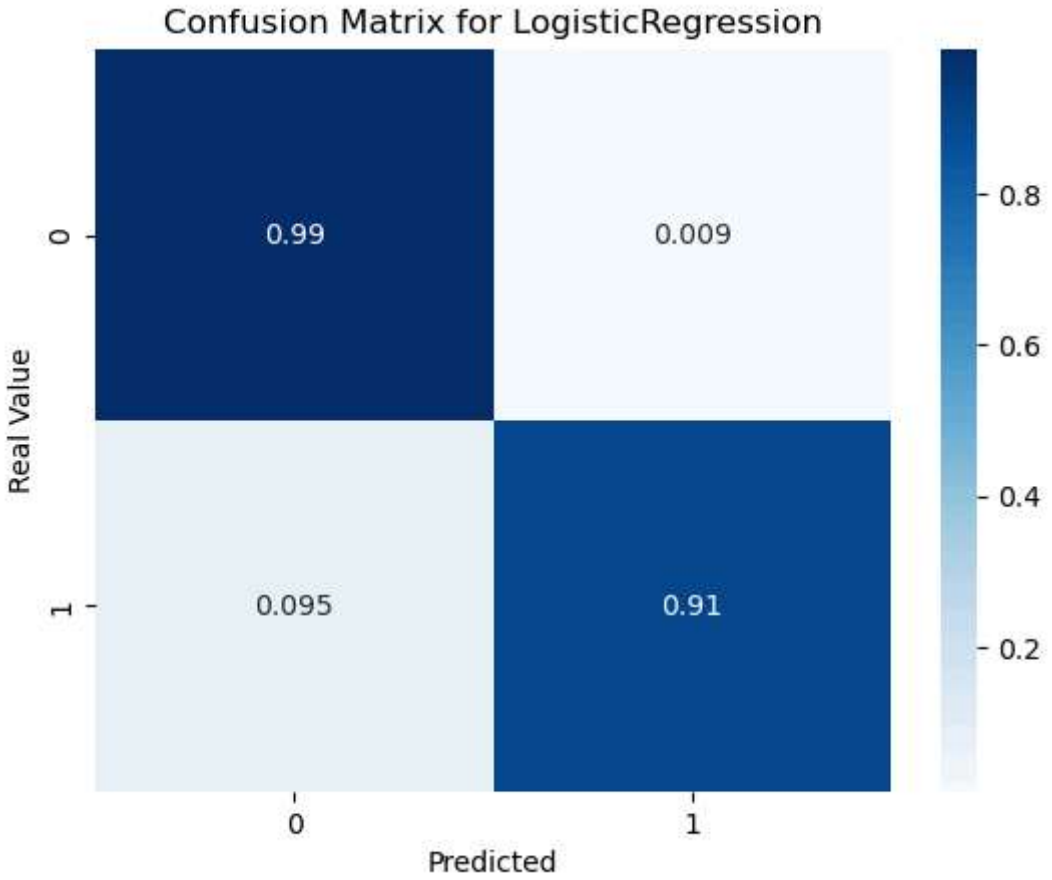
```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best Score 0.9997538267139271
Best Parameters {'model__n_estimators': 75}
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           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
```



In [47]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    f1_score,
    accuracy_score,
    make_scorer,
    matthews_corrcoef,
    confusion_matrix
)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Define the pipeline and parameter grid for LogisticRegression
pipeline_lr = Pipeline([
    ('model', LogisticRegression(random_state=1))
])
param_grid_lr = {
    'model__penalty': ['l2'],
    'model__class_weight': [None, 'balanced']
}

MCC_scorer = make_scorer(matthews_corrcoef)
grid_lr = GridSearchCV(estimator=pipeline_lr, param_grid=param_grid_lr, scoring=MCC_scorer, n_jobs=-1, pre_dispatch='2*

# Assuming X_res and y_res are defined and available
grid_lr.fit(X_res, y_res)

# Evaluate the grid search results
grid_eval(grid_lr)
y_pred_lr = evaluation(y_test, grid_lr, X_test)

# Confusion matrix for LogisticRegression
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred_lr, normalize='true'), annot=True, ax=ax, cmap='Blues')

ax.set_title("Confusion Matrix for LogisticRegression")
ax.set_ylabel("Real Value")
ax.set_xlabel("Predicted")

