In [24]:
```
# Homework-3 Credit Card Data Fraud Analsysis
# Read the credit card csv file
# Review, Explore and Analyze Aspects of Credit Card Data
# Plot/Visualize the dataset independent/dependent variables
# Prepare Train and Test Data for the Machine Learning Model
# Perform Machine Learning using Logistic Regression and Random Forest
# Train and Test the model and show model scores/performance
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import random
import seaborn as sns
import datetime
import sklearn
import missingno as msno
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import ensemble
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_confusion_matrix

############################ Get the data from .csv #########################
##################################

# Read the credit card csv file into dataframe;
# By default, low_memory is True when loading data into memory and processes t
he file in chunks while parsing
print ('\033[1m \033[4mStep 1- Read Kaggle Credit Card .CSV File Data into Dat
aframe\033[0m' + "\n")
df_Card = pd.read_csv(r'creditcard.csv', low_memory=False)

############################# Review, Explore and Analyze Aspects of Credit Ca
rd Data ##################################
# Print Kaggle credit card records; Note that out of 30 columns of independent
data, only 2 columns are named/defined;
# The remaining column are masked/hidden through principal component analysis
 (dimensionality reduction)
# First column is for 'Time' (seconds between the transactions) and 2nd last c
olumn is for 'Amount' (Transaction amount)
# Last columns is the dependent variable and denotes 'Count' (0=No Fraud, 1=Fr
aud)

print ('\x1b[43m \033[1m \033[4mStep 2- Review, Explore and Analyze Credit Car
d Data \033[0m \x1b[0m' + "\n")
print ('\033[1m \033[4m2A- Print Dataframe Header Data \033[0m' + "\n")
print(df_Card.head())
```

```
print("\n")

# Lets find total number of columns and rows of credit card data
print('\033[1m Total Number of Rows and columns in Credit Card Data: \033[0m
{}'.format(df_Card.shape))
print ('\033[1m Note: There are 30 Independent Variables and 1 Dependent Varia
ble - Class (Boolean, 1-Fraud/0-No Fraud) \033[0m')
print ('\033[1m Independent Variables in Colunm 1-28 are masked/hidden for pri
vacy reasons \033[0m')
print("\n")

# Explore the 'Class' column ('Class'= 1-Fraud/0-No Fraud)
print ('\033[1m \033[4m2B- Show Fraud and Non Fraud Record Details: Class 1=Fr
aud, 0=No Fraud \033[0m' + "\n")

# lets check the 'class' column to see how many credit card data records are f
raud and non-fraud
fraud = df_Card.loc[df_Card['Class'] == 1]
fraud_indices = df_Card[df_Card.Class == 1].index
non_fraud = df_Card.loc[df_Card['Class'] == 0]
non_fraud_indices = df_Card[df_Card.Class == 0].index
print('Number of Fraud Records : {}'.format(len(fraud)))
print('Number of Nonfraud Records : {}'.format(len(non_fraud)))
print('Note that the Number of Fraudulant Transactions are quite Small relativ
e to Total credit card tTransactions')
print('\n')
print('Fraud Record Indices : {}'.format(fraud_indices))
print('\n')
print('Nonfraud Record indices {}'.format(non_fraud_indices))
print('\n')

# Use describe function to get further insight and view each independent/depen
dent variable's mean, std, min, max, percentile etc.
print ('\033[1m \033[4m2C- Show Statistical Details About Independent and Depe
ndent Variables\033[0m' + "\n")
print(df_Card.describe())
print("\n")

# Use info() function to get insight into info (count, non-null, type (float/i
nt) of each independent/dependent variable
print ('\033[1m \033[4m2D- Show Credit Card Data Info for Each Column or Depen
dent/Independent Variables\033[0m' + "\n")
print(df_Card.info())
print("\n")

# Check for total number of missing or null values in each column
print ('\033[1m \033[4m2E- Show Total Number of Null Values in Each Column\033
[0m' + "\n")
print(df_Card.isnull().sum())
print("\n" + 'No Null Values Found' + "\n")

############################## Plot/Visualize the dataset independent/dependent
variables  ##############################

print ('\x1b[43m \033[1m \033[4m3- Plot Credit Card Data \033[0m \x1b[0m' + "
\n")
```

```
# Visualize the dataframe with no missing value.
print ('\033[1m \033[4m3A- Visualize the Dataframe With No Missing Values \033
[0m')
msno.matrix(df_Card)
plt.show()
print('\n')

# Plots ot show that fruad cases are for smaller amount transactions
print('\033[1m \033[4m3B- Note: Fraud Transactions are centered around Lower A
mount Transactions under 3000 \033[0m' + "\n")
plt.figure(figsize=(10,6))
plt.subplot(1,2,1)
Fraud_Amt = df_Card[df_Card["Class"]==1].Amount
plt.xlabel('Transaction Amount')
plt.ylabel('Transaction Frequency')
plt.title('Fraud Transacations')
Fraud_Amt.hist(figsize=(10,6))
plt.subplot(1,2,2)
plt.xlabel('Transaction Amount')
plt.title('Non Fraud Transacations')
Non_Fraud_Amt = df_Card[df_Card["Class"]==0].Amount
Non_Fraud_Amt.hist(figsize=(10,6))
plt.show()

# Scatter Plot
plt.figure(figsize=(10,6))
plt.scatter(x=df_Card['Amount'], y=df_Card['Class'], marker='x', color='Orang
e', label='Fraud Transactions')
plt.scatter(x=df_Card['Amount'], y=df_Card['Class'], marker='s', color='blue',
label='Non_Fraud Transactions')
plt.legend(loc='upper right')
plt.title('Credit Card Fraud Info')
plt.xlabel('Amount')
plt.ylabel('Class: 1=Fraud, 0=No Fraud')
plt.show()

# Plot the Histograms for each independent/dependent variable
print('\033[1m \033[4m3B- "Plot Histograms...Note that most independent variab
les are centered around 0" \033[0m' + "\n")
df_Card.hist(figsize=(20,20))
plt.show()

# Plot the Histograms for each independent variable (V1...V28) against depende
nt variable class (fraud and non fraud)
for i in range(1,29):
    plt.subplot()
    sns.distplot( df_Card.iloc[:,i][df_Card.Class == 1] , color="orange", labe
l="Fraud")
    sns.distplot( df_Card.iloc[:,i][df_Card.Class == 0] , color="deeppink", la
bel="Non Fraud")
    plt.show()

############################# Prepare Train and Test Data for the Machine Lear
ning Model ##############################

print ('\x1b[43m \033[1m \033[4m4- Prepare Train and Test Data \033[0m \x1b[0
m' '\n')
```

```
# Create X (independent variables) and y (dependent variable that we are tryin
g to predict, Class column) data sets for logistic regression
X_carddata = df_Card.iloc[:,:-1]
y_classdata = df_Card.iloc[:,-1]

# StandardScaler function helps standardize the data into same scale and distr
ibution for machine learning
scaler = StandardScaler()
scaler.fit(X_carddata)

# Divide the data set into training data and test data 65/35 split using 'trai
n_test_split' function
from sklearn.model_selection import train_test_split
X_carddata_train, X_carddata_test, y_classdata_train, y_classdata_test = train
_test_split(X_carddata, y_classdata, test_size=0.35)

print('Total data size is : {}'.format(len(df_Card)))
#Keep 65% data for training
train_size = 0.65
test_size = 1 - train_size
print('Training data size is ' + str(train_size*100) + '% : ' + '{}'.format(in
t(round(train_size * len(df_Card)))))
#Keep 35% data for testing
print('Test data size is ' + str((1-train_size)*100) + '% : ' + '{}'.format(in
t(round((len(df_Card)*(1 - train_size))))))
print('\n')

#################### Logistic Regression To Train and Test #################
####################################################
# Use Logistic regression (predictive analysis) to find dependent binary varia
ble (Fraud/No fraud) from independent variables.
# Logistic regression to predict the probability of a categorical dependent va
riable

print ('\x1b[43m \033[1m \033[4m5- Starting Logistic Regression \033[0m \x1b[0
m' '\n')

# Create Logistic Regression Estimator/Model Object for training, testing/pred
iction
# Use C is inverse of cross regularization parameter for principal component a
nalysis (PCA) or dimensionality reduction
# C=100000 so we are not overfitting on our trained dataset
Card_LogisticReg = linear_model.LogisticRegression(C=100000)

# Train the Logistic Regression model using training set
Card_LogisticReg_Result = Card_LogisticReg.fit(X_carddata_train, y_classdata_t
rain)

# Test to predict with this model using training data
y_Training_Predictions = Card_LogisticReg.predict(X_carddata_train)

# Test to predict with this model using test data
y_Test_Predictions = Card_LogisticReg.predict(X_carddata_test)

# Combine the training and testing predictions; Joining the two arrays along a
xis 0
```

```
y_All_Predictions = np.concatenate((y_Training_Predictions, y_Test_Predictions
), axis=0)

# Generate the classification report to qualify the quality of predictions
#The precision is the ratio tp / (tp + fp) where tp is the number of true posi
tives and fp the number of false positives. The precision is intuitively the a
bility of the classifier to not label a sample as positive if it is negative.
#The recall is the ratio tp / (tp + fn) where tp is the number of true positiv
es and fn the number of false negatives. The recall is intuitively the ability
of the classifier to find all the positive samples.
#The F-beta score can be interpreted as a weighted harmonic mean of the precis
ion and recall, where an F-beta score reaches its best value at 1 and worst sc
ore at 0.
#The F-beta score weights the recall more than the precision by a factor of be
ta. beta = 1.0 means recall and precision are equally important.
#The support is the number of occurrences of each class in y_test.

print('\033[1m \033[4m5A- Evaluate Logistic Regression with Precision, Recall,
F1-score in Classification Report\033[0m' '\n')
print(classification_report(y_classdata_test, y_Test_Predictions))
print('\n')

# calculate the Intercepts and coefficients
print('\033[1m \033[4m5B- Logistic Regression Intercepts and Coefficients \033
[0m' '\n')
print ('  Intercepts: ', list(Card_LogisticReg.intercept_))
print ('  Coeficiencts: ', list(Card_LogisticReg.coef_))
print('\n')

# calculate and plot the Confusion Matrix
print('\033[1m \033[4m5C- Logistic Regression Confusion Matrix \033[0m' '\n')
CM = confusion_matrix(y_classdata_test, y_Test_Predictions.round())
print(CM)
print('\n')
plot_confusion_matrix(conf_mat=CM, figsize=(5,4), hide_ticks=True, cmap=plt.cm
.Blues)
plt.title('Logistic Regression Confusion Matrix')
plt.show()

# Use the score method to evaluate the trained model
LogisticReg_Accuracy_Score = Card_LogisticReg.score(X_carddata_test, y_classda
ta_test)
print('\033[1m \033[4m5D-  Logistic Regression Accuracy \033[0m'  '\n')
print(' The Logistic Regression Accuracy Score is ' + str(LogisticReg_Accuracy
_Score) + '\n')
print('\n')

# Evaluate Outcome
print('\033[1m Note: While Logistic Regression Accuracy score is high, its Con
fusion Matrix is showing incorrect \033[0;0m')
print('\033[1m (false positives+false negatives) predictions. We should try ot
her models like Random Forest to compare. \033[0;0m')
print('\n')
#################### Random Forest  To Train and Test #####################
############################################
# Use Random Forest to for fraud analysis
#Random forest, like its name implies, consists of a large number of individua
```

```
l decision trees that operate as an ensemble.
#Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction

print ('\x1b[43m \033[1m \033[4m6- Starting Random Forest Analysis \033[0m \x1b[0m' '\n')

# Create Logistic Regression Estimator/Model Object for training, testing/prediction
Card_RandomFrst = ensemble.RandomForestClassifier(n_estimators=100)

# Train the Logistic Regression model using training set
Card_RandomFrst.fit(X_carddata_train, y_classdata_train)

# Test to predict with this model using training data
y_Training_Predictions = Card_RandomFrst.predict(X_carddata_train)

# Test to predict with this model using test data
y_Test_Predictions = Card_RandomFrst.predict(X_carddata_test)

# Combine the training and testing predictions; Joining the two arrays along axis 0
y_All_Predictions = np.concatenate((y_Training_Predictions, y_Test_Predictions), axis=0)

# calculate the Confusion Matrix
print('\033[1m \033[4m6A- Random Forest Confusion Matrix \033[0m' '\n')
CM = confusion_matrix(y_classdata_test, y_Test_Predictions.round())
print(CM)
print('\n')
plot_confusion_matrix(conf_mat=CM, figsize=(5,4), hide_ticks=True, cmap=plt.cm.Blues)
plt.title('Random Forest Confusion Matrix')
plt.show()

# Use the score method to evaluate the model
Card_RandomFrst_Accuracy_Score = Card_RandomFrst.score(X_carddata_test, y_classdata_test)
print('\033[1m \033[4m6B-  Random Forest Accuracy \033[0m'  '\n')
print(' The Random Forest Accuracy Score is ' + str(Card_RandomFrst_Accuracy_Score) + '\n')

# Evaluate Outcome
print('\033[1m Note: While Accuracy score of both Random Forest and Logistic Regression are high, Random Forest Confusion Matrix is showing \033[0m')
print('\033[1m more correct predictions and less incorrect (false positives+false negatives) predictions than Logistics Regression confusion matrix \033[0m')
print('\n')
```

