Credit Card Fraud Detection

Aveline Mariya Shaji, Kyuri Kim, Satya Bharath Reddy Duvvi

# Introduction

Credit card fraud detection is the collective term for the policies, tools, methodologies, and practices that credit card companies and financial institutions take to combat identity fraud and stop fraudulent transactions. It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

In this kernel we will use various predictive models to see how accurate they are in detecting whether a transaction is a normal payment or a fraud. As described in the dataset, the features are scaled and the names of the features are not shown due to privacy reasons. Nevertheless, we can still analyze some important aspects of the dataset.

The aim of this project is to use different methods to handle “Unbalanced Data” and implement different ML Algorithms and identify the best model for predicting Credit Card Fraud Detection.

## Customer Analysis Overview

The dataset contains transactions made by credit cards in September 2013 by European cardholders and this dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are ‘Time’ and ‘Amount’. Feature ‘Time’ contains the seconds elapsed between each transaction and the first transaction in the dataset.

The feature ‘Amount’ is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature ‘Class’ is the response variable and it takes value 1 in case of fraud and 0 otherwise.

# SMART Question

***Which ML model performs best in predictive analysis on credit card transaction dataset to detect the fraudulent transactions from the given dataset***

# Data Preprocessing

# Importing the libraries  
  
import numpy as np   
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from pylab import rcParams  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix   
from sklearn.metrics import precision\_score, recall\_score, f1\_score  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.tree import DecisionTreeRegressor  
from matplotlib import gridspec  
import warnings  
warnings.filterwarnings("ignore")

## Load the Credit Card File into a DataFrame (CCFD\_DATA)

***Shape of the dataframe***

CCFD\_DATA =pd.read\_csv('creditcard.csv')  
CCFD\_DATA.shape

(284807, 31)

### Analyzing the DataFrame

***Viewing the Data***

CCFD\_DATA.head()

|  | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| 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 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| 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 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| 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 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| 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 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |

CCFD\_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

## Cleaning Data

***Checking for Empty Cells***

CCFD\_DATA.isnull().values.sum()

0

***Checking for Duplicates Cells***

CCFD\_DATA.duplicated().sum()

1081

### Removing Duplicates

CCFD\_DATA.drop\_duplicates(keep=False,inplace=True)

***Re-checking the dataframe for duplicates***

CCFD\_DATA.duplicated().sum()

0

***let’s check the class column***

CCFD\_DATA['Class'].value\_counts()

0 282493  
1 460  
Name: Class, dtype: int64

## Column Data Type Assessment

CCFD\_DATA.dtypes.value\_counts()

float64 30  
int64 1  
dtype: int64

non\_fraud = len(CCFD\_DATA[CCFD\_DATA.Class == 0])  
fraud = len(CCFD\_DATA[CCFD\_DATA.Class == 1])  
fraud\_percent = (fraud / (fraud + non\_fraud)) \* 100  
  
print("Number of Genuine transactions: ", non\_fraud)  
print("Number of Fraud transactions: ", fraud)  
print("Percentage of Fraud transactions: {:.4f}".format(fraud\_percent))

Number of Genuine transactions: 282493  
Number of Fraud transactions: 460  
Percentage of Fraud transactions: 0.1626

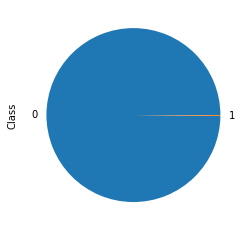
## Summary Statistics

CCFD\_DATA.describe()

|  | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | ... | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 | 282953.000000 |
| mean | 94816.256714 | 0.010161 | -0.006837 | 0.002906 | -0.004665 | 0.003311 | -0.001734 | 0.002985 | -0.002038 | -0.002651 | ... | -0.000316 | 0.000184 | 0.000332 | 0.000372 | -0.000347 | 0.000317 | 0.002830 | 0.000740 | 88.534756 | 0.001626 |
| std | 47479.631543 | 1.940990 | 1.643708 | 1.504189 | 1.413356 | 1.374938 | 1.331984 | 1.223249 | 1.173378 | 1.094047 | ... | 0.721104 | 0.724223 | 0.623093 | 0.605599 | 0.521199 | 0.481876 | 0.391139 | 0.327223 | 250.567570 | 0.040287 |
| min | 0.000000 | -56.407510 | -72.715728 | -48.325589 | -5.683171 | -113.743307 | -26.160506 | -43.557242 | -73.216718 | -13.434066 | ... | -34.830382 | -10.933144 | -44.807735 | -2.836627 | -10.295397 | -2.604551 | -22.565679 | -15.430084 | 0.000000 | 0.000000 |
| 25% | 54213.000000 | -0.912989 | -0.601721 | -0.888987 | -0.851101 | -0.688407 | -0.769506 | -0.551470 | -0.209036 | -0.645213 | ... | -0.228236 | -0.542743 | -0.161658 | -0.354423 | -0.317659 | -0.326567 | -0.070453 | -0.052736 | 5.590000 | 0.000000 |
| 50% | 84704.000000 | 0.022459 | 0.062929 | 0.180273 | -0.023625 | -0.052817 | -0.275914 | 0.041333 | 0.021522 | -0.052847 | ... | -0.029370 | 0.007041 | -0.011184 | 0.041074 | 0.016162 | -0.052152 | 0.001564 | 0.011312 | 22.000000 | 0.000000 |
| 75% | 139294.000000 | 1.316582 | 0.797751 | 1.027190 | 0.737319 | 0.612704 | 0.395220 | 0.570666 | 0.324281 | 0.594912 | ... | 0.186184 | 0.528316 | 0.147729 | 0.439880 | 0.350621 | 0.239885 | 0.091310 | 0.078270 | 77.710000 | 0.000000 |
| max | 172792.000000 | 2.454930 | 22.057729 | 9.382558 | 16.875344 | 34.801666 | 73.301626 | 120.589494 | 20.007208 | 15.594995 | ... | 22.614889 | 10.503090 | 22.528412 | 4.584549 | 7.519589 | 3.517346 | 31.612198 | 33.847808 | 25691.160000 | 1.000000 |

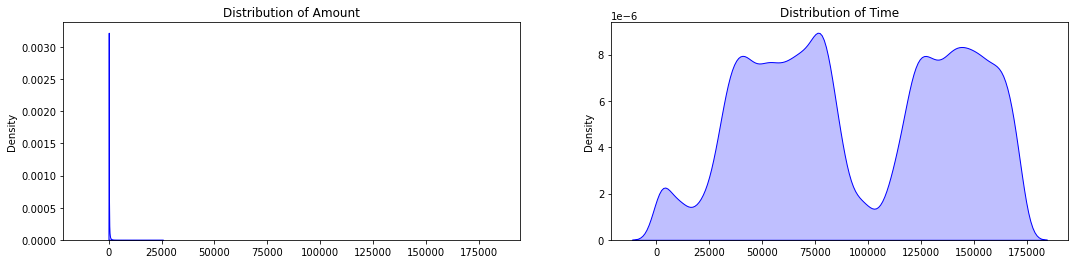
CCFD\_DATA["Class"].value\_counts().plot(kind="pie")

<AxesSubplot:ylabel='Class'>



***Plot the named features***

f, axes = plt.subplots(1, 2, figsize=(18,4), sharex = True)  
  
amount\_value = CCFD\_DATA['Amount'].values   
time\_value = CCFD\_DATA['Time'].values  
  
sns.distplot(amount\_value, hist=False, color="b", kde\_kws={"shade": True}, ax=axes[0]).set\_title('Distribution of Amount')  
sns.distplot(time\_value, hist=False, color="b", kde\_kws={"shade": True}, ax=axes[1]).set\_title('Distribution of Time')  
  
plt.show()



print("Average Amount in a Fraudulent Transaction: " + str(CCFD\_DATA[CCFD\_DATA["Class"] == 1]["Amount"].mean()))  
print("Average Amount in a Valid Transaction: " + str(CCFD\_DATA[CCFD\_DATA["Class"] == 0]["Amount"].mean()))

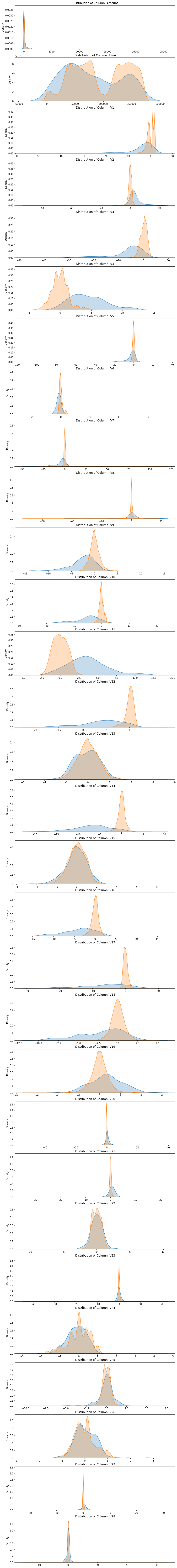
Average Amount in a Fraudulent Transaction: 124.0452391304348  
Average Amount in a Valid Transaction: 88.47693213631382

print("Summary of the feature - Amount" + "\n-------------------------------")  
print(CCFD\_DATA["Amount"].describe())