plt.show()
```

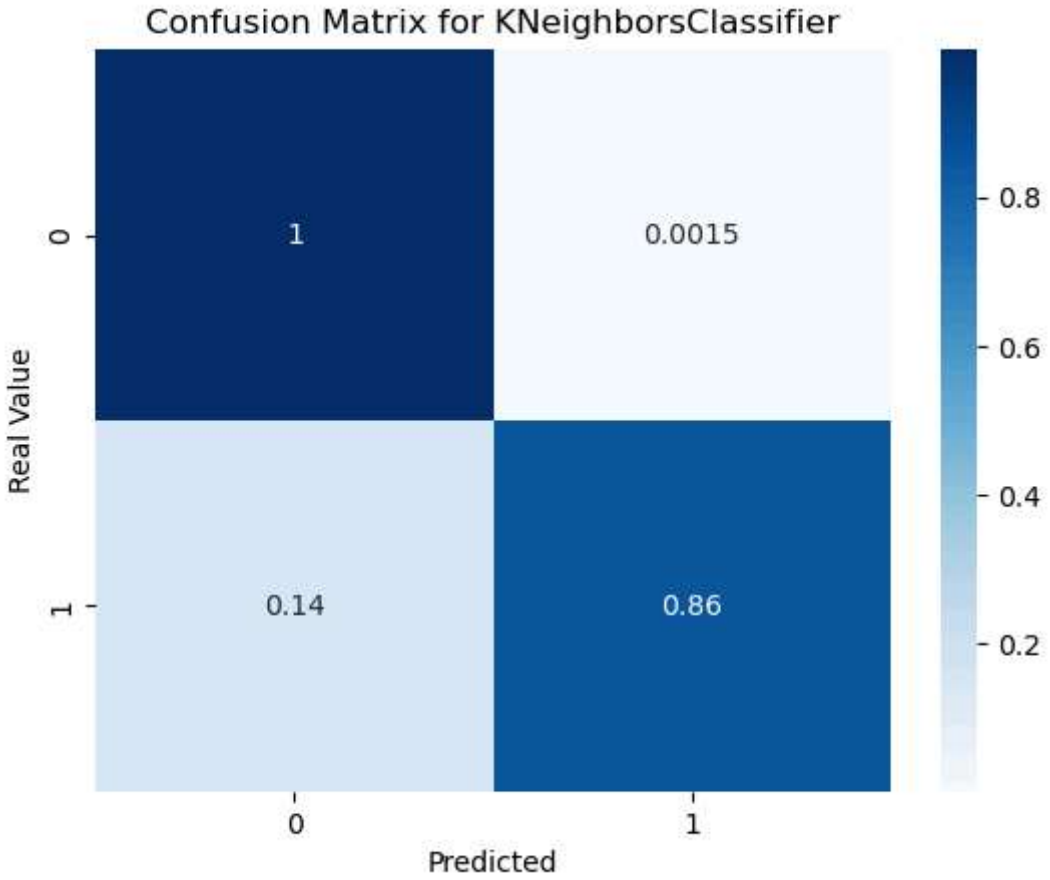
```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
Best Score 0.959816277887179
Best Parameters {'model__class_weight': None, 'model__penalty': 'l2'}
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           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
```



In [48]:
```
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    f1_score,
    accuracy_score,
    make_scorer,
    matthews_corrcoef,
    confusion_matrix
)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Define the pipeline and parameter grid for KNeighborsClassifier
pipeline_knn = Pipeline([
    ('model', KNeighborsClassifier(n_neighbors=5))
])
param_grid_knn = {'model__p': [2]}

MCC_scorer = make_scorer(matthews_corrcoef)
grid_knn = GridSearchCV(estimator=pipeline_knn, param_grid=param_grid_knn, scoring=MCC_scorer, n_jobs=-1, pre_dispatch=

# Assuming X_res and y_res are defined and available
grid_knn.fit(X_res, y_res)

# Evaluate the grid search results
grid_eval(grid_knn)
y_pred_knn = evaluation(y_test, grid_knn, X_test)

# Confusion matrix for KNeighborsClassifier
fig, ax = plt.subplots()
sns.heatmap(confusion_matrix(y_test, y_pred_knn, normalize='true'), annot=True, ax=ax, cmap='Blues')

ax.set_title("Confusion Matrix for KNeighborsClassifier")
ax.set_ylabel("Real Value")
ax.set_xlabel("Predicted")

plt.show()
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best Score 0.9980623930056313
Best Parameters {'model__p': 2}
CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.50      0.86      0.63       148

    accuracy                           1.00     85443
   macro avg       0.75      0.93      0.82     85443
weighted avg       1.00      1.00      1.00     85443

AUC-ROC
0.9283095789968995
F1-Score
0.6318407960199005
Accuracy
0.9982678510820079
```


Confusion Matrix for KNeighborsClassifier
1
0.0015
0.14
0.86
0
1
Real Value
0
1
Predicted
0.8
0.6
0.4
0.2

In [ ]: