C3879C Capstone Project

Credit Card Fraud Detection System

*Kandasamy Subbaian **Karthik***

ID# 18061866

INTRODUCTION:

- It is about automation of identifying fraudulent Credit Card transaction with the help of supervised machine learning (ML) algorithms such as Naïve Bayes, Support Vector Machine (SVM) and Logistics Regression.

- In the Artificial Intelligence (AI) era, It helps Financial Institutions (FI) to leverage Machine Learning (ML) to perform anomaly detection.

PROBLEM STATEMENTS:

- With the legacy systems, Financial Institutions (FIs) are struggling to automate detection of credit card fraudulent transaction precisely.
- As a pre-emptive approach to tackle fraud, Machine Learning (ML) helps FIs to detect anomaly precisely in a cost-effective manner.
- As a counter measure, the “Credit Card Fraud Detection” system helps FIs to find anomaly and increase customer experience (or) satisfaction.

PROJECT REQUIREMENTS:

- There is a need to develop a system to identify/classify whether the new credit card transaction is a fraud or normal transaction with the help of Machine Learning Models.

- Fraudulent transaction will be notified to the concerned financial officer for their further action, if the transaction is fraudulent.

SYSTEM REQUIREMENTS:

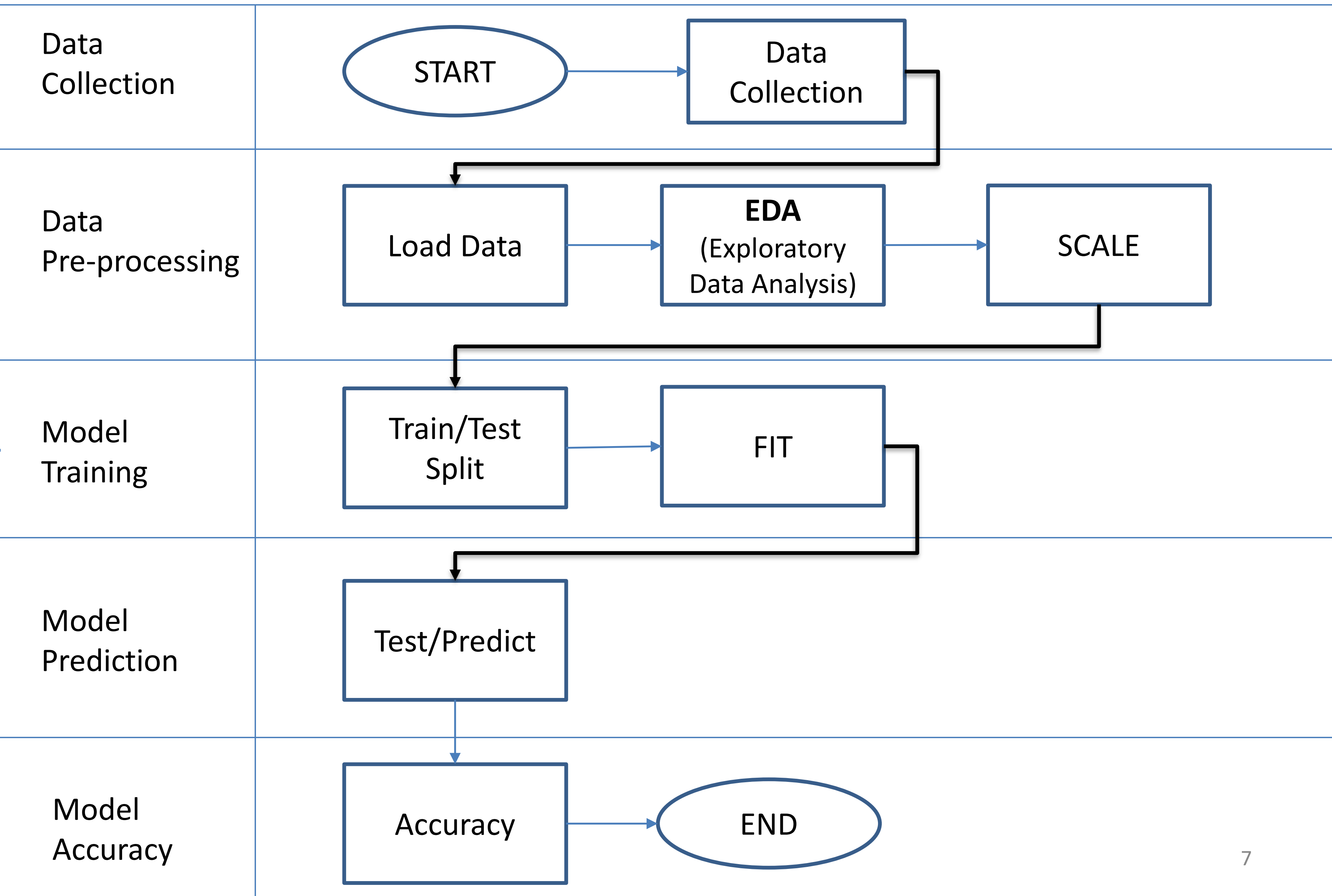
SOFTWARE REQUIREMENT	VERSION
Python	2.7 (Or above)
Scipy	1.3.0
Numpy	1.3.0
matplotlib	3.1.0
Pandas	0.24.2
scikit-learn	0.21.1
Windows 64 bit	10