Summary of the feature - Amount  
-------------------------------  
count 282953.000000  
mean 88.534756  
std 250.567570  
min 0.000000  
25% 5.590000  
50% 22.000000  
75% 77.710000  
max 25691.160000  
Name: Amount, dtype: float64

***Plot the distributions of the features***

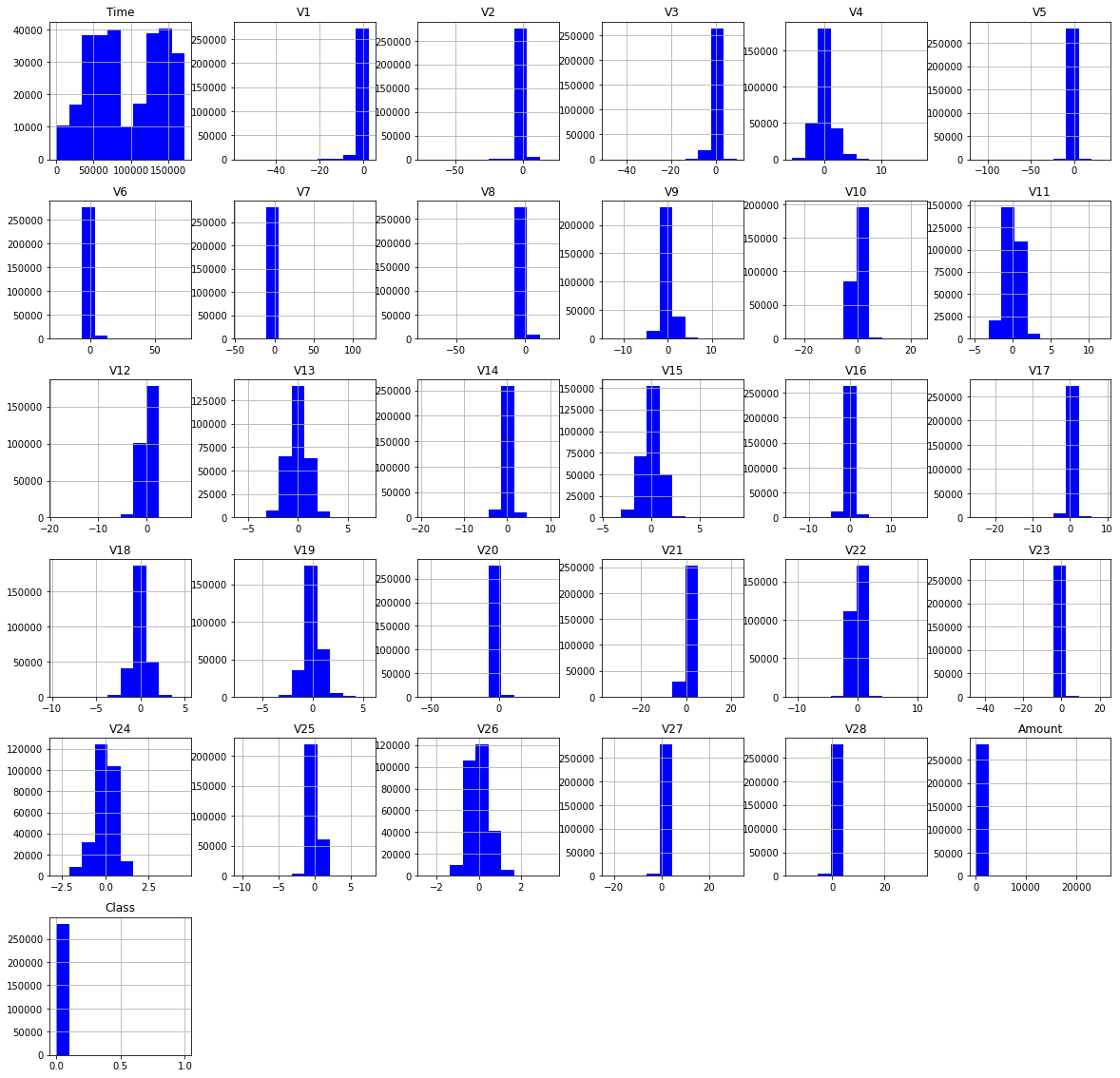
# Reorder the columns Amount, Time then the rest  
data\_plot = CCFD\_DATA.copy()  
amount = data\_plot['Amount']  
data\_plot.drop(labels=['Amount'], axis=1, inplace = True)  
data\_plot.insert(0, 'Amount', amount)  
  
  
columns = data\_plot.iloc[:,0:30].columns  
plt.figure(figsize=(12,30\*4))  
grids = gridspec.GridSpec(30, 1)  
for grid, index in enumerate(data\_plot[columns]):  
 ax = plt.subplot(grids[grid])  
 sns.distplot(data\_plot[index][data\_plot.Class == 1], hist=False, kde\_kws={"shade": True}, bins=50)  
 sns.distplot(data\_plot[index][data\_plot.Class == 0], hist=False, kde\_kws={"shade": True}, bins=50)  
 ax.set\_xlabel("")  
 ax.set\_title("Distribution of Column: " + str(index))  
  
plt.show()



# Exploratory Data Analysis

## Exploratory Data Analysis for Unbalanced Data

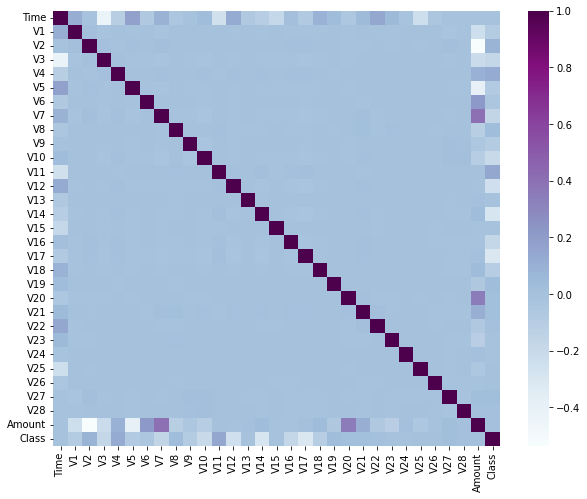
CCFD\_DATA.hist(figsize=(20,20),color='blue')  
plt.show()



***Correlation matrix before Sampling***

plt.figure(figsize=(10,8))  
corr=CCFD\_DATA.corr()  
sns.heatmap(corr,cmap='BuPu')

<AxesSubplot:>



Since the data is highly imbalanced, we will move ahead with Undersampling/Upsampling. But inorder to see which method works for the data, let’s create model using upsampled and undersampled data. We need balanced data!!!

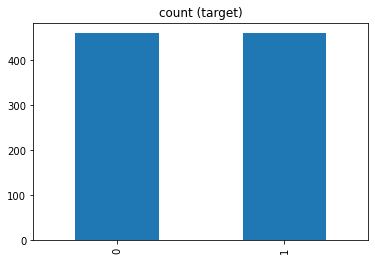
# class count  
class\_count\_0, class\_count\_1 = CCFD\_DATA['Class'].value\_counts()  
  
# Separate class  
# It takes value 1 in case of fraud and 0 otherwise  
class\_0 = CCFD\_DATA[CCFD\_DATA['Class'] == 0]  
class\_1 = CCFD\_DATA[CCFD\_DATA['Class'] == 1]

***Undersampling Technique***

class\_0\_under = class\_0.sample(class\_count\_1)  
data\_under = pd.concat([class\_0\_under, class\_1], axis=0)  
print("total class of 1 and 0:",data\_under['Class'].value\_counts())  
  
