CREDIT CARD FRAUD DETECTION

IMPORTING REQUIRED LIBRARIES

```
import pandas as pd
import numpy as np
import itertools
import time

from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import train_test_split, cross_val_score, KFold, Stratifie
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

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

import warnings
warnings.simplefilter('ignore')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
  import pandas.util.testing as tm
```

LOAD IN THE DATASET

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

	Time	V1	V2	V3	V4	V5	V6	V7
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098

EXPLORATORY DATA ANALYSIS

```
# Find the rows and columns size
data.shape
```

```
(93181, 31)
```

```
# Columns present for the dataset
data.columns
```

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

```
# Checking the datatypes of attributes
data.dtypes
```

```
Time        int64
```

```
1 # Summary of the attributes
2 data.describe(include='all')
```

	Time	V1	V2	V3	V4	
count	93181.000000	93181.000000	93181.000000	93181.000000	93181.000000	93181.0000
mean	40720.460405	-0.263002	-0.040780	0.676306	0.163064	-0.2800
std	16392.444505	1.869011	1.664308	1.341575	1.355105	1.3686
min	0.000000	-56.407510	-72.715728	-33.680984	-5.172595	-42.1478
25%	32678.000000	-1.028016	-0.605706	0.179018	-0.716783	-0.8987
50%	42848.000000	-0.258489	0.072818	0.756532	0.189117	-0.3149
75%	53622.000000	1.153021	0.728656	1.381324	1.034477	0.2509
max	64283.000000	1.960497	18.902453	4.226108	16.715537	34.8016

```
V22       float64
```

```
1 # Checking for null values
2 data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
```

```
# Null values are present
# Replacing null values with mean
data.fillna(data.mean(), inplace=True)
```

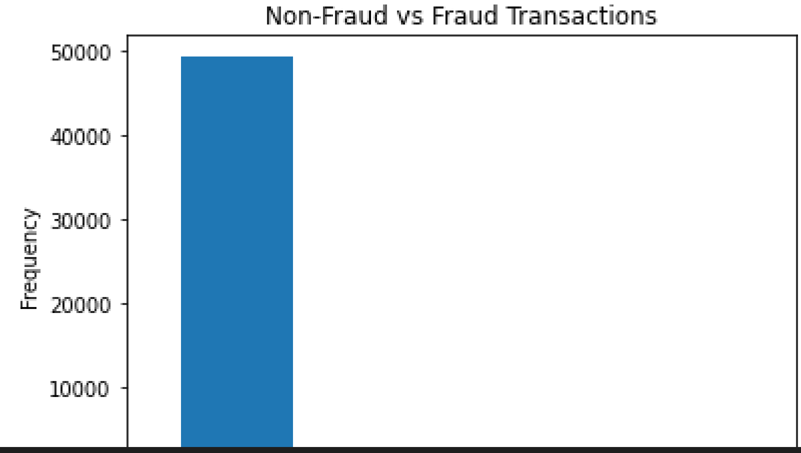
```
# No null values
data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

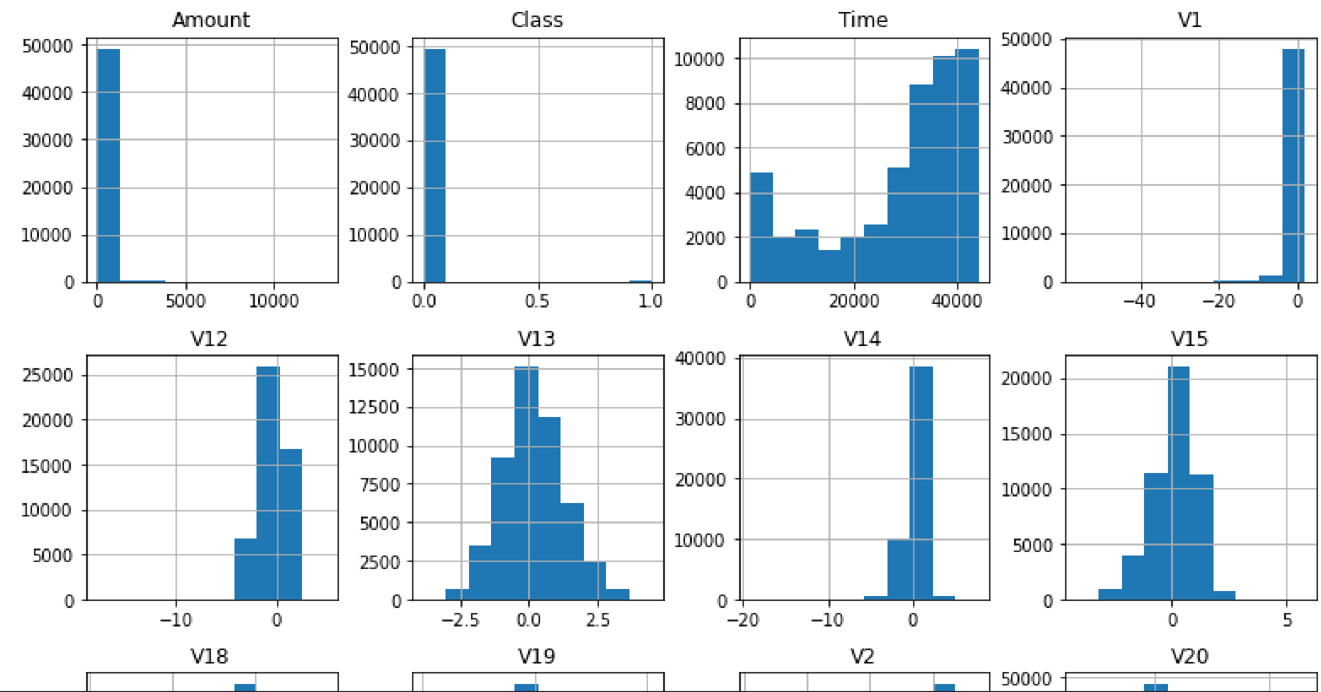
```
# 'Class' is the target variable
# It has two values counts - Fraud and Non-fraud
# Plot showing the counts of both counts
# Most of them are Non-fraud
# This is an unbalanced dataset

count_classes = pd.value_counts(data['Class'], sort=True)
count_classes.plot(kind='bar', rot=0)
plt.title('Non-Fraud vs Fraud Transactions')
plt.xticks(range(2), ['Non-Fraud', 'Fraud'])
plt.xlabel('Class')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
1 # Plotting a histogram for each attribute
2 data.hist(figsize=(20,20));
```




```
# Finding exact number of Fraud and Non-fraud values
# Fraud percentage is less than 1% of total transactions

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

print('Fraud shape: {}'.format(fraud.shape))
print('Non-Fraud shape: {}'.format(normal.shape))
print('Percentage of fraud transcation: {:.4f}%'.format(len(fraud)/len(normal) * 100))
```

```
Fraud shape: (148, 31)
Non-Fraud shape: (49461, 31)
Percentage of fraud transcation: 0.2992%
```

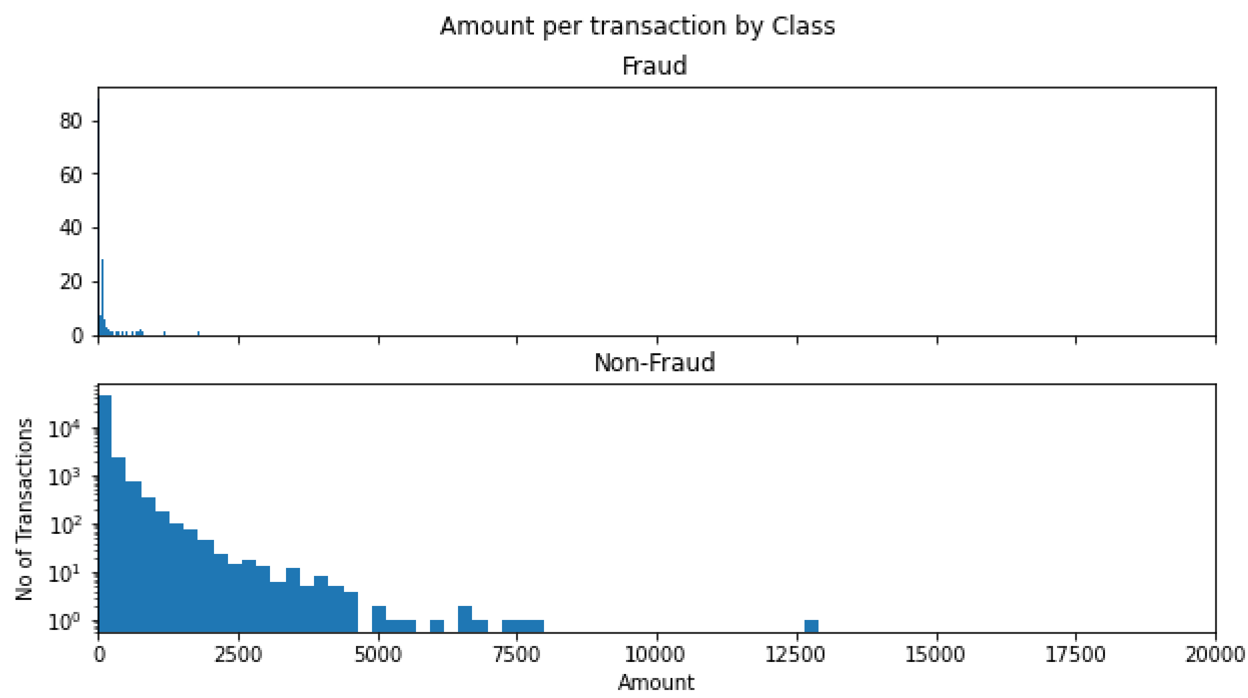
```
# Summary of Fraud transactions
fraud.Amount.describe()
```

```
count     148.000000
mean      100.170676
std       233.347471
min         0.000000
25%         1.000000
50%         9.560000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

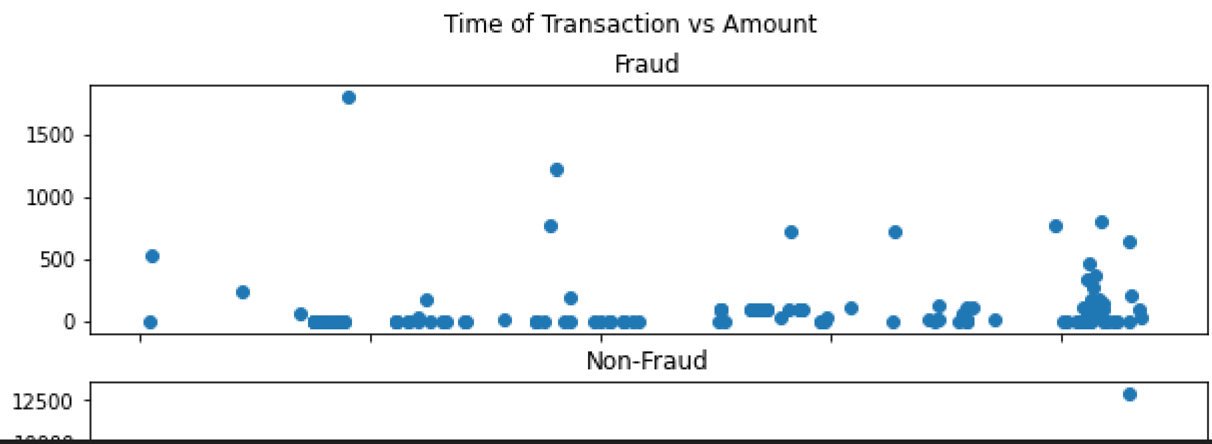
```
# Summary of Normal transactions
normal.Amount.describe()
```

```
count    49461.000000
mean        93.099593
std        253.325102
min          0.000000
```

```python
# Plot for showing the amount for both classes
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,5))
fig.suptitle('Amount per transaction by Class')
ax1.hist(fraud.Amount, bins=50)
ax1.set_title('Fraud')
ax2.hist(normal.Amount, bins=50)
ax2.set_title('Non-Fraud')
plt.xlabel('Amount')
plt.ylabel('No of Transactions')
plt.xlim(0, 20000)
plt.yscale('log')
```