HARDWARE REQUIREMENT
RAM 8 GB
CPU 1,80 GHz
Storage 40 GB (minimum)

CONSTRAINT/ASSUMPTIONS:

- Since the data collection is a time-consuming process, I intend to use the dataset available on www.kaggle.com.
- The dataset has 31 features, of which 28 features do not have any description about it.
- https://www.kaggle.com/mlg-ulb/creditcardfraud

DATA FLOW DIAGRAM:



IMPLEMENTAION:

Dataset Features:

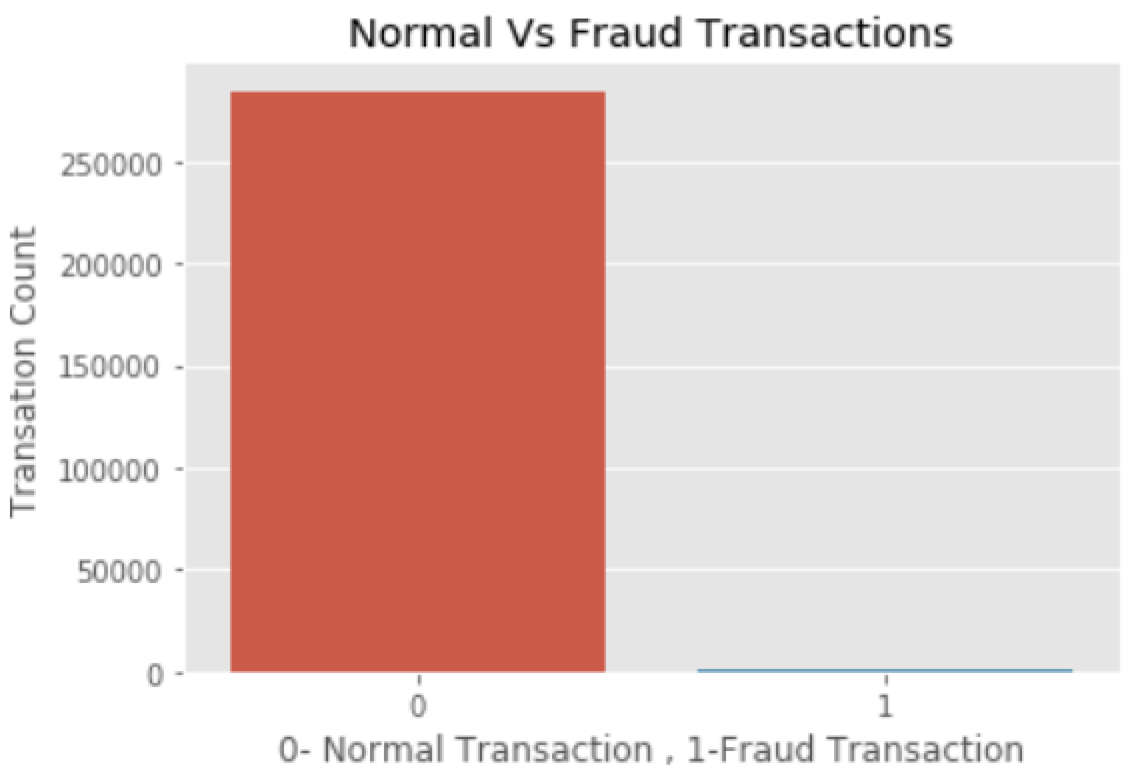
- The dataset has 31 features.
  - 28 anonymized features
  - 3 non-anonymized features (Time, Amount and Class)
- The 28 features are already in the form of a PCA (Principal Component Analysis) complaint.
- They are labeled V1 through V28.
- Both Time and Amount are not in the form of PCA.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
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	0.12853
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	0.16717
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	-0.32764
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	0.64737
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	-0.20601