Step 1- Read Kaggle Credit Card .CSV File Data into Dataframe

Step 2- Review, Explore and Analyze Credit Card Data

2A- Print Dataframe Header Data

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

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

Total Number of Rows and columns in Credit Card Data: (284807, 31)
Note: There are 30 Independent Variables and 1 Dependent Variable - Class (Boolean, 1-Fraud/0-No Fraud)
Independent Variables in Colunm 1-28 are masked/hidden for privacy reasons

2B- Show Fraud and Non Fraud Record Details: Class 1=Fraud, 0=No Fraud

```
Number of Fraud Records : 492
Number of Nonfraud Records : 284315
Note that the Number of Fraudulant Transactions are quite Small relative to T
otal credit card tTransactions


Fraud Record Indices : Int64Index([   541,    623,   4920,   6108,   6329,
 6331,   6334,   6336,
             6338,   6427,
            ...
            274382, 274475, 275992, 276071, 276864, 279863, 280143, 280149,
            281144, 281674],
           dtype='int64', length=492)


Nonfraud Record indices Int64Index([     0,      1,      2,      3,      4,
5,      6,      7,
```

```
              8,      9,
           ...
           284797, 284798, 284799, 284800, 284801, 284802, 284803, 284804,
           284805, 284806],
          dtype='int64', length=284315)
```

2C- Show Statistical Details About Independent and Dependent Variables

```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  3.919560e-15  5.688174e-16 -8.769071e-15  2.782312e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.537294e-16  7.959909e-16  5.367590e-16  4.458112e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
```

```
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

2D- Show Credit Card Data Info for Each Column or Dependent/Independent Variables

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
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
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
```

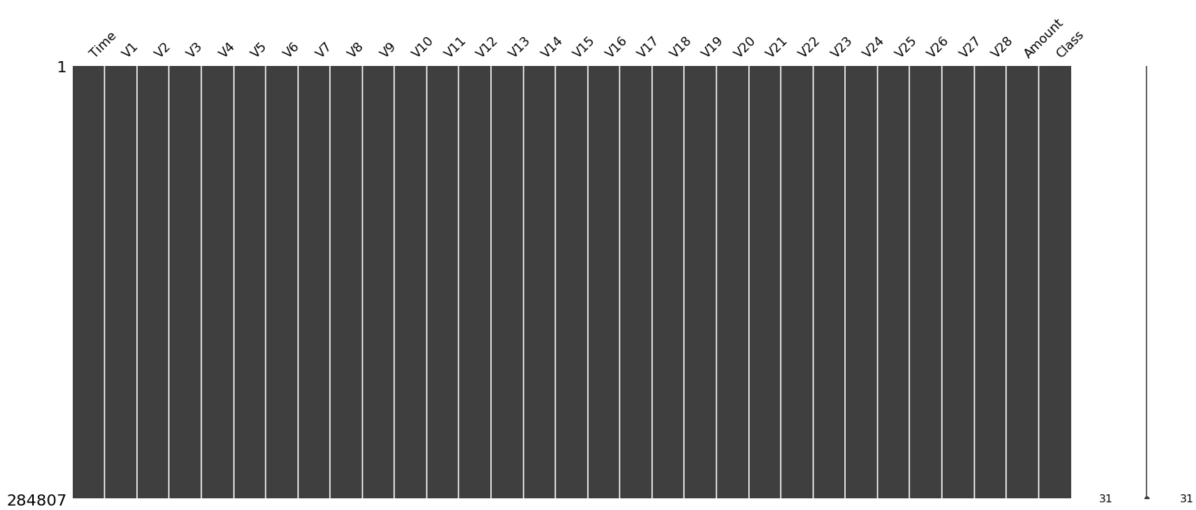
2E- Show Total Number of Null Values in Each Column

```
Time      0
V1        0
V2        0
V3        0
V4        0
```

