Importing required libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

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

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]
```

```
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
```

```
data.shape
```

```
(284807, 31)
```

```
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]
```

Fraudulent & Non-Fraudulent data

```
fraud = data[data['Class']==1]
normal = data[data['Class']==0]
```

```
data.size
```

```
8829017
```

```
len(fraud)
```

```
492
```

```
len(normal)
```

```
284315
```

```
data.Amount.unique()
```

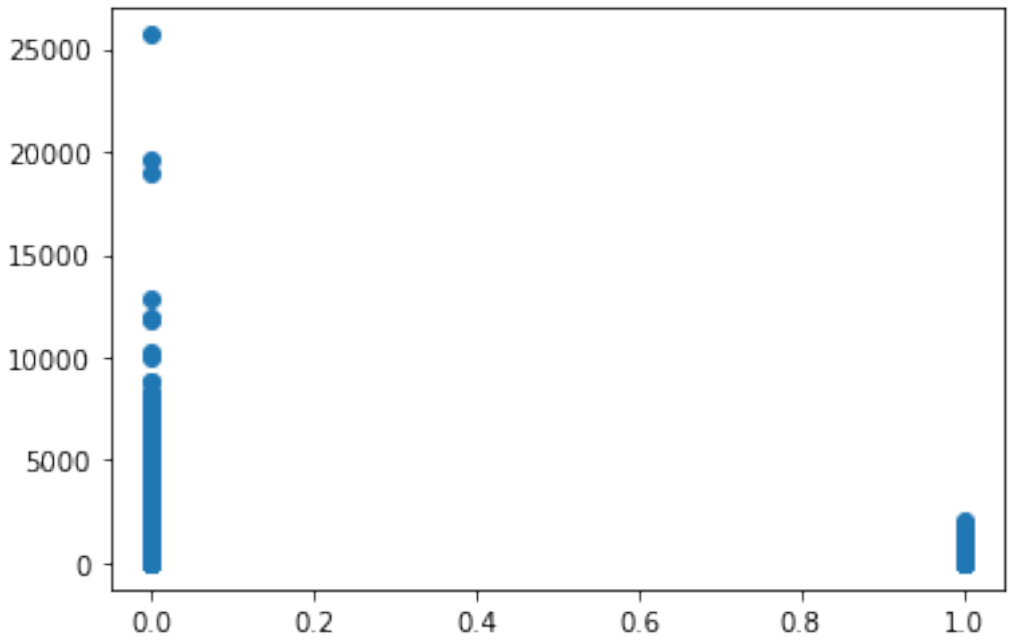
```
array([149.62,   2.69, 378.66, ..., 381.05, 337.54,  95.63])
```

```
data['Class'].value_counts()
```

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

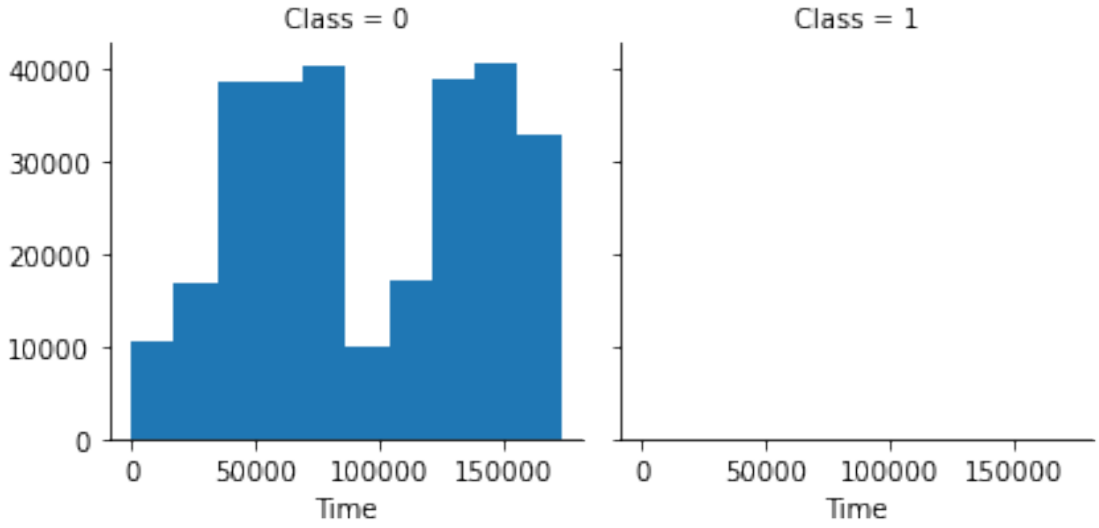
Data Visualization

```
plt.scatter('Class', 'Amount', c=None, cmap='rainbow', data=data)
plt.show()
```

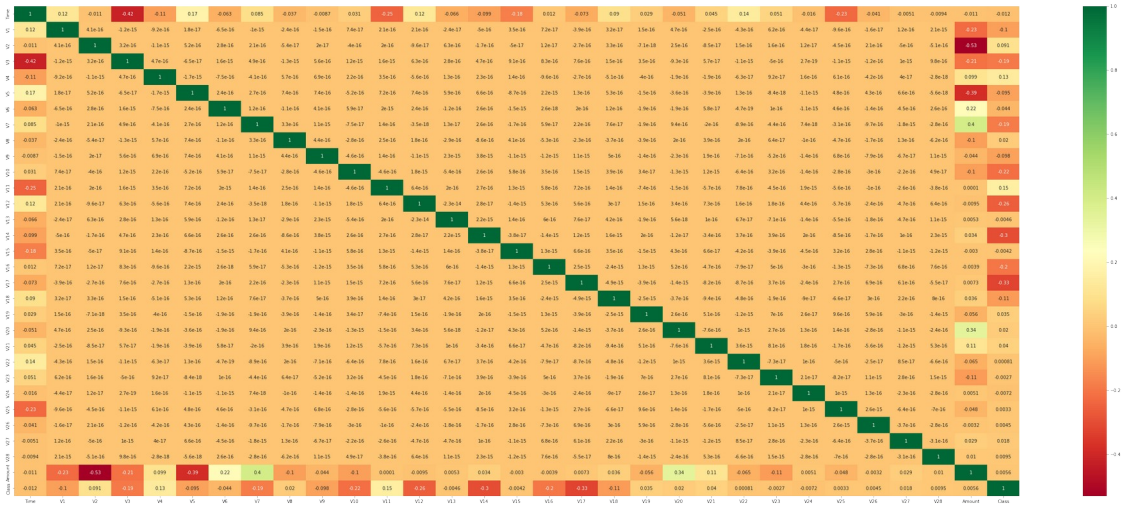



```
a=sns.FacetGrid(data,col="Class")
a.map(plt.hist,"Time")
```

```
<seaborn.axisgrid.FacetGrid at 0x16e70449280>
```




```
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(50,20))
g = sns.heatmap(data[top_corr_features].corr(),annot=True,
cmap="RdYlGn")
```