Exploratory Data Analysis (EDA):

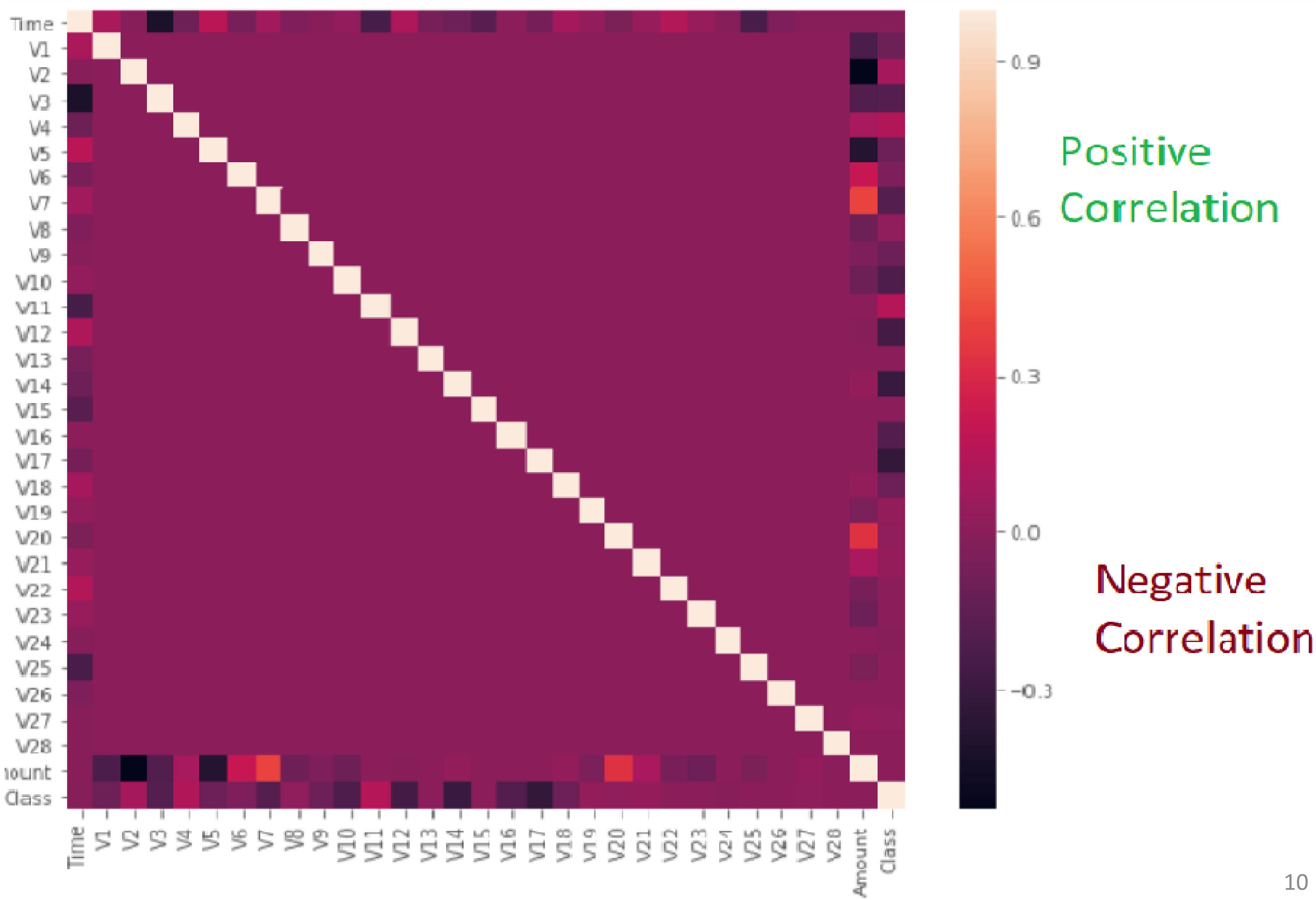
Imbalanced Dataset:

- Positive (1) – Fraud Transactions: 0.173%
- Negative(0) – Normal Transactions : 99.827%



FEATURES CORRELATION:

- Correlations between our **features** w.r.t the **Class**.



IMPUTATION OF MISSING VALUE:

- All predictors are PCA-compliant.
- No missing/Null values (i.e., No Imputation)
- All values ➔ Int (or) float data type.

```
V1      284807 non-null float64
V2      284807 non-null float64
V3      284807 non-null float64
V4      284807 non-null float64
V5      284807 non-null float64
V6      284807 non-null float64
V7      284807 non-null float64
V8      284807 non-null float64
V9      284807 non-null float64
V10     284807 non-null float64
V11     284807 non-null float64
V12     284807 non-null float64
V13     284807 non-null float64
V14     284807 non-null float64
```

```
V15     284807 non-null float64
V16     284807 non-null float64
V17     284807 non-null float64
V18     284807 non-null float64
V19     284807 non-null float64
V20     284807 non-null float64
V21     284807 non-null float64
V22     284807 non-null float64
V23     284807 non-null float64
V24     284807 non-null float64
V25     284807 non-null float64
V26     284807 non-null float64
V27     284807 non-null float64
V28     284807 non-null float64
Amount  284807 non-null float64
Class   284807 non-null int64
```

DATA PREPARATION – DATA SCALING:

- Anonymized (v1 to 28) features ➔ Scaled (i.e., centered around 0).
- Time and Amount ➔ Not in line with other features in terms of scaling. It affects the model performance.

```python
# Consider Non-anonymized (Amount and Time) features for normalisation.
from sklearn.preprocessing import StandardScaler

scaleTime = StandardScaler()
scaled_Time = scaleTime.fit_transform(cc_df[['Time']])
list_scaledTime = [stime for sublist in scaled_Time.tolist() for stime in sublist]
series_scaledTime = pd.Series(list_scaledTime)
```

```python
scaleAmount = StandardScaler()
scaled_Amount = scaleAmount.fit_transform(cc_df[['Amount']])
list_scaledAmt = [amt for sublist in scaled_Amount.tolist() for amt in sublist]
series_scaledAmt = pd.Series(list_scaledAmt)
```

```python
# Concatenating scaled time and amount with original dataframe
scaled_df = pd.concat([cc_df, series_scaledAmt.rename("scaled_amount"), series_scaledTime.rename("scaled_time")], axis=1)
scaled_df.head()
```

CREATE DATASET (Techniques):

- Since the dataset is highly imbalanced, it will lead ML Algorithms "to generalize the unseen data incorrectly". In other words, it will classify most of the (unseen) data as "Non-Fraud" Transaction.
- In order to avoid this, we can use below resampling techniques to create a balanced dataset.
  - Oversampling (minority class – Fraud Transaction)
  - Undersampling (majority class – Normal Transaction)
  - Generate synthetic samples (SMOTE or **S**ynthetic **M**inority **O**versampling **T**echnique uses a nearest neighbors algorithm)