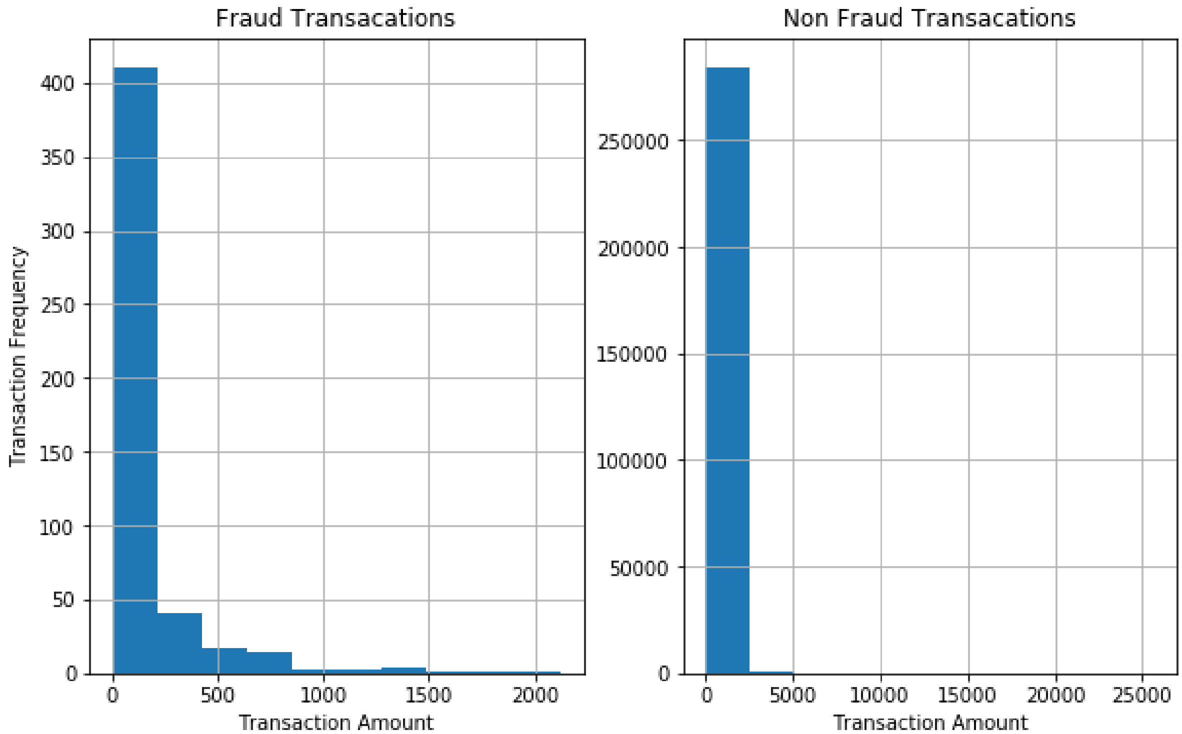
```
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount  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

No Null Values Found
```