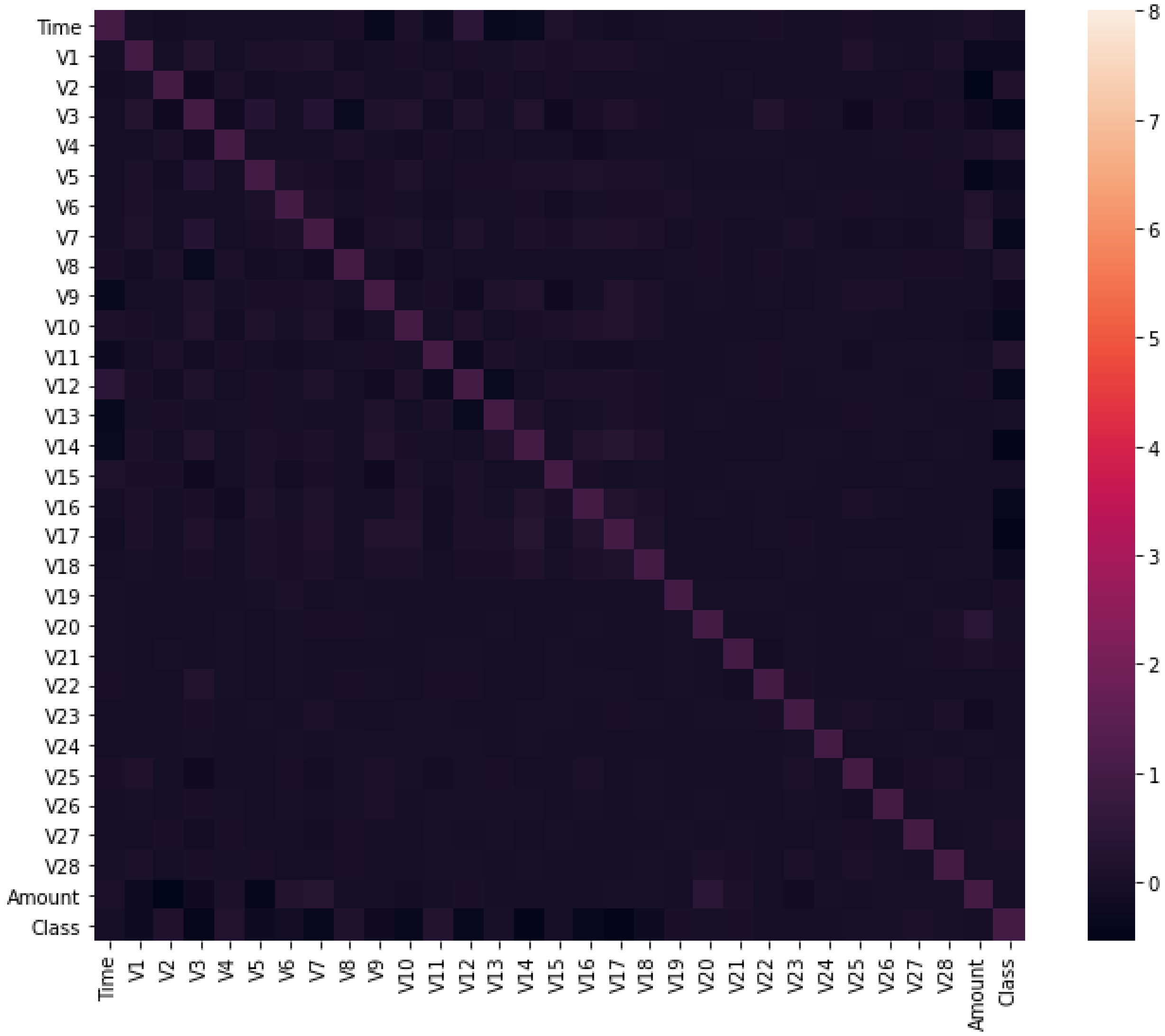



```python
# Plot for Fraud and Normal transcations over time intervals
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(10,5))
fig.suptitle('Time of Transaction vs Amount')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(normal.Time, normal.Amount)
ax2.set_title('Non-Fraud')
plt.xlabel('Time in seconds')
plt.ylabel('Amount')
plt.show()
```




```
# Heatmap to find any correlation between attributes
corr_matrix = data.corr()
fig = plt.figure(figsize=(12,9))
sns.heatmap(corr_matrix, vmax=8, square=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2468830e48>
```



