Credit Card Fraud Detection

In this project you will predict fraudulent credit card transactions with the help of Machine learning models. Please import the following libraries to get started.

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn import metrics
from sklearn import preprocessing
```

Exploratory data analysis

In [2]:

```
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

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

5 rows × 31 columns

Here we will observe the distribution of our classes

In [4]:

```
classes=df['Class'].value_counts()
normal_share=classes[0]/df['Class'].count()*100
fraud_share=classes[1]/df['Class'].count()*100
```

In [5]:

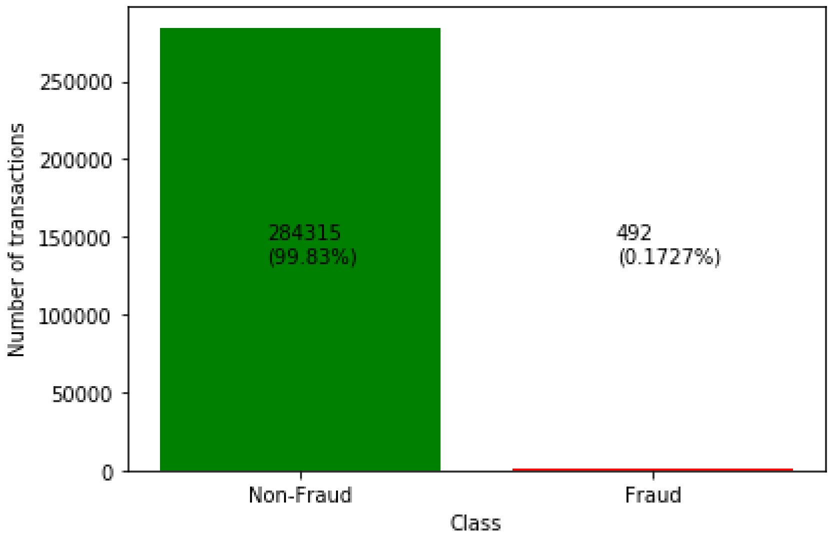
```
data1 = {'Class':df['Class'].unique(),
        'Counts': classes,
        'Percentage': [normal_share,fraud_share],
        }

df1 = pd.DataFrame (data1, columns = ['Class','Counts','Percentage'])
df1
```

Out[5]:

	Class	Counts	Percentage
0	0	284315	99.827251
1	1	492	0.172749

In [6]:

```
# Create a bar plot for the number and percentage of fraudulent vs non-fraudulent trans
cations

plt.bar(['Non-Fraud','Fraud'], df['Class'].value_counts(), color=['g','r'])
plt.xlabel('Class')
plt.ylabel('Number of transactions')
plt.annotate('{}\n({:.4}%)'.format(df['Class'].value_counts()[0],
                                         df['Class'].value_counts()[0]/df['Class'].coun
t()*100),
             (0.20, 0.45), xycoords='axes fraction')
plt.annotate('{}\n({:.4}%)'.format(df['Class'].value_counts()[1],
                                         df['Class'].value_counts()[1]/df['Class'].coun
t()*100),
             (0.70, 0.45), xycoords='axes fraction')
plt.tight_layout()
plt.show()
```

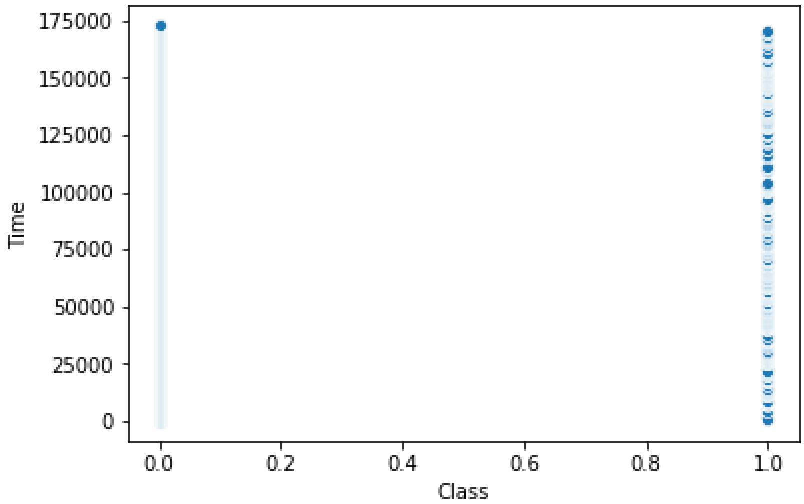


In [7]:

```
# Create a scatter plot to observe the distribution of classes with time
sns.scatterplot(x='Class', y='Time',data=df)
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18b46a2b128>
```

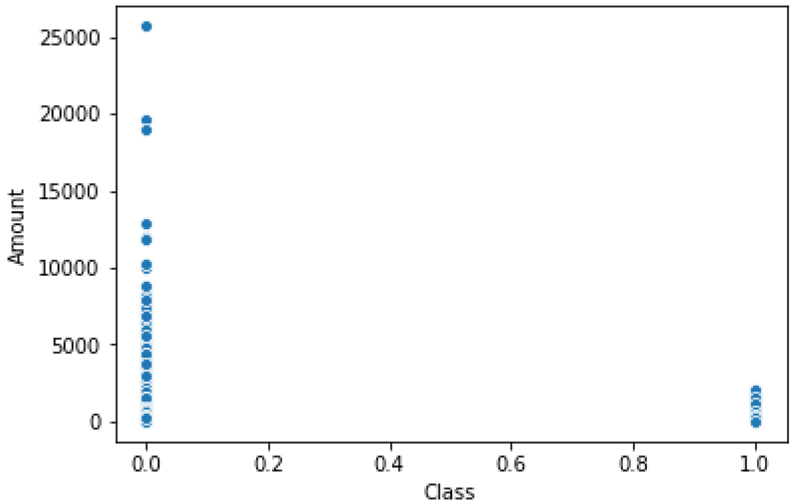


In [8]:

```
# Create a scatter plot to observe the distribution of classes with Amount
sns.scatterplot(x='Class', y='Amount', data=df)
```

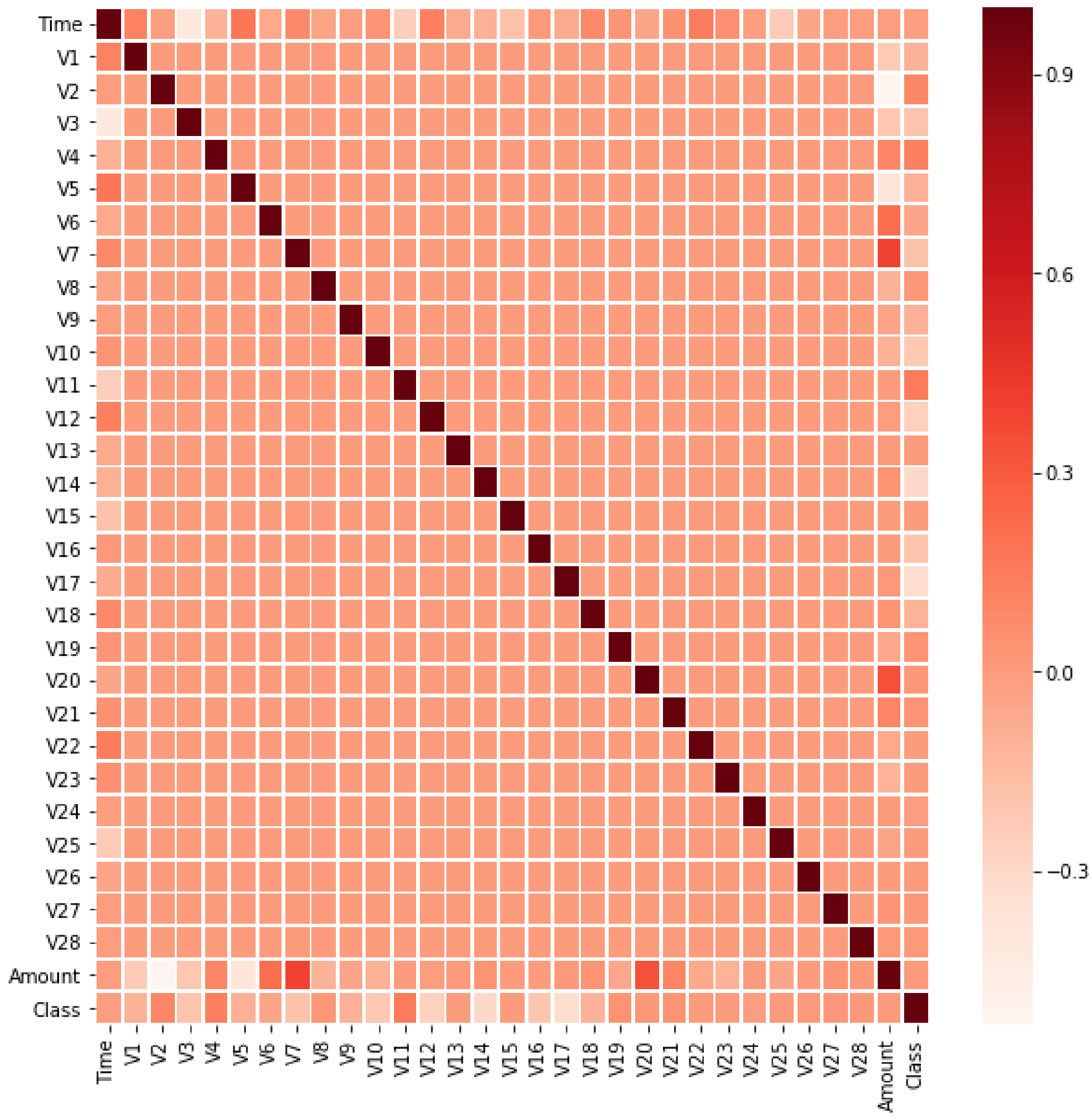
Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18b472240b8>
```



In [9]:

```
# Drop unnecessary columns
fig, ax = plt.subplots(figsize=(10,10))
corr = df.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.8,cmap=
"Reds",ax=ax)
plt.show()
```



Splitting the data into train & test data

In [10]:

```
y= df['Class']#class variable
X = df.drop(columns=['Class'])
```

In [11]:

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y,ran
dom_state=42)
```

Preserve X_test & y_test to evaluate on the test data once you build the model

In [12]:

```
print(np.sum(y))
print(np.sum(y_train))
print(np.sum(y_test))
```

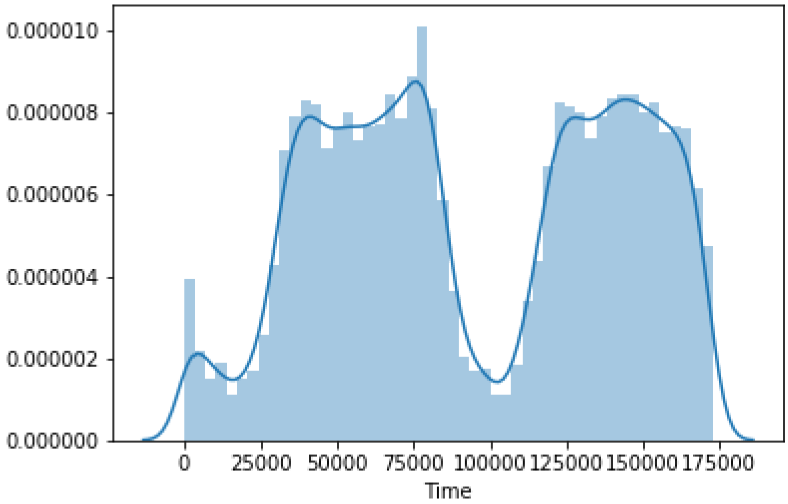
```
492
344
148
```

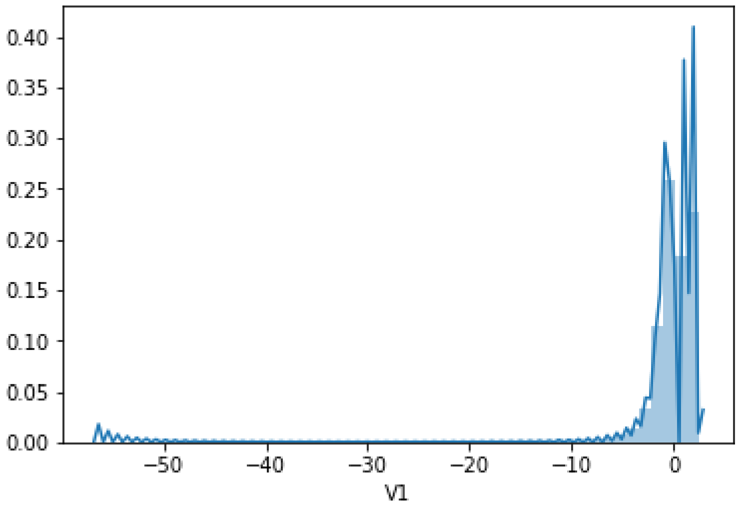
Plotting the distribution of a variable

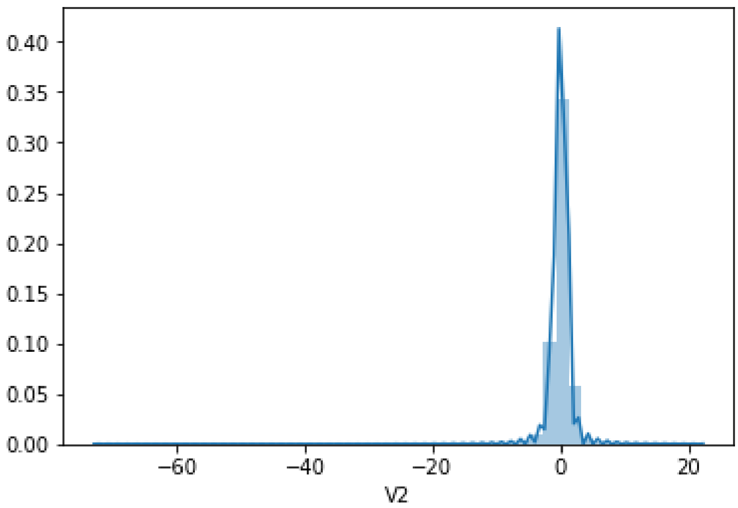
In [13]:

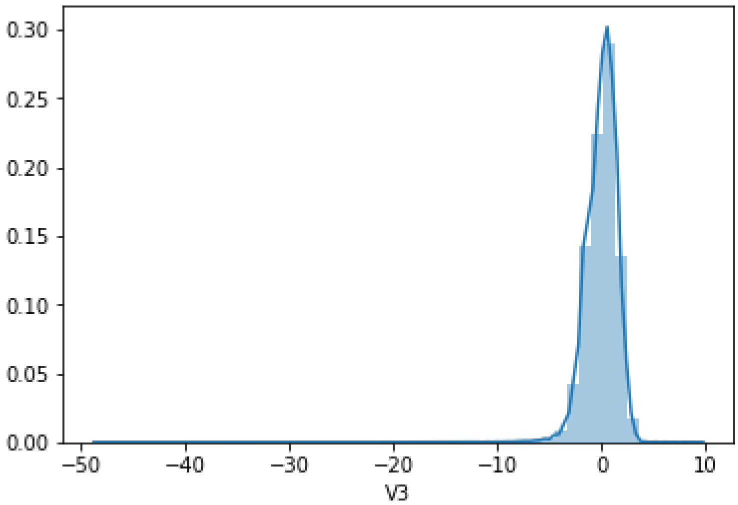
```
# plot the histogram of a variable from the dataset to see the skewness
for i, col in enumerate(X_train.columns):
    plt.figure(i)
    sns.distplot(X_train[col])
```