CREATE DATASET - Undersampling:

Split the scaled dataset into Train and Test

```
mask = np.random.rand(len(scaled_df)) < 0.9
train = scaled_df[mask]
test = scaled_df[~mask]
print('Train Shape: {}\nTest Shape: {}'.format(train.shape, test.shape))
```

```
Train Shape: (256293, 31)
Test Shape: (28514, 31)
```

```
train.reset_index(drop=True, inplace=True)
test.reset_index(drop=True, inplace=True)
```

Create a sub-sample dataset with balanced class distribution

```
# Find how many Fraud Transactions are there in the (Random) Training Data
no_of_fraud_trans = train['Class'].value_counts()[1]
print('There are {} fraudulent transactions in the train data.'.format(no_of_fraud_trans))
```

```
There are 443 fraudulent transactions in the train data.
```

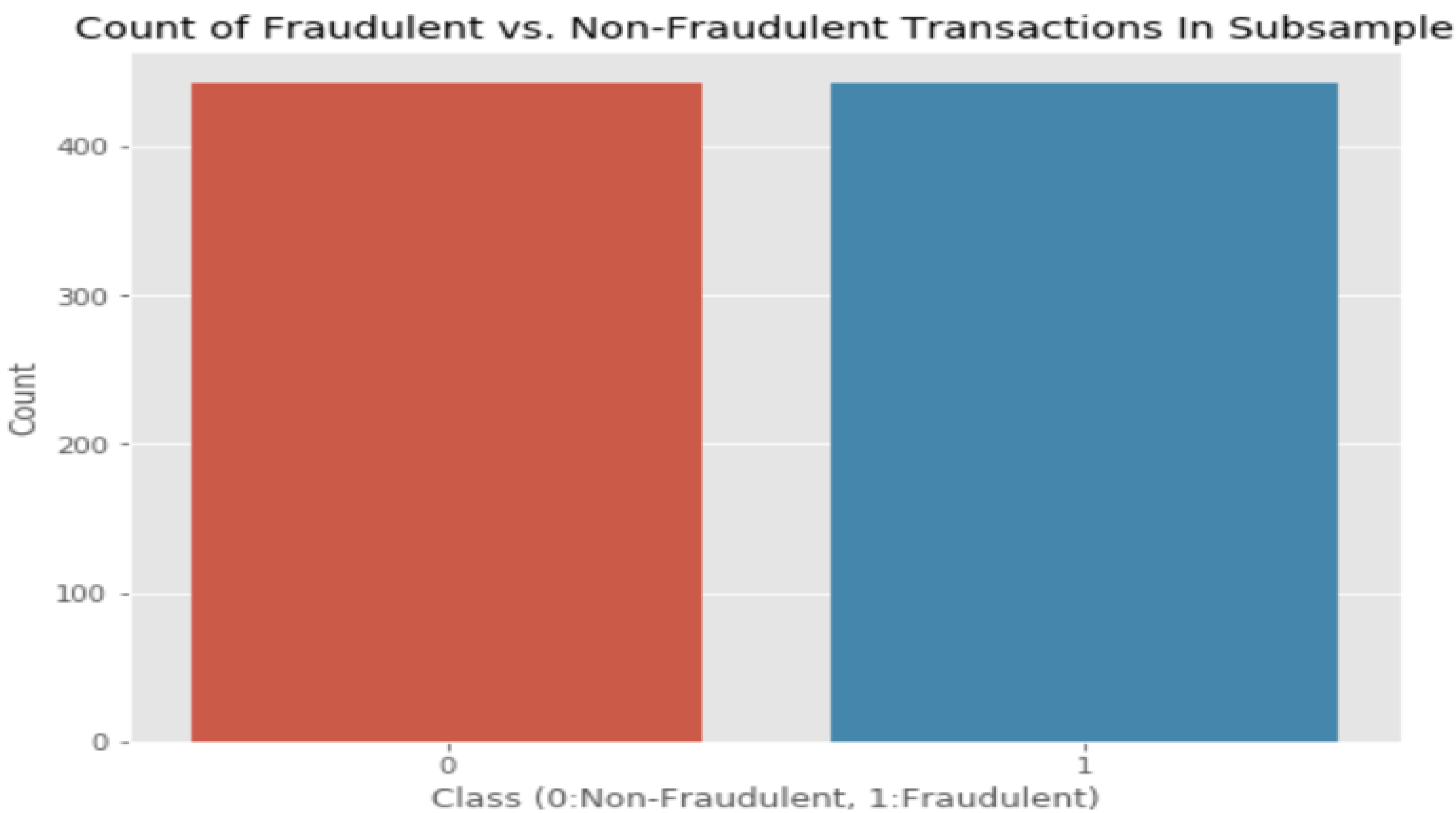
```
# Segregate Normal and Fraud Transactions from the Train Data
normal_trans_df = train[train['Class']==0]
fraud_trans_df = train[train['Class']==1]
```

```
# Randomly selecting the same no of Normal Trans (that is, 445) as Fraud Trans from Normal Trans Data
selected_norml_trans_df = normal_trans_df.sample(no_of_fraud_trans)
selected_norml_trans_df.head()
```