# plot the count after under-sampling  
data\_under['Class'].value\_counts().plot(kind='bar', title='count (target)')

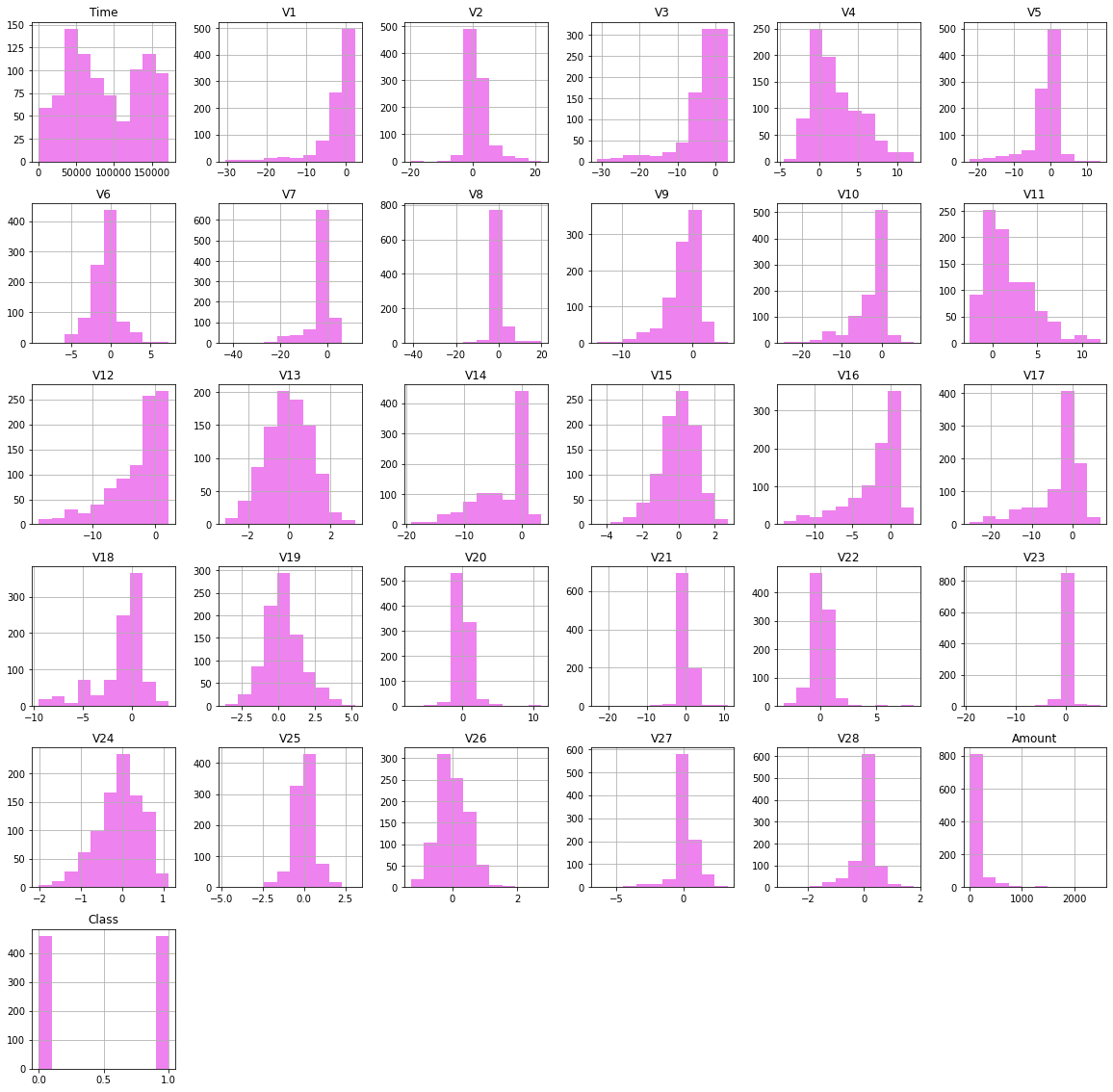
total class of 1 and 0: 0 460  
1 460  
Name: Class, dtype: int64

<AxesSubplot:title={'center':'count (target)'}>

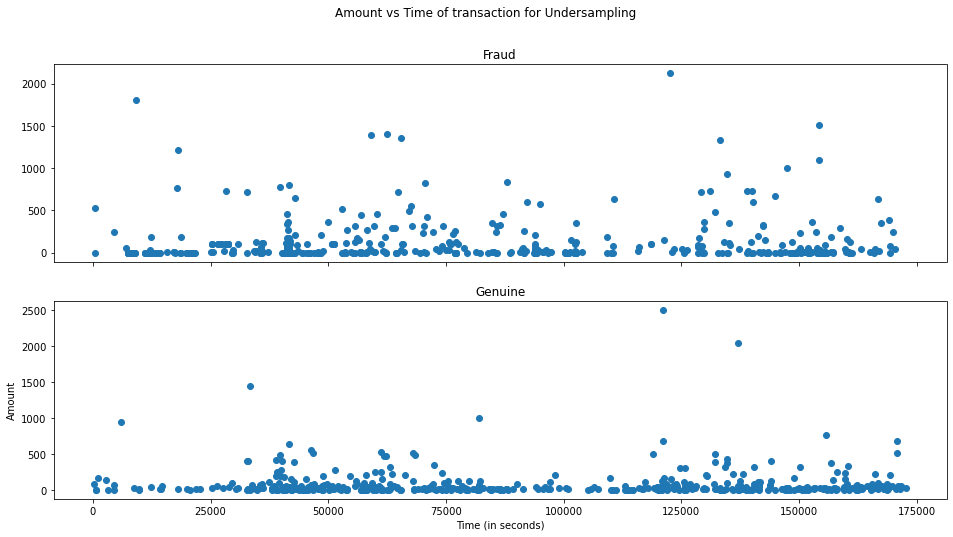


## Exploratory Data Analysis for Undersampling

data\_under.hist(figsize=(20,20),color='violet')  
plt.show()



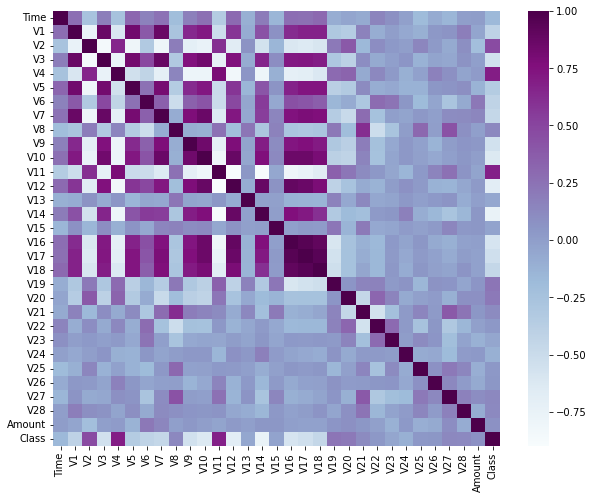
genuine = data\_under[data\_under['Class'] == 0]  
fraud = data\_under[data\_under['Class'] == 1]  
  
rcParams['figure.figsize'] = 16, 8  
f,(ax1, ax2) = plt.subplots(2, 1, sharex=True)  
f.suptitle('Amount vs Time of transaction for Undersampling')  
ax1.scatter(fraud.Time, fraud.Amount)  
ax1.set\_title('Fraud')  
ax2.scatter(genuine.Time, genuine.Amount)  
ax2.set\_title('Genuine')  
plt.xlabel('Time (in seconds)')  
plt.ylabel('Amount')  
plt.show()



***Correlation matrix after Under Sampling***

plt.figure(figsize=(10,8))  
corr=data\_under.corr()  
sns.heatmap(corr,cmap='BuPu')

<AxesSubplot:>

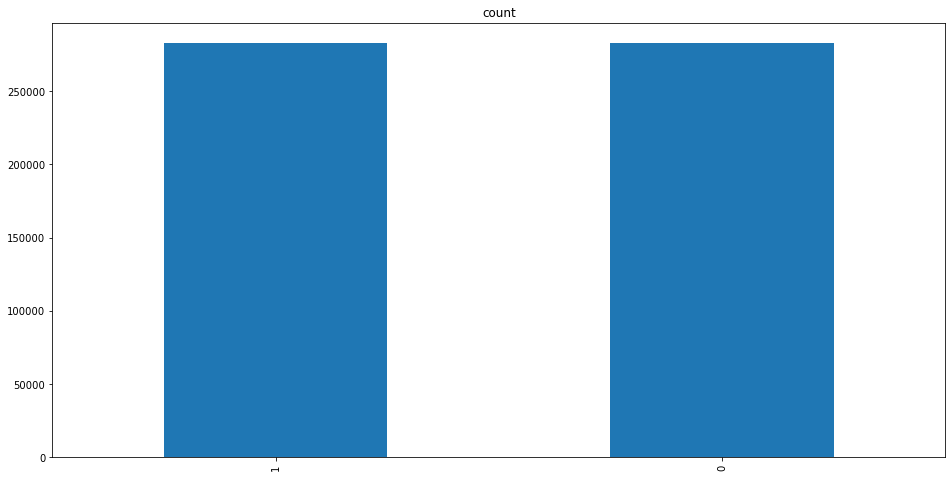


***Oversampling Technique***

class\_1\_over = class\_1.sample(class\_count\_0, replace = True)  
data\_over = pd.concat([class\_1\_over, class\_0], axis=0)  
print("Total class of 1 and 0:", data\_under['Class'].value\_counts())  
  