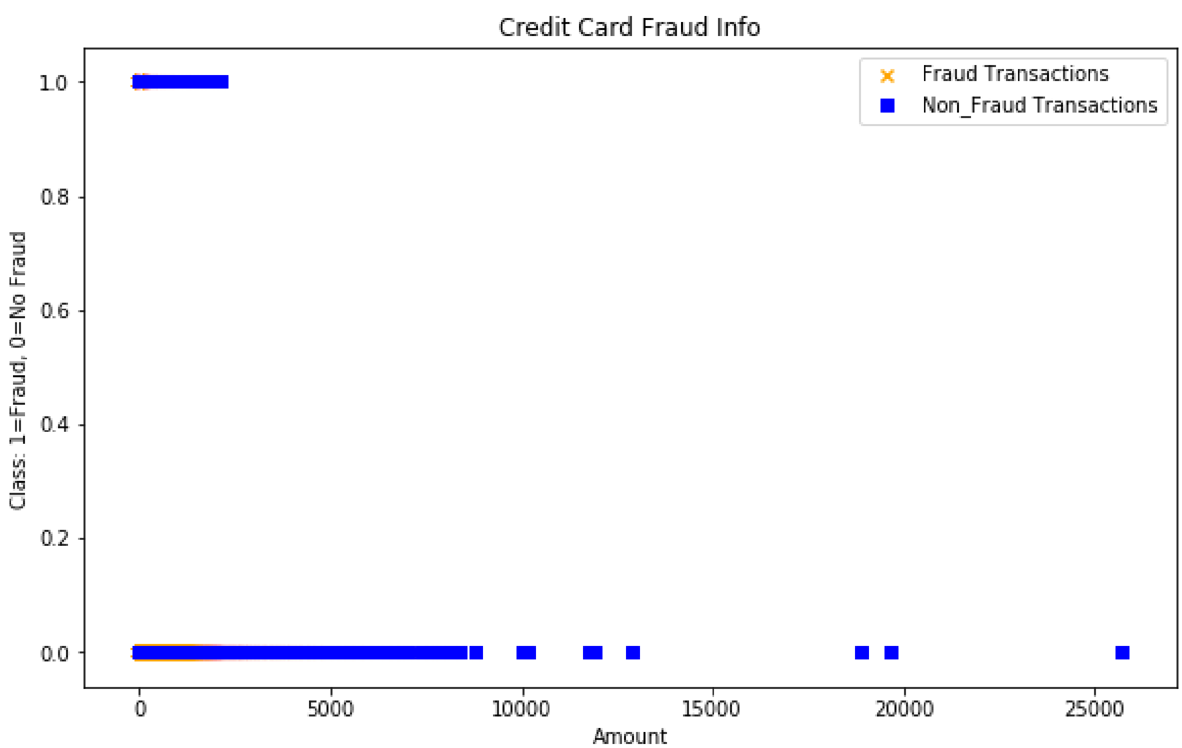
3- Plot Credit Card Data

3A- Visualize the Dataframe With No Missing Values

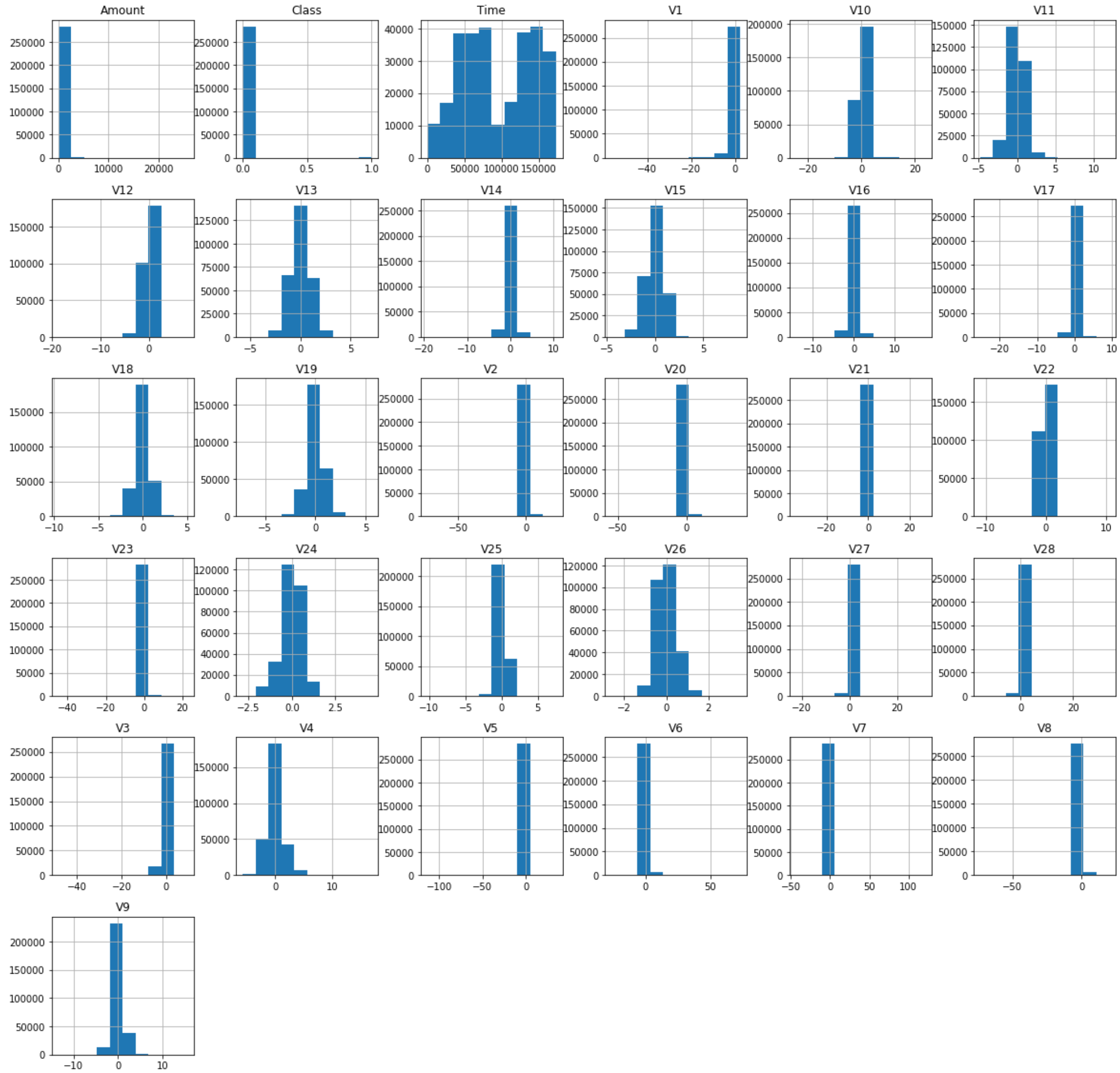


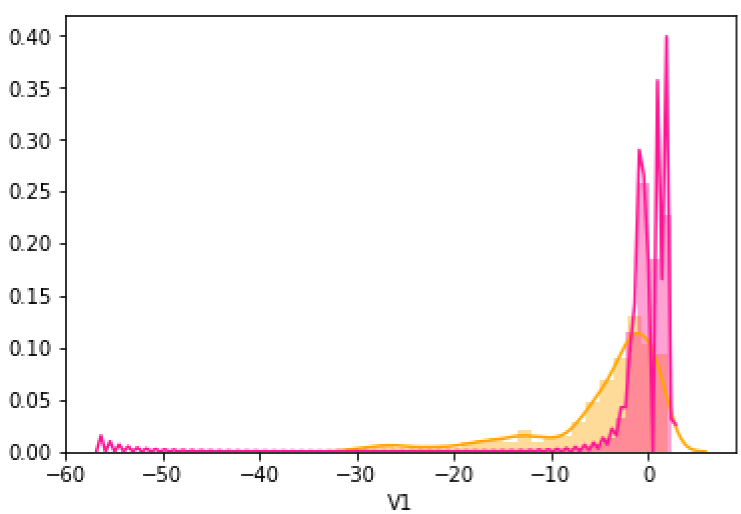
3B- Note: Fraud Transactions are centered around Lower Amount Transactions under 3000

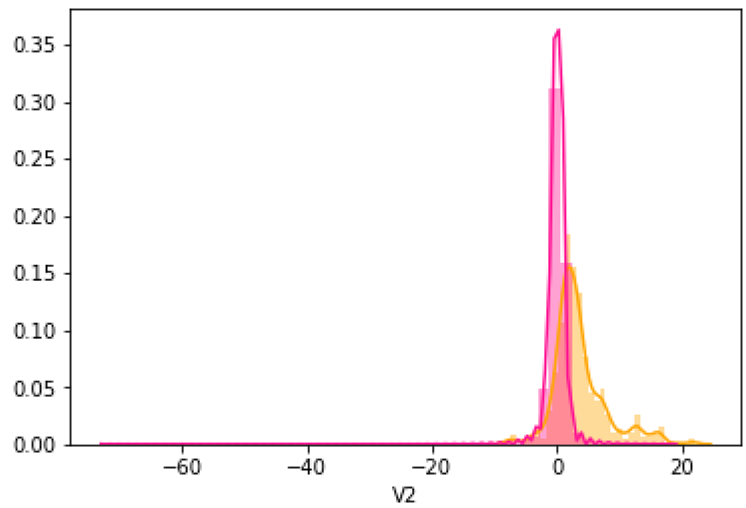


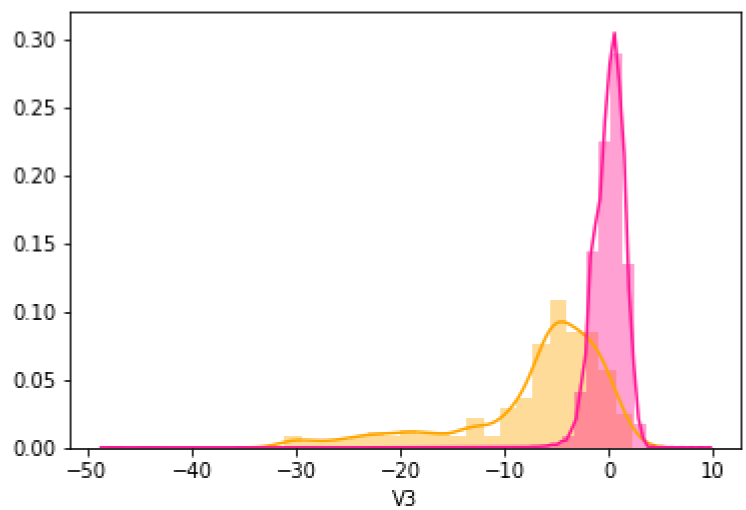


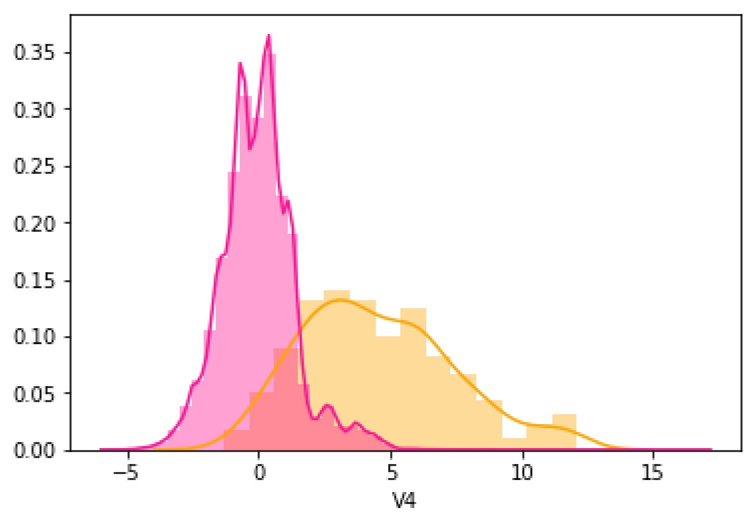
3B- "Plot Histograms...Note that most independent variables are centered around 0"

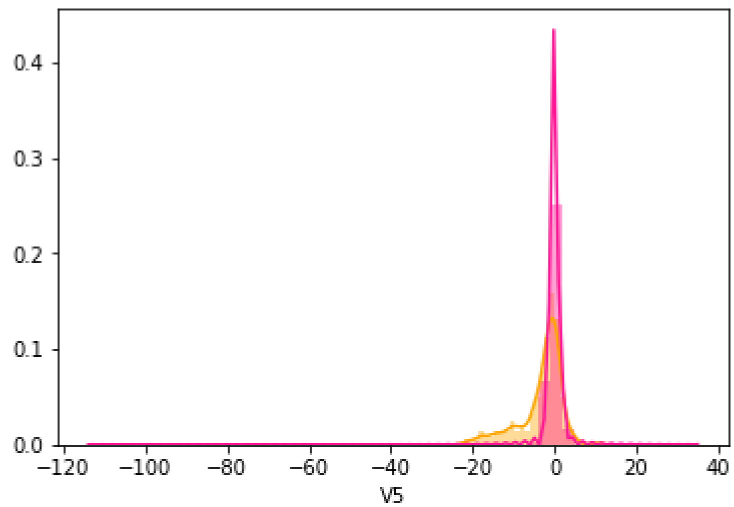

Amount
Class
Time
V1
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V2
V20
V21
V22
V23
V24
V25
V26
V27
V28
V3
V4
V5
V6
V7
V8
V9