CREATE DATASET – Final Dataset:

```
# Concatenate both selected_norml_trans_df and fraud_trans_df
subsample_df = pd.concat([selected_norml_trans_df, fraud_trans_df])
subsample_df.shape
```

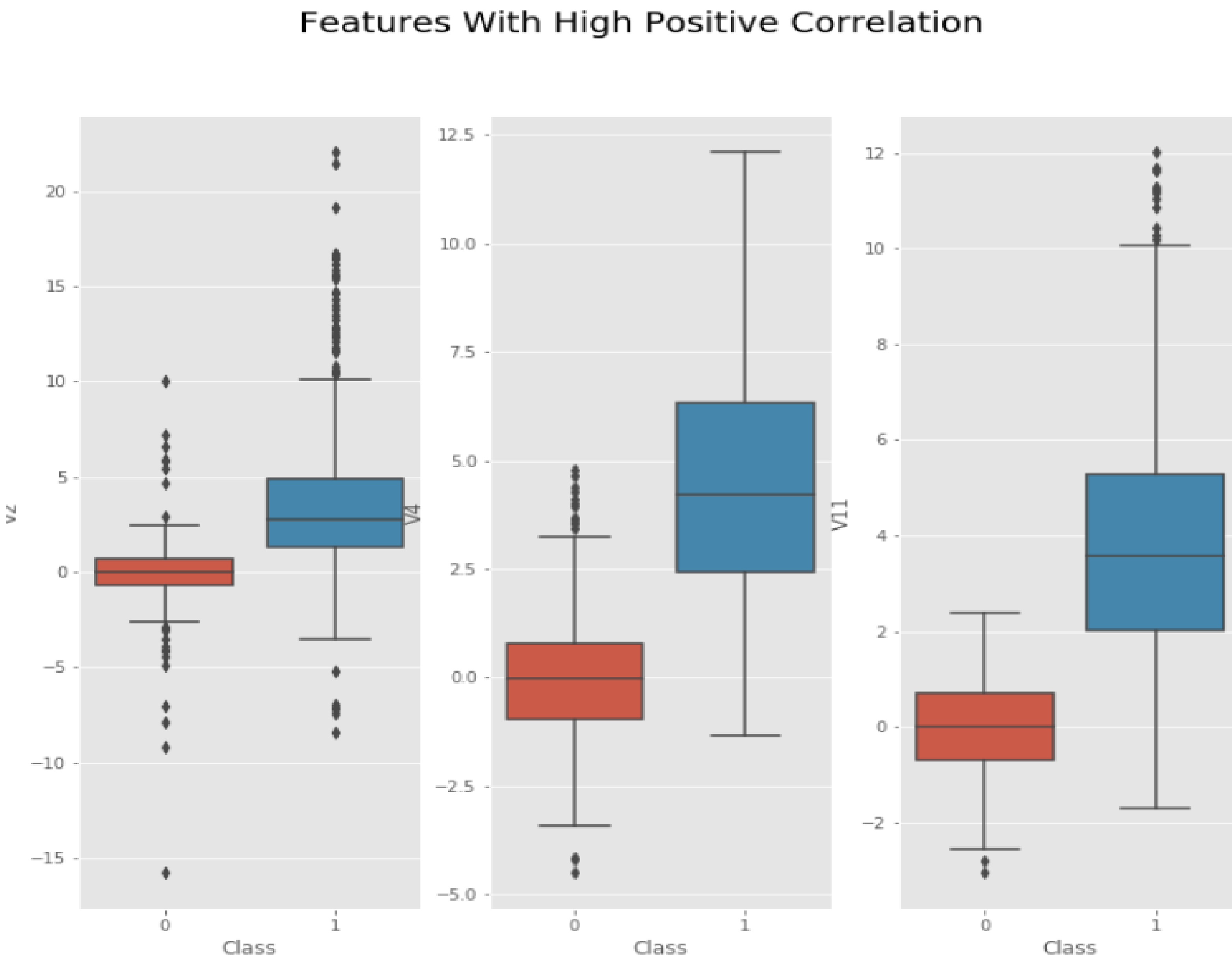
```
#shuffle the subsample df/dataset
subsample_df = subsample_df.sample(frac=1).reset_index(drop=True)
```