APPLYING TRANSFORMATION ON DATA

```
# The columns from 'V1' to 'V28' are alread normalized
```

```
# Applying scaling on the 'Time' and 'Amount' attributes
# Using RobustScaler as its less prone to outliers
# and we need to detect outliers

rob_scaler = RobustScaler()

data['Scaled_time'] = rob_scaler.fit_transform(data['Time'].values.reshape(-1, 1))
data['Scaled_amount'] = rob_scaler.fit_transform(data['Amount'].values.reshape(-1, 1))

data.drop(['Time', 'Amount'], axis=1, inplace=True)
```

```
# Adding and removing columns

scaled_amount = data['Scaled_amount']
scaled_time = data['Scaled_time']

data.drop(['Scaled_amount', 'Scaled_time'], axis=1, inplace=True)
data.insert(0, 'Scaled_amount', scaled_amount)
data.insert(1, 'Scaled_time', scaled_time)

# Amount and Time are Scaled!
data.head()
```

	Scaled_amount	Scaled_time	V1	V2	V3	V4	V5	
0	1.488894	-2.045837	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.4
1	-0.294453	-2.045837	1.191857	0.266151	0.166480	0.448154	0.060018	-0.0
2	4.268843	-2.045789	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.8
3	1.171866	-2.045789	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.2
4	0.522393	-2.045741	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.0

```
# Splitting of data for model
# X - Predictor variable
# y - Target variable

X = data.iloc[:,:-1]
y = data[['Class']]
```

```
# Splitting into train and test values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=42
print('X_train shape: {}'.format(X_train.shape))
print('X_test shape: {}'.format(X_test.shape))
```

```
X_train shape: (74544, 30)
X_test shape: (18637, 30)
```