```
class_count = data['Class'].value_counts()
class_count
```

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

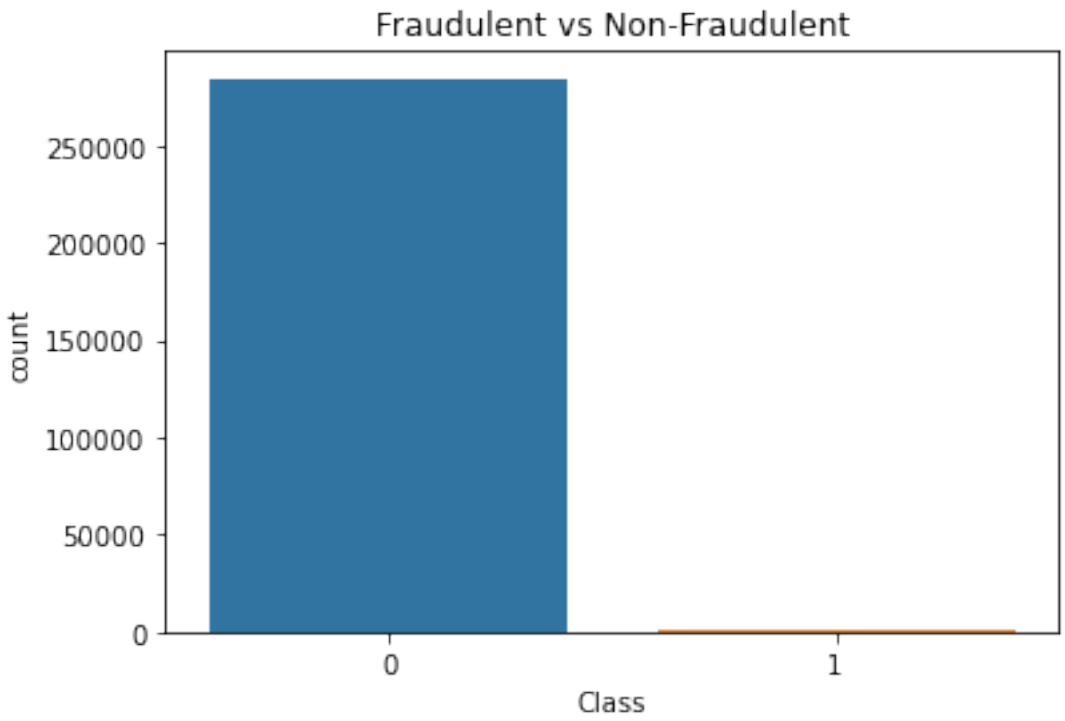
```
normal_share = round((class_count[0]/data['Class'].count()*100),2)
normal_share
```

```
99.83
```

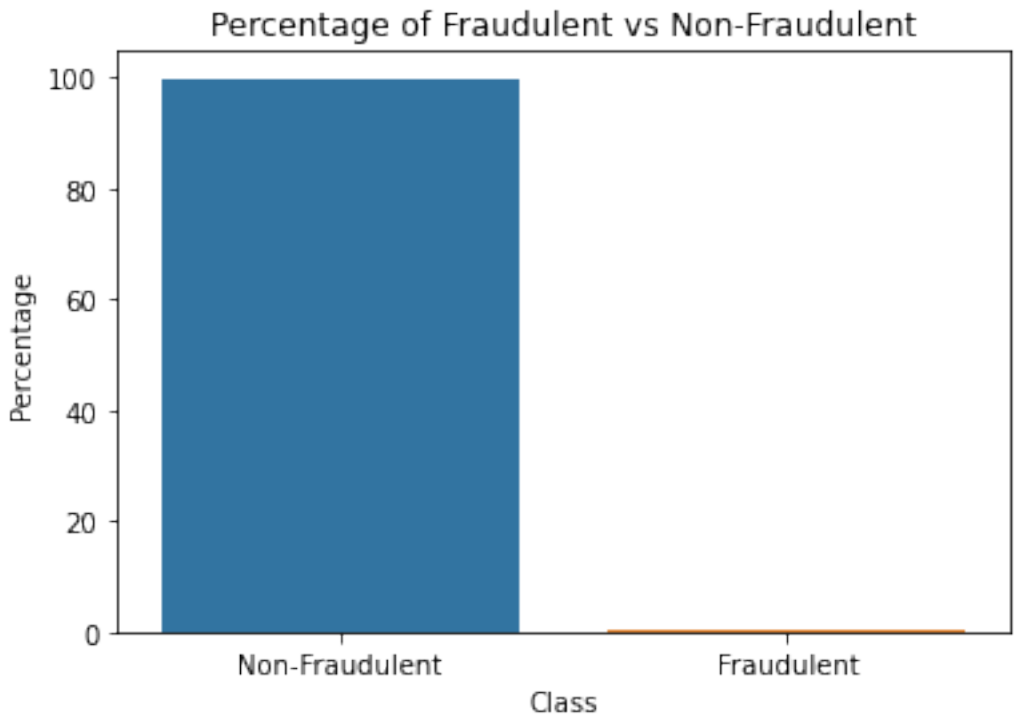
```
fraud_share = round((class_count[1]/data['Class'].count()*100),2)
fraud_share
```