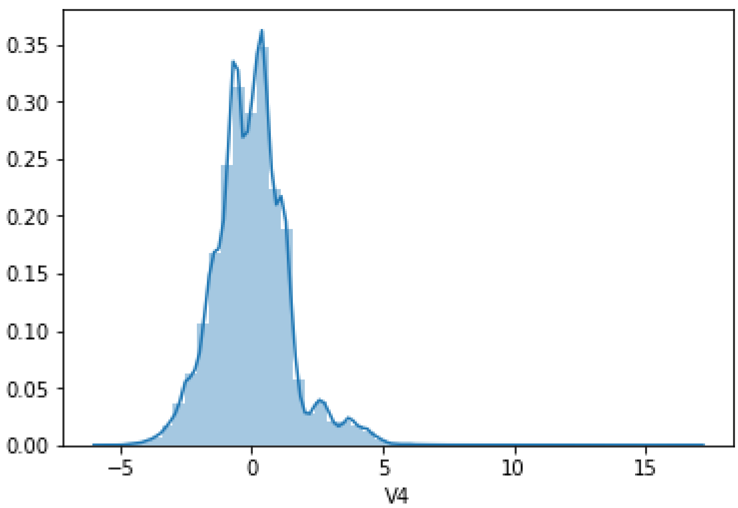
```
C:\Users\family\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: Fut
ureWarning: Using a non-tuple sequence for multidimensional indexing is de
precated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this
will be interpreted as an array index, `arr[np.array(seq)]`, which will re
sult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\Users\family\Anaconda3\lib\site-packages\matplotlib\pyplot.py:514: Runt
imeWarning: More than 20 figures have been opened. Figures created through
the pyplot interface (`matplotlib.pyplot.figure`) are retained until expli
citly closed and may consume too much memory. (To control this warning, se
e the rcParam `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

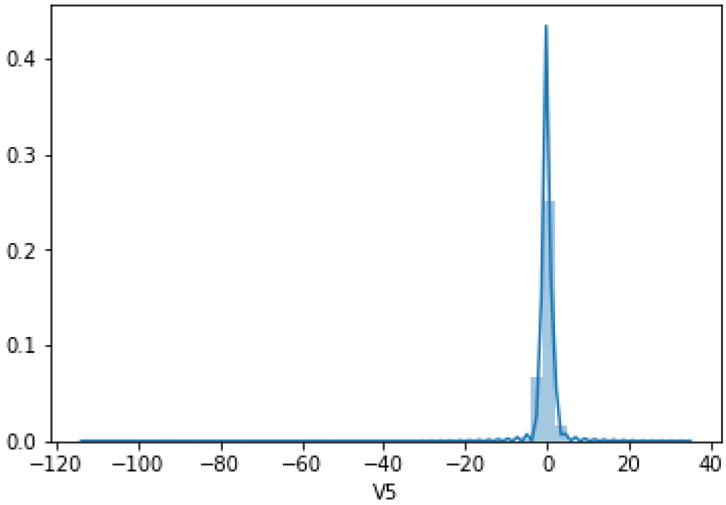


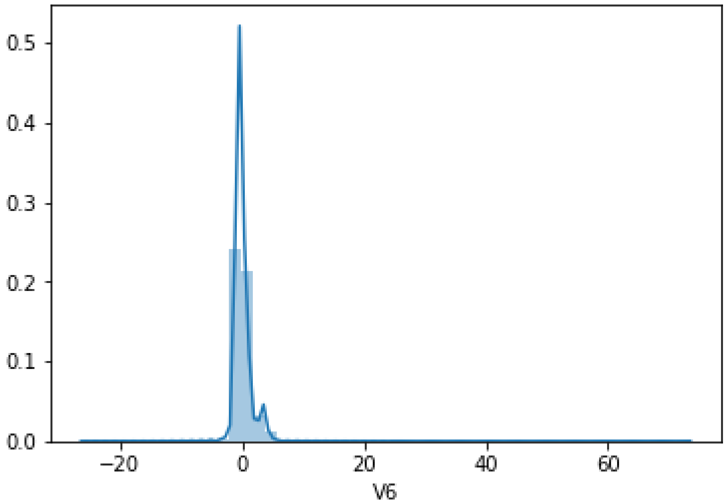


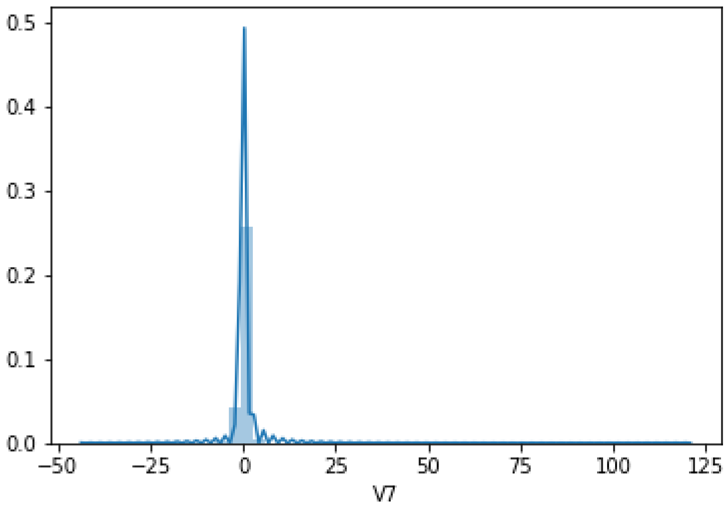

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
V2

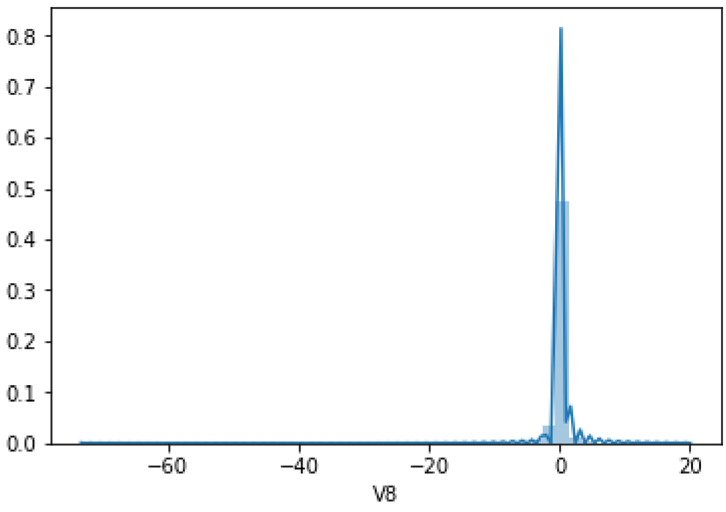

0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
10
V3

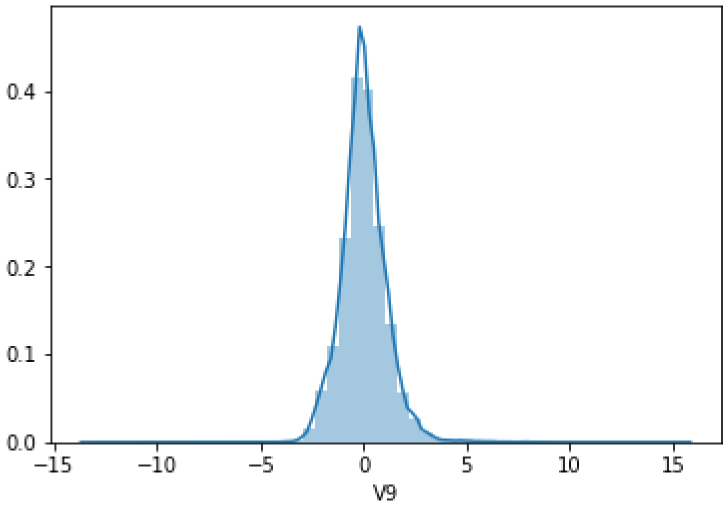

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
V4

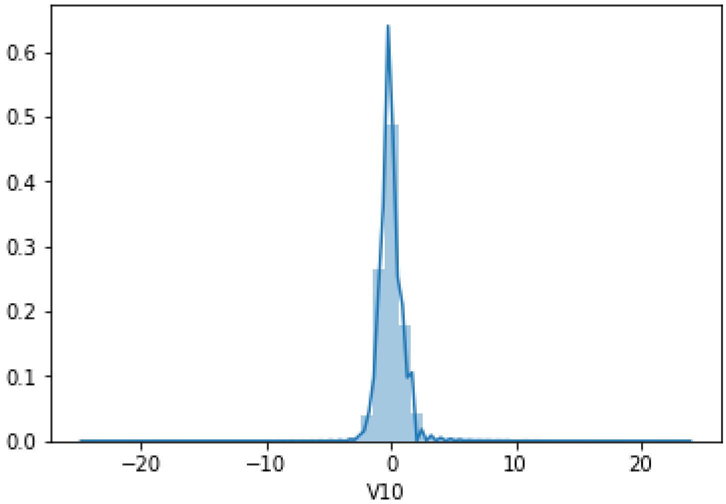

0.4
0.3
0.2
0.1
0.0
−120
−100
−80
−60
−40
−20
0
20
40
V5

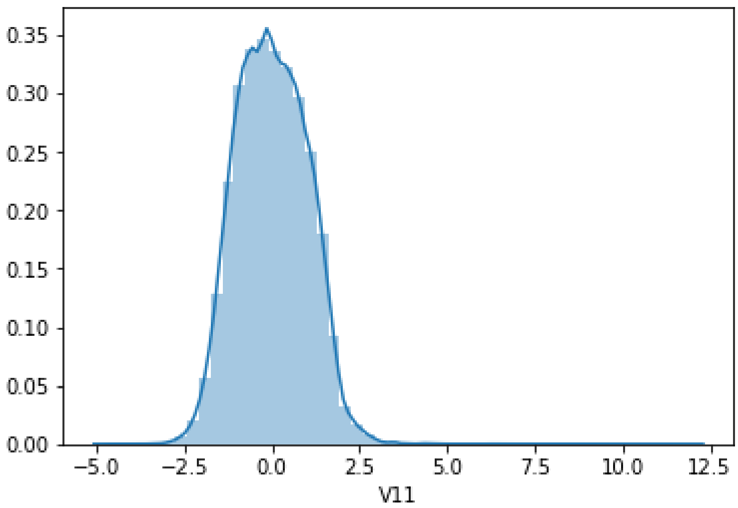

0.5
0.4
0.3
0.2
0.1
0.0
−20
0
20
40
60
V6

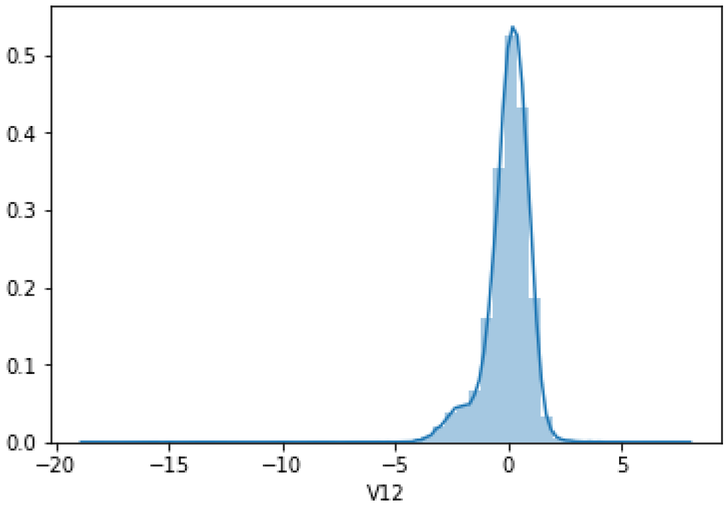

0.5
0.4
0.3
0.2
0.1
0.0
−50
−25
0
25
50
75
100
125
V7

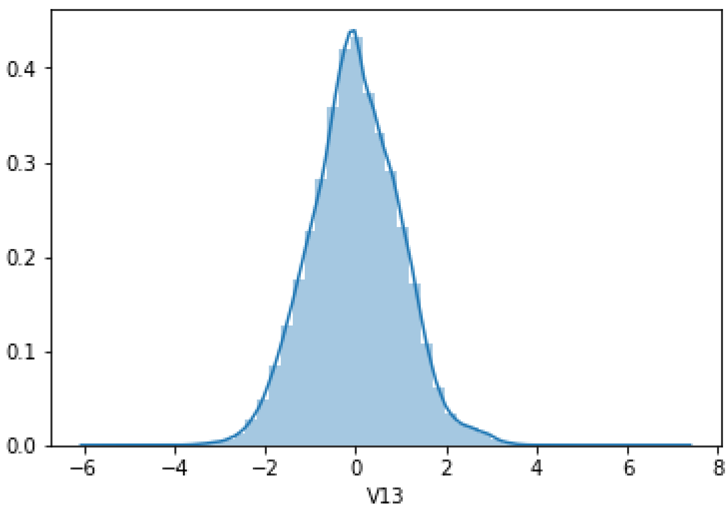

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−60
−40
−20
0
20
V8

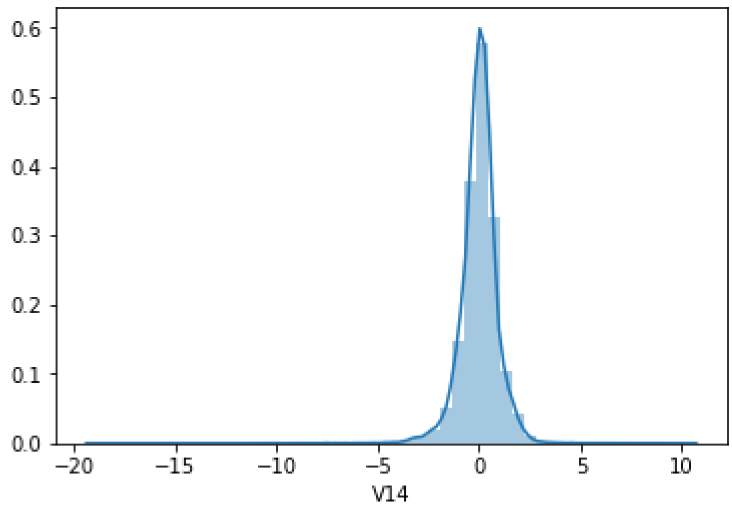

0.4
0.3
0.2
0.1
0.0
−15
−10
−5
0
5
10
15
V9

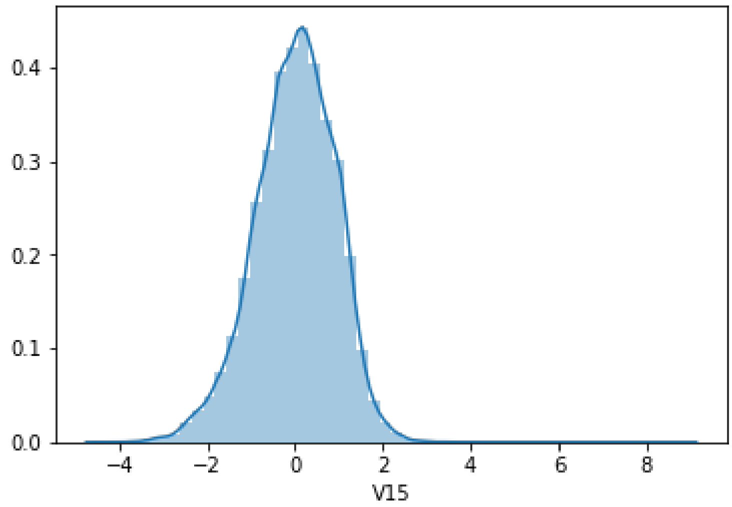

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−20
−10
0
10
20
V10

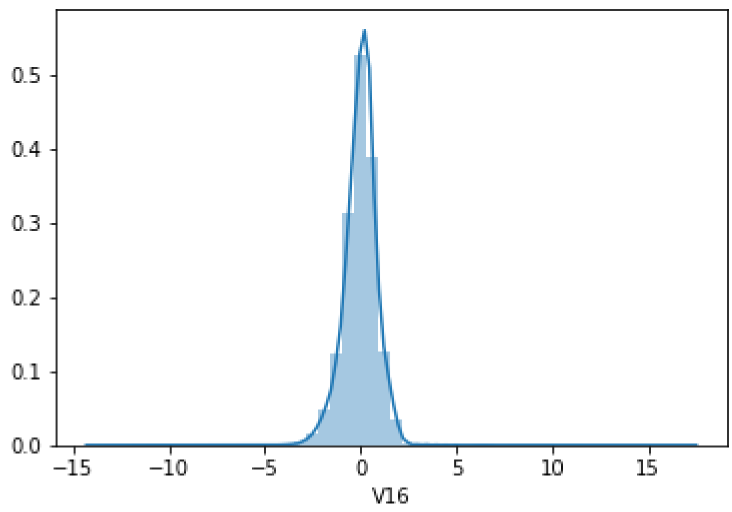

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
V11

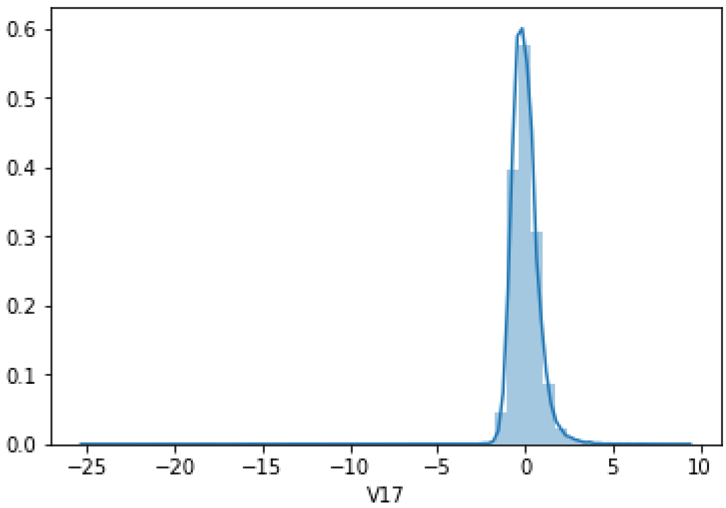

0.5
0.4
0.3
0.2
0.1
0.0
−20
−15
−10
−5
0
5
V12

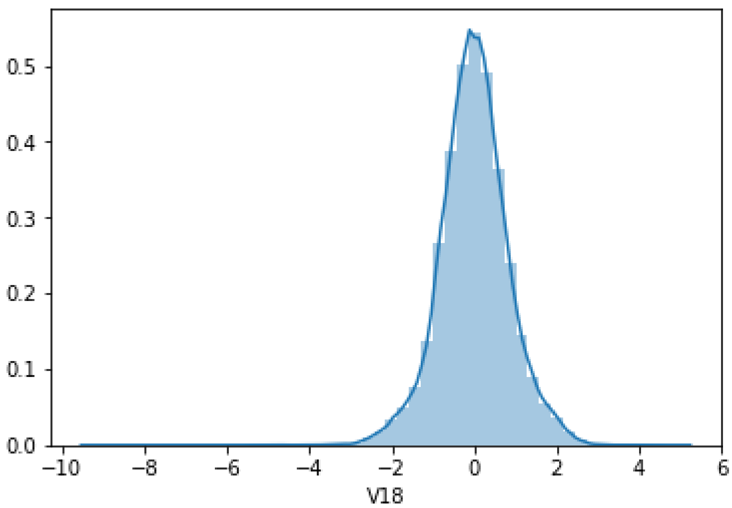

0.4
0.3
0.2
0.1
0.0
−6
−4
−2
0
2
4
6
8
V13

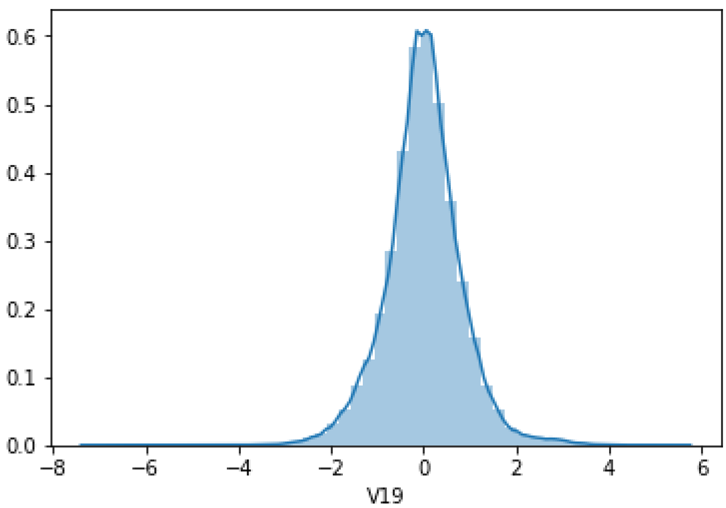

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−20
−15
−10
−5
0
5
10
V14

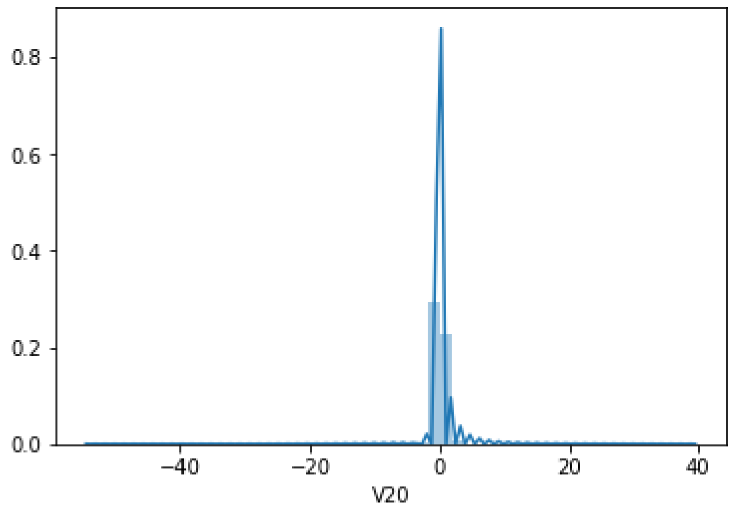

0.4
0.3
0.2
0.1
0.0
−4
−2
0
2
4
6
8
V15

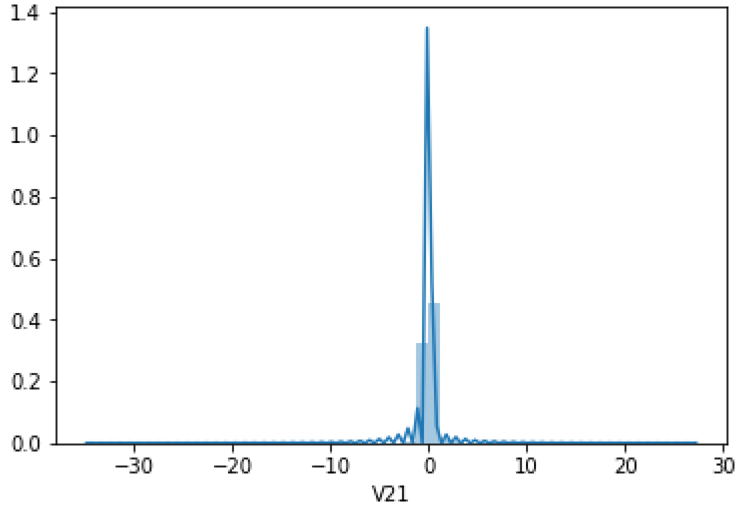

0.5
0.4
0.3
0.2
0.1
0.0
−15
−10
−5
0
5
10
15
V16

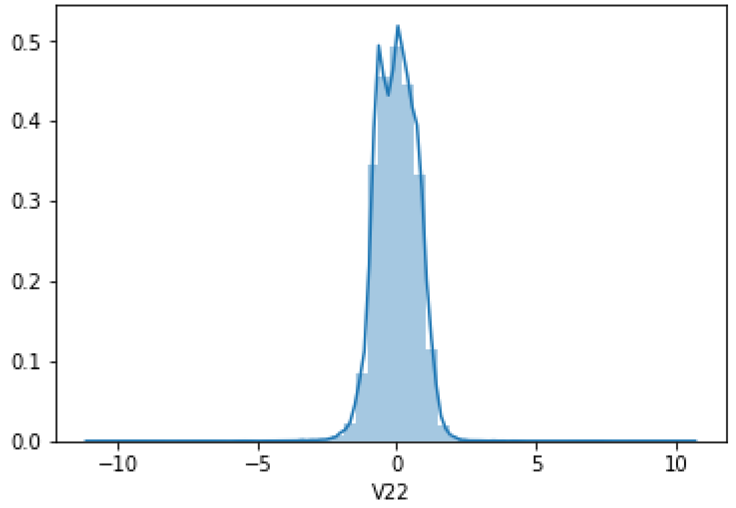

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−25 −20 −15 −10 −5 0 5 10
V17

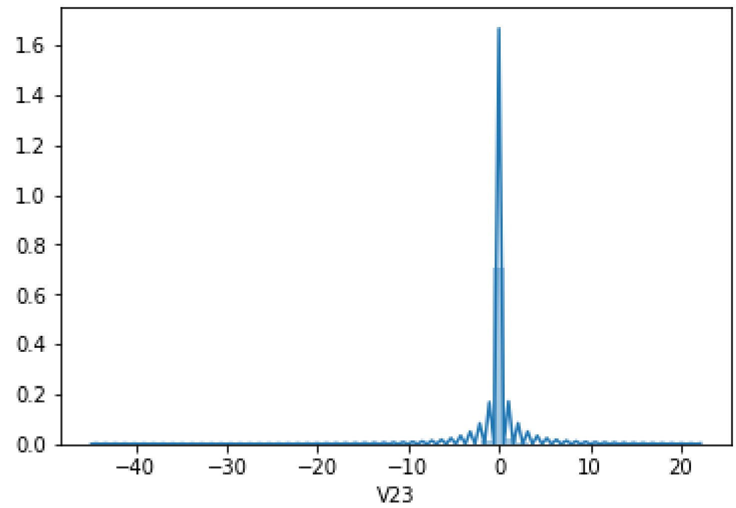

0.5
0.4
0.3
0.2
0.1
0.0
−10 −8 −6 −4 −2 0 2 4 6
V18

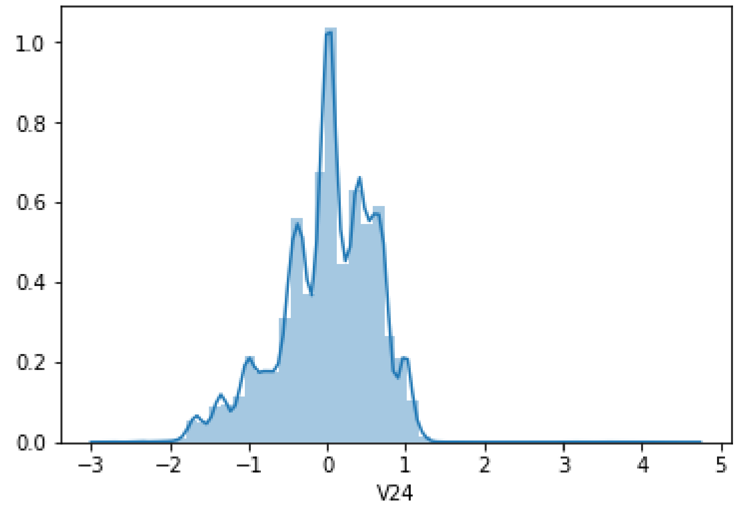

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−8 −6 −4 −2 0 2 4 6
V19

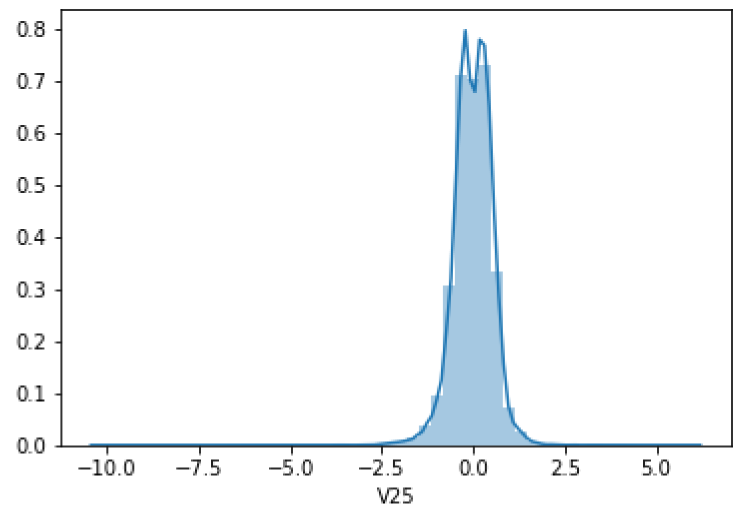

0.8
0.6
0.4
0.2
0.0
−40
−20
0
20
40
V20

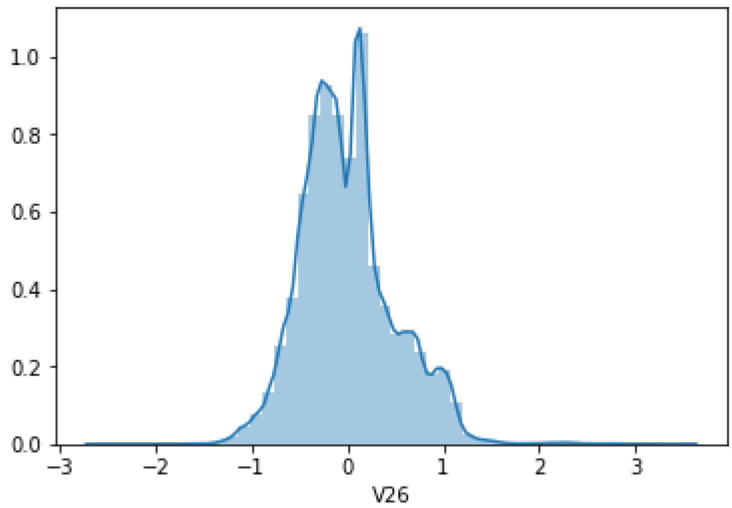

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−30
−20
−10
0
10
20
30
V21

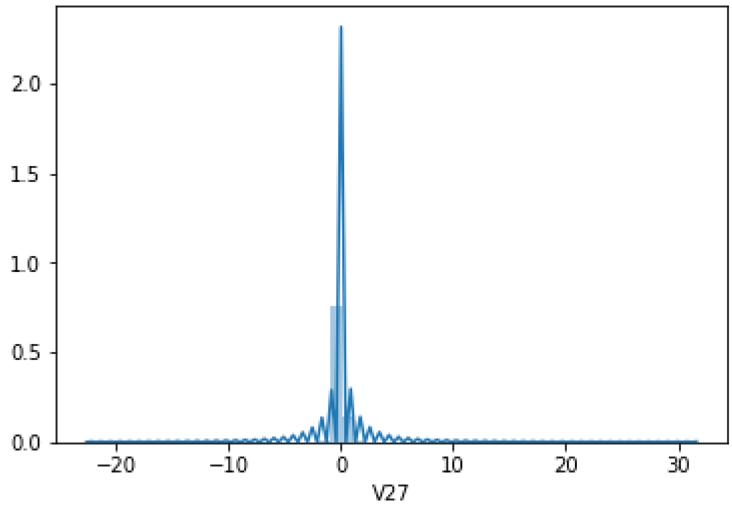

0.5
0.4
0.3
0.2
0.1
0.0
−10
−5
0
5
10
V22

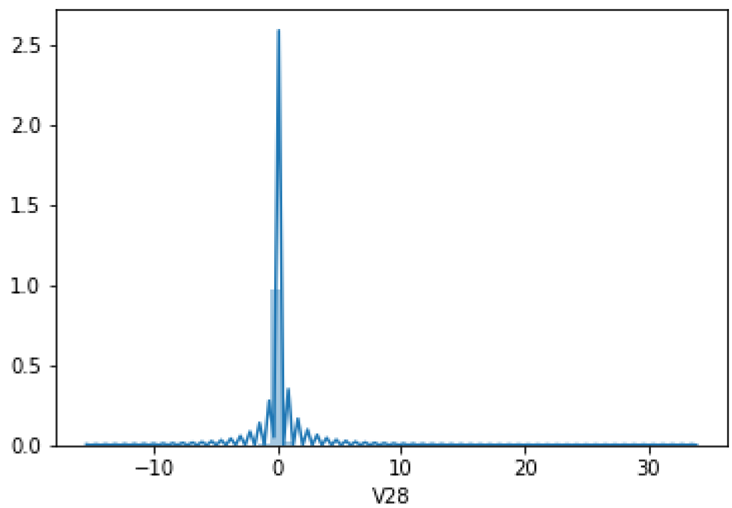

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−40
−30
−20
−10
0
10
20
V23

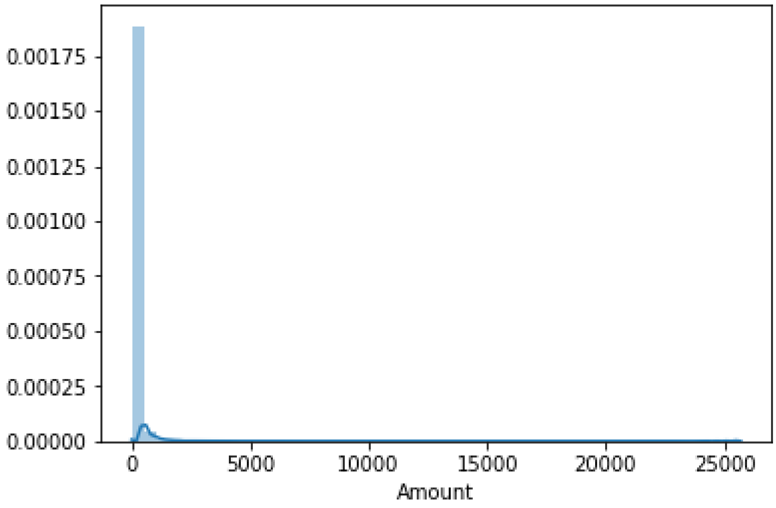

1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3
4
5
V24


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
V25


1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3
V26


2.0
1.5
1.0
0.5
0.0
−20
−10
0
10
20
30
V27


2.5
2.0
1.5
1.0
0.5
0.0
−10
0
10
20
30
V28



If there is skewness present in the distribution use:

- **Power Transformer** package present in the **preprocessing library provided by sklearn** to make distribution more gaussian

In [14]:

```
print(type(X_train))
print(type(y_train))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

In [63]:

```
# - Apply : preprocessing.PowerTransformer(copy=False) to fit & transform the train & t
est data
#Below code is commented since sklearn.preprocessing is giving error after XGBoost inst
allation

#from sklearn.preprocessing import PowerTransformer
#df['scaled_amount'] = RobustScaler().fit_transform(df['Amount'].values.reshape(-1,1))
#df['scaled_time'] = RobustScaler().fit_transform(df['Time'].values.reshape(-1,1))

# Make a new dataset named "df_scaled" dropping out original "Time" and "Amount"
#df_scaled = df.drop(['Time','Amount'],axis = 1,inplace=False)
#df_scaled.head()                      ## Fit the PT on training data
```

Model Building

- Build different models on the imbalanced dataset and see the result

Similarly explore other algorithms by building models like:

- KNN
- SVM
- Decision Tree
- Random Forest
- XGBoost

Proceed with the model which shows the best result

- Apply the best hyperparameter on the model
- Predict on the test dataset

In [20]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import itertools
import xgboost as xgb
from xgboost import XGBClassifier
```

```
C:\Users\family\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosti
ng.py:29: DeprecationWarning: numpy.core.umath_tests is an internal NumPy
module and should not be imported. It will be removed in a future NumPy re
lease.
  from numpy.core.umath_tests import inner1d
```

In [21]:

```
classifiers = []

classifiers.append(('Logistic Regression', LogisticRegression()))
classifiers.append(('KNN', KNeighborsClassifier()))
#classifiers.append(('SVM', SVC(random_state=42)))
classifiers.append(('Decision Tree', DecisionTreeClassifier(random_state=42)))
classifiers.append(('Random Forest', RandomForestClassifier(random_state=42)))
classifiers.append(('XGBoost', XGBClassifier(random_state=42)))
#Ensemble classifier - All classifiers have the same weight
eclf = VotingClassifier(estimators=classifiers, voting='soft', weights=np.ones(len(clas
sifiers)))
```

In [22]:

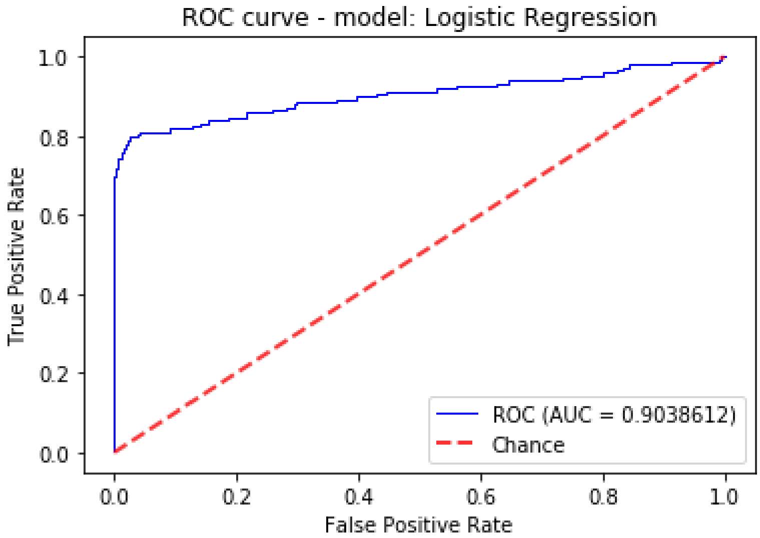
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    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)

    #if normalize:
    #    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    #    print("Normalized confusion matrix")
    #else:
    #    print('Confusion matrix, without normalization')

    #print(cm)

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

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

In [23]:

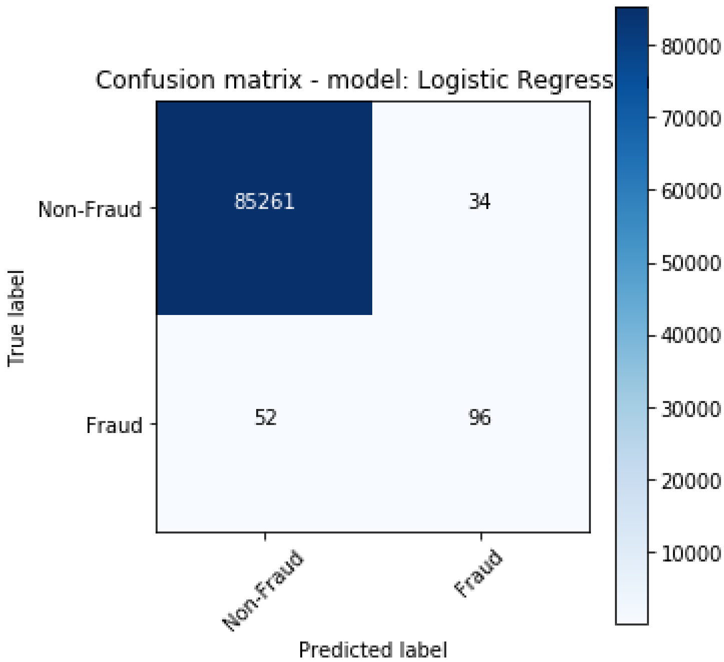
```
from sklearn import svm
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from scipy import interp

def plot_CM_and_ROC_curve(classifier, X_train, y_train, X_test, y_test):
    '''Plots the ROC curve and the confusion matrix, and calculates AUC, recall and pre
cision.'''

    name = classifier[0]
    classifier = classifier[1]

    mean_fpr = np.linspace(0, 1, 100)
    class_names = ['Non-Fraud', 'Fraud']
    confusion_matrix_total = [[0, 0], [0, 0]]

    #Obtain probabilities for each class
    probas_ = classifier.fit(X_train, y_train).predict_proba(X_test)

    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=1, alpha=1, color='b', label='ROC (AUC = %0.7f)' % (roc_auc))
    plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
             label='Chance', alpha=.8)
    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve - model: ' + name)
    plt.legend(loc="lower right")
    plt.figure(figsize=(5,5))
    plt.show()

    #Store the confusion matrix result to plot a table later
    y_pred=classifier.predict(X_test)
    cnf_matrix = confusion_matrix(y_test, y_pred)
    confusion_matrix_total += cnf_matrix

    #Print precision and recall
    tn, fp = confusion_matrix_total.tolist()[0]
    fn, tp = confusion_matrix_total.tolist()[1]
    accuracy = (tp+tn)/(tp+tn+fp+fn)
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    print('Accuracy = {:2.2f}%'.format(accuracy*100))
    print('Precision = {:2.2f}%'.format(precision*100))
    print('Recall = {:2.2f}%'.format(recall*100))

    # Plot confusion matrix
    plt.figure(figsize=(5,5))
    plot_confusion_matrix(confusion_matrix_total, classes=class_names, title='Confusion
matrix - model: ' + name)
    plt.show()
```

In [24]:

```
#for clf in classifiers:
plot_CM_and_ROC_curve(classifiers[0], X_train, y_train, X_test, y_test)
```

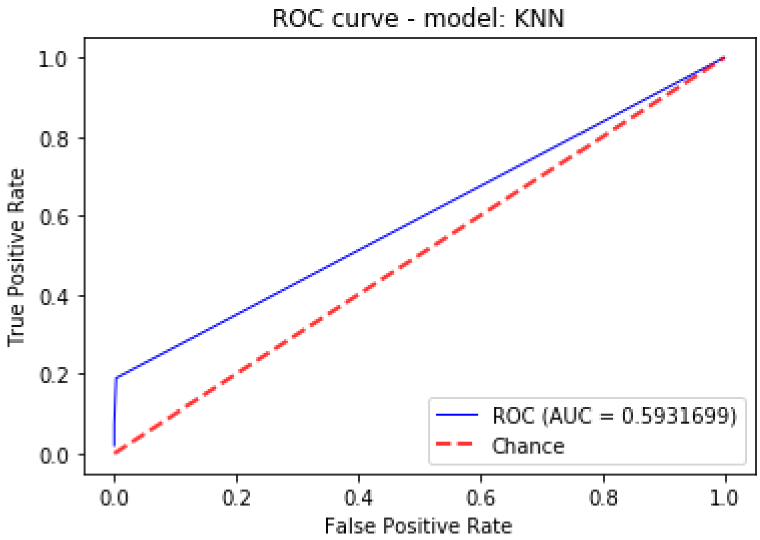



```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.90%
Precision = 73.85%
Recall = 64.86%
```

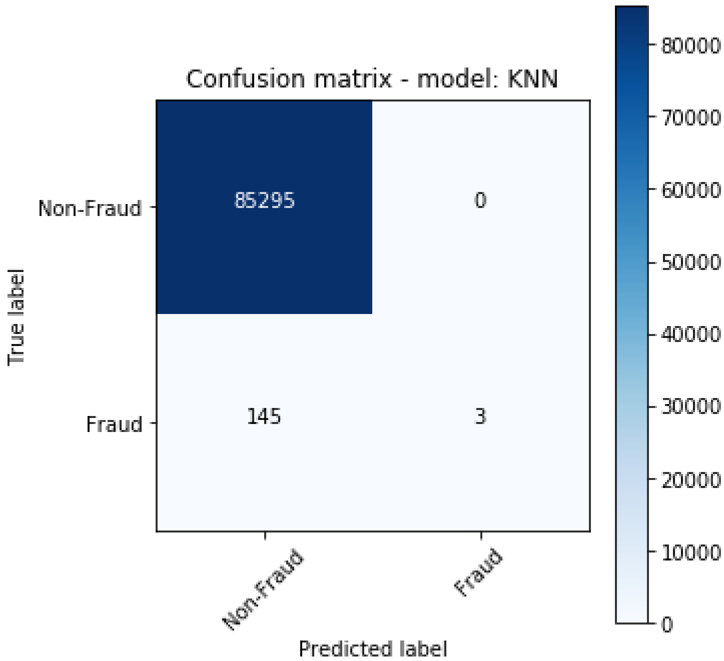


In [25]:

```
plot_CM_and_ROC_curve(classifiers[1], X_train, y_train, X_test, y_test)
```

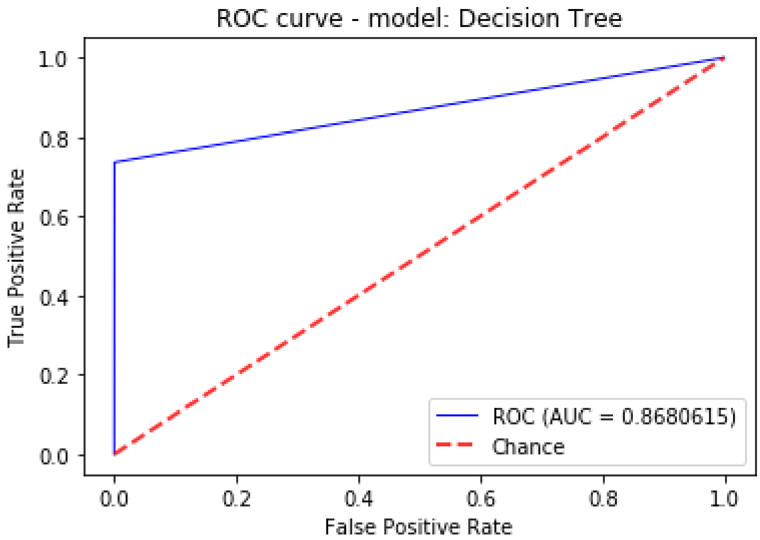



```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.83%
Precision = 100.00%
Recall = 2.03%
```



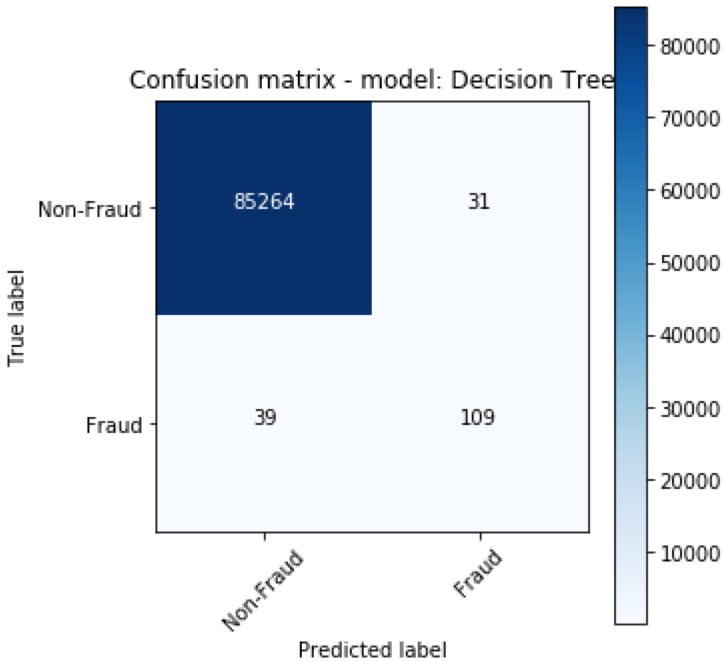
In [26]:

```
plot_CM_and_ROC_curve(classifiers[2], X_train, y_train, X_test, y_test)
```



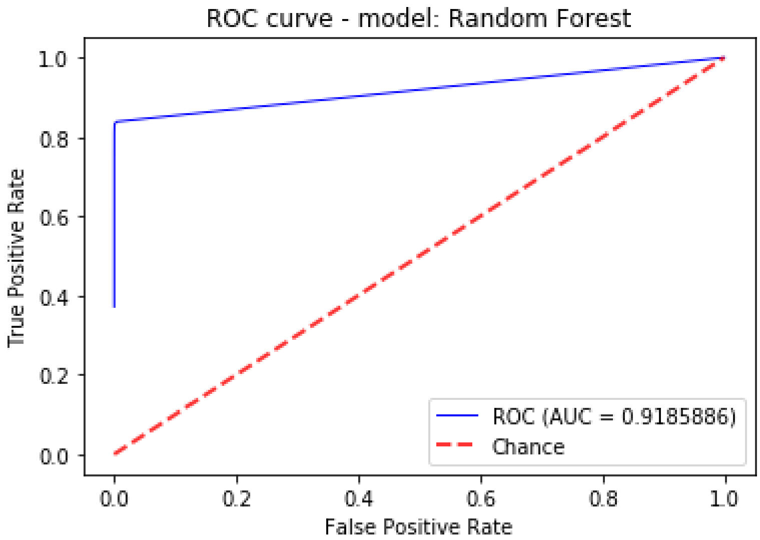

```
<Figure size 360x360 with 0 Axes>
```

```
Accuracy = 99.92%
Precision = 77.86%
Recall = 73.65%
```

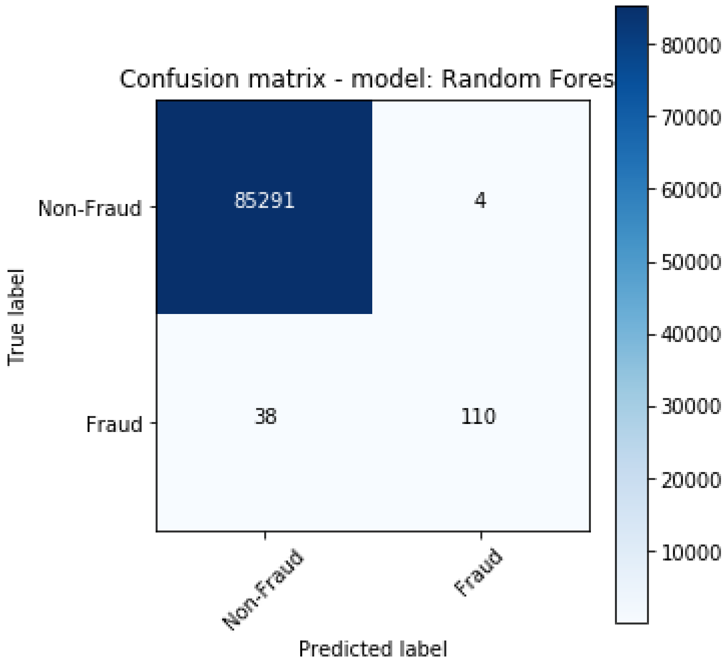


In [27]:

```
plot_CM_and_ROC_curve(classifiers[3], X_train, y_train, X_test, y_test)
```

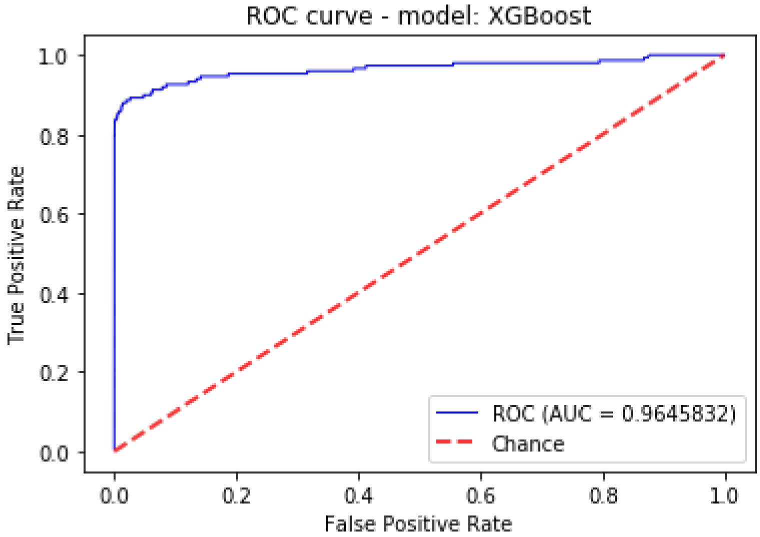



```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.95%
Precision = 96.49%
Recall = 74.32%
```

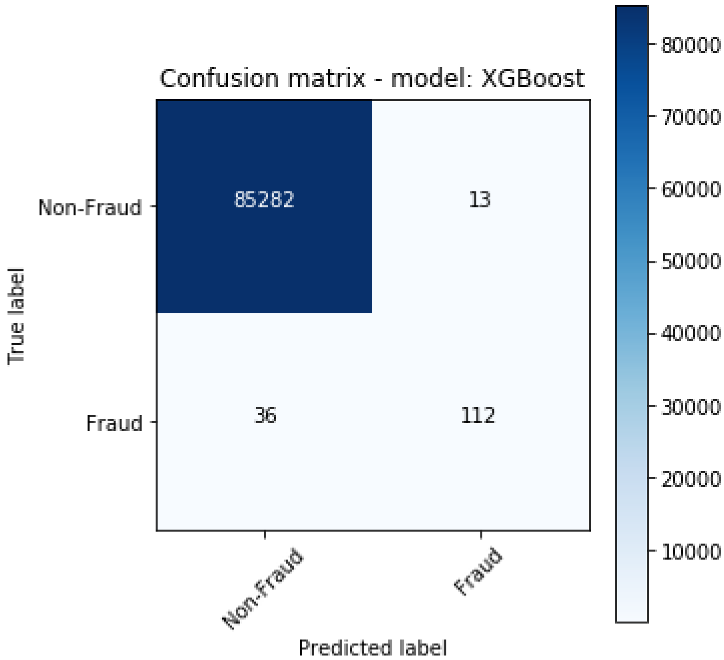


In [28]:

```
plot_CM_and_ROC_curve(classifiers[4], X_train, y_train, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>

C:\Users\family\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:

Accuracy = 99.94%
Precision = 89.60%
Recall = 75.68%
```



Print the important features of the best model to understand the dataset

- This will not give much explanation on the already transformed dataset
- But it will help us in understanding if the dataset is not PCA transformed

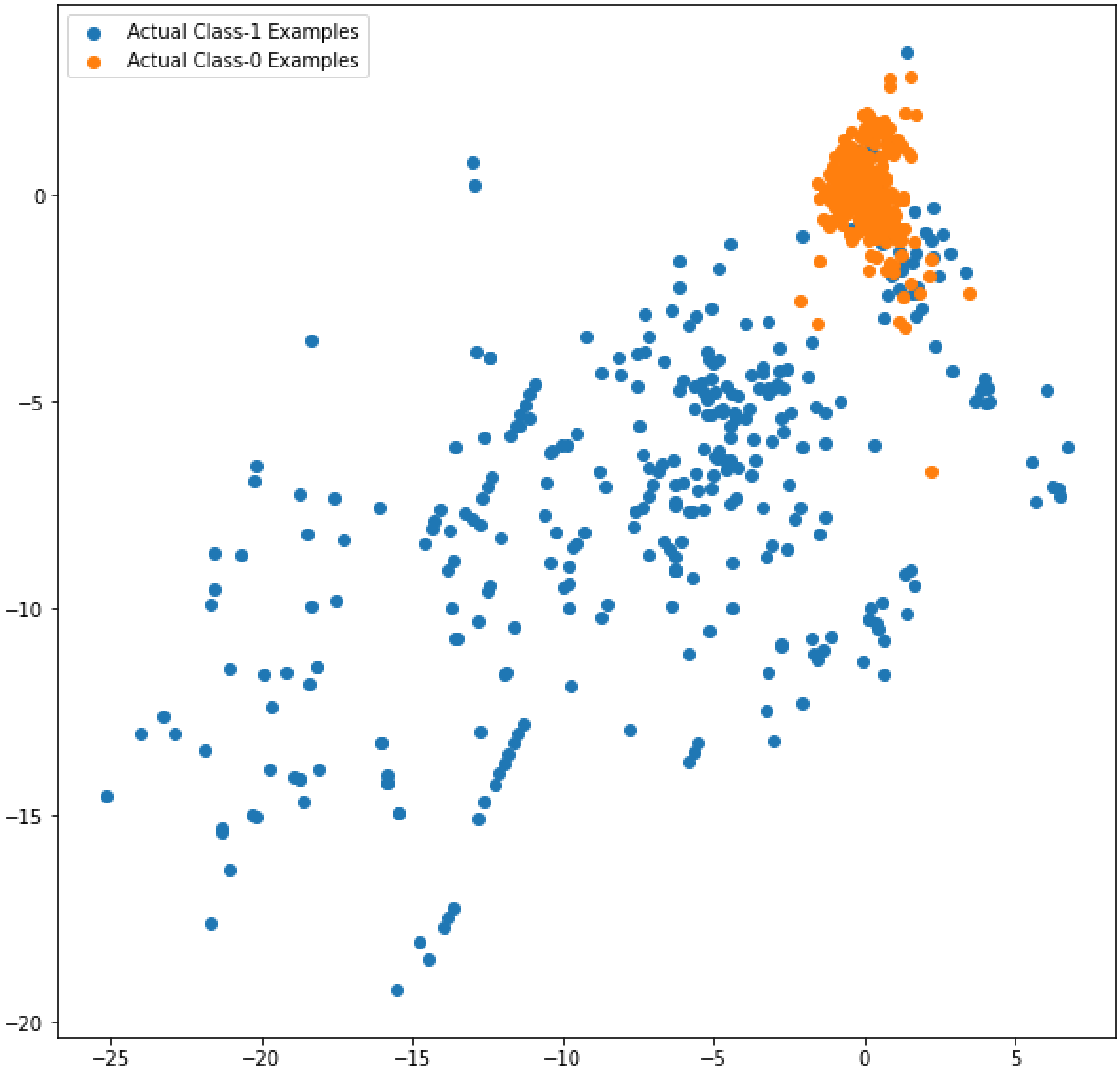
In [29]:

```
clf = classifiers[4][1]
var_imp = []
for i in clf.feature_importances_:
    var_imp.append(i)
print('Top var =', var_imp.index(np.sort(clf.feature_importances_)[-1])+1)
print('2nd Top var =', var_imp.index(np.sort(clf.feature_importances_)[-2])+1)
print('3rd Top var =', var_imp.index(np.sort(clf.feature_importances_)[-3])+1)

# Variable on Index-16 and Index-13 seems to be the top 2 variables
top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-1])
second_top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-2])

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]

np.random.shuffle(X_train_0)

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [10, 10]

plt.scatter(X_train_1[:, top_var_index], X_train_1[:, second_top_var_index], label='Act
ual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], top_var_index], X_train_0[:X_train_1.shape[0
], second_top_var_index],
            label='Actual Class-0 Examples')
plt.legend()
```

```
Top var = 18
2nd Top var = 15
3rd Top var = 8
```

Out[29]:

```
<matplotlib.legend.Legend at 0x18b4830ea58>
```



Model building with balancing Classes

Perform class balancing with :

- Random Oversampling
- SMOTE
- ADASYN

Model Building

- Build different models on the balanced dataset and see the result

Random Oversampling

In [30]:

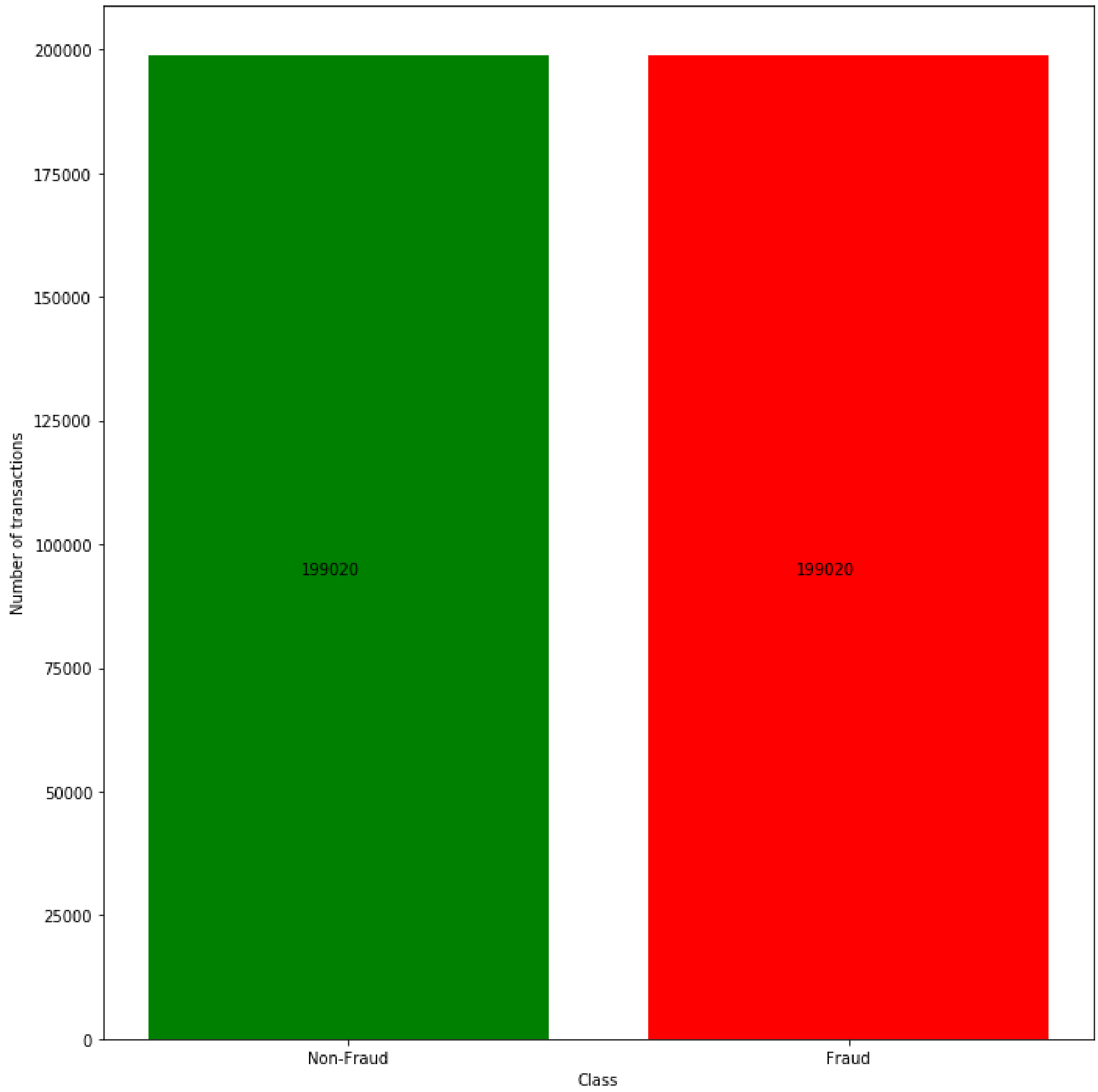
```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn import over_sampling #- import the packages

#perform cross validation & then balance classes on X_train_cv & y_train_cv using Rando
m Oversampling
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_sample(X_train, y_train)
```

In [31]:

```
from collections import Counter
```

In [32]:

```
plt.bar(['Non-Fraud','Fraud'], [Counter(y_ros)[0], Counter(y_ros)[1]], color=['g','r'])
plt.xlabel('Class')
plt.ylabel('Number of transactions')
plt.annotate('{}'.format(Counter(y_ros)[0]), (0.20, 0.45), xycoords='axes fraction')
plt.annotate('{}'.format(Counter(y_ros)[1]), (0.70, 0.45), xycoords='axes fraction')

plt.tight_layout()
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```

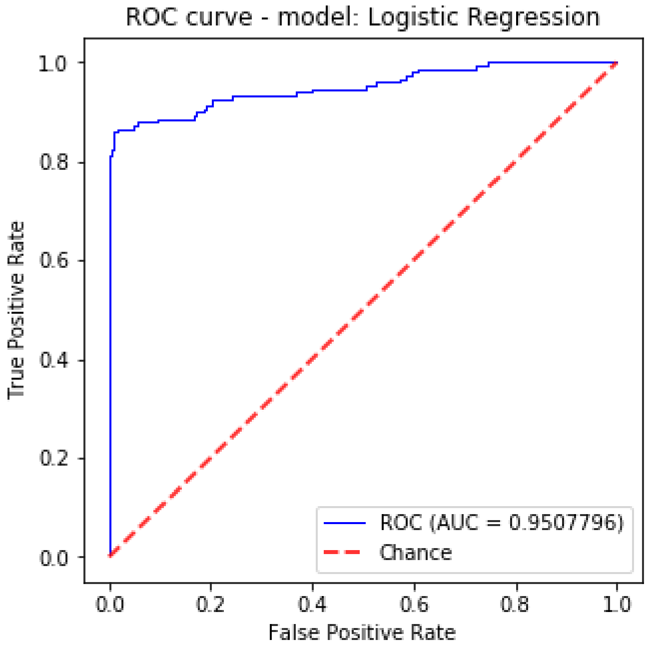


Similarly explore other algorithms on balanced dataset by building models like:

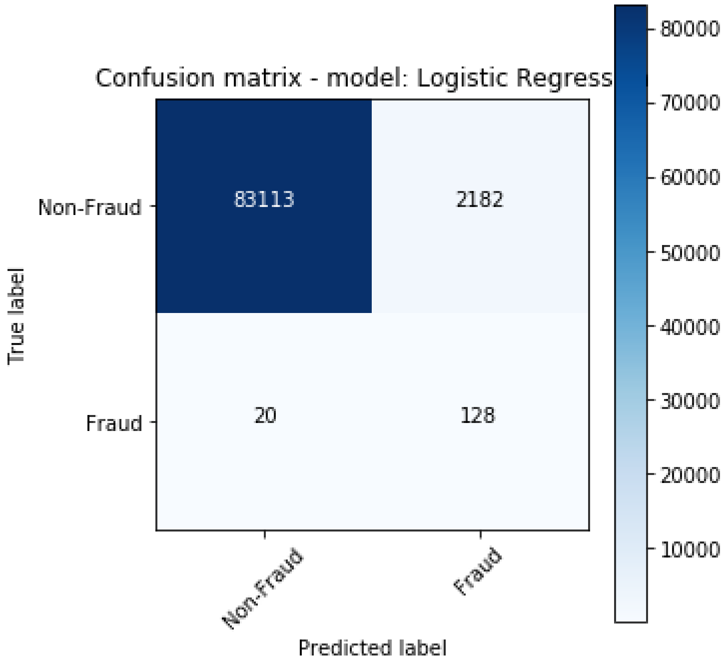
- KNN
- SVM
- Decision Tree
- Random Forest
- XGBoost

In [33]:

```
plot_CM_and_ROC_curve(classifiers[0], X_ros, y_ros, X_test, y_test)
```

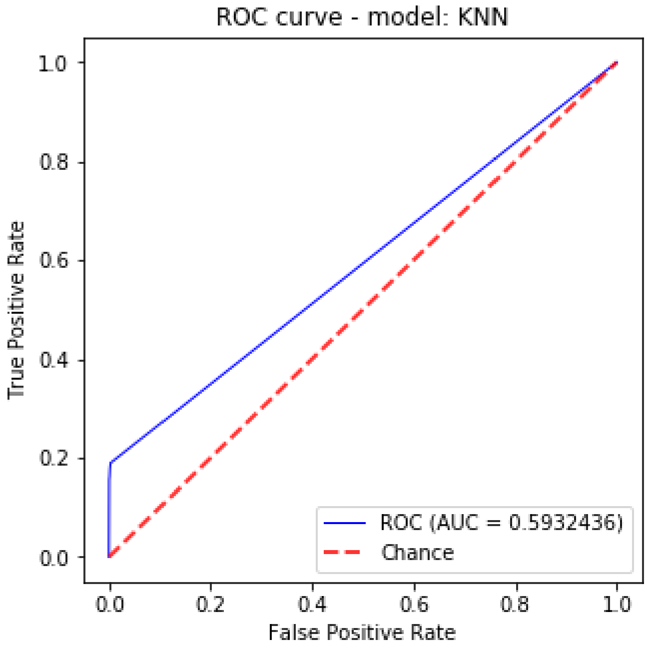



```
<Figure size 360x360 with 0 Axes>

Accuracy = 97.42%
Precision = 5.54%
Recall = 86.49%
```


Confusion matrix - model: Logistic Regress
True label
Non-Fraud
Fraud
83113
2182
20
128
Non-Fraud
Fraud
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