# plot the count after over-sampling  
data\_over['Class'].value\_counts().plot(kind='bar', title='count')

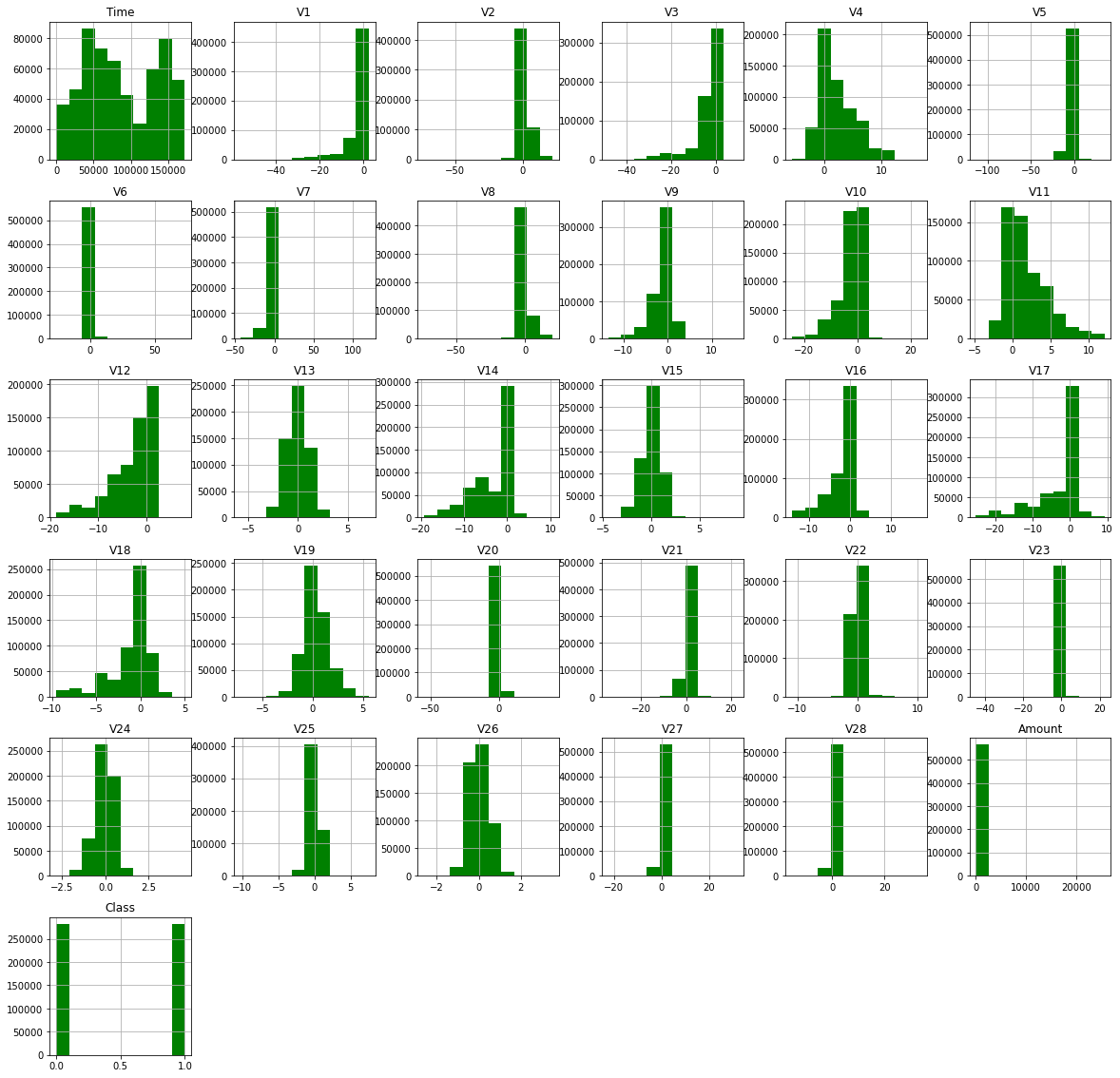
Total class of 1 and 0: 0 460  
1 460  
Name: Class, dtype: int64

<AxesSubplot:title={'center':'count'}>

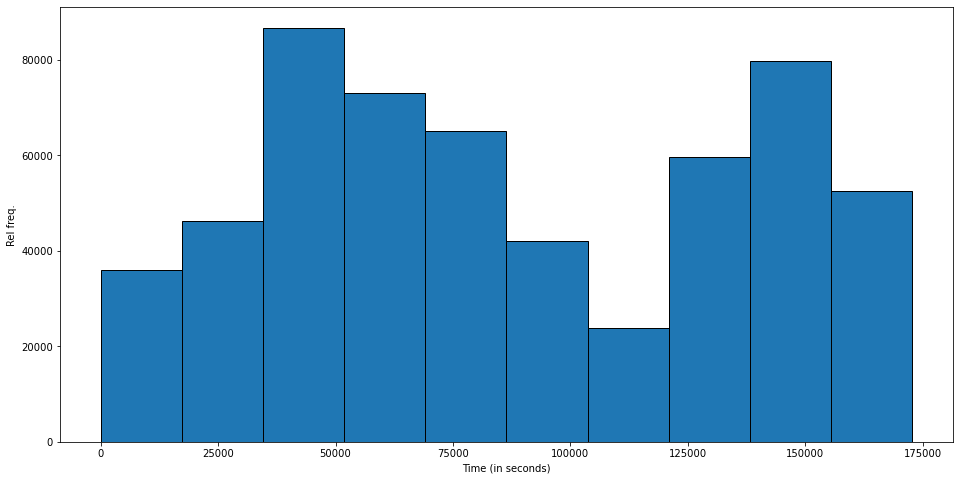


## Exploratory Data Analysis for Oversampling

data\_over.hist(figsize=(20,20),color='green')  
plt.show()



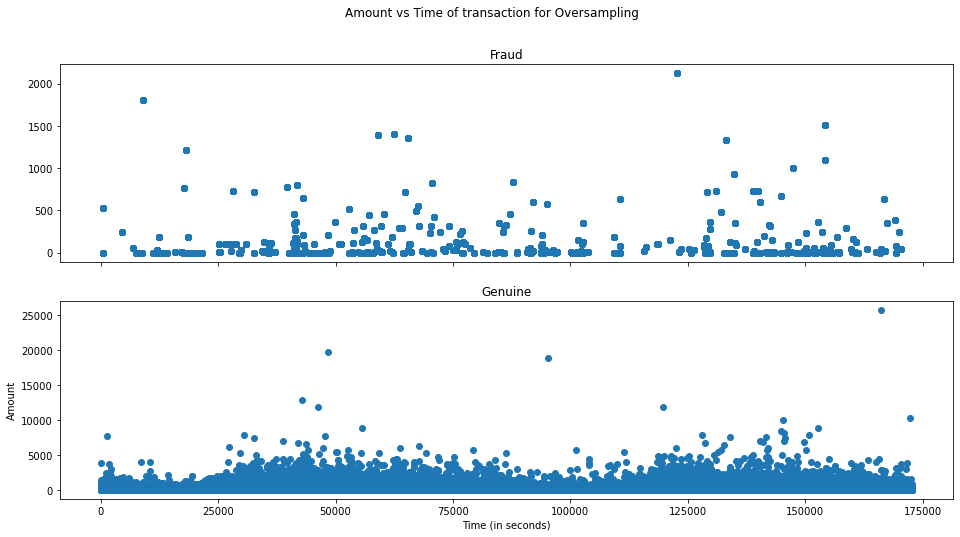
plt.hist(data\_over.Time, label='time', edgecolor='black', linewidth=1)  
plt.xlabel('Time (in seconds)')  
plt.ylabel('Rel freq.')  
plt.show()



plt.hist(data\_over.Amount, label='time', edgecolor='black', linewidth=1)  
plt.xlabel('Amount')  
plt.ylabel('Rel freq.')  
plt.show()



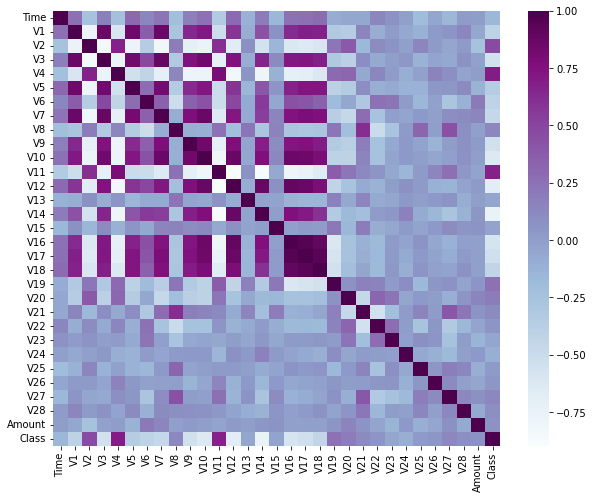
genuine = data\_over[data\_over['Class'] == 0]  
fraud = data\_over[data\_over['Class'] == 1]  
  
rcParams['figure.figsize'] = 16, 8  
f,(ax1, ax2) = plt.subplots(2, 1, sharex=True)  
f.suptitle('Amount vs Time of transaction for Oversampling')  
ax1.scatter(fraud.Time, fraud.Amount)  
ax1.set\_title('Fraud')  
ax2.scatter(genuine.Time, genuine.Amount)  
ax2.set\_title('Genuine')  
plt.xlabel('Time (in seconds)')  
plt.ylabel('Amount')  
plt.show()



***Correlation matrix after Over Sampling***

plt.figure(figsize=(10,8))  
corr=data\_over.corr()  
sns.heatmap(corr,cmap='BuPu')

<AxesSubplot:>



# Modeling

## Modeling for Unbalanced Data

Before proceeding with sampling techniques we implemented the modeling for Unbalanced Data to check how the Accuracy, Precision, Recall, and F1 Score will be.