```
0.17
```

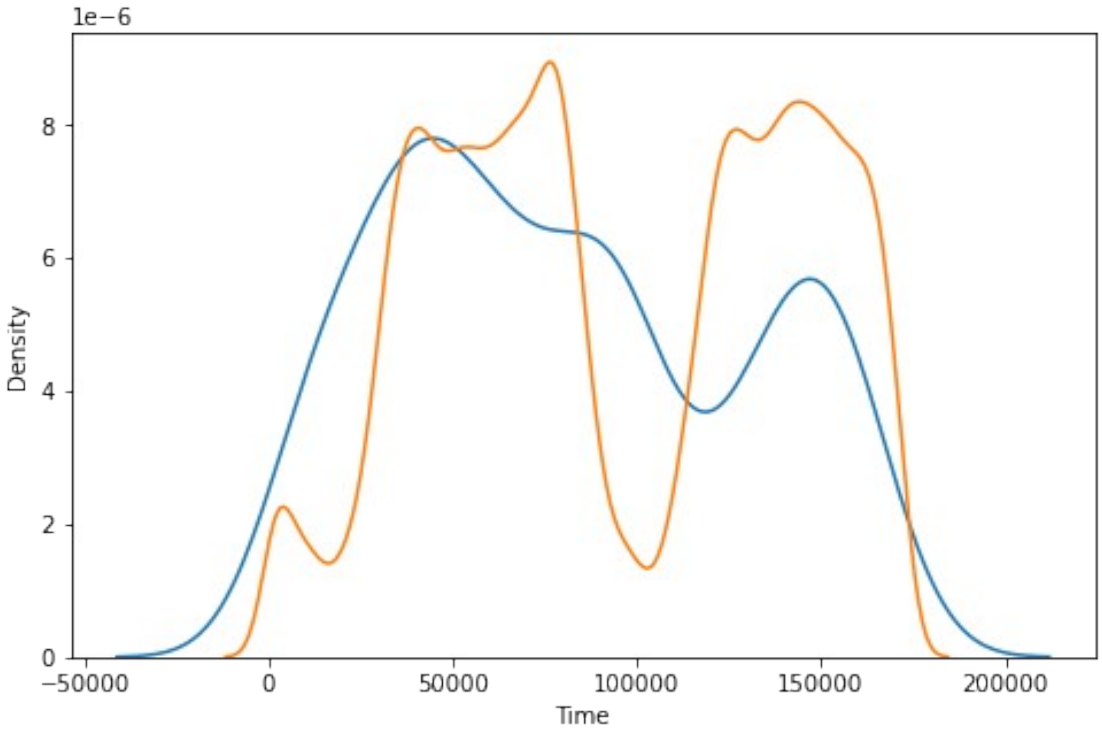
```
sns.countplot(x='Class', data=data)
plt.title("Fraudulent vs Non-Fraudulent")
plt.show()
```



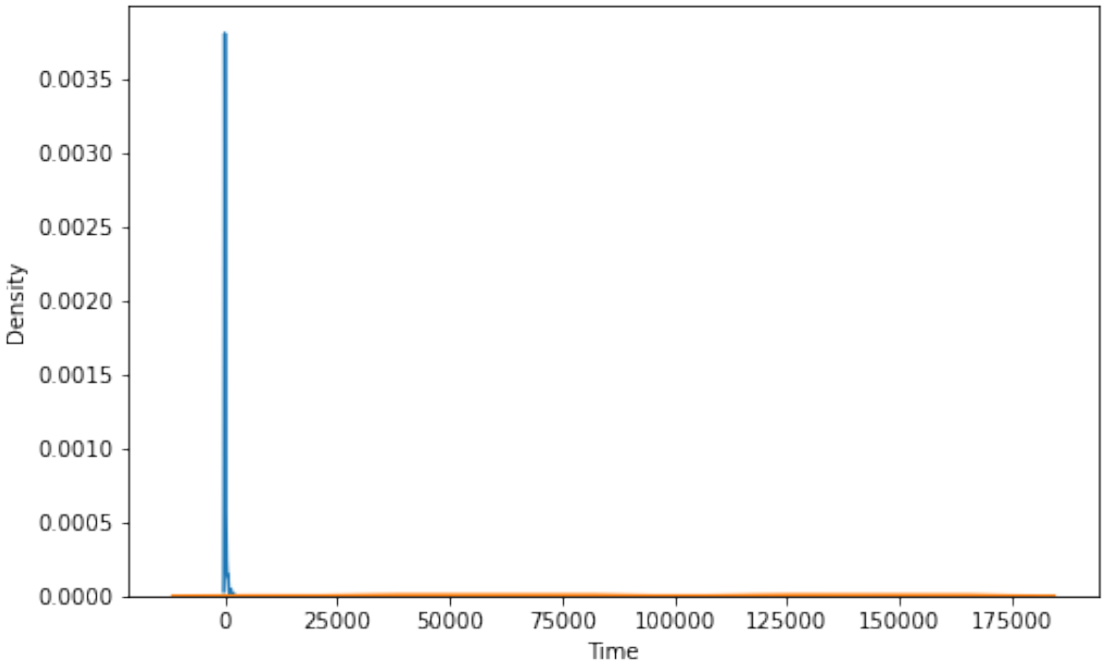

```
fraud_percentage = {'Class':['Non-Fraudulent', 'Fraudulent'],
'Percentage':[normal_share, fraud_share]}
data_fraud_percentage = pd.DataFrame(fraud_percentage)
sns.barplot(x='Class', y='Percentage', data=data_fraud_percentage)
plt.title("Percentage of Fraudulent vs Non-Fraudulent")
plt.show()
```




```
plt.figure(figsize=(8,5))
ax = sns.distplot(fraud['Time'], label='fraudulent', hist=False)
ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
plt.show()
```

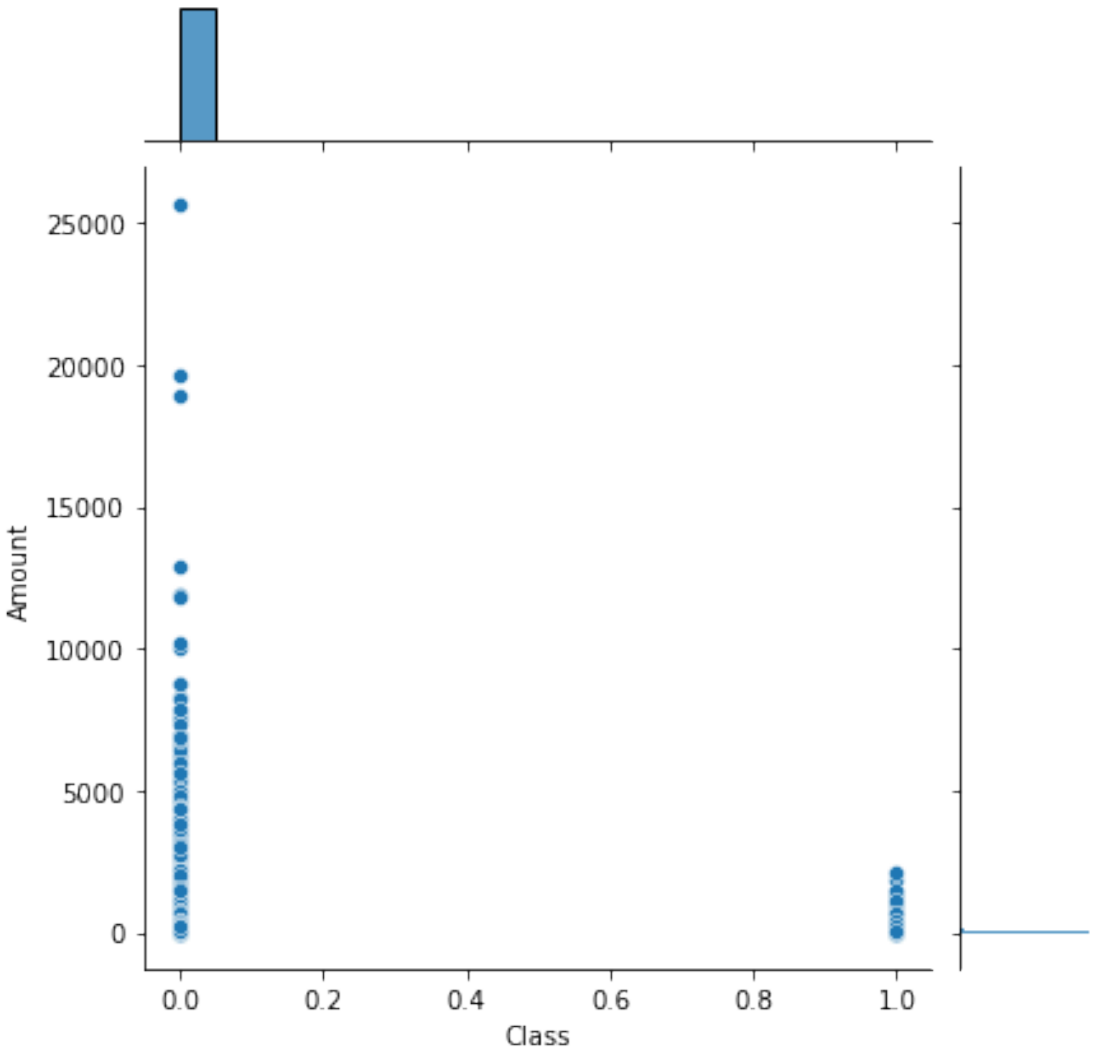