TRAINING THE MODELS

```
# This is a classification task
# We will use the following models

classifiers = {
    'LogisticRegression': LogisticRegression(),
    'RandomForestClassifier': RandomForestClassifier(),
    'AdaBoostClassifier': AdaBoostClassifier(),
    'DecisionTreeClassifier': DecisionTreeClassifier(),
    'SVC': SVC(),
    'MLPClassifier': MLPClassifier(),
    'KNeighborsClassifier': KNeighborsClassifier(),
    'XGBClassifier': XGBClassifier()
}
```

```
# Function to train model and do predictions
# We want a model with high recall as to detect outliers - Fraud transcations

precision = []
recall = []

for name, clf in classifiers.items():
    start = time.time()
    name = clf.fit(X_train, y_train)
    end = time.time()
    y_pred = name.predict(X_test)

    print('**************************************************************************
    print('\nModel: {}'.format(name))
    print('\nTime taken: {:.2f}min'.format((end-start)/60))
    print('\nTrainig Accuracy: {:.2f}%'.format(name.score(X_train, y_train)*100))
    print('\nTest Accuracy: {:.2f}%'.format(name.score(X_test, y_test)*100))
    #print('\nConfusion Matrix: \n')
    #print(confusion_matrix(y_test, y_pred))
    print('\nPrecision Score: {:.3f}'.format(precision_score(y_test, y_pred)))
    print('\nRecall Score: {:.3f}'.format(recall_score(y_test, y_pred)))
    precision.append(precision_score(y_test, y_pred))
    recall.append(recall_score(y_test, y_pred))
    print('\n')
    print('**************************************************************************

# i = recall.index(max(recall))
# print('\nModel with best recall: {}'.format(classifiers[i]))
# print('\nRecall: {}, Precision: {}'.format(recall[i], precision[i]))
```

```
*************************************************************************************

Model: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

Time taken: 0.01min

Trainig Accuracy: 99.87%

Test Accuracy: 99.82%

Precision Score: 0.692

Recall Score: 0.692


*************************************************************************************
*************************************************************************************

Model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

Time taken: 0.09min

Trainig Accuracy: 100.00%

Test Accuracy: 99.93%

Precision Score: 0.917

Recall Score: 0.846


*************************************************************************************
*************************************************************************************

Model: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0
                   n_estimators=50, random_state=None)

Time taken: 0.07min

Trainig Accuracy: 100.00%

Test Accuracy: 99.89%

Precision Score: 0.900

Recall Score: 0.692


*************************************************************************************
*************************************************************************************
```

```
Model: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

Time taken: 0.01min

Trainig Accuracy: 100.00%

Test Accuracy: 99.89%

Precision Score: 0.833

Recall Score: 0.769


************************************************************************************
************************************************************************************

Model: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

Time taken: 0.01min

Trainig Accuracy: 99.95%

Test Accuracy: 99.86%

Precision Score: 0.889

Recall Score: 0.615


************************************************************************************
************************************************************************************

Model: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='adam',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)

Time taken: 0.11min

Trainig Accuracy: 99.97%

Test Accuracy: 99.91%

Precision Score: 0.846

Recall Score: 0.846


************************************************************************************
```

```
*************************************************************************************

Model: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')

Time taken: 0.00min

Trainig Accuracy: 99.91%

Test Accuracy: 99.89%

Precision Score: 0.833

Recall Score: 0.769


*************************************************************************************
*************************************************************************************

Model: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

Time taken: 0.05min

Trainig Accuracy: 100.00%

Test Accuracy: 99.91%
```

```python
# Wow our scores are getting even high scores even when applying cross validation.

for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    print(classifier.__class__.__name__, ": ", round(training_score.mean(), 2) * 100, "
```

```
LogisticRegression :  100.0 % accuracy score
RandomForestClassifier :  100.0 % accuracy score
AdaBoostClassifier :  100.0 % accuracy score
DecisionTreeClassifier :  100.0 % accuracy score
SVC :  100.0 % accuracy score
MLPClassifier :  100.0 % accuracy score
KNeighborsClassifier :  100.0 % accuracy score
XGBClassifier :  100.0 % accuracy score
```

```python
# Use GridSearchCV to find the best parameters.


# Logistic Regression
log_reg_params = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
grid_log_reg = GridSearchCV(LogisticRegression(), log_reg_params)
grid_log_reg.fit(X_train, y_train)
# We automatically get the logistic regression with the best parameters
```

```
# We automatically get the logistic regression with the best parameters.
log_reg = grid_log_reg.best_estimator_


# K-Nears
knears_params = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['auto', 'ball_tree',
grid_knears = GridSearchCV(KNeighborsClassifier(), knears_params)
grid_knears.fit(X_train, y_train)
# KNears best estimator
knears_neighbors = grid_knears.best_estimator_


# Support Vector Classifier
svc_params = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}
grid_svc = GridSearchCV(SVC(), svc_params)
grid_svc.fit(X_train, y_train)
# SVC best estimator
svc = grid_svc.best_estimator_


# DecisionTree Classifier
tree_params = {"criterion": ["gini", "entropy"], "max_depth": list(range(2,4,1)),
              "min_samples_leaf": list(range(5,7,1))}
grid_tree = GridSearchCV(DecisionTreeClassifier(), tree_params)
grid_tree.fit(X_train, y_train)
# tree best estimator
tree_clf = grid_tree.best_estimator_

# Random
random_params =  {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']}
grid_random = GridSearchCV(RandomForestClassifier(), random_params)
grid_random.fit(X_train, y_train)
# random best estimator
random_clf = grid_random.best_estimator_