***Splitting the Data before Sampling***

Before proceeding with the modeling for Unbalanced Data we have to separate the original data frame into training and testing sets and separate the features from the labels.

X=CCFD\_DATA.drop(['Class'],axis=1)  
y=CCFD\_DATA['Class']  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.30,random\_state=123)

### Model 1 : Random Forest Classifier - Before Sampling

# Fit and predict  
rfc = RandomForestClassifier()   
rfc.fit(X\_train, y\_train)   
y\_pred = rfc.predict(X\_test)  
   
print("The accuracy is", accuracy\_score(y\_test, y\_pred))   
print("The precision is", precision\_score(y\_test, y\_pred))  
print("The recall is", recall\_score(y\_test, y\_pred))  
print("The F1 score is", f1\_score(y\_test, y\_pred))

The accuracy is 0.99949343825837  
The precision is 0.9519230769230769  
The recall is 0.7226277372262774  
The F1 score is 0.8215767634854771

***Classification report***

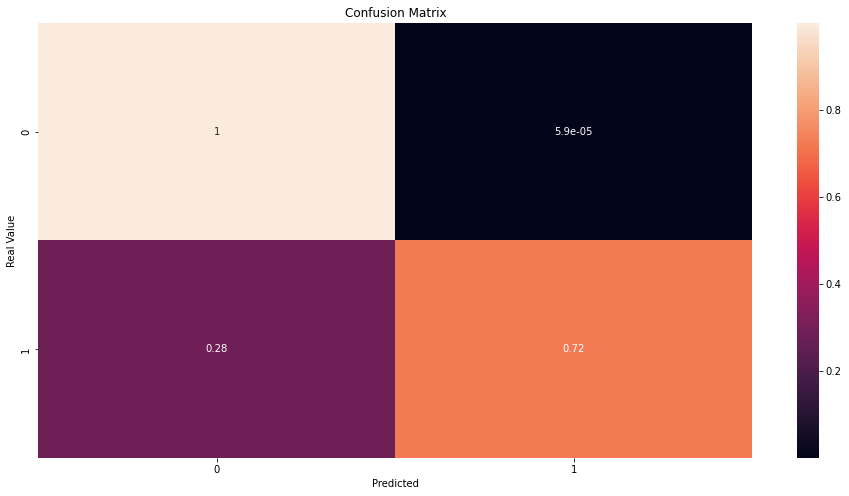
print(classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84749  
 1 0.95 0.72 0.82 137  
  
 accuracy 1.00 84886  
 macro avg 0.98 0.86 0.91 84886  
weighted avg 1.00 1.00 1.00 84886

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 2 : Logistic Regression - Before Sampling

lr=LogisticRegression()  
lr.fit(X\_train,y\_train)  
y\_pred\_2=lr.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_2))   
print("The precision is", precision\_score(y\_test, y\_pred\_2))  
print("The recall is", recall\_score(y\_test, y\_pred\_2))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_2))

The accuracy is 0.9988455104493085  
The precision is 0.6585365853658537  
The recall is 0.5912408759124088  
The F1 score is 0.6230769230769231

***Classification report***

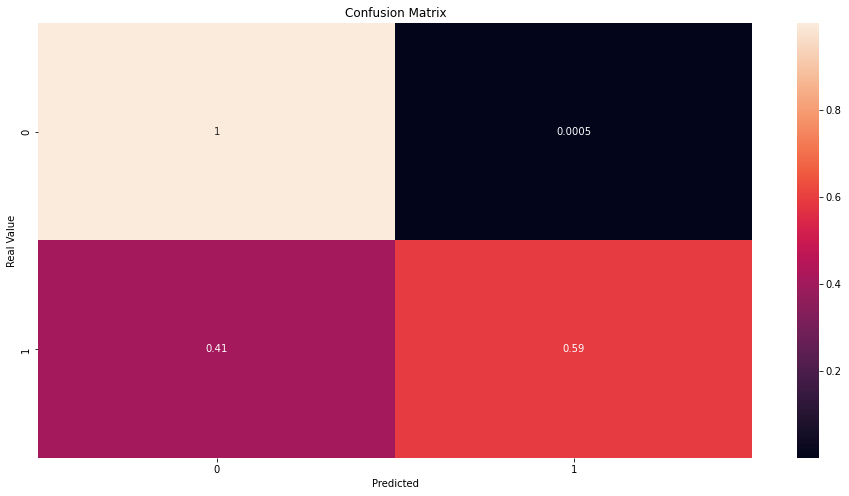
print(classification\_report(y\_test, y\_pred\_2))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84749  
 1 0.66 0.59 0.62 137  
  
 accuracy 1.00 84886  
 macro avg 0.83 0.80 0.81 84886  
weighted avg 1.00 1.00 1.00 84886

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_2, normalize='true'), annot=True, ax=ax)  
ax.set\_title ("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 3 : Decision Tree - Before Sampling

dt=DecisionTreeRegressor()  
dt.fit(X\_train,y\_train)  
y\_pred\_3=dt.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_3))   
print("The precision is", precision\_score(y\_test, y\_pred\_3))  
print("The recall is", recall\_score(y\_test, y\_pred\_3))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_3))

The accuracy is 0.9992460476403647  
The precision is 0.8067226890756303  
The recall is 0.7007299270072993  
The F1 score is 0.75

***Classification report***

print(classification\_report(y\_test, y\_pred\_3))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84749  
 1 0.81 0.70 0.75 137  
  
 accuracy 1.00 84886  
 macro avg 0.90 0.85 0.87 84886  
weighted avg 1.00 1.00 1.00 84886

***Confusion matrix***

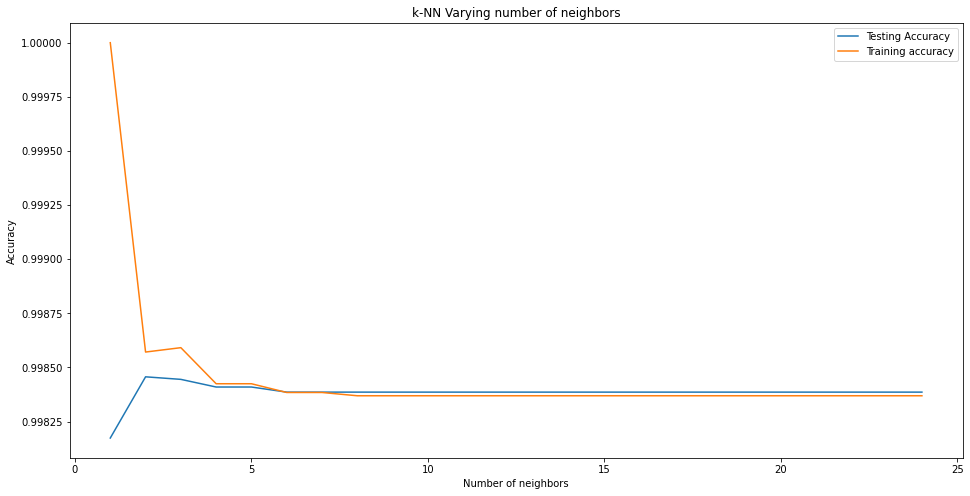
fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_3, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 4 : KNN Model - Before Sampling

neighbours = np.arange(1,25)  
train\_accuracy =np.empty(len(neighbours))  
test\_accuracy = np.empty(len(neighbours))  
  
for i,k in enumerate(neighbours):  
 #Setup a knn classifier with k neighbors  
 knn=KNeighborsClassifier(n\_neighbors=k,algorithm="kd\_tree",n\_jobs=-1)  
   
 #Fit the model  
 knn.fit(X\_train,y\_train.ravel())  
   
 #Compute accuracy on the training set  
 train\_accuracy[i] = knn.score(X\_train, y\_train.ravel())  
   
 #Compute accuracy on the test set  
 test\_accuracy[i] = knn.score(X\_test, y\_test.ravel())   
  
  
#Generate plot  
plt.title('k-NN Varying number of neighbors')  
plt.plot(neighbours, test\_accuracy, label='Testing Accuracy')  
plt.plot(neighbours, train\_accuracy, label='Training accuracy')  
plt.legend()  
plt.xlabel('Number of neighbors')  
plt.ylabel('Accuracy')  
plt.show()  
  
idx = np.where(test\_accuracy == max(test\_accuracy))  
x = neighbours[idx]  
  