```
plt.figure(figsize=(8,5))
ax = sns.distplot(fraud['Amount'], label='fraudulent', hist=False)
ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
plt.show()
```

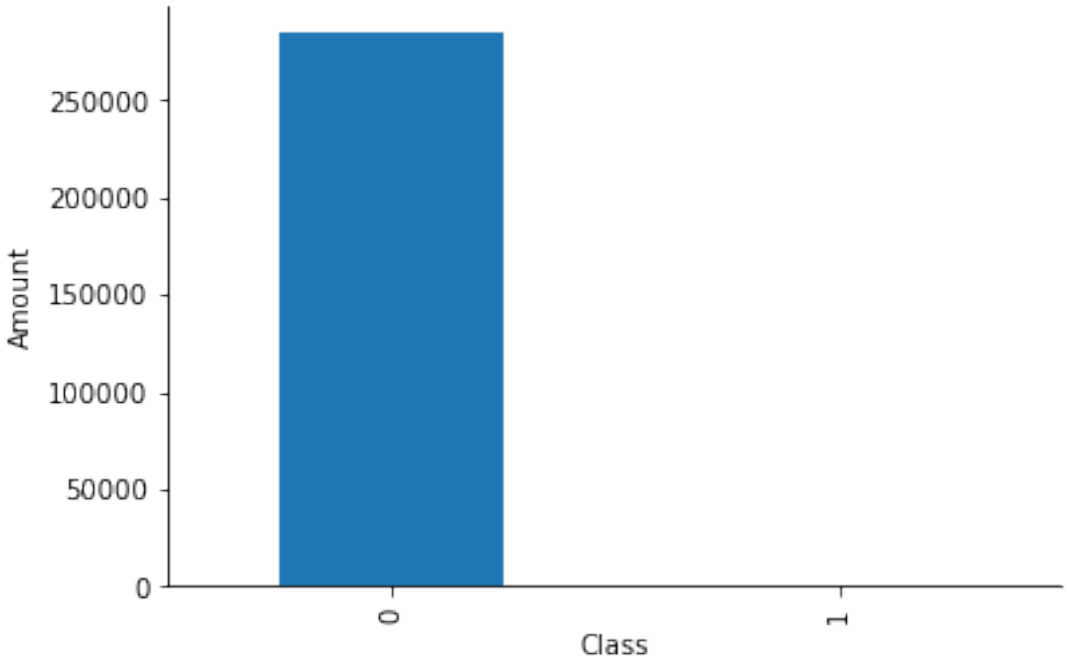



```
sns.jointplot('Class','Amount',data=data)
```

```
<seaborn.axisgrid.JointGrid at 0x16e08438dc0>
```

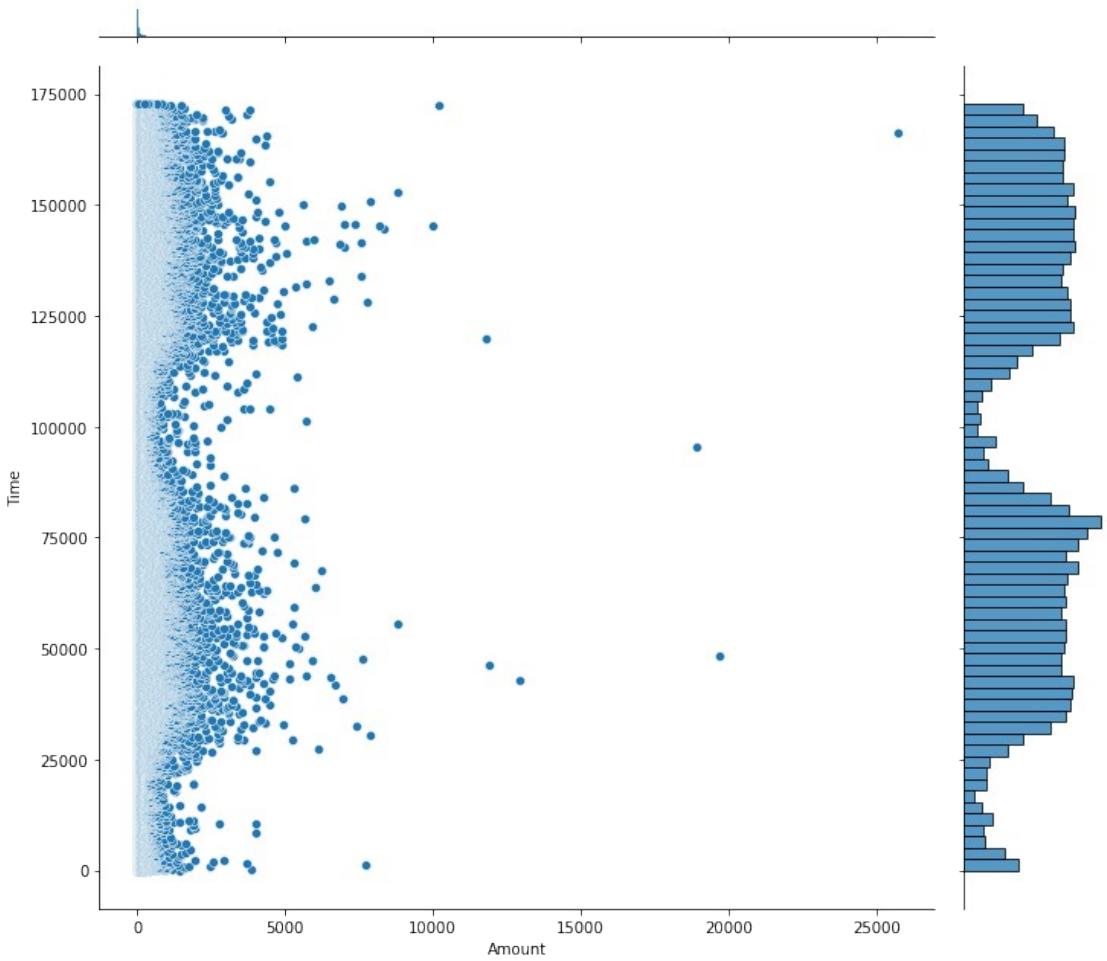