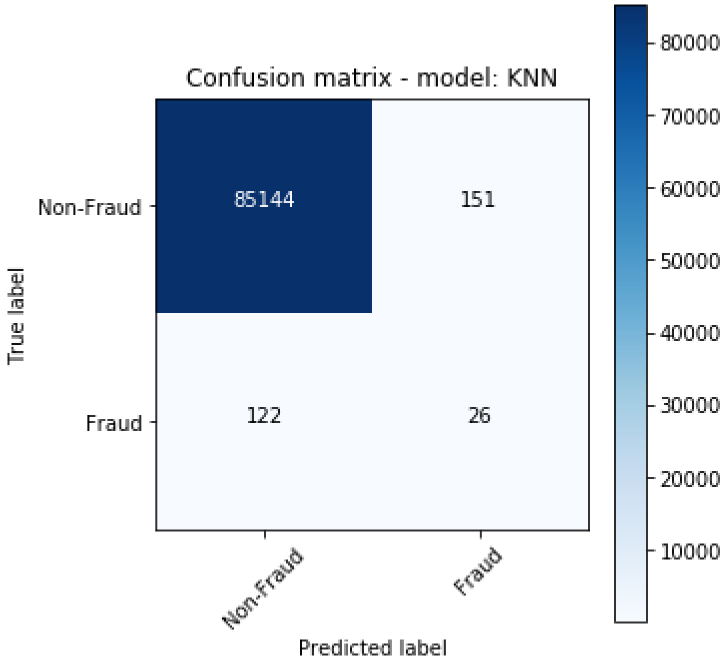
In [34]:

```
plot_CM_and_ROC_curve(classifiers[1], X_ros, y_ros, X_test, y_test)
```



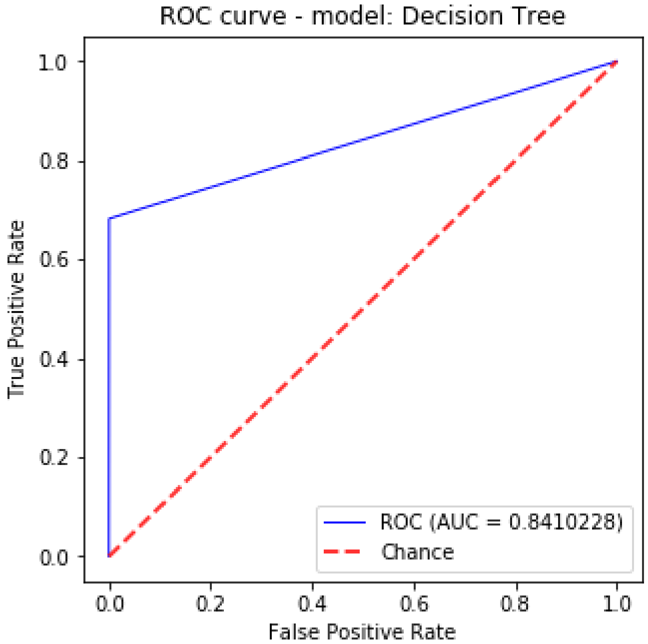

```
<Figure size 360x360 with 0 Axes>
```

```
Accuracy = 99.68%
Precision = 14.69%
Recall = 17.57%
```

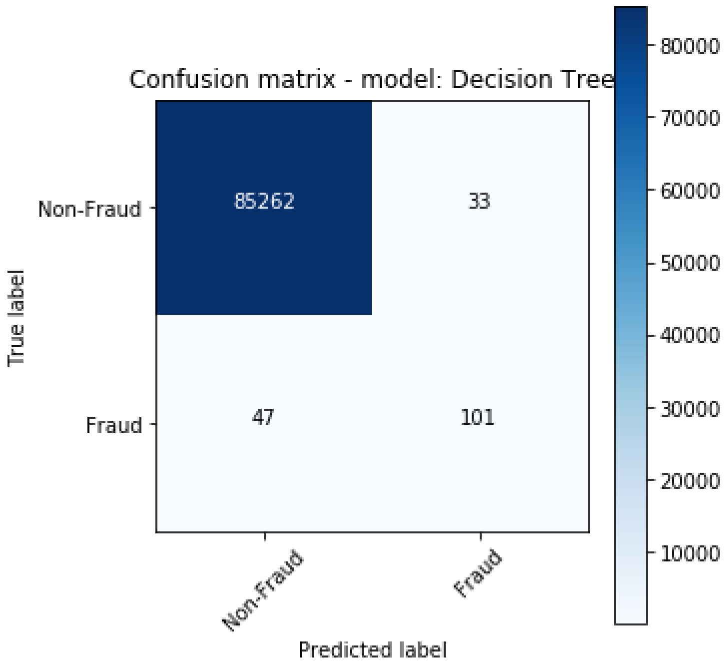

Confusion matrix - model: KNN
Non-Fraud
Fraud
True label
85144
151
122
26
Non-Fraud
Fraud
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

In [35]:

```
plot_CM_and_ROC_curve(classifiers[2], X_ros, y_ros, X_test, y_test)
```

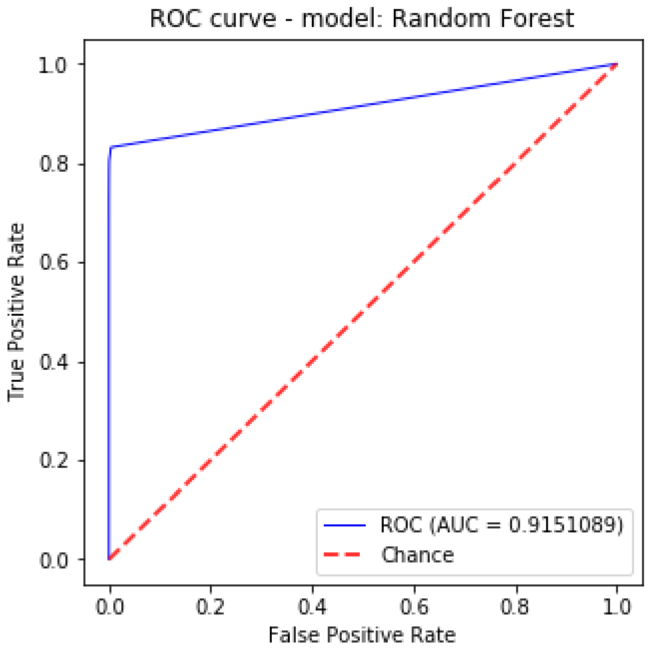



```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.91%
Precision = 75.37%
Recall = 68.24%
```



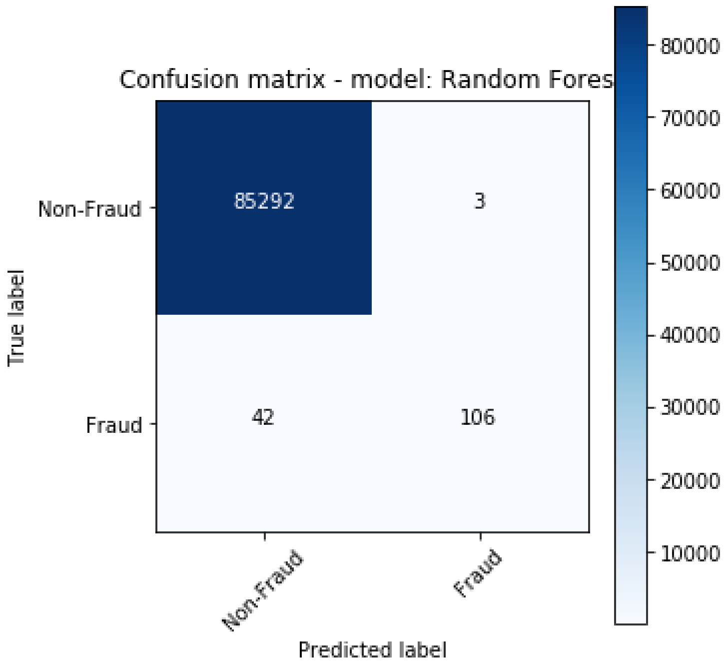
In [36]:

```
plot_CM_and_ROC_curve(classifiers[3], X_ros, y_ros, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>
```

```
Accuracy = 99.95%
Precision = 97.25%
Recall = 71.62%
```

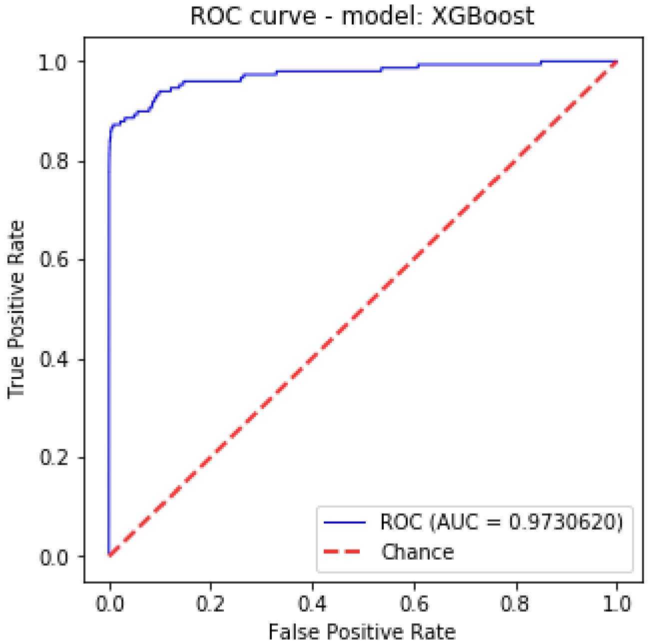


In [37]:

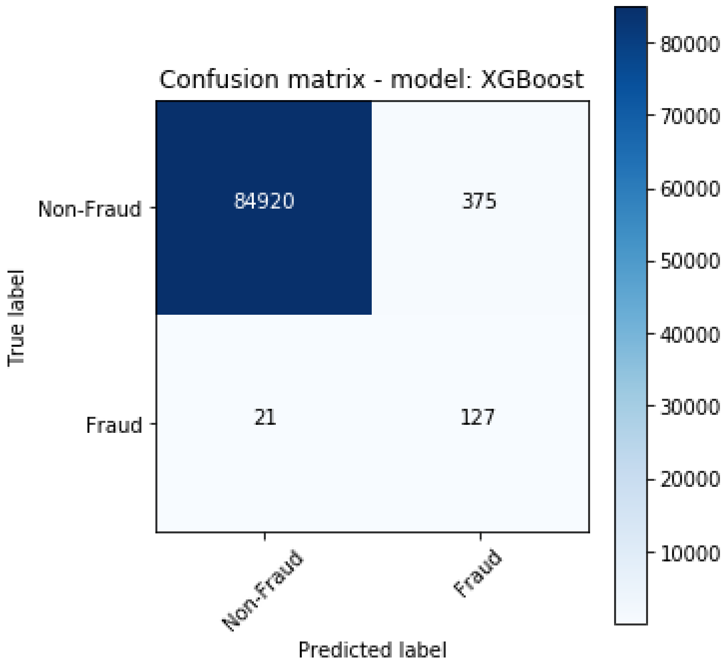
```
X_Cols = X_train.columns
X_ros = pd.DataFrame(data=X_ros, columns=X_Cols)
y_ros = pd.Series(y_ros)
```

In [38]:

```
plot_CM_and_ROC_curve(classifiers[4], X_ros, y_ros, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>

C:\Users\family\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:

Accuracy = 99.54%
Precision = 25.30%
Recall = 85.81%
```



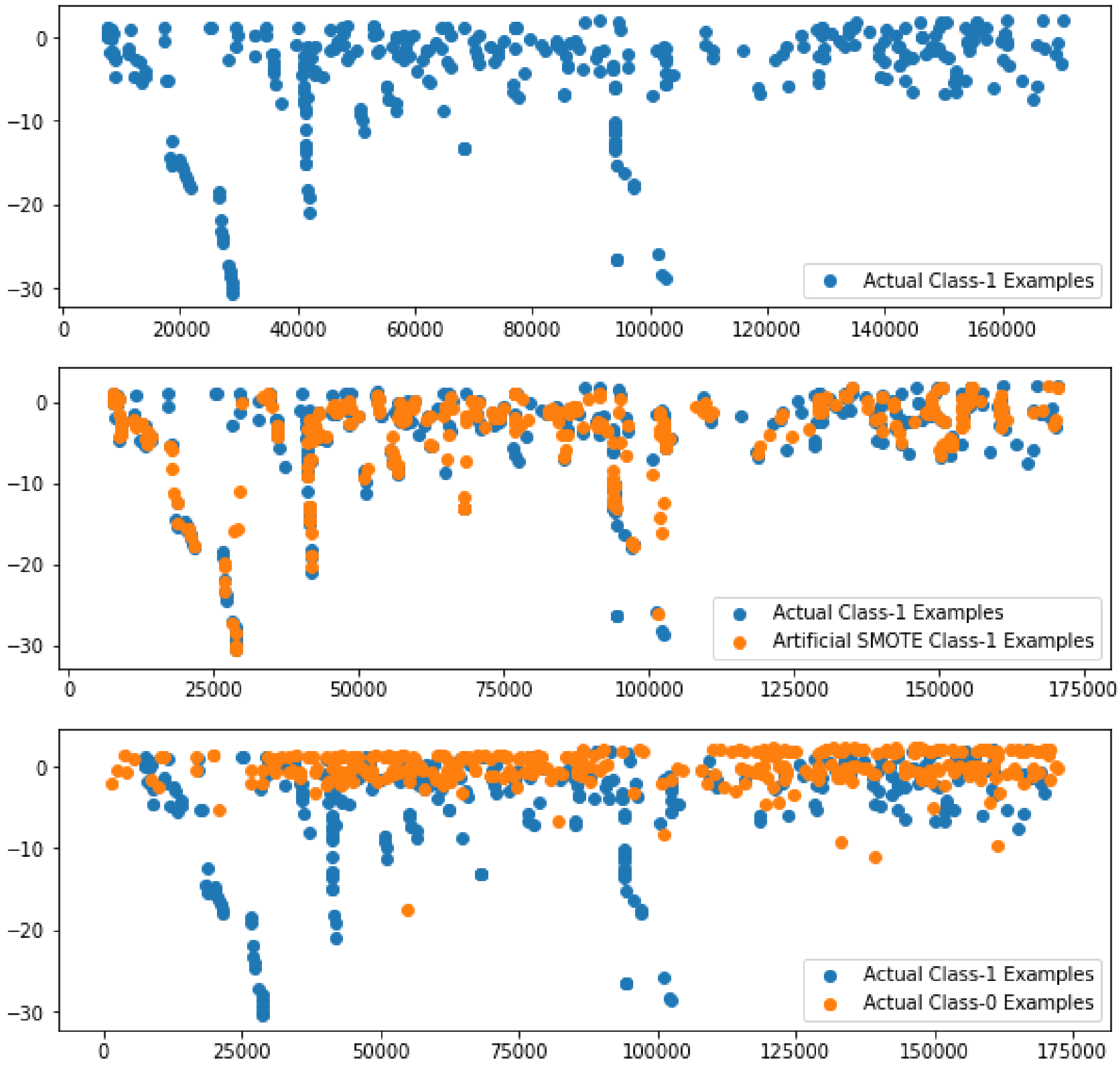
Print the class distribution after applying SMOTE

In [40]:

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


sm = over_sampling.SMOTE(random_state=0)
X_train_smote, y_train_smote = sm.fit_sample(X_train, y_train)
# Artificial minority samples and corresponding minority labels from SMOTE are appended
# below X_train and y_train respectively
# So to exclusively get the artificial minority samples from SMOTE, we do
X_train_smote_1 = X_train_smote[X_train.shape[0]:]

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]


plt.rcParams['figure.figsize'] = [10, 10]
fig = plt.figure()

plt.subplot(3, 1, 1)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.legend()

plt.subplot(3, 1, 2)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.scatter(X_train_smote_1[:X_train_1.shape[0], 0], X_train_smote_1[:X_train_1.shape[0
], 1],
            label='Artificial SMOTE Class-1 Examples')
plt.legend()

plt.subplot(3, 1, 3)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], 0], X_train_0[:X_train_1.shape[0], 1], label
='Actual Class-0 Examples')
plt.legend()
```

Out[40]:

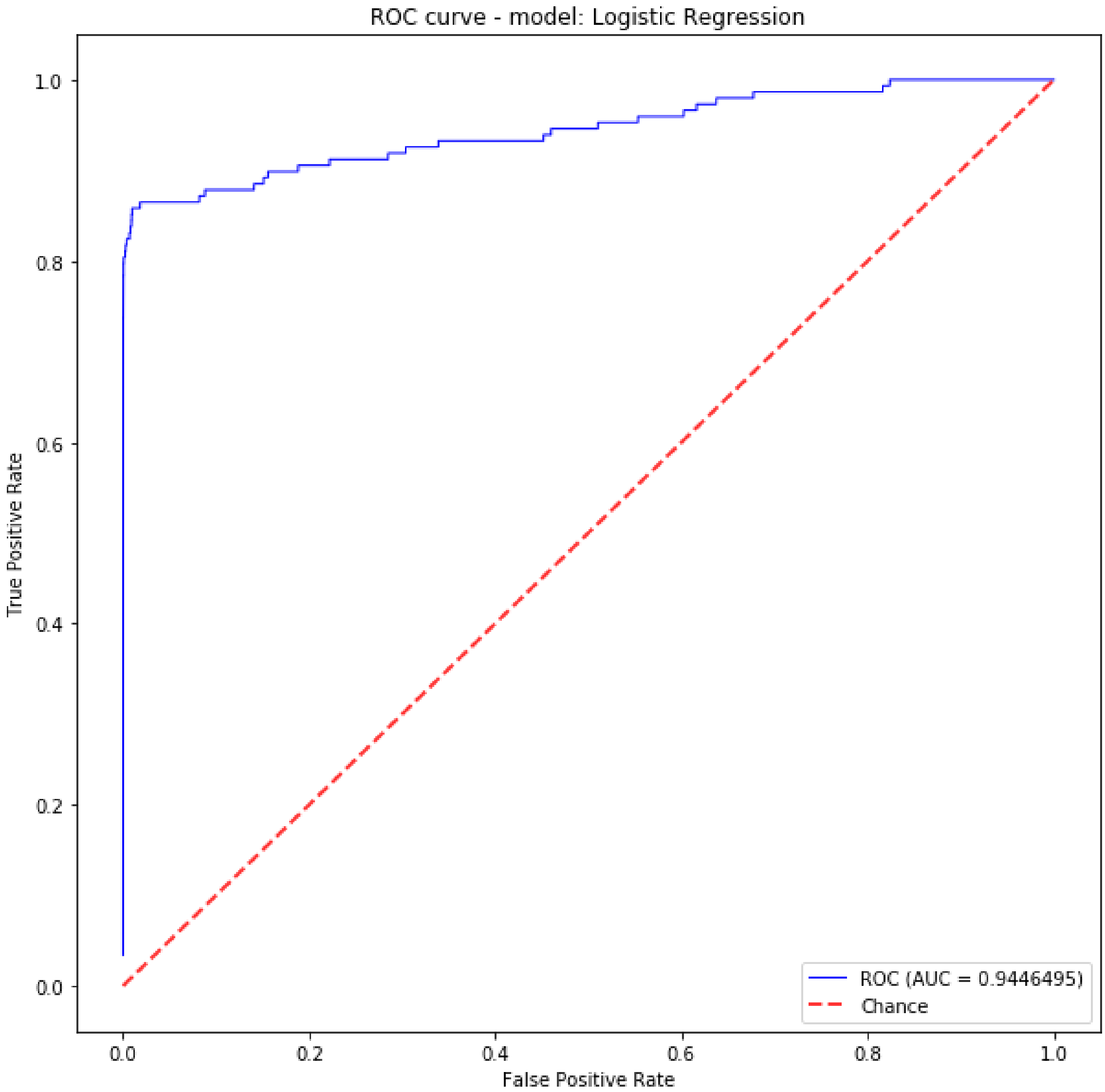
```
<matplotlib.legend.Legend at 0x18b53002c18>
```



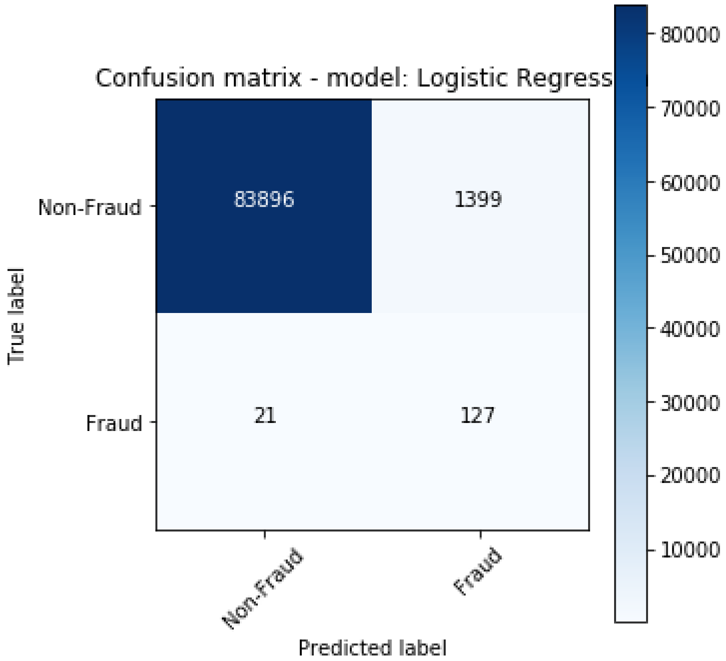
Build models on other algorithms to see the better performing on SMOTE

In [41]:

```
plot_CM_and_ROC_curve(classifiers[0], X_train_smote, y_train_smote, X_test, y_test)
```



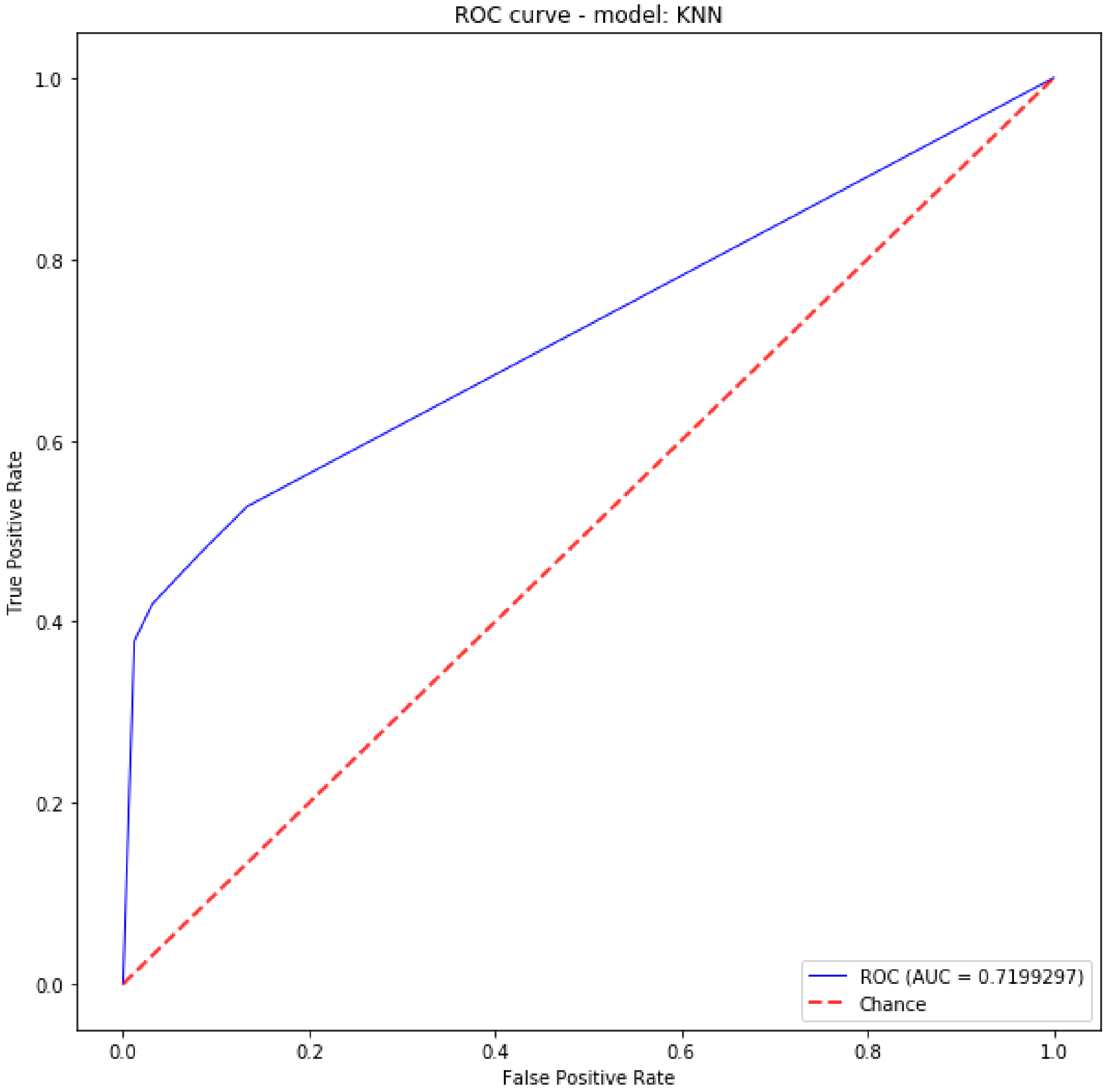

```
<Figure size 360x360 with 0 Axes>

Accuracy = 98.34%
Precision = 8.32%
Recall = 85.81%
```



Confusion matrix - model: Logistic Regress
True label
Non-Fraud
Fraud
83896
1399
21
127
Non-Fraud
Fraud
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

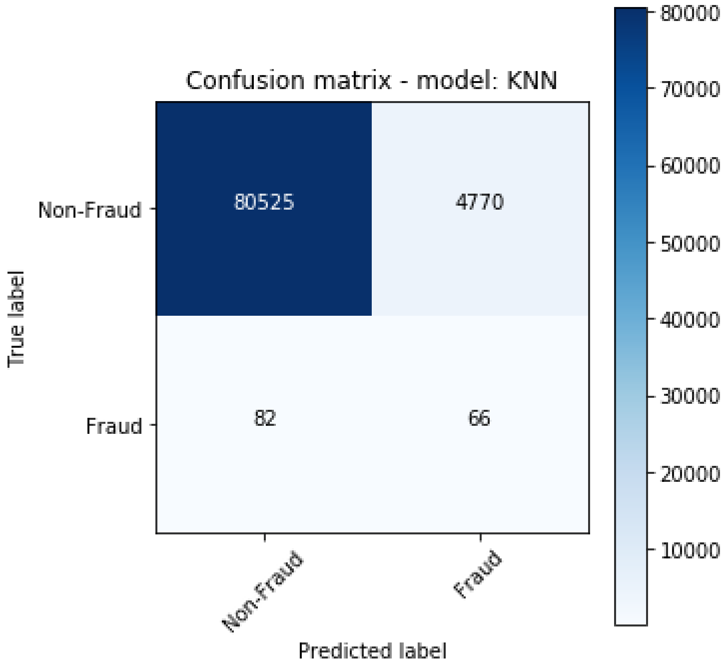
In [42]:

```
plot_CM_and_ROC_curve(classifiers[1], X_train_smote, y_train_smote, X_test, y_test)
```



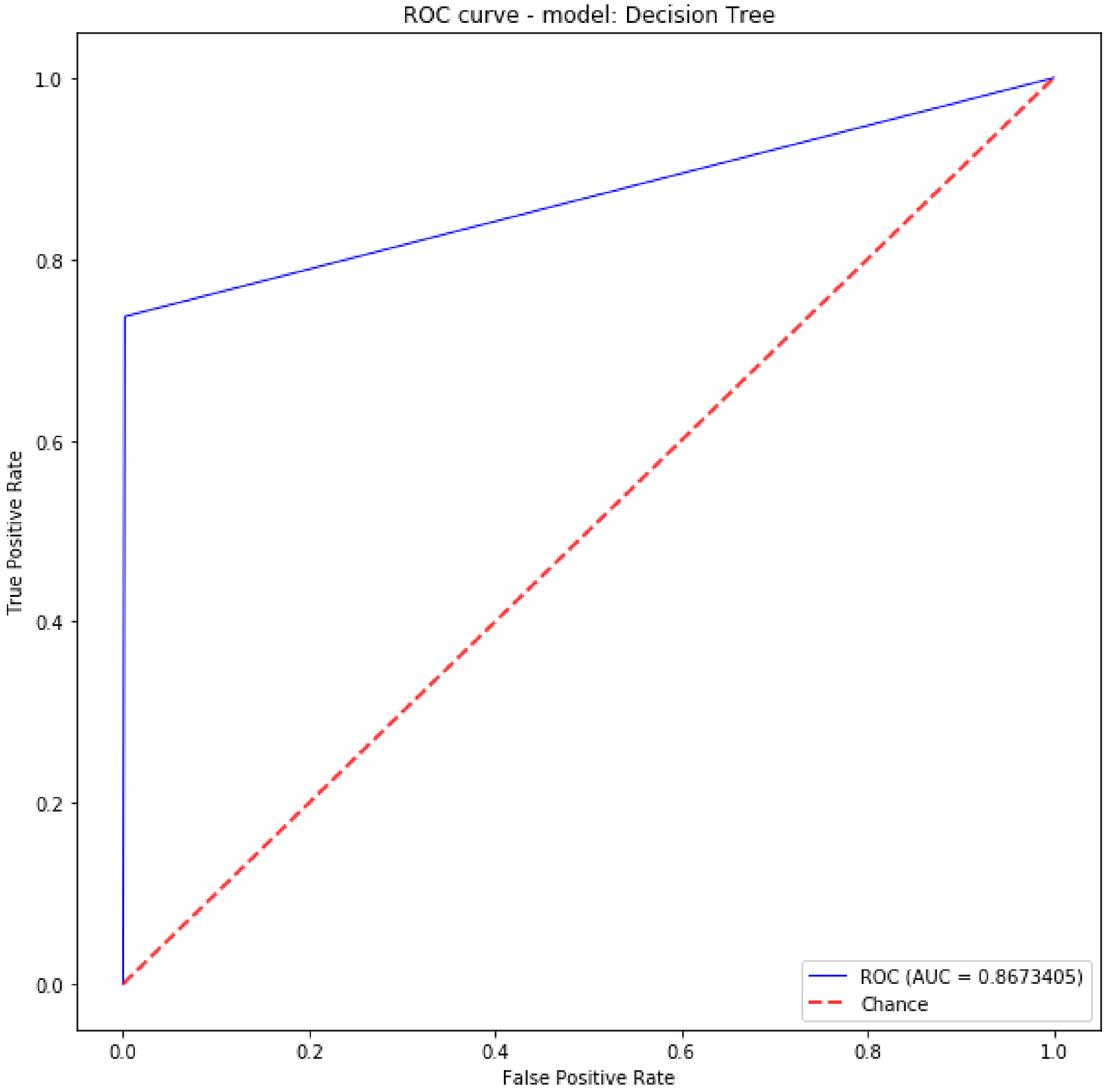

```
&lt;Figure size 360x360 with 0 Axes&gt;
```

```
Accuracy = 94.32%
Precision = 1.36%
Recall = 44.59%
```

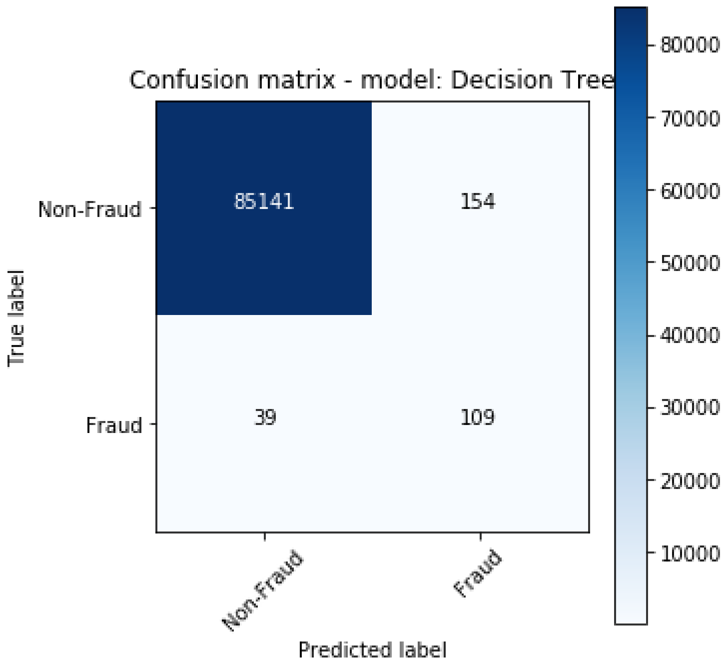

Confusion matrix - model: KNN
True label
Non-Fraud
Fraud
80525
4770
82
66
Predicted label
Non-Fraud
Fraud
80000
70000
60000
50000
40000
30000
20000
10000

In [43]:

```
plot_CM_and_ROC_curve(classifiers[2], X_train_smote, y_train_smote, X_test, y_test)
```

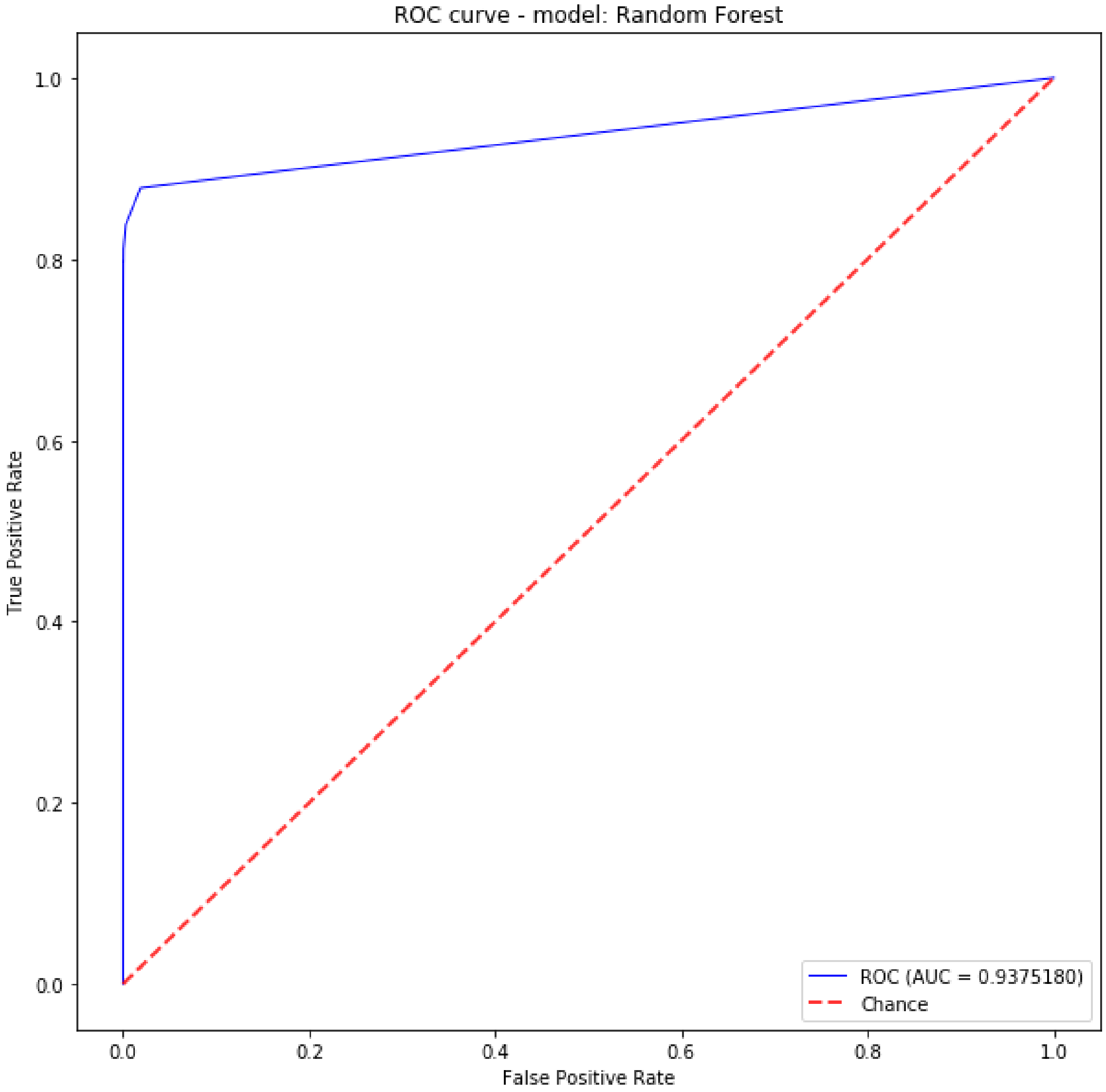



```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.77%
Precision = 41.44%
Recall = 73.65%
```

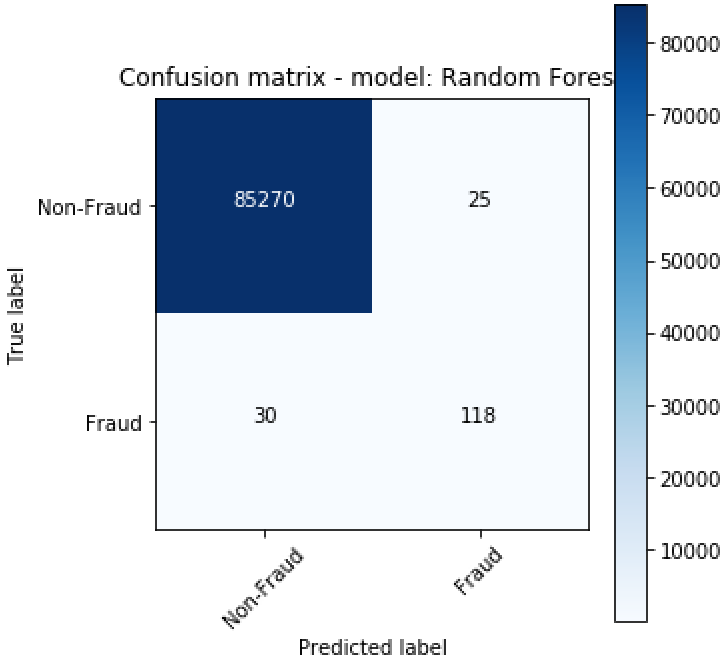


In [44]:

```
plot_CM_and_ROC_curve(classifiers[3], X_train_smote, y_train_smote, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.94%
Precision = 82.52%
Recall = 79.73%
```

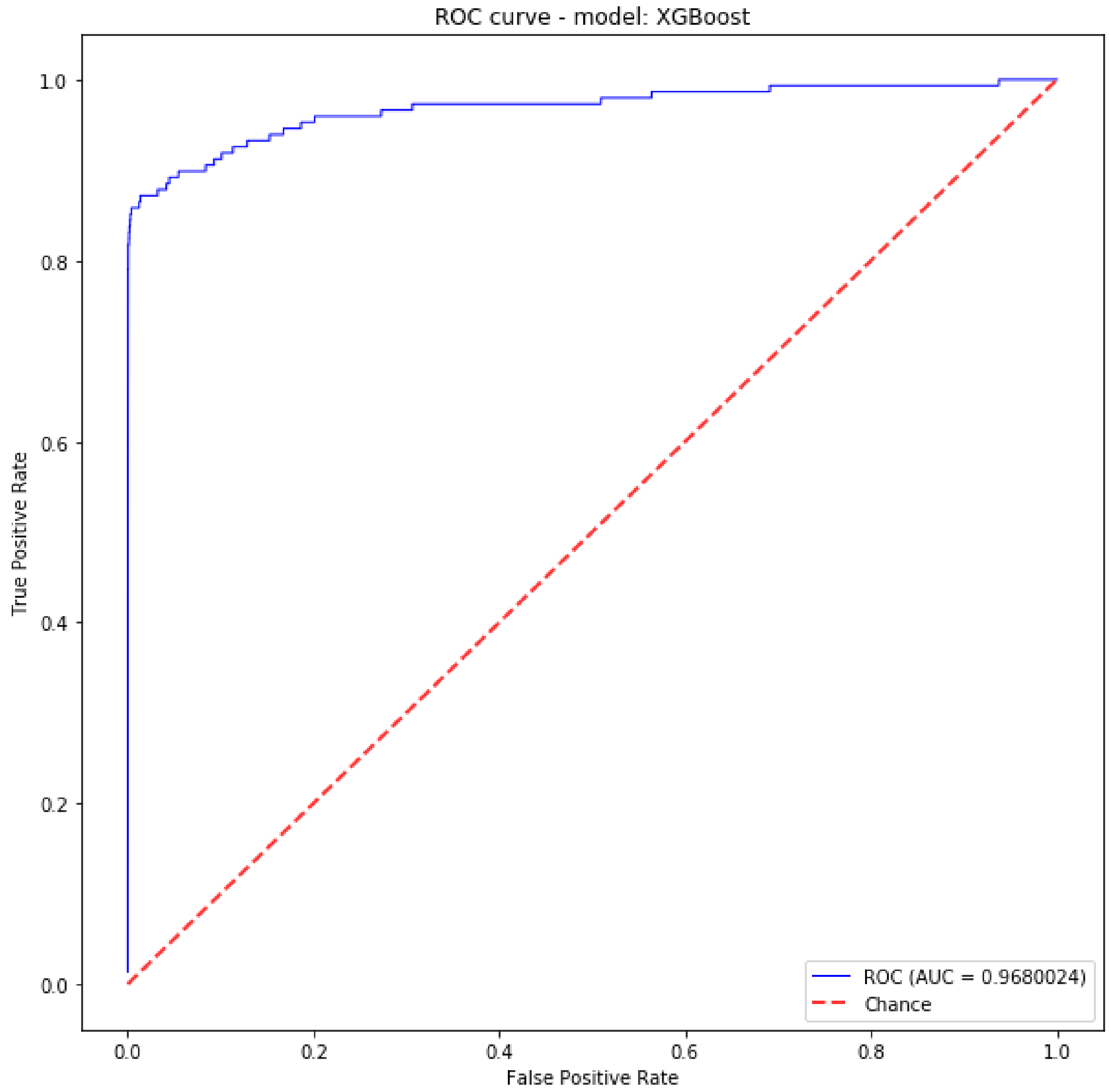


In [45]:

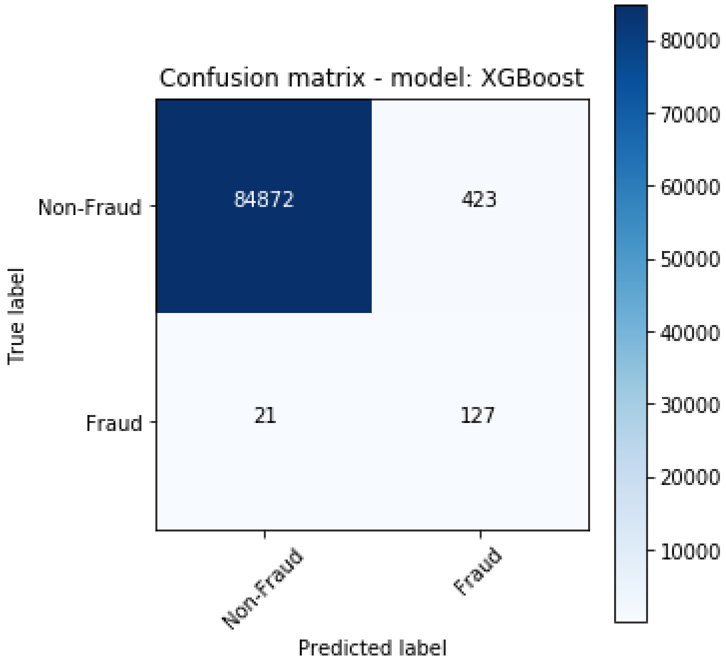
```
X_Cols = X_train.columns
X_train_smote = pd.DataFrame(data=X_train_smote, columns=X_Cols)
y_train_smote = pd.Series(y_train_smote)
```

In [46]:

```
plot_CM_and_ROC_curve(classifiers[4], X_train_smote, y_train_smote, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.48%
Precision = 23.09%
Recall = 85.81%
```



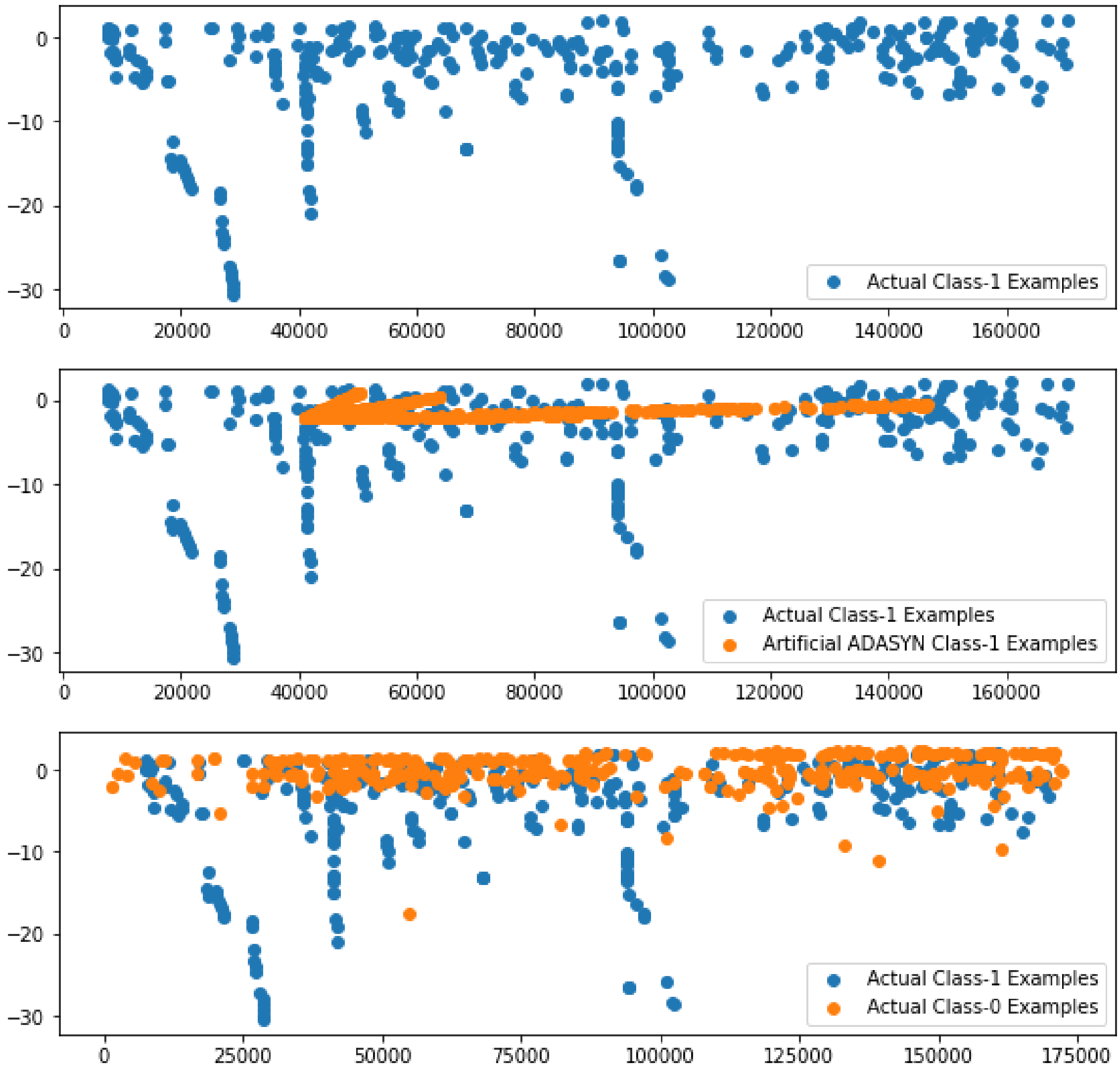
Print the class distribution after applying ADASYN

In [47]:

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

from imblearn import over_sampling

ada = over_sampling.ADASYN(random_state=0)
X_train_adasyn, y_train_adasyn = ada.fit_sample(X_train, y_train)
# Artificial minority samples and corresponding minority labels from ADASYN are appende
d
# below X_train and y_train respectively
# So to exclusively get the artificial minority samples from ADASYN, we do
X_train_adasyn_1 = X_train_adasyn[X_train.shape[0]:]

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]



import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [10, 10]
fig = plt.figure()

plt.subplot(3, 1, 1)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.legend()

plt.subplot(3, 1, 2)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.scatter(X_train_adasyn_1[:X_train_1.shape[0], 0], X_train_adasyn_1[:X_train_1.shape
[0], 1],
            label='Artificial ADASYN Class-1 Examples')
plt.legend()

plt.subplot(3, 1, 3)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], 0], X_train_0[:X_train_1.shape[0], 1], label
='Actual Class-0 Examples')
plt.legend()
```

Out[47]:

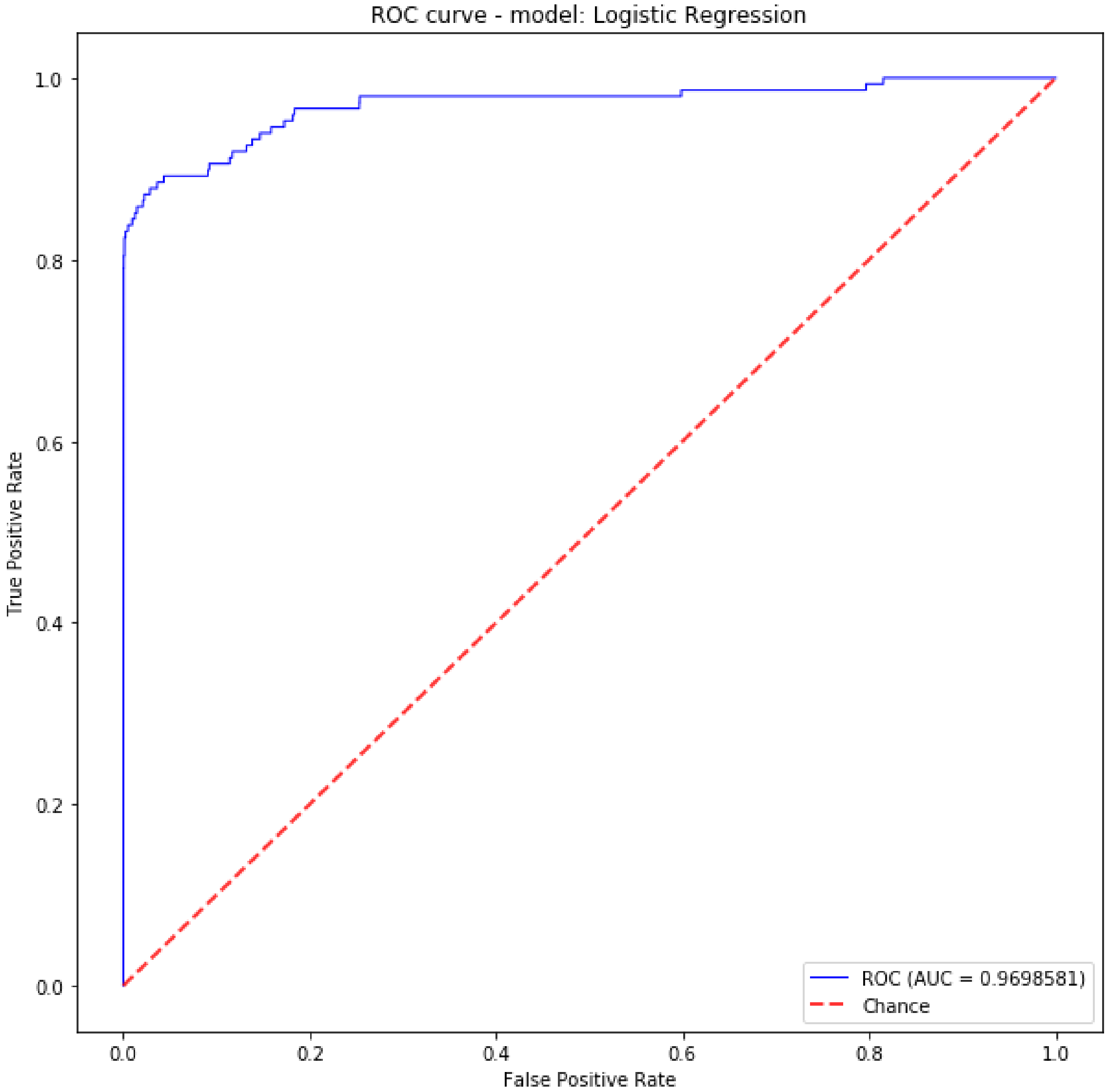
```
<matplotlib.legend.Legend at 0x18b522e9518>
```



Build models on other algorithms to see the better performing on ADASYN

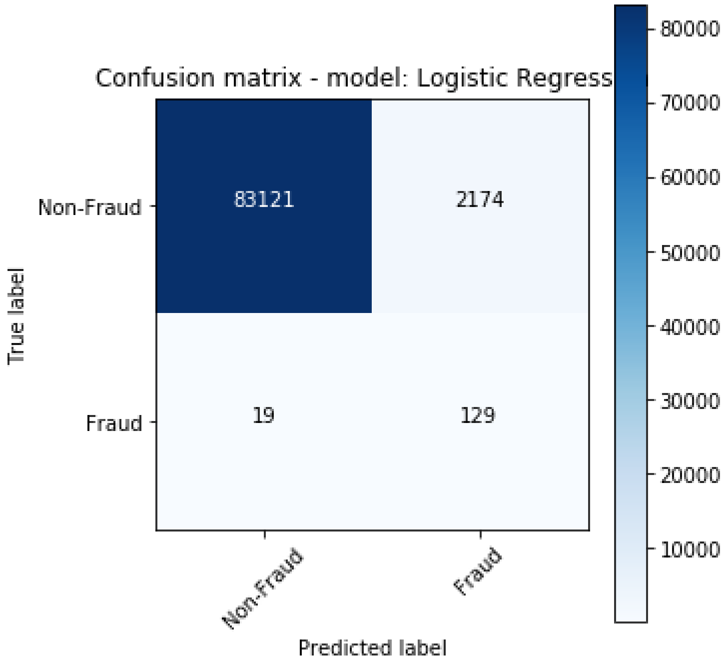
In [49]:

```
plot_CM_and_ROC_curve(classifiers[0], X_train_adasyn, y_train_adasyn, X_test, y_test)
```



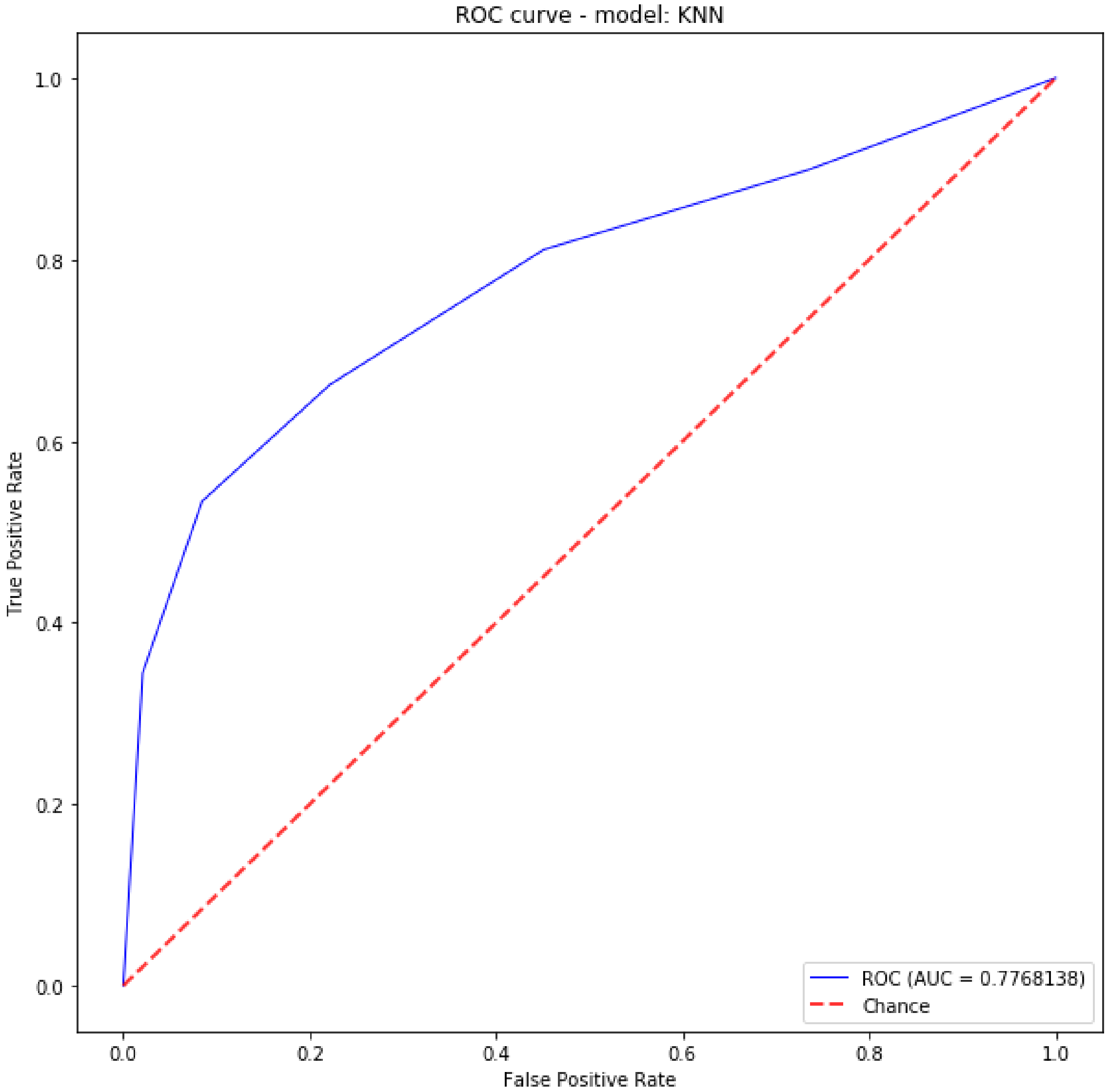

```
<Figure size 360x360 with 0 Axes>
```

```
Accuracy = 97.43%
Precision = 5.60%
Recall = 87.16%
```

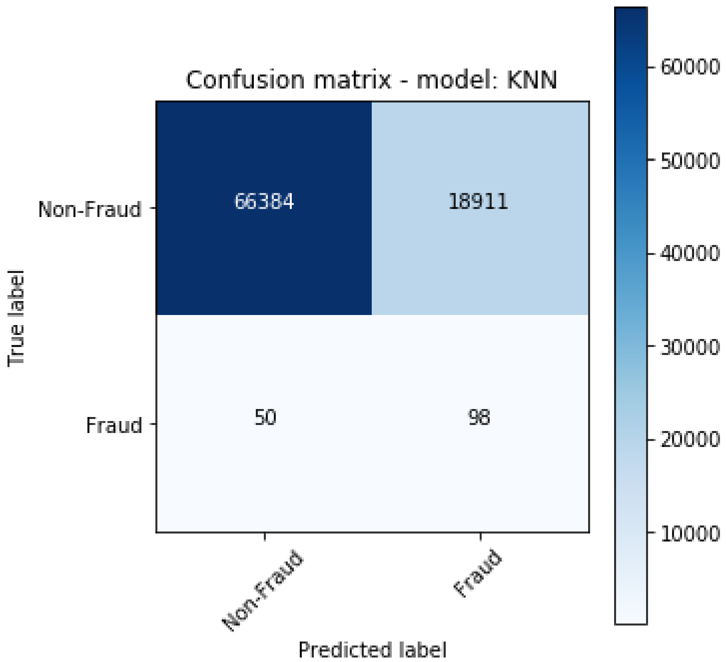

Confusion matrix - model: Logistic Regress
Non-Fraud
Fraud
True label
83121
2174
19
129
Non-Fraud
Fraud
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

In [50]:

```
plot_CM_and_ROC_curve(classifiers[1], X_train_adasyn, y_train_adasyn, X_test, y_test)
```

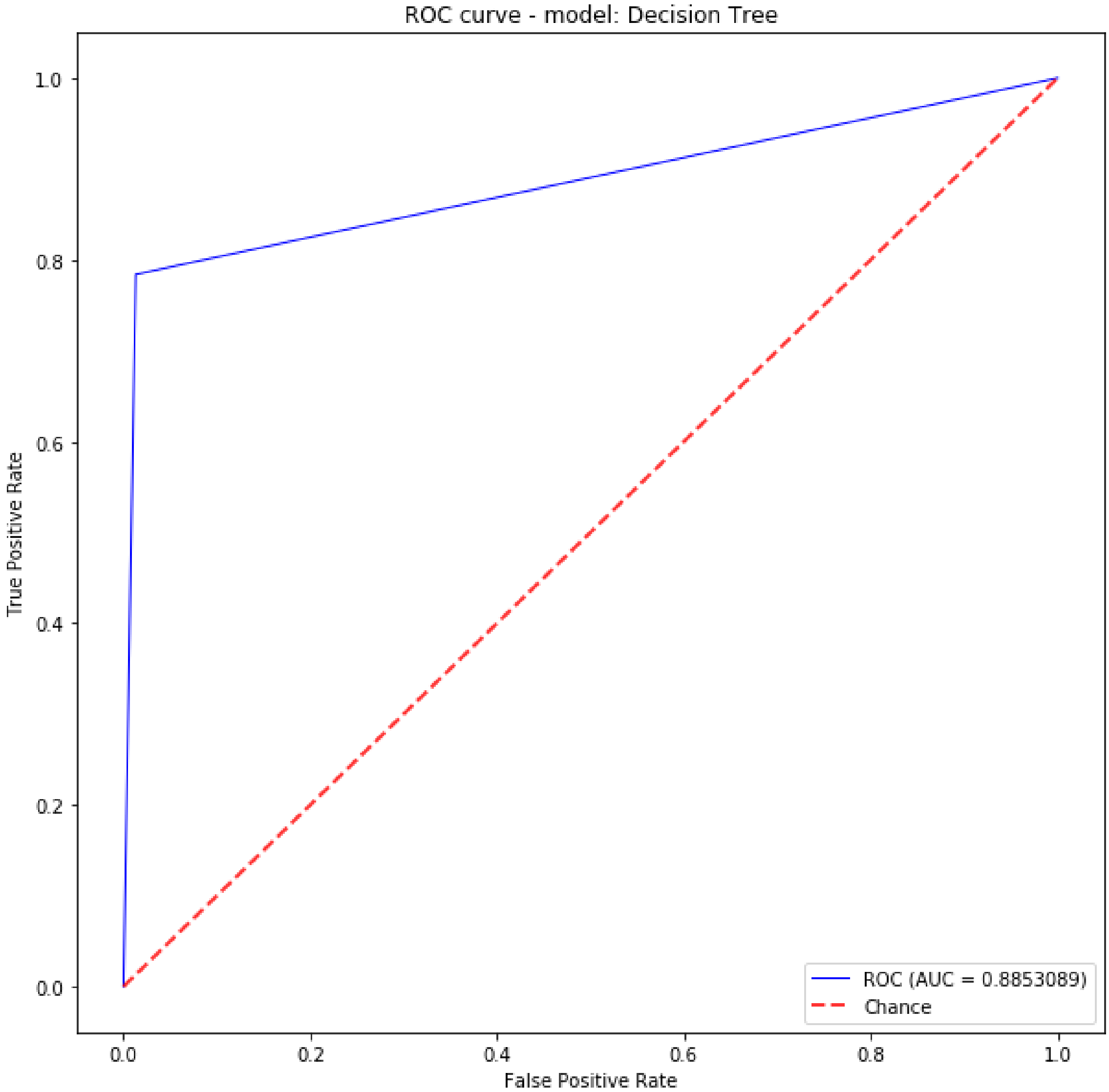



```
<Figure size 360x360 with 0 Axes>

Accuracy = 77.81%
Precision = 0.52%
Recall = 66.22%
```