#k\_nearest\_neighbours\_classification  
knn=KNeighborsClassifier(n\_neighbors=x[0],algorithm="kd\_tree",n\_jobs=-1)  
knn.fit(X\_train,y\_train.ravel())



KNeighborsClassifier(algorithm='kd\_tree', n\_jobs=-1, n\_neighbors=2)

***KNN results***

knn = KNeighborsClassifier(n\_neighbors = 2)   
knn.fit(X\_train,y\_train)  
y\_pred\_4 = knn.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_4))   
print("The precision is", precision\_score(y\_test, y\_pred\_4))  
print("The recall is", recall\_score(y\_test, y\_pred\_4))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_4))

The accuracy is 0.9984567537638716  
The precision is 1.0  
The recall is 0.043795620437956206  
The F1 score is 0.08391608391608392

***Classification report***

print(classification\_report(y\_test, y\_pred\_4))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84749  
 1 1.00 0.04 0.08 137  
  
 accuracy 1.00 84886  
 macro avg 1.00 0.52 0.54 84886  
weighted avg 1.00 1.00 1.00 84886

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_4, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



We can observe that sampling is necessary to fit these models either with the data frames that were undersampled or oversampled (in order for our models to detect the patterns) and test it on the original testing set to improve Precision, Recall, and F1 Score.

## Modeling for Under Sampling Data

In this phase of the project, we will implement “Random Under Sampling” which basically consists of removing data in order to have a more balanced dataset and thus preventing our models from overfitting.

***Splitting the Data for Under Sampling***

X=data\_under.drop(['Class'],axis=1)  
y=data\_under['Class']  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.30,random\_state=123)

In this section, we will train four types of classifiers and decide which classifier will be more effective in detecting fraud transactions.

### Model 1 : Random Forest Classifier - Under Sampling

# Fit and predict  
rfc = RandomForestClassifier()   
rfc.fit(X\_train, y\_train)   
y\_pred = rfc.predict(X\_test)  
   
print("The accuracy is", accuracy\_score(y\_test, y\_pred))   
print("The precision is", precision\_score(y\_test, y\_pred))  
print("The recall is", recall\_score(y\_test, y\_pred))  
print("The F1 score is", f1\_score(y\_test, y\_pred))

The accuracy is 0.927536231884058  
The precision is 0.975609756097561  
The recall is 0.8759124087591241  
The F1 score is 0.923076923076923

***Classification report***

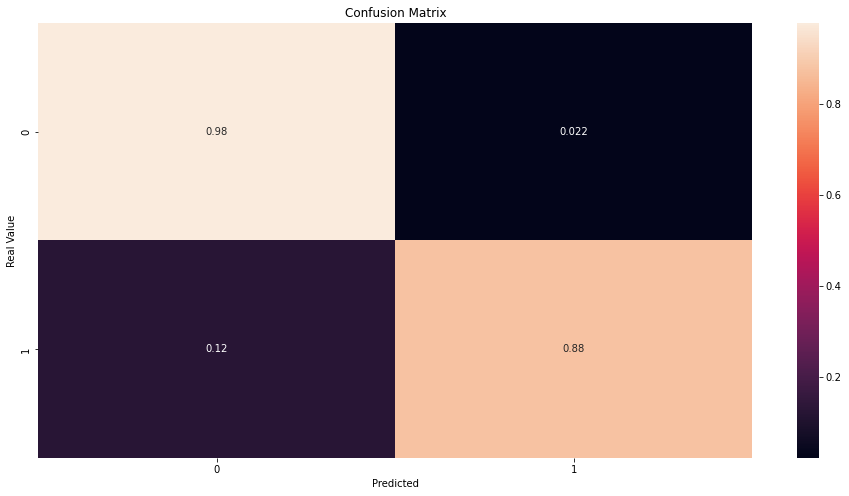
print(classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 0.89 0.98 0.93 139  
 1 0.98 0.88 0.92 137  
  
 accuracy 0.93 276  
 macro avg 0.93 0.93 0.93 276  
weighted avg 0.93 0.93 0.93 276

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 2 : Logistic Regression - Under Sampling

lr=LogisticRegression()  
lr.fit(X\_train,y\_train)  
y\_pred\_2=lr.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_2))   
print("The precision is", precision\_score(y\_test, y\_pred\_2))  
print("The recall is", recall\_score(y\_test, y\_pred\_2))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_2))

The accuracy is 0.9311594202898551  
The precision is 0.9538461538461539  
The recall is 0.9051094890510949  
The F1 score is 0.9288389513108616

***Classification report***

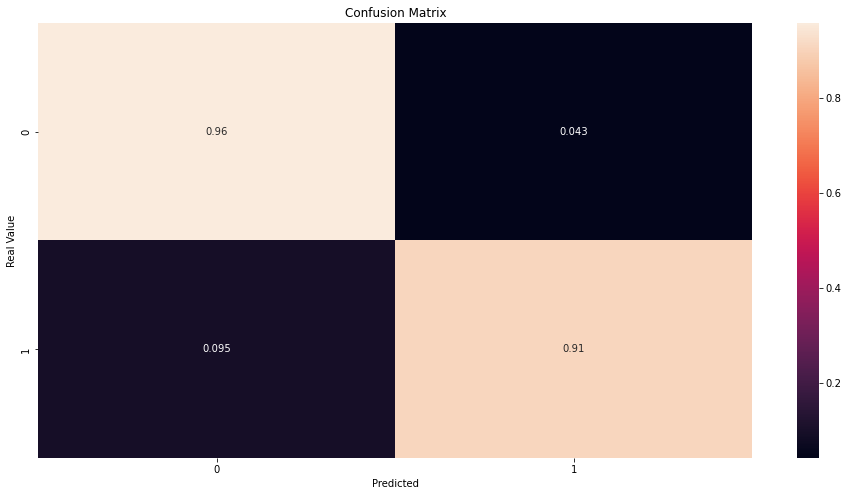
print(classification\_report(y\_test, y\_pred\_2))

precision recall f1-score support  
  
 0 0.91 0.96 0.93 139  
 1 0.95 0.91 0.93 137  
  
 accuracy 0.93 276  
 macro avg 0.93 0.93 0.93 276  
weighted avg 0.93 0.93 0.93 276

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_2, normalize='true'), annot=True, ax=ax)  
ax.set\_title ("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 3 : Decision Tree - Under Sampling

dt=DecisionTreeRegressor()  
dt.fit(X\_train,y\_train)  
y\_pred\_3=dt.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_3))   
print("The precision is", precision\_score(y\_test, y\_pred\_3))  
print("The recall is", recall\_score(y\_test, y\_pred\_3))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_3))

The accuracy is 0.9166666666666666  
The precision is 0.9384615384615385  
The recall is 0.8905109489051095  
The F1 score is 0.9138576779026217

***Classification report***

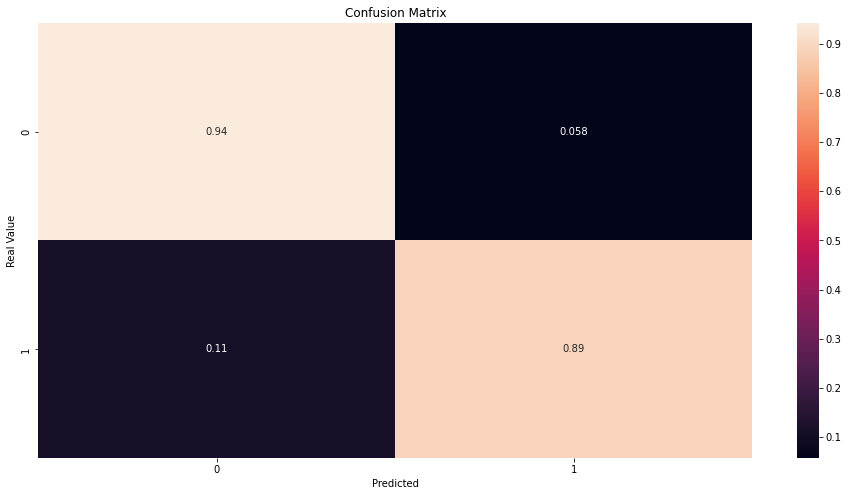
print(classification\_report(y\_test, y\_pred\_3))

precision recall f1-score support  
  
 0 0.90 0.94 0.92 139  
 1 0.94 0.89 0.91 137  
  
 accuracy 0.92 276  
 macro avg 0.92 0.92 0.92 276  
weighted avg 0.92 0.92 0.92 276