```
data['Class'].value_counts().plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Amount')
sns.despine()
```



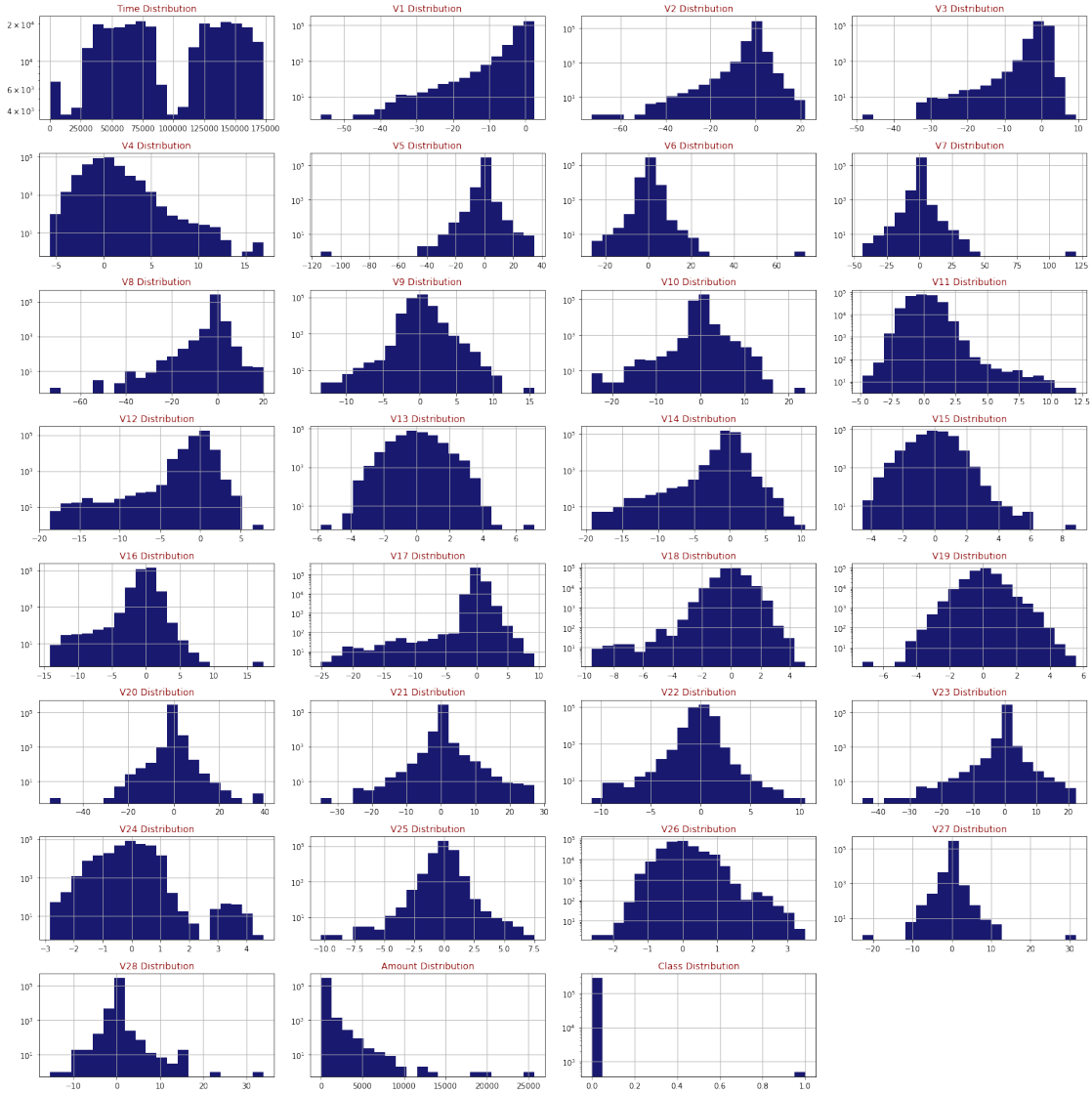

```
plt.figure(figsize=(10,10))
sns.jointplot(x=data.Amount, y=data.Time, height=10)
plt.show()
sns.despine()
```

```
<Figure size 720x720 with 0 Axes>
```



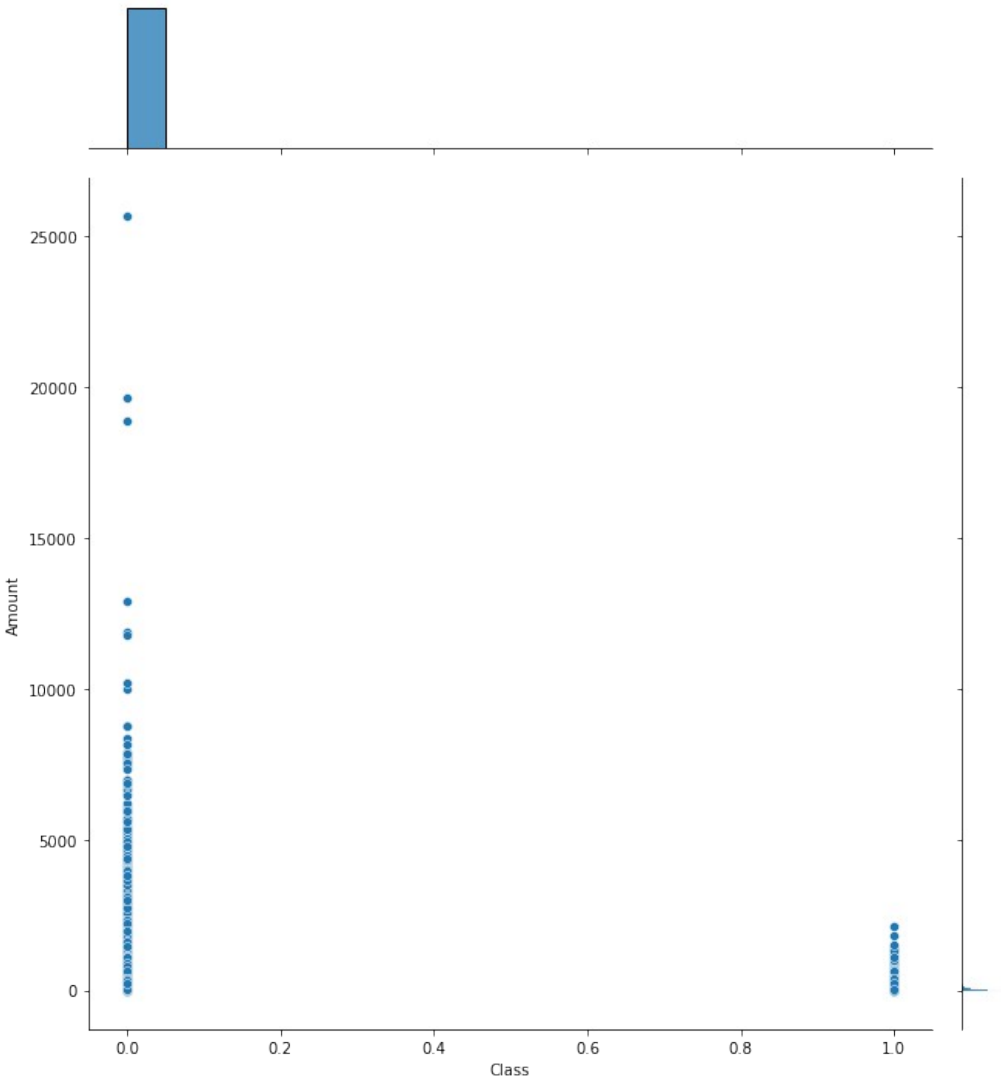