REMOVE OUTLINERS:

```
# Remove Extreme outliers
Q1_feature_val = subsample_df.quantile(0.25)
Q3_feature_val= subsample_df.quantile(0.75)
IQR = (Q3_feature_val - Q1_feature_val)
wo_outliner_df = subsample_df[~((subsample_df < (Q1_feature_val - 2.5 * IQR)) |(subsample_df > (Q3_feature_val + 2.5 * IQR))).an
```

REMOVE OUTLINERS:



SPLIT TRAIN & TEST:

- Split the dataset into Train and Test for learning.

Split Final Dataset into Train and Test

```
# Train and Test Split
from sklearn.model_selection import train_test_split
X = wo_outliner_df.drop('Class', axis=1)
y = wo_outliner_df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

SPOT-CHECK ALGORITHMS:

```
##Spot-Checking Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('SVM', SVC()))
models.append(('GNB', GaussianNB()))
```

```
#testing models
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=42)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring='roc_auc')
    results.append(cv_results)
    names.append(name)
    msg = '%s: %f (%f)' % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.971002 (0.028816)
SVM: 0.955585 (0.043666)
GNB: 0.955635 (0.037525)
```

PREDICTION ON TEST DATA:

- $\boxed{\text{Model = Algorithm(DATA)}}$
- Predict on Test Data
- Find Accuracy Score
- Confusion Matrix
- Classification Report

```python
# Make predictions on Test data
for name, model in models:
    model.fit(X_train,y_train)
    print("Predictions with " + name +":")
    predictions = model.predict(X_test)
    print("\n accuracy_score:")
    print(accuracy_score(y_test, predictions))
    print("\n Confusion Matrix:")
    print(confusion_matrix(y_test, predictions))
    print("\n Classification Report:")
    print(classification_report(y_test, predictions))
    print("--------------------------------------------------------")
```

MODEL PERFORMANCE:

- Precision: The number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. **It is a measure of a classifier's exactness.** Low precision indicates a high number of false positives.
- Recall: The number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. **It is a measure of a classifier's completeness.**
- F1-Score: The weighted average of precision and recall.
- Confusion Matrix:

	Prediction	
Actual	0 (Negative)	1 (Positive)
0 (-ve)	TN (True Positive)	FP (False Positive)
1 (+ve)	FN (False Negative)	TP (True Positive)

PERFORMANCE TEST DATA:

```
Predictions with LR:

 accuracy_score:
0.9186991869918699

 Confusion Matrix:
TN  [[78  2]   FP
     [ 8 35]]  TP
FN
 Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        80
           1       0.95      0.81      0.88        43

   micro avg       0.92      0.92      0.92       123
   macro avg       0.93      0.89      0.91       123
weighted avg       0.92      0.92      0.92       123

-------------------------------------------------------
```

```
Predictions with SVM:

 accuracy_score:
0.9186991869918699

 Confusion Matrix:
[[78  2]
 [ 8 35]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        80
           1       0.95      0.81      0.88        43

   micro avg       0.92      0.92      0.92       123
   macro avg       0.93      0.89      0.91       123
weighted avg       0.92      0.92      0.92       123

-------------------------------------------------------
```

```
Predictions with GNB:

 accuracy_score:
0.9024390243902439

 Confusion Matrix:
[[78  2]
 [10 33]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        80
           1       0.94      0.77      0.85        43

   micro avg       0.90      0.90      0.90       123
   macro avg       0.91      0.87      0.89       123
weighted avg       0.91      0.90      0.90       123
```

Accuracy = (TP+TN)/Total = (78+35)/123 = 0.9186991869918699

Error Rate = 1 - Accuracy = (FN + FP)/Total = (8+2)/123 = 0.08130081300

Sensitivity/Recall = TP /(TP + FN) = 35/(8+35) = 0.813953488

F1 Score = (Precision + Recall)/2 = (0.95 + 0.88)/2 = 0.88

HYPERPARAMETERS - GridSearchCV:

- Of 3 Algorithms compared above, Logistic Regression is better than both SVM and GNB.
- Performance of both Logistic Regression and SVM are sometime equal.

Hyperparameters Tuning - Logistic Regression

```
# Grid search cross validation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
grid={"C":np.logspace(-3, 3, 20), "penalty":['l1', 'l2'], "solver":["liblinear"]} # l1 ->lasso l2->ridge
logreg=LogisticRegression()
logreg_cv=GridSearchCV(logreg,grid,cv=10,scoring='roc_auc')
logreg_cv= logreg_cv.fit(X_train,y_train)

print("accuracy :",logreg_cv.best_score_)
print('Best Penalty:', logreg_cv.best_estimator_.get_params()['penalty'])
print('Best C:', logreg_cv.best_estimator_.get_params()['C'])
```

```
accuracy : 0.9797634107406404
Best Penalty: l2
Best C: 0.1623776739188721
```

PREDICT ON TEST DATA WITH BEST MODEL:

Predict on Test Data with the best model

```
logreg_cv.predict(X_test)
```

```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0], dtype=int64)
```

Accuracy Before Tuning	Accuracy After Tuning
0.9186991869918699	0.979763410740640 4

```
logreg_cv
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': array([1.00000e-03, 2.06914e-03, 4.28133e-03, 8.85867e-03, 1.83298e-02,
       3.79269e-02, 7.84760e-02, 1.62378e-01, 3.35982e-01, 6.95193e-01,
       1.43845e+00, 2.97635e+00, 6.15848e+00, 1.27427e+01, 2.63665e+01,
       5.45559e+01, 1.12884e+02, 2.33572e+02, 4.83293e+02, 1.00000e+03]), 'penalty': ['l1', 'l2'], 'solver': ['liblinear']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

ROC-AUC:

- The *receiver operating characteristic* (ROC) curve is another common tool used with binary classifiers.

```
from sklearn.metrics import roc_auc_score, roc_curve
y_pred_proba = logreg_cv.predict_proba(X_test)[::,1]
fpr, tpr, _ = roc_curve(y_test,  y_pred_proba)
plt.figure(figsize=(8,8))
plt.plot(fpr, tpr, linewidth=2, label=None)
plt.plot([0, 1], [0, 1],"--")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
# ROC AUC SCORE
auc = roc_auc_score(y_test, y_pred_proba)
print(auc)
```

```
0.9558823529411765
```

ROC-AUC:

- Perfect classifier will have a ROC AUC equal to 1.
- Random classifier will have a ROC AUC equal to 0.5.