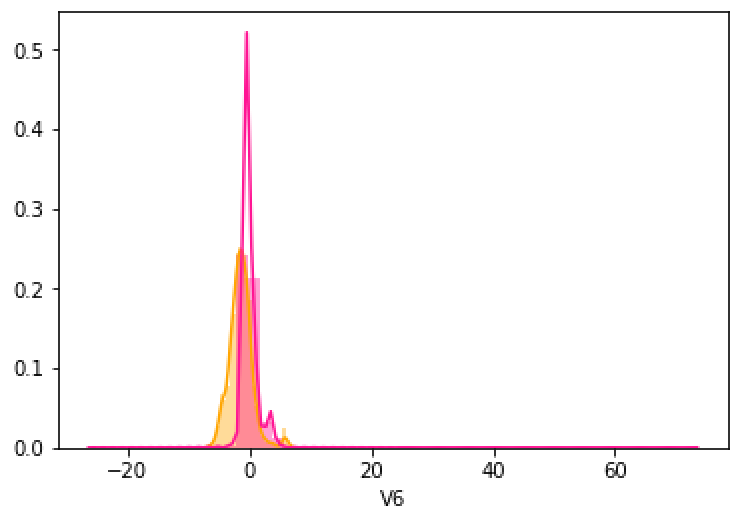

V1

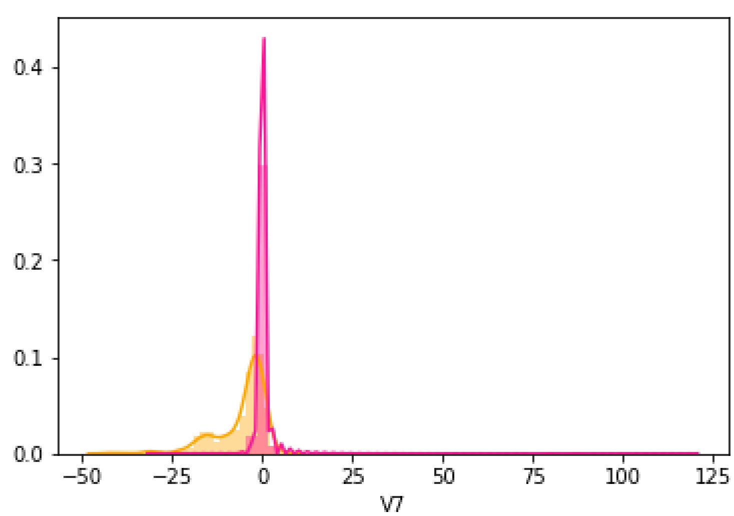

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
V2

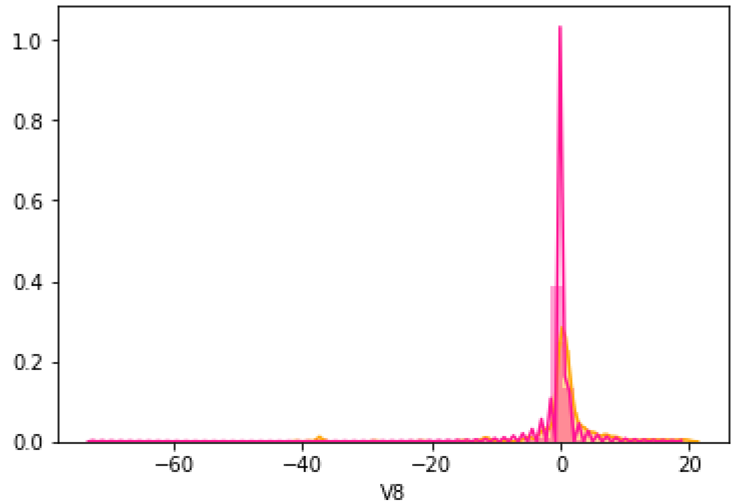

0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
10
V3

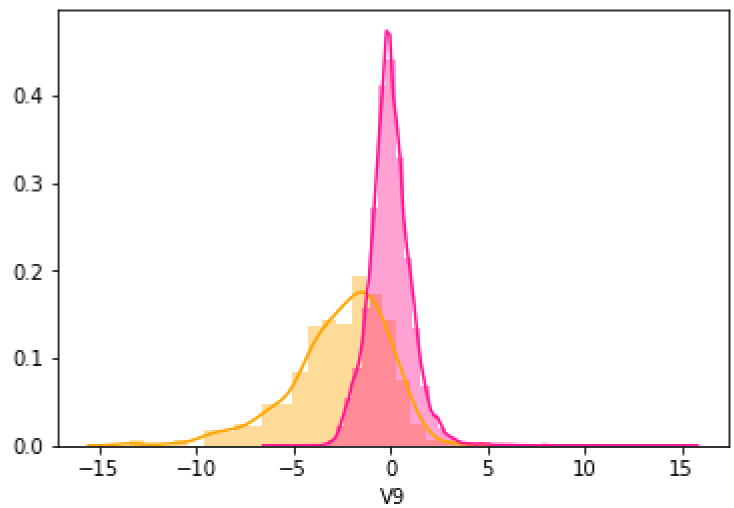

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
V4

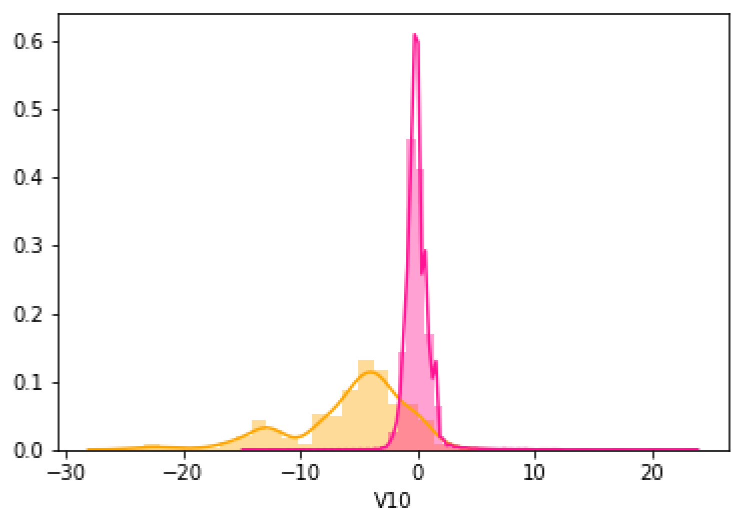

0.4
0.3
0.2
0.1
0.0
−120
−100
−80
−60
−40
−20
0
20
40
V5

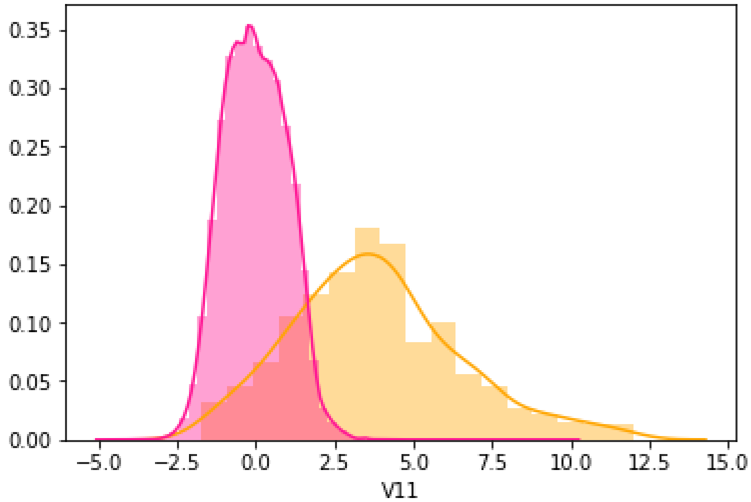