```
<Figure size 432x288 with 0 Axes>
```

```
def draw_hist(dataframe, features, rows, cols):
    fig = plt.figure(figsize=(20,20))
    for i, feature in enumerate(features):
        ax=fig.add_subplot(rows,cols,i+1)

dataframe[feature].hist(bins=20,ax=ax,facecolor='midnightblue')
        ax.set_title(feature+ " Distribution",color='DarkRed')
        ax.set_yscale('log')
    fig.tight_layout()
    plt.show()

draw_hist(data,data.columns,8,4)
```




```
plt.figure(figsize=(10,10))
sns.jointplot(x=data.Class, y=data.Amount, height=10)
plt.show()
sns.despine()
```

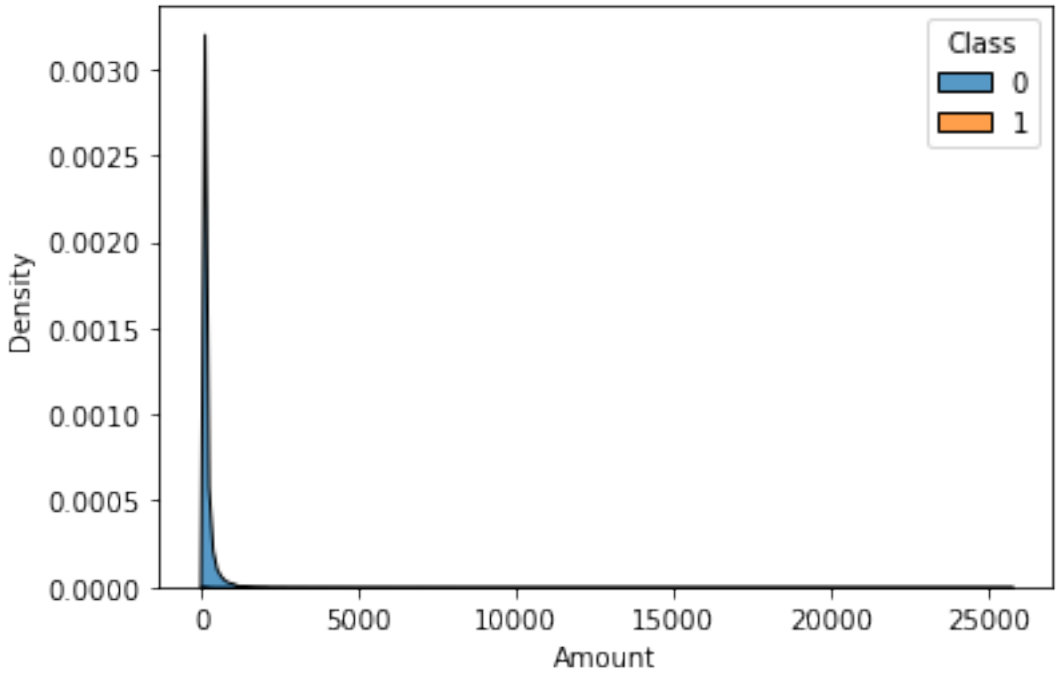
```
<Figure size 720x720 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

```
sns.kdeplot(data=data,x='Amount',hue='Class',multiple='stack')
```

```
<AxesSubplot:xlabel='Amount', ylabel='Density'>
```




```
#
sns.kdeplot(x=data.Amount,y=data.Class,fill=True,thresh=0,cmap='mako')
# plt.show()
```

Training Model

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn import tree

X = data.drop('Class', axis = 1).values
y = data['Class'].values

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

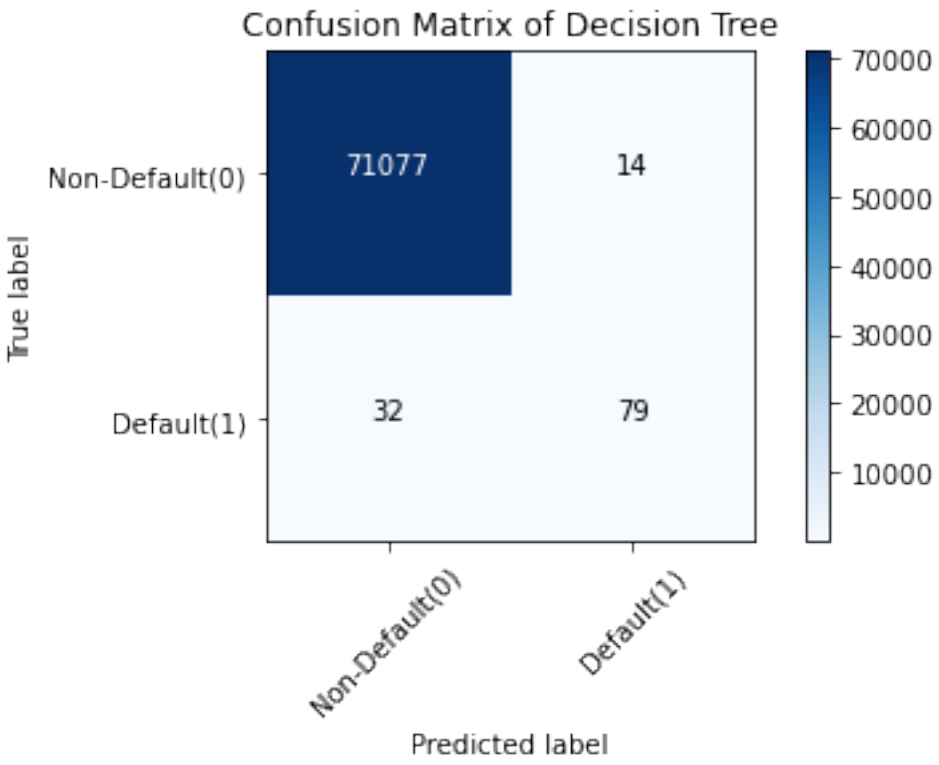
DecisionTree Classifier Algorithm

```
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import accuracy_score

DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
DT.fit(X_train, y_train)
dt_yhat = DT.predict(X_test)

print("Accuracy = {}".format(accuracy_score(y_test, dt_yhat)))

Accuracy = 0.9993539507317211

confusion_matrix(y_test, dt_yhat, labels = [0, 1])
```

```
array([[71077,    14],
       [   32,    79]], dtype=int64)
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap
= plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
import itertools
tree_matrix = confusion_matrix(y_test, dt_yhat, labels = [0,1])
tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                     classes = ['Non-
Default(0)','Default(1)'],
                                     normalize = False, title = 'Decision
Tree')
plt.savefig('tree_cm_plot.png')
plt.show()
```




```
confusion_tree = confusion_matrix(y_test, dt_yhat, labels = [0, 1])

TP = confusion_tree[1,1]
TN = confusion_tree[0,0]
FP = confusion_tree[0,1]
FN = confusion_tree[1,0]

print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
print("Precision : ", TP/float(TP+FP))
print("Sensitivity : ", TP/float(TP+FN))
```

```
Accuracy :  0.9993539507317211
Precision :  0.8494623655913979
Sensitivity :  0.7117117117117117
```