# MLP
mlp_params = {'solver': ['lbfgs'],
              'max_iter': [1000,1100,1200,1300,1400,1500,1600,1700,1800,1900,2000],
              'alpha': 10.0 ** -np.arange(1, 10),
              'hidden_layer_sizes':np.arange(10, 15)}
grid_mlp = GridSearchCV(MLPClassifier(), mlp_params)
grid_mlp.fit(X_train, y_train)
# random best estimator
mlp_clf = grid_mlp.best_estimator_

# Ada
ada_params = {'base_estimator__max_depth':[1,50],
          'base_estimator':[DecisionTreeClassifier(max_features=2), DecisionTreeClassif
grid_ada = GridSearchCV(AdaBoostClassifier(base_estimator=DecisionTreeClassifier()), ad
grid_ada.fit(X_train, y_train)
# best
ada_clf = grid_ada.best_estimator_
```

```
ada_clf = grid_ada.best_estimator_
```

```
# Now we will train models again with the parameters obtained from
# GridSearchCV
# Lets see if we can get higher recall values

estimators = [log_reg, random_clf, ada_clf ,tree_clf, svc, mlp_clf, knears_neighbors]

names = ["Logistic Regression", "Nearest Neighbors",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    LogisticRegression(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    DecisionTreeClassifier(),
    SVC(),
    MLPClassifier(),
    KNeighborsClassifier()]
```

```
# Function to train model and do predictions, with best parameters
precision = []
recall = []

for name, clf, est in zip(names, classifiers, estimators):
    start = time.time()
    name = est.fit(X_train, y_train)
    end = time.time()
    y_pred = name.predict(X_test)

    print('****************************************************************************
    print('\nModel: {}'.format(name))
    print('\nTime taken: {:.2f}min'.format((end-start)/60))
    print('\nTrainig Accuracy: {:.2f}%'.format(name.score(X_train, y_train)*100))
    print('\nTest Accuracy: {:.2f}%'.format(name.score(X_test, y_test)*100))
    #print('\nConfusion Matrix: \n')
    #print(confusion_matrix(y_test, y_pred))
    print('\nPrecision Score: {:.3f}'.format(precision_score(y_test, y_pred)))
    print('\nRecall Score: {:.3f}'.format(recall_score(y_test, y_pred)))
    precision.append(precision_score(y_test, y_pred))
    recall.append(recall_score(y_test, y_pred))
    print('\n')
    print('****************************************************************************

i = recall.index(max(recall))
print('\nModel with best recall: {}'.format(names[i]))
print('\nRecall: {}, Precision: {}'.format(recall[i], precision[i]))
```

```
*************************************************************************************

Model: LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)

Time taken: 0.01min

Trainig Accuracy: 99.89%

Test Accuracy: 99.84%

Precision Score: 0.750

Recall Score: 0.692


*************************************************************************************
*************************************************************************************

Model: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=7, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

Time taken: 0.16min

Trainig Accuracy: 99.98%

Test Accuracy: 99.89%

Precision Score: 0.900

Recall Score: 0.692


*************************************************************************************
*************************************************************************************

Model: AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=1,
                                                         max_features=2,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1.0, n_estimators=50, random_state=None)
```

```
Time taken: 0.01min

Trainig Accuracy: 100.00%

Test Accuracy: 99.89%

Precision Score: 0.900

Recall Score: 0.692


************************************************************************************
************************************************************************************

Model: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=6, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')

Time taken: 0.00min

Trainig Accuracy: 99.90%

Test Accuracy: 99.89%

Precision Score: 0.900

Recall Score: 0.692


************************************************************************************
************************************************************************************

Model: SVC(C=0.5, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

Time taken: 0.00min

Trainig Accuracy: 99.97%

Test Accuracy: 99.93%

Precision Score: 0.917

Recall Score: 0.846


************************************************************************************
************************************************************************************

Model: MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1600,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
```

```
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)

Time taken: 0.01min

Trainig Accuracy: 100.00%

Test Accuracy: 99.93%

Precision Score: 0.917

Recall Score: 0.846


*************************************************************************************
*************************************************************************************

Model: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')

Time taken: 0.00min

Trainig Accuracy: 99.92%

Test Accuracy: 99.89%

Precision Score: 0.900

Recall Score: 0.692


*************************************************************************************

Model with best recall: Neural Net
```

```
# As Neural net and KNN has same score we will test using both
# The better one will be chosen

mlp = MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1600,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)

print('Confusion Matrix:\n',confusion_matrix(y_test.round(), y_pred_mlp.round()))
print('Recall score: ',recall_score(y_test.round(),y_pred_mlp.round()))
print('Precision score:',precision_score(y_test.round(),y_pred_mlp.round()))
```

```
Confusion Matrix:
 [[18586     8]
 [   12    31]]
Recall score:  0.7209302325581395
Precision score: 0.7948717948717948
```

```
# KNN has lower recall than MLP on test set

knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

print('Confusion Matrix: \n',confusion_matrix(y_test.round(), y_pred_knn.round()))
print('Recall score: ',recall_score(y_test.round(),y_pred_knn.round()))
print('Precision score: ',precision_score(y_test.round(),y_pred_knn.round()))
```

```
Confusion Matrix:
 [[18592     2]
 [   13    30]]
Recall score:  0.6976744186046512
Precision score:  0.9375
```

```
# As Neural net has higher recall we will use that and
# Prepare submission file

output = pd.DataFrame({'Time': X_test.Scaled_time, 'Amount': X_test.Scaled_amount, 'Pre
output.to_csv('submission.csv', index=False)
```

```

```