0.5
0.4
0.3
0.2
0.1
0.0
−20
0
20
40
60
V6

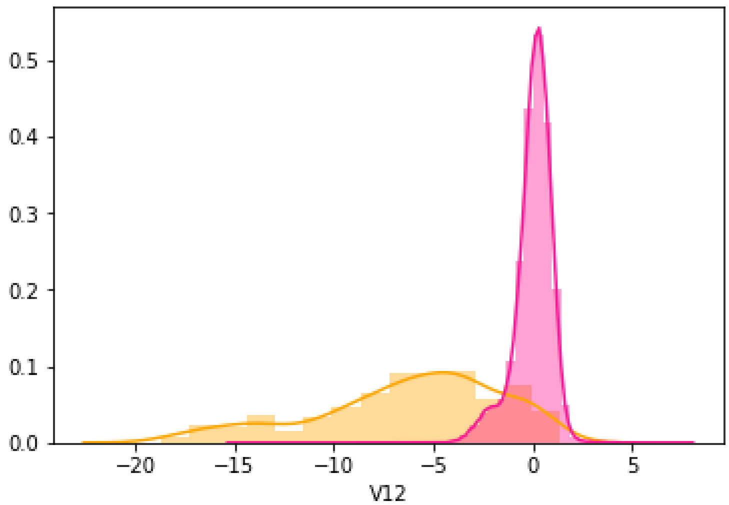

0.4
0.3
0.2
0.1
0.0
−50
−25
0
25
50
75
100
125
V7

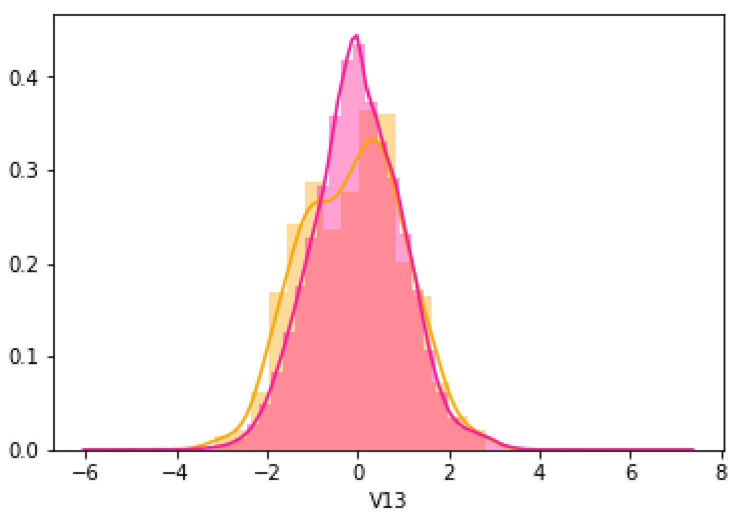

V8

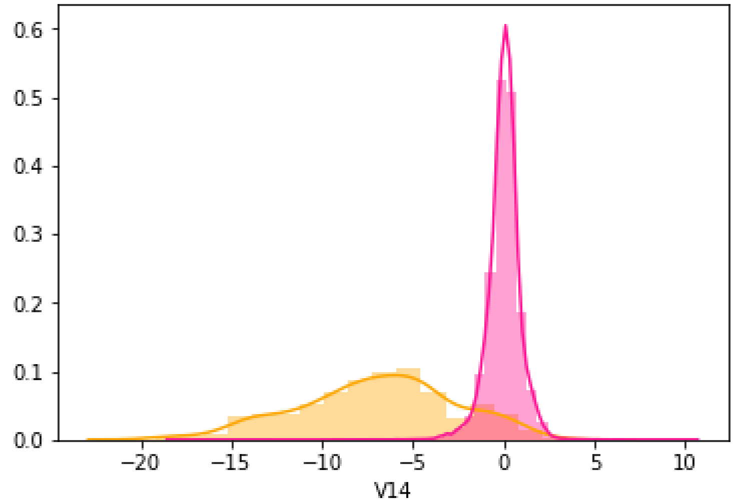

V9

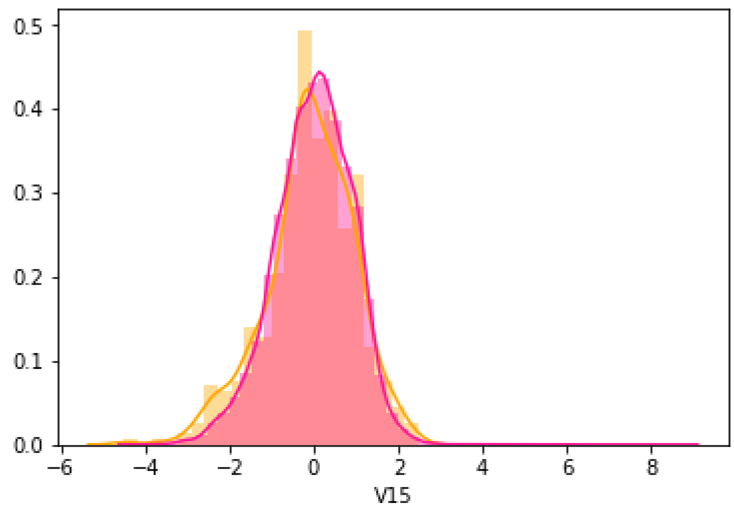

V10

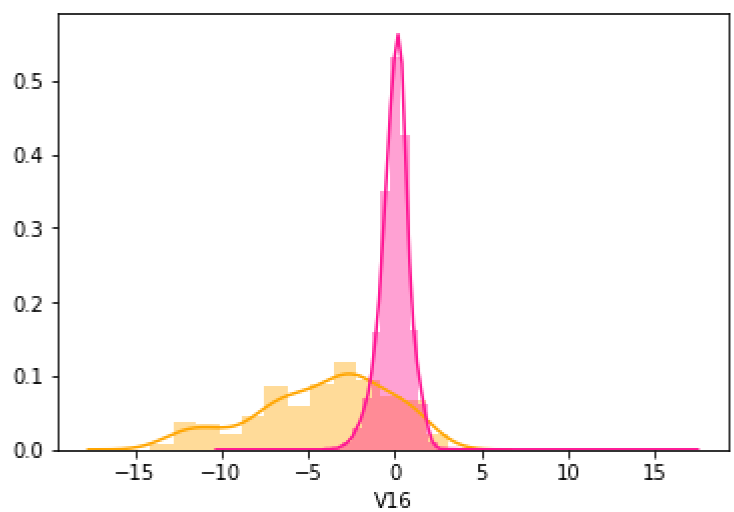

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
V11

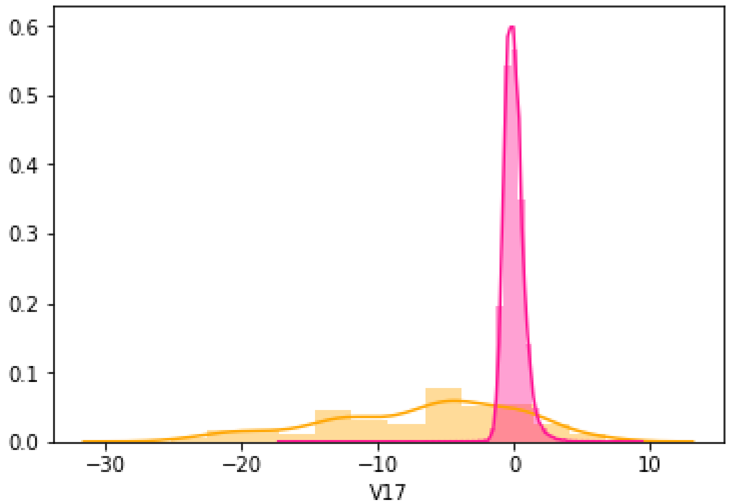

0.5
0.4
0.3
0.2
0.1
0.0
−20
−15
−10
−5
0