Logistic Regression Algorithm

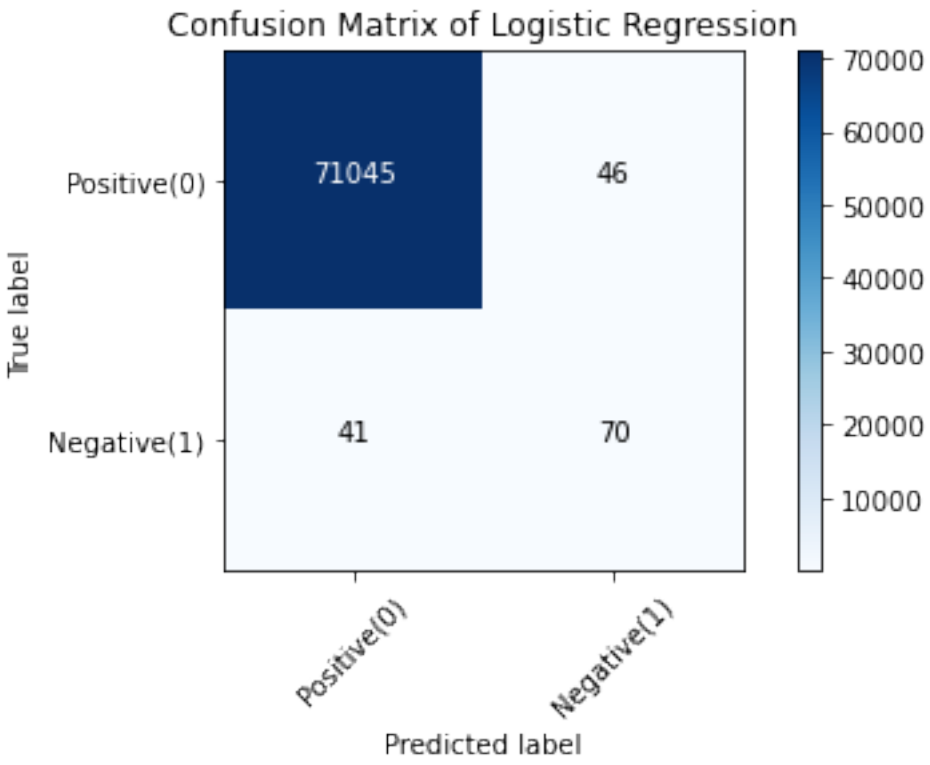
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)

print("Accuracy = {}".format(accuracy_score(y_test, lr_yhat)))
```

```
Accuracy = 0.9987781242099941
```

```
confusion_matrix(y_test, lr_yhat, labels = [0, 1])
```

```
array([[71045,    46],
       [   41,    70]], dtype=int64)
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap
= plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0,1])
lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                   classes =
['Positive(0)','Negative(1)'],
                                   normalize = False, title = 'Logistic
Regression')
plt.savefig('lr_cm_plot.png')
plt.show()
```




```
confusion_lr = confusion_matrix(y_test, lr_yhat, labels = [0, 1])

TP = confusion_lr[1,1]
TN = confusion_lr[0,0]
FP = confusion_lr[0,1]
FN = confusion_lr[1,0]

print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
print("Precision : ", TP/float(TP+FP))
print("Sensitivity : ", TP/float(TP+FN))
```

```
Accuracy :  0.9987781242099941
Precision :  0.603448275862069
Sensitivity :  0.6306306306306306
```

RandomForestClassifier Algorithm

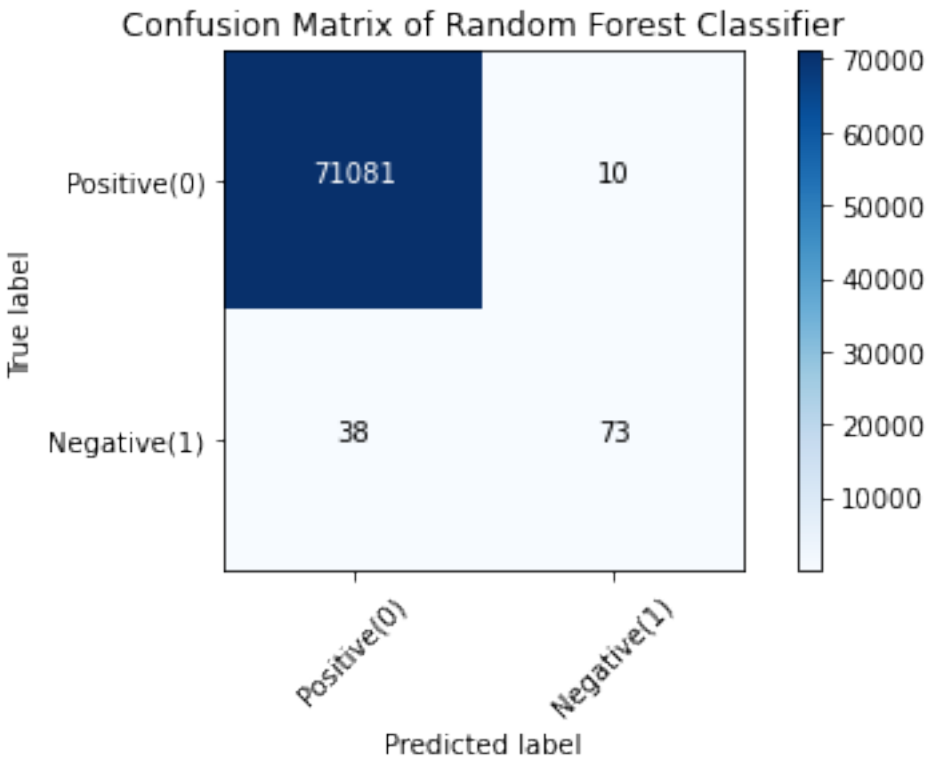
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)

print("Accuracy = {}".format(accuracy_score(y_test, rf_yhat)))
```

```
Accuracy = 0.9993258616331002
```

```
confusion_matrix(y_test, rf_yhat, labels = [0, 1])
```

```
array([[71081,    10],
       [   38,    73]], dtype=int64)
```

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap
= plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0,1])
rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                   classes =
['Positive(0)','Negative(1)'],
                                   normalize = False, title = 'Random
Forest Classifier')
plt.savefig('rf_cm_plot.png')
plt.show()
```




```
confusion_rf = confusion_matrix(y_test, rf_yhat, labels = [0, 1])

TP = confusion_rf[1,1]
TN = confusion_rf[0,0]
FP = confusion_rf[0,1]
FN = confusion_rf[1,0]

print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
print("Precision : ", TP/float(TP+FP))
print("Sensitivity : ", TP/float(TP+FN))
```

```
Accuracy :  0.9993258616331002
Precision :  0.8795180722891566
Sensitivity :  0.6576576576576577
```

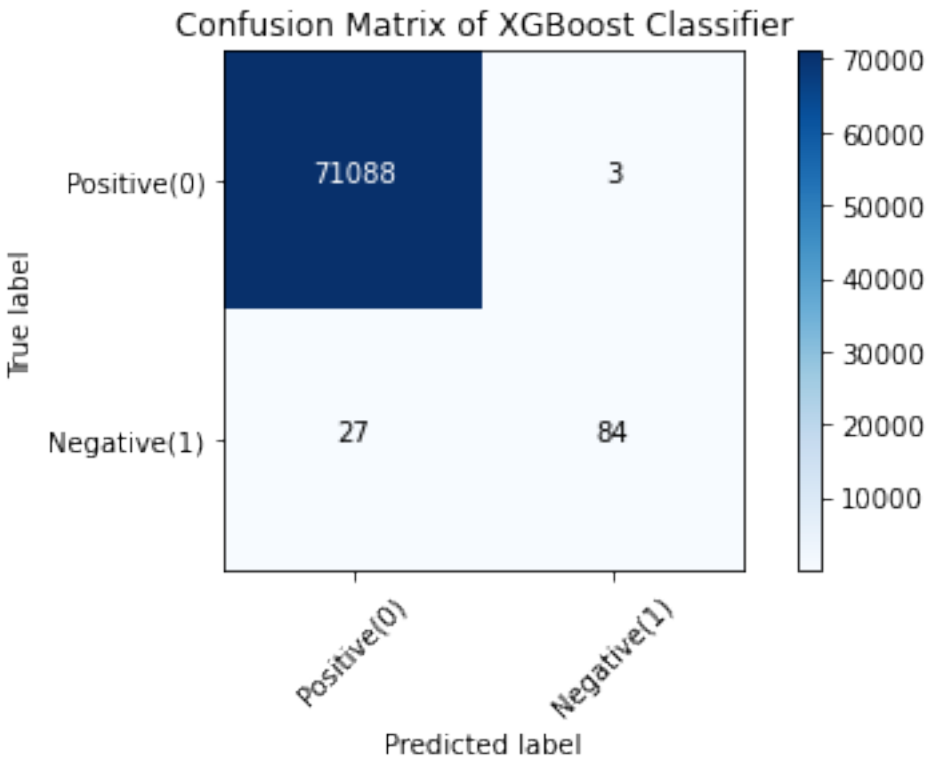
XGBoostClassifier Algorithm