***Confusion matrix***

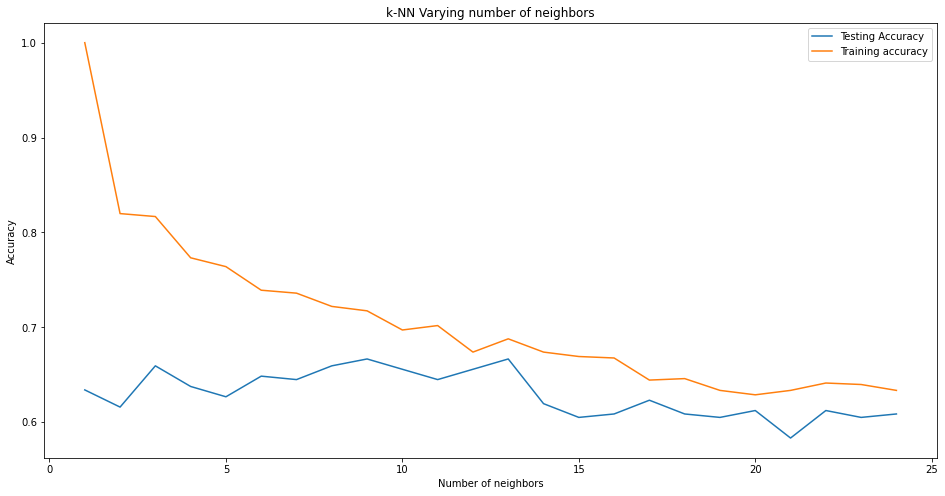
fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_3, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 4 : KNN Model - Under Sampling

neighbours = np.arange(1,25)  
train\_accuracy =np.empty(len(neighbours))  
test\_accuracy = np.empty(len(neighbours))  
  
for i,k in enumerate(neighbours):  
 #Setup a knn classifier with k neighbors  
 knn=KNeighborsClassifier(n\_neighbors=k,algorithm="kd\_tree",n\_jobs=-1)  
   
 #Fit the model  
 knn.fit(X\_train,y\_train.ravel())  
   
 #Compute accuracy on the training set  
 train\_accuracy[i] = knn.score(X\_train, y\_train.ravel())  
   
 #Compute accuracy on the test set  
 test\_accuracy[i] = knn.score(X\_test, y\_test.ravel())   
  
  
#Generate plot  
plt.title('k-NN Varying number of neighbors')  
plt.plot(neighbours, test\_accuracy, label='Testing Accuracy')  
plt.plot(neighbours, train\_accuracy, label='Training accuracy')  
plt.legend()  
plt.xlabel('Number of neighbors')  
plt.ylabel('Accuracy')  
plt.show()  
  
  
idx = np.where(test\_accuracy == max(test\_accuracy))  
x = neighbours[idx]  
  
#k\_nearest\_neighbours\_classification  
knn=KNeighborsClassifier(n\_neighbors=x[0],algorithm="kd\_tree",n\_jobs=-1)  
knn.fit(X\_train,y\_train.ravel())



KNeighborsClassifier(algorithm='kd\_tree', n\_jobs=-1, n\_neighbors=9)

***KNN results***

knn = KNeighborsClassifier(n\_neighbors = 2)   
knn.fit(X\_train,y\_train)  
y\_pred\_4 = knn.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_4))   
print("The precision is", precision\_score(y\_test, y\_pred\_4))  
print("The recall is", recall\_score(y\_test, y\_pred\_4))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_4))

The accuracy is 0.6159420289855072  
The precision is 0.7012987012987013  
The recall is 0.39416058394160586  
The F1 score is 0.5046728971962617

***Classification report***

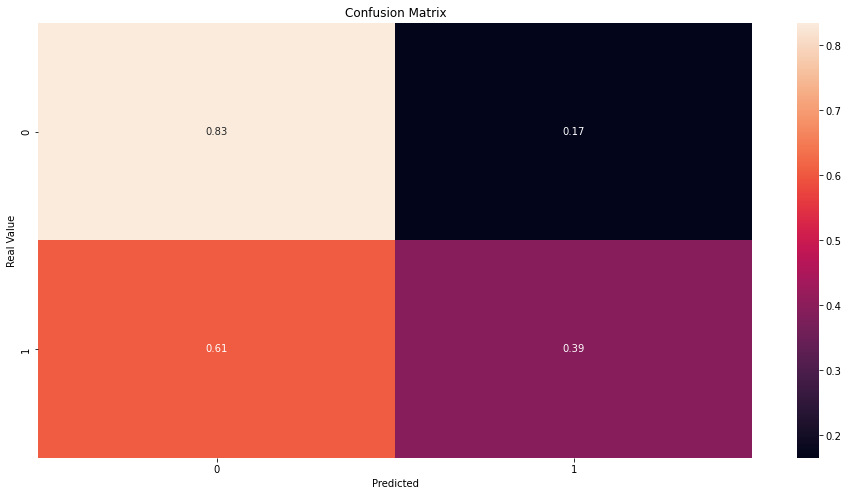
print(classification\_report(y\_test, y\_pred\_4))

precision recall f1-score support  
  
 0 0.58 0.83 0.69 139  
 1 0.70 0.39 0.50 137  
  
 accuracy 0.62 276  
 macro avg 0.64 0.61 0.60 276  
weighted avg 0.64 0.62 0.60 276

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_4, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



## Modeling for Over Sampling Data

***Splitting the Data for Over Sampling***

X=data\_over.drop(['Class'],axis=1)  
y=data\_over['Class']  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.30,random\_state=123)

### Model 1 : Random Forest Classifier - Over Sampling

# Fit and predict  
rfc = RandomForestClassifier()   
rfc.fit(X\_train, y\_train)   
y\_pred = rfc.predict(X\_test)  
   
print("The accuracy is", accuracy\_score(y\_test, y\_pred))   
print("The precision is", precision\_score(y\_test, y\_pred))  
print("The recall is", recall\_score(y\_test, y\_pred))  
print("The F1 score is", f1\_score(y\_test, y\_pred))

The accuracy is 0.9999705007787795  
The precision is 0.9999408794769016  
The recall is 1.0  
The F1 score is 0.9999704388646159

***Classification report***

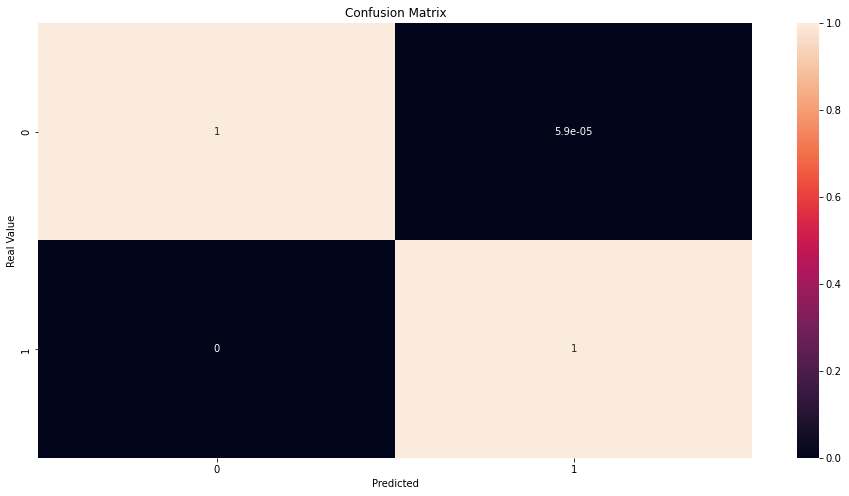
print(classification\_report(y\_test, y\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84928  
 1 1.00 1.00 1.00 84568  
  
 accuracy 1.00 169496  
 macro avg 1.00 1.00 1.00 169496  
weighted avg 1.00 1.00 1.00 169496

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 2 : Logistic Regression - Over Sampling

lr=LogisticRegression()  
lr.fit(X\_train,y\_train)  
y\_pred\_2=lr.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_2))   
print("The precision is", precision\_score(y\_test, y\_pred\_2))  
print("The recall is", recall\_score(y\_test, y\_pred\_2))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_2))

The accuracy is 0.9383348279605418  
The precision is 0.9597652663705615  
The recall is 0.9147549900671649  
The F1 score is 0.9367197432947871