5
V12

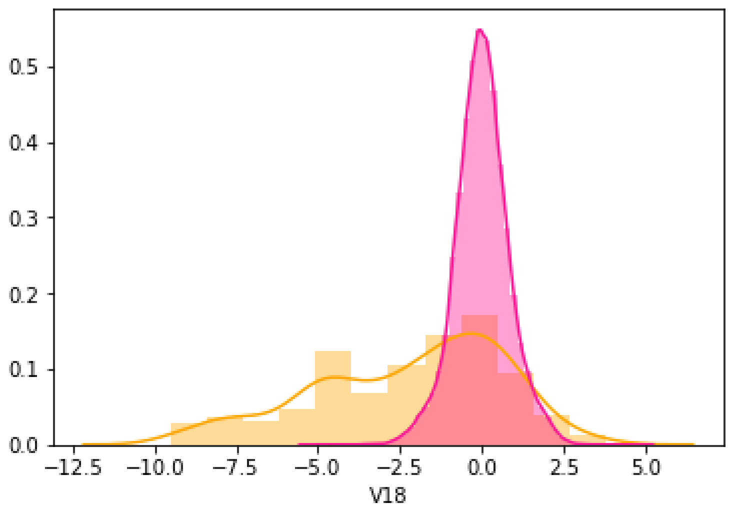

0.4
0.3
0.2
0.1
0.0
−6
−4
−2
0
2
4
6
8
V13

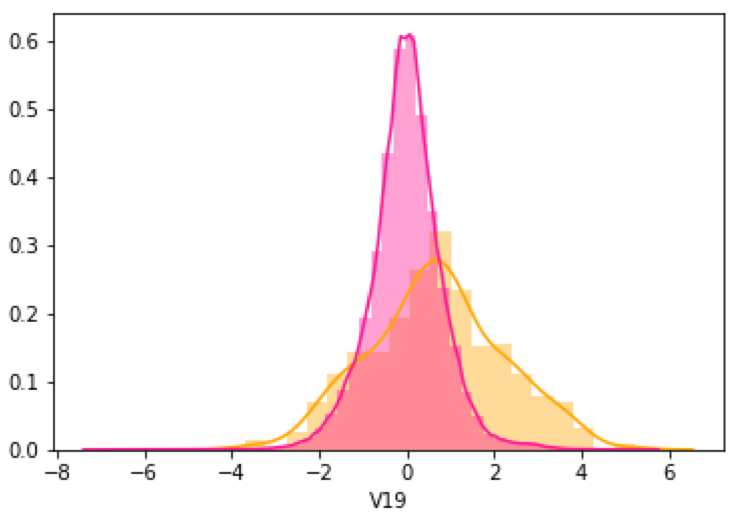

V14

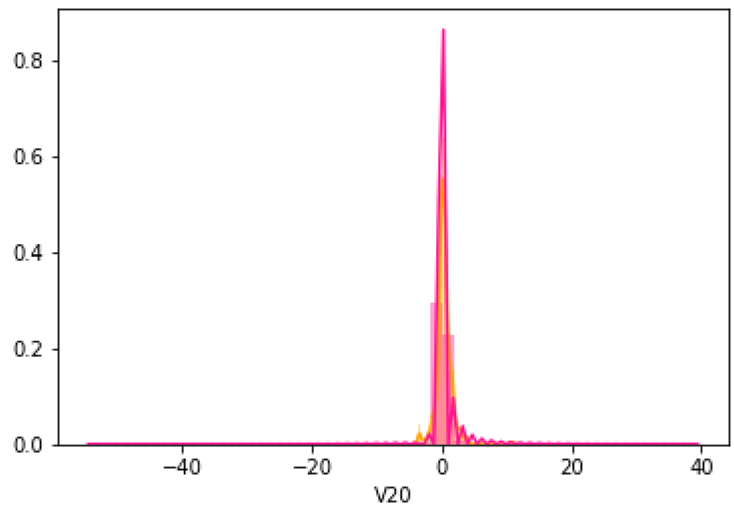

V15

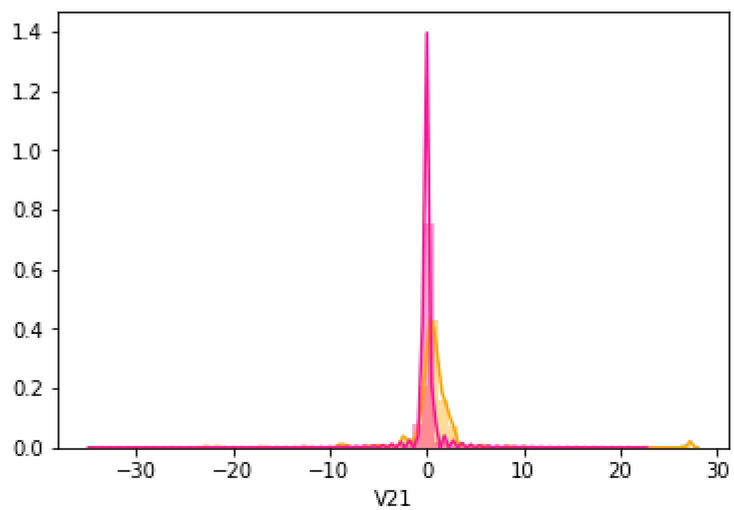

V16

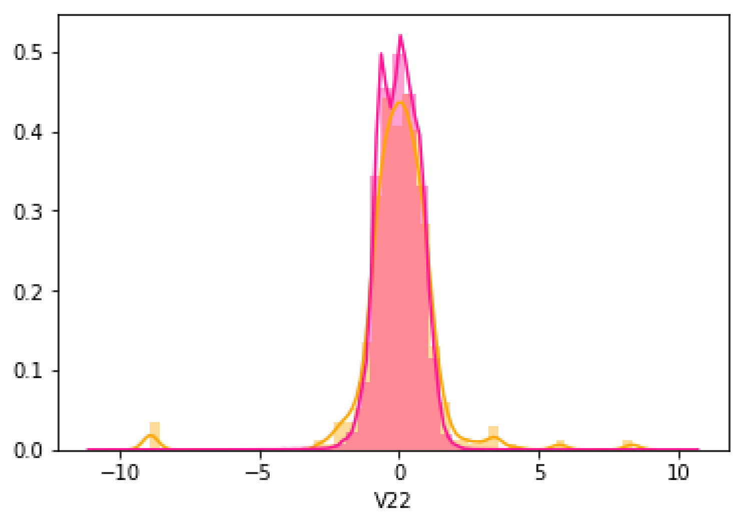

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−30
−20
−10
0
10
V17

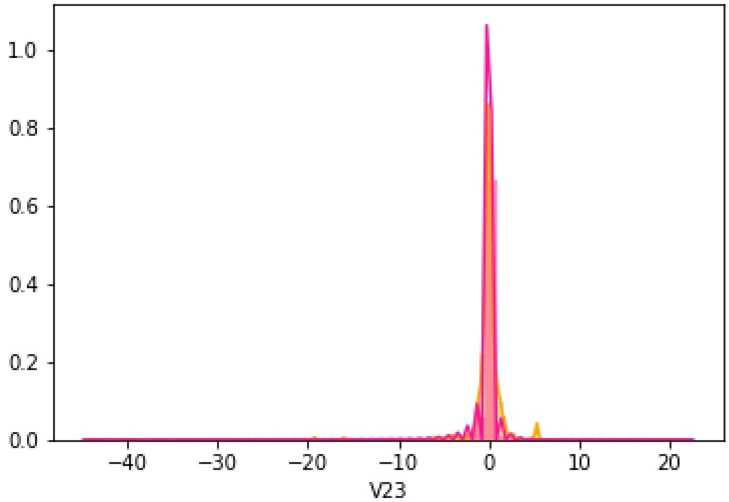

0.5
0.4
0.3
0.2
0.1
0.0
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
V18