```
from xgboost import XGBClassifier

xgb = XGBClassifier(max_depth = 4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)
```

```
[22:25:40] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective
'binary:logistic' was changed from 'error' to 'logloss'. Explicitly
set eval_metric if you'd like to restore the old behavior.
```

```
print("Accuracy = {}".format(accuracy_score(y_test, xgb_yhat)))
```

```
Accuracy = 0.9995786635206876

confusion_matrix(y_test, xgb_yhat, labels = [0, 1])

array([[71088,     3],
       [   27,    84]], dtype=int64)

def plot_confusion_matrix(cm, classes, title, normalize = False, cmap
= plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

xgb_matrix = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])

xgb_cm_plot = plot_confusion_matrix(xgb_matrix,
                                    classes =
['Positive(0)','Negative(1)'],
                                    normalize = False, title = 'XGBoost
Classifier')
plt.savefig('xgb_cm_plot.png')
plt.show()
```




```
confusion_xgb = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
confusion_xgb
```

```
array([[71088,     3],
       [   27,    84]], dtype=int64)
```

```
TP = confusion_xgb[1,1]
TN = confusion_xgb[0,0]
FP = confusion_xgb[0,1]
FN = confusion_xgb[1,0]
```

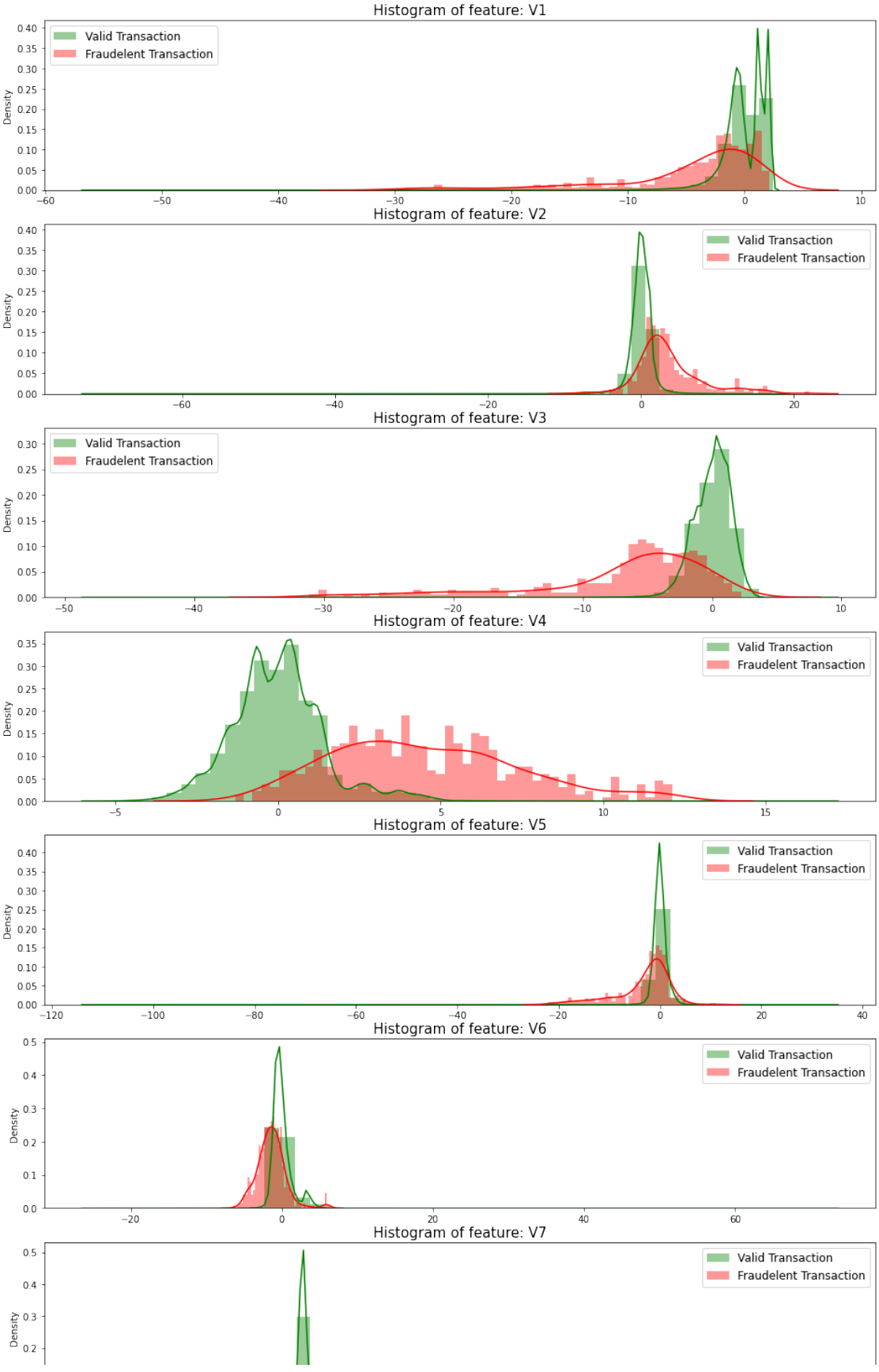
```
print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
print("Precision : ", TP/float(TP+FP))
print("Sensitivity : ", TP/float(TP+FN))
```

```
Accuracy :  0.9995786635206876
Precision :  0.9655172413793104
Sensitivity :  0.7567567567567568
```

```
import matplotlib.gridspec as gridspec
plt.clf()
pca_features = data.columns[1:29]
plt.figure(figsize=(16,28*4))
gs = gridspec.GridSpec(28, 1)
for i, col in enumerate(data[pca_features]):
    ax = plt.subplot(gs[i])
    sns.distplot(data[col][data.Class == 0], bins=50, label='Valid
Transaction', color='green')
```

```
    sns.distplot(data[col][data.Class == 1], bins=50,
label='Fraudelent Transaction', color='red')
    ax.set_xlabel('')
    ax.set_title('Histogram of feature: ' + str(col),fontsize=15)
    plt.legend(loc='best',fontsize=12)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```


Histogram of feature: V1
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V2
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V3
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V4
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V5
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V6
Valid Transaction
Fraudelent Transaction
Density
Histogram of feature: V7
Valid Transaction
Fraudelent Transaction
Density