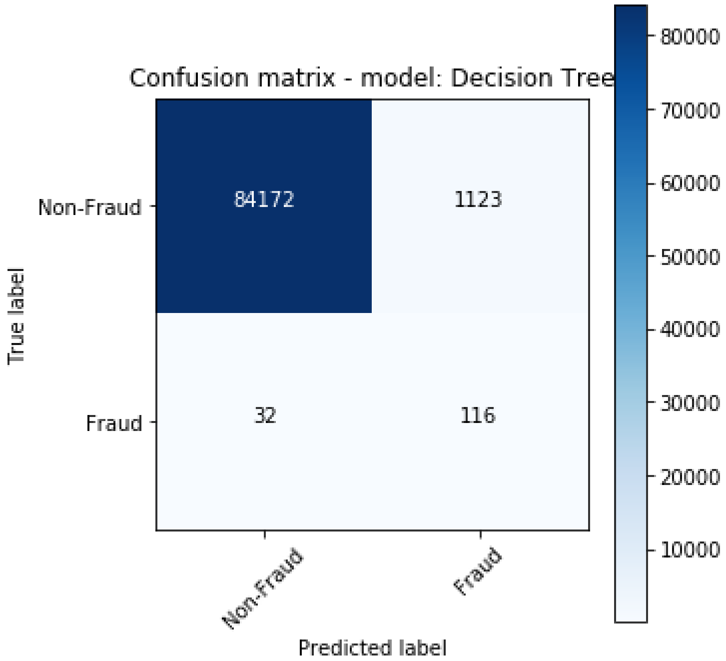


In [51]:

```
plot_CM_and_ROC_curve(classifiers[2], X_train_adasyn, y_train_adasyn, X_test, y_test)
```

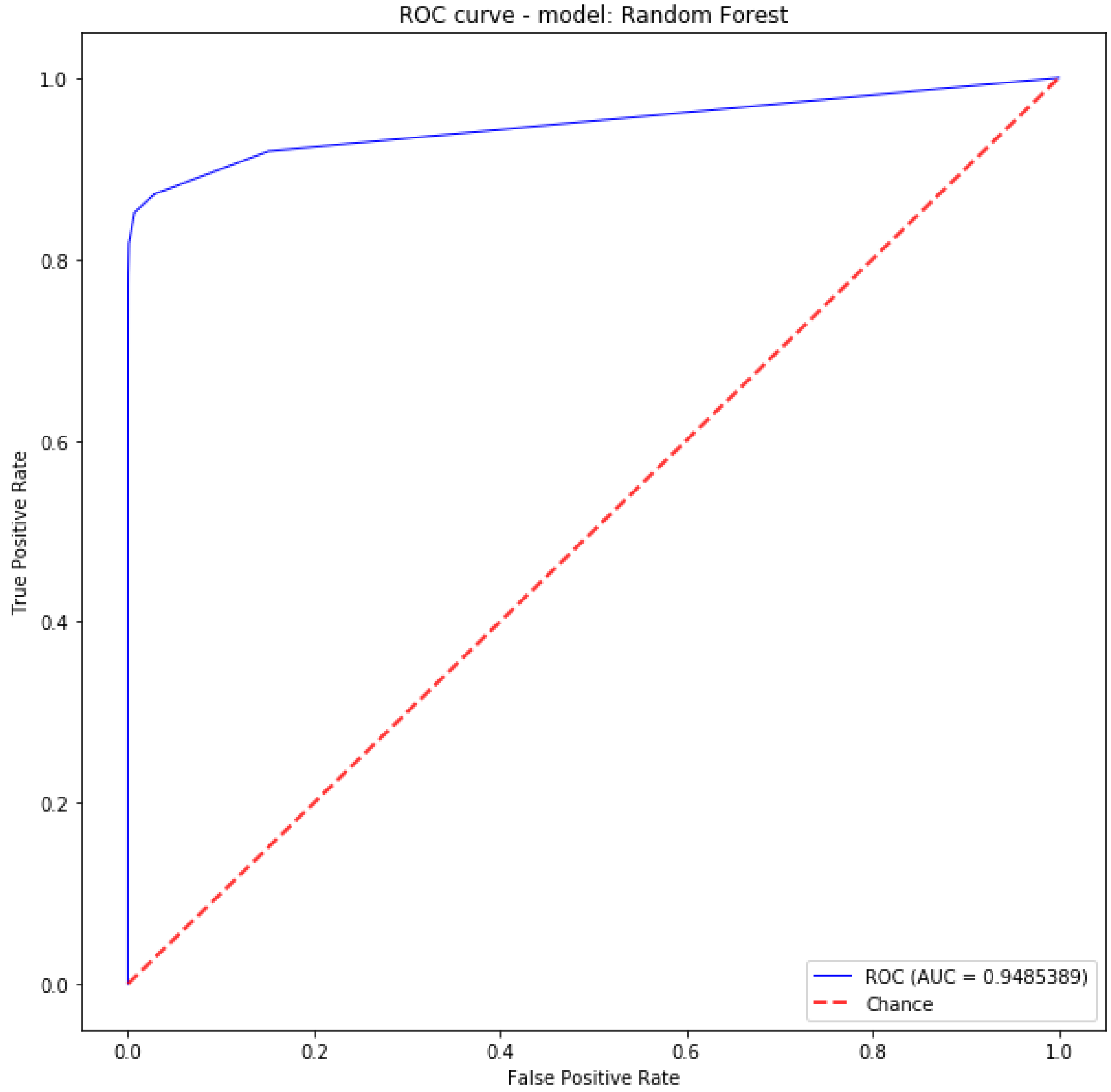



```
<Figure size 360x360 with 0 Axes>

Accuracy = 98.65%
Precision = 9.36%
Recall = 78.38%
```

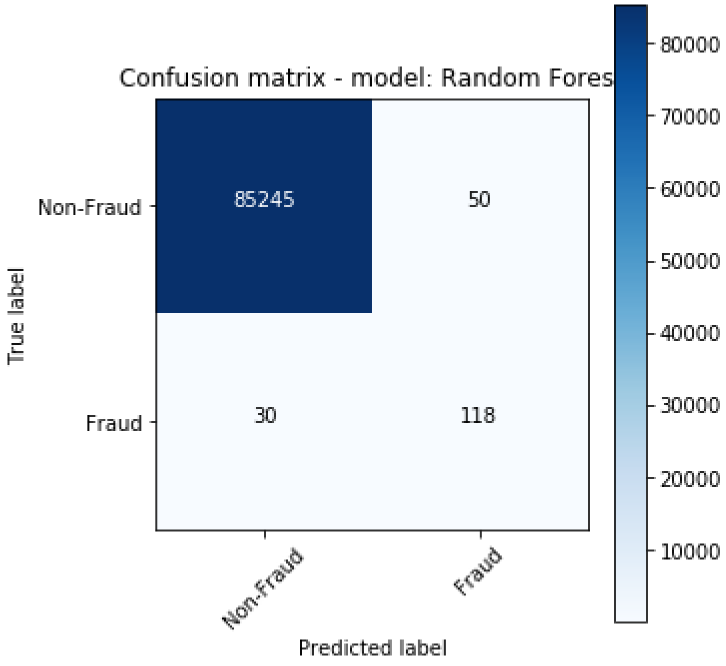

Confusion matrix - model: Decision Tree
Non-Fraud
Fraud
84172
1123
32
116
True label
Predicted label
80000
70000
60000
50000
40000
30000
20000
10000

In [52]:

```
plot_CM_and_ROC_curve(classifiers[3], X_train_adasyn, y_train_adasyn, X_test, y_test)
```




```
<Figure size 360x360 with 0 Axes>

Accuracy = 99.91%
Precision = 70.24%
Recall = 79.73%
```

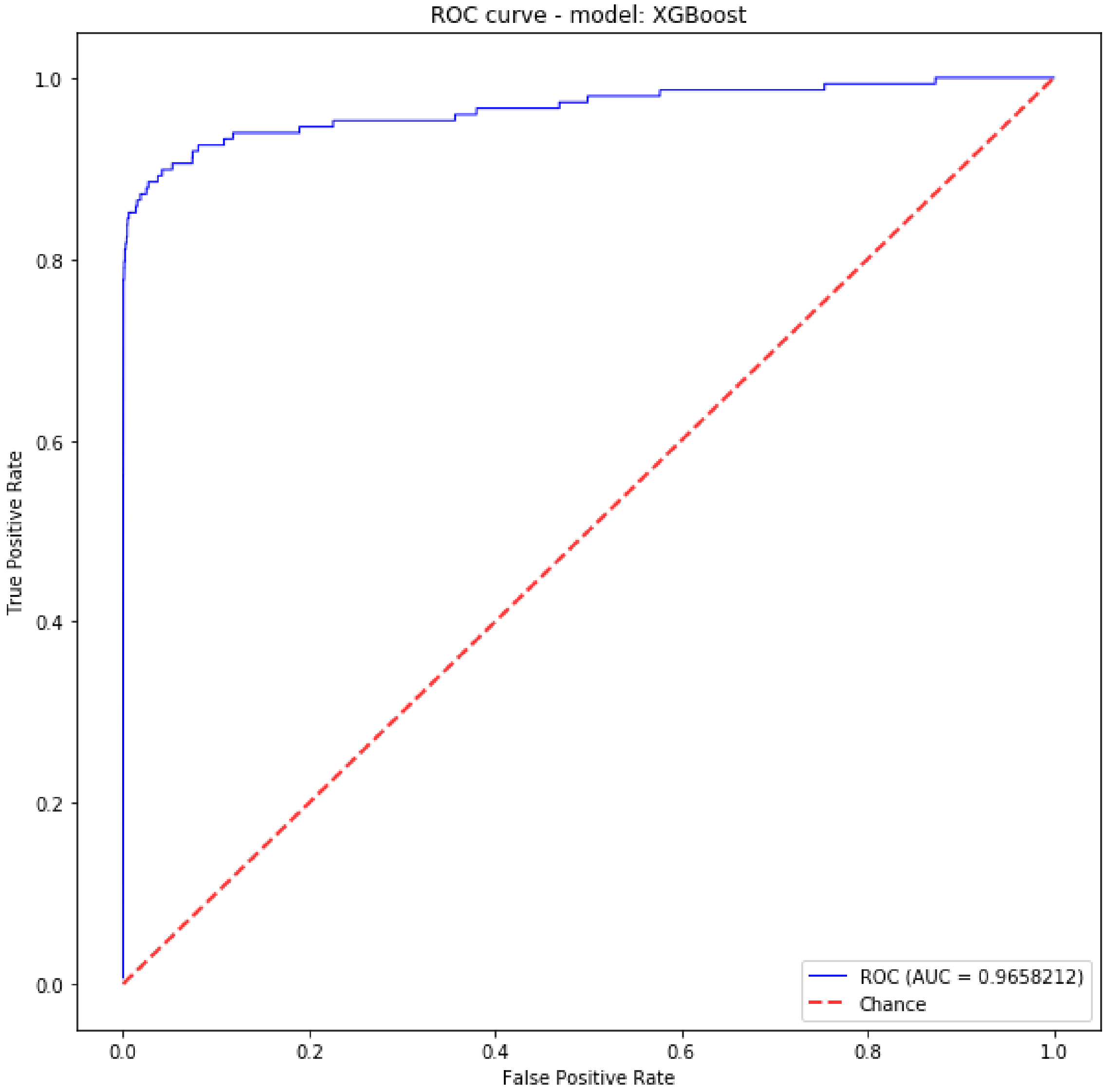


In [53]:

```
X_Cols = X_train.columns
X_train_adasyn = pd.DataFrame(data=X_train_adasyn, columns=X_Cols)
y_train_adasyn = pd.Series(y_train_adasyn)
```

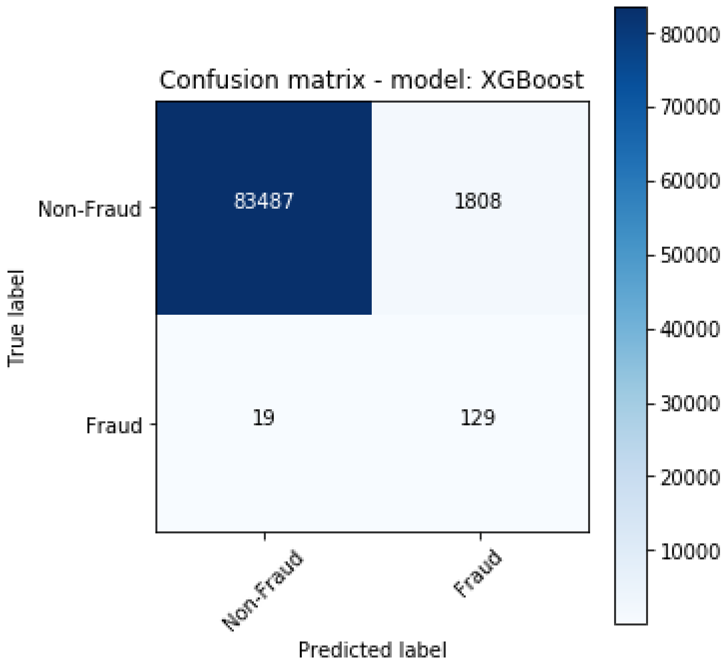
In [54]:

```
plot_CM_and_ROC_curve(classifiers[4], X_train_adasyn, y_train_adasyn, X_test, y_test)
```


ROC curve - model: XGBoost
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC (AUC = 0.9658212)
Chance

```
<Figure size 360x360 with 0 Axes>
```

```
Accuracy = 97.86%
Precision = 6.66%
Recall = 87.16%
```



Select the oversampling method which shows the best result on a model

- Apply the best hyperparameter on the model
- Predict on the test dataset

In [56]:

```
# perform the best oversampling method on X_train & y_train
#Best results achieved is from XGBoost with Random oversampling

confusion_matrix_total = [[0, 0], [0, 0]]
clf = XGBClassifier(random_state=42)  #initialise the model with optimum hyperparameter
s
probas_ = clf.fit(X_ros, y_ros).predict_proba(X_test) # fit on the balanced dataset
fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
roc_auc = auc(fpr, tpr)
y_pred=clf.predict(X_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
confusion_matrix_total += cnf_matrix
tn, fp = confusion_matrix_total.tolist()[0]
fn, tp = confusion_matrix_total.tolist()[1]
accuracy = (tp+tn)/(tp+tn+fp+fn)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
print('ROC_AUC = {:2.2f}%'.format(roc_auc*100)) #print the evaluation score on the X_te
st by choosing the best evaluation metric
print('Accuracy = {:2.2f}%'.format(accuracy*100))
print('Precision = {:2.2f}%'.format(precision*100))
print('Recall = {:2.2f}%'.format(recall*100))
```

```
ROC_AUC = 97.31%
Accuracy = 99.54%
Precision = 25.30%
Recall = 85.81%
```

Print the important features of the best model to understand the dataset

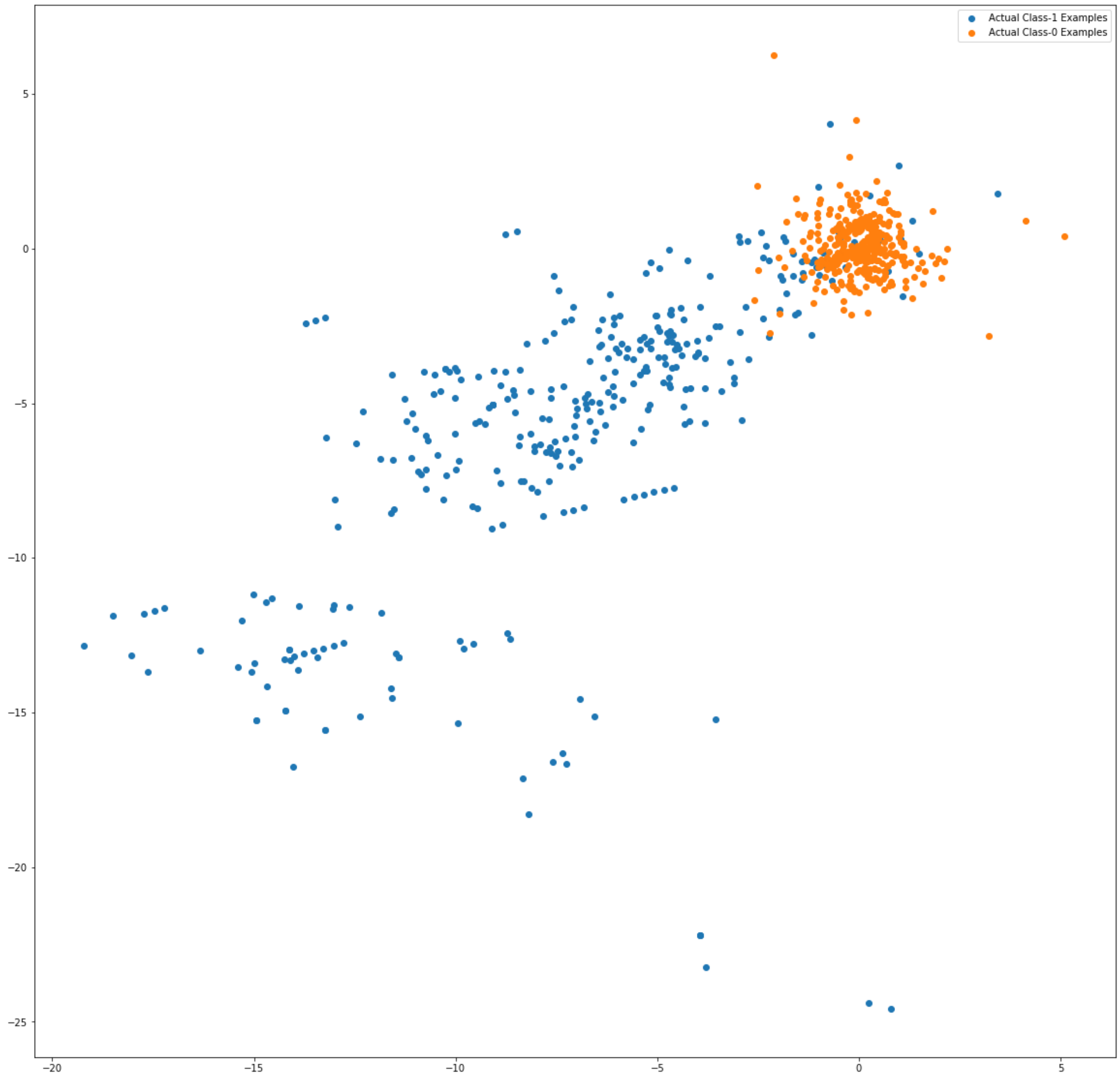
In [57]:

```
var_imp = []
for i in clf.feature_importances_:
    var_imp.append(i)
print('Top var =', var_imp.index(np.sort(clf.feature_importances_)[-1])+1)
print('2nd Top var =', var_imp.index(np.sort(clf.feature_importances_)[-2])+1)
print('3rd Top var =', var_imp.index(np.sort(clf.feature_importances_)[-3])+1)

# Variable on Index-13 and Index-9 seems to be the top 2 variables
top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-1])
second_top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-2])

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]

np.random.shuffle(X_train_0)

import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [20, 20]

plt.scatter(X_train_1[:, top_var_index], X_train_1[:, second_top_var_index], label='Act
ual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], top_var_index], X_train_0[:X_train_1.shape[0
], second_top_var_index],
            label='Actual Class-0 Examples')
plt.legend()
```

Top var = 15
2nd Top var = 11
3rd Top var = 5

Out[57]:

<matplotlib.legend.Legend at 0x18b521cbf98>



In [ ]:

```
#### Print the FPR,TPR & select the best threshold from the roc curve
```

In [62]:

```
print('Train auc =', metrics.roc_auc_score(y_test,y_pred))
fpr, tpr, thresholds = metrics.roc_curve(y_test, probas_[:, 1])
threshold = thresholds[np.argmax(tpr-fpr)]
print(threshold)
```

Train auc = 0.9268558009325346
0.35989735

In [ ]:

```

```