***Classification report***

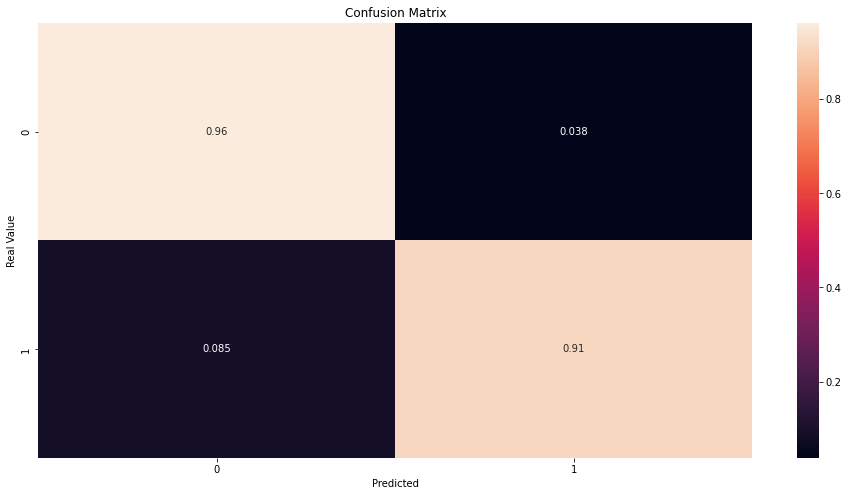
print(classification\_report(y\_test, y\_pred\_2))

precision recall f1-score support  
  
 0 0.92 0.96 0.94 84928  
 1 0.96 0.91 0.94 84568  
  
 accuracy 0.94 169496  
 macro avg 0.94 0.94 0.94 169496  
weighted avg 0.94 0.94 0.94 169496

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_2, normalize='true'), annot=True, ax=ax)  
ax.set\_title ("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 3 : Decision Tree - Over Sampling

dt=DecisionTreeRegressor()  
dt.fit(X\_train,y\_train)  
y\_pred\_3=dt.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_3))   
print("The precision is", precision\_score(y\_test, y\_pred\_3))  
print("The recall is", recall\_score(y\_test, y\_pred\_3))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_3))

The accuracy is 0.9997994052957002  
The precision is 0.9995981182477955  
The recall is 1.0  
The F1 score is 0.999799018738547

***Classification report***

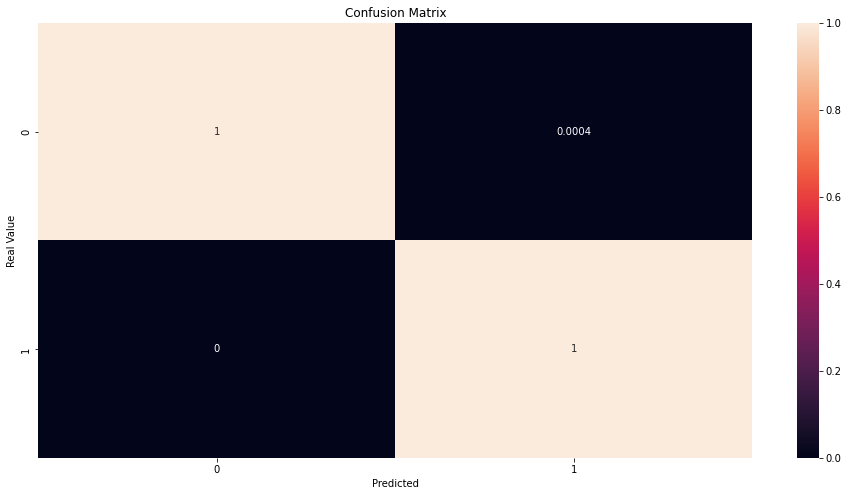
print(classification\_report(y\_test, y\_pred\_3))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84928  
 1 1.00 1.00 1.00 84568  
  
 accuracy 1.00 169496  
 macro avg 1.00 1.00 1.00 169496  
weighted avg 1.00 1.00 1.00 169496

***Confusion matrix***

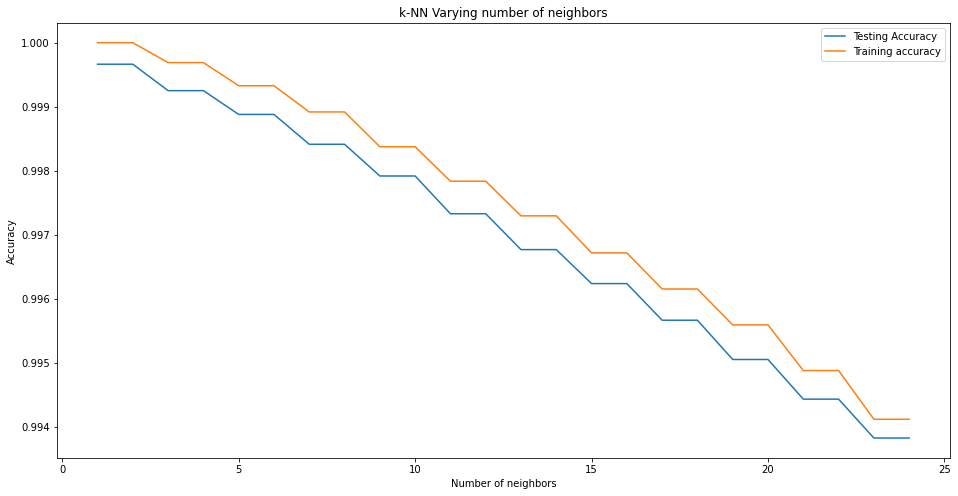
fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_3, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



### Model 4 : KNN Model - Over Sampling

neighbours = np.arange(1,25)  
train\_accuracy =np.empty(len(neighbours))  
test\_accuracy = np.empty(len(neighbours))  
  
for i,k in enumerate(neighbours):  
 #Setup a knn classifier with k neighbors  
 knn=KNeighborsClassifier(n\_neighbors=k,algorithm="kd\_tree",n\_jobs=-1)  
   
 #Fit the model  
 knn.fit(X\_train,y\_train.ravel())  
   
 #Compute accuracy on the training set  
 train\_accuracy[i] = knn.score(X\_train, y\_train.ravel())  
   
 #Compute accuracy on the test set  
 test\_accuracy[i] = knn.score(X\_test, y\_test.ravel())   
  
  
#Generate plot  
plt.title('k-NN Varying number of neighbors')  
plt.plot(neighbours, test\_accuracy, label='Testing Accuracy')  
plt.plot(neighbours, train\_accuracy, label='Training accuracy')  
plt.legend()  
plt.xlabel('Number of neighbors')  
plt.ylabel('Accuracy')  
plt.show()  
  
  
idx = np.where(test\_accuracy == max(test\_accuracy))  
x = neighbours[idx]  
  
#k\_nearest\_neighbours\_classification  
knn=KNeighborsClassifier(n\_neighbors=x[0],algorithm="kd\_tree",n\_jobs=-1)  
knn.fit(X\_train,y\_train.ravel())



KNeighborsClassifier(algorithm='kd\_tree', n\_jobs=-1, n\_neighbors=1)

***KNN results***

knn = KNeighborsClassifier(n\_neighbors = 2)   
knn.fit(X\_train,y\_train)  
y\_pred\_4 = knn.predict(X\_test)  
  
print("The accuracy is", accuracy\_score(y\_test, y\_pred\_4))   
print("The precision is", precision\_score(y\_test, y\_pred\_4))  
print("The recall is", recall\_score(y\_test, y\_pred\_4))  
print("The F1 score is", f1\_score(y\_test, y\_pred\_4))

The accuracy is 0.9996637088780856  
The precision is 0.9993264401772526  
The recall is 1.0  
The F1 score is 0.9996631066297069

***Classification report***

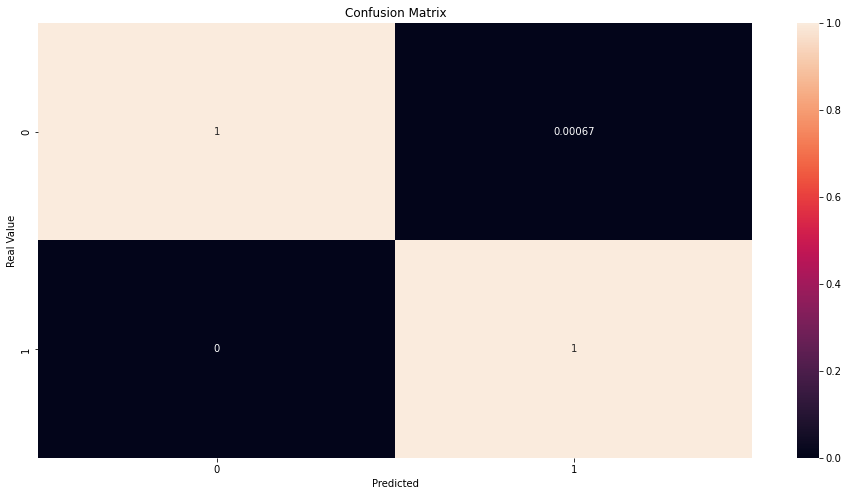
print(classification\_report(y\_test, y\_pred\_4))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 84928  
 1 1.00 1.00 1.00 84568  
  
 accuracy 1.00 169496  
 macro avg 1.00 1.00 1.00 169496  
weighted avg 1.00 1.00 1.00 169496

***Confusion matrix***

fig, ax = plt.subplots()  
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_4, normalize='true'), annot=True, ax=ax)  
ax.set\_title("Confusion Matrix")  
ax.set\_ylabel("Real Value")  
ax.set\_xlabel("Predicted")

Text(0.5, 51.0, 'Predicted')



# Conclusion

# References