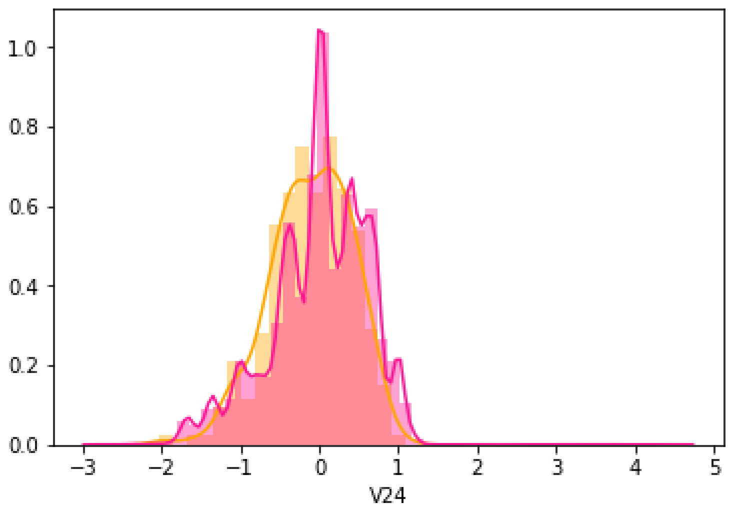

0.6
0.5
0.4
0.3
0.2
0.1
0.0
−8
−6
−4
−2
0
2
4
6
V19

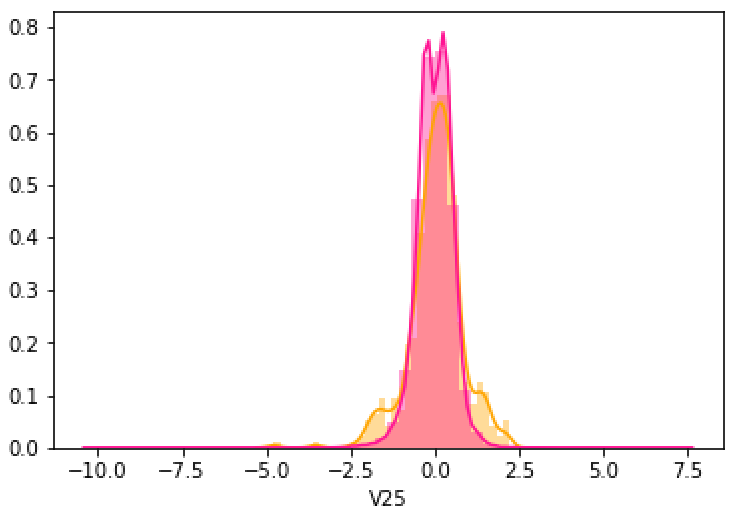

0.8
0.6
0.4
0.2
0.0
−40
−20
0
20
40
V20

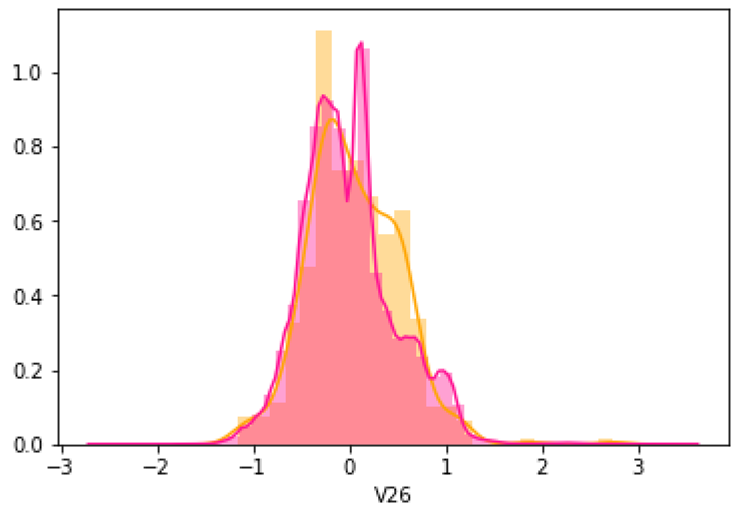

1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−30
−20
−10
0
10
20
30
V21

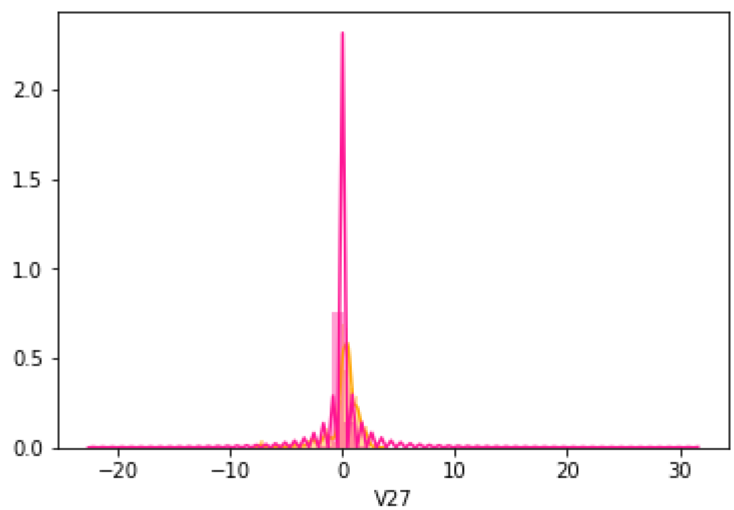

0.5
0.4
0.3
0.2
0.1
0.0
−10
−5
0
5
10
V22

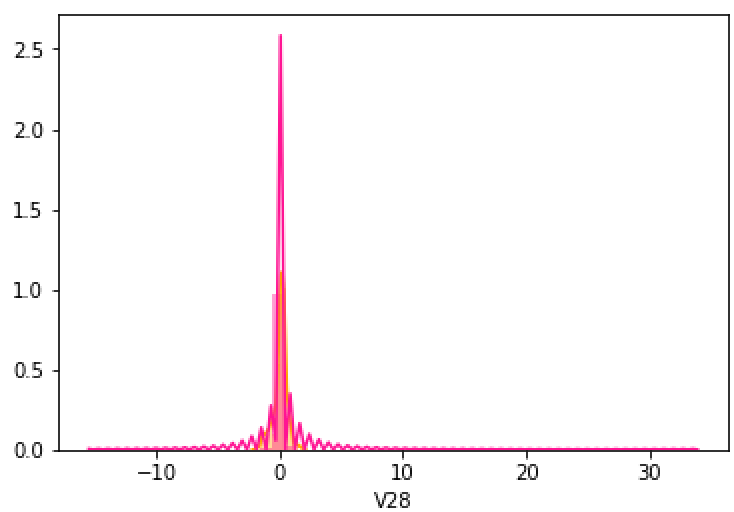

1.0
0.8
0.6
0.4
0.2
0.0
−40
−30
−20
−10
0
10
20
V23


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
7.5
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

4- Prepare Train and Test Data

```
Total data size is : 284807
Training data size is 65.0% : 185125
Test data size is 35.0% : 99682
```

5- Starting Logistic Regression

5A- Evaluate Logistic Regression with Precision, Recall, F1-score in Classification Report

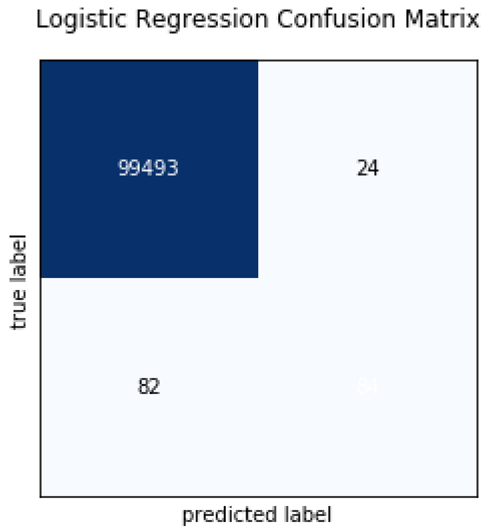
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     99517
           1       0.78      0.51      0.61       166

   micro avg       1.00      1.00      1.00     99683
   macro avg       0.89      0.75      0.81     99683
weighted avg       1.00      1.00      1.00     99683
```

5B- Logistic Regression Intercepts and Coefficients

```
  Intercepts:  [-1.7507512653723882]
  Coeficiencts:  [array([-6.55814048e-05,  3.85843259e-01, -4.87791509e-01, -
7.55022432e-01,
        7.25900953e-02,  1.91746326e-02, -2.54983649e-02,  3.79613067e-01,
       -4.23831047e-01, -3.61316369e-01, -1.58384794e-01, -3.52457959e-01,
        5.60523882e-02, -3.05708087e-01, -6.03302238e-01, -4.74655873e-01,
       -3.12654611e-01, -4.64275413e-01,  3.60960009e-03,  1.56784530e-02,
        1.00357905e-01,  2.95866206e-01,  3.67637831e-01,  2.12791956e-02,
       -2.50347351e-02, -3.96639182e-01,  4.91415090e-02, -1.07935305e-01,
        3.72367086e-02, -4.41222979e-03])]
```

5C- Logistic Regression Confusion Matrix

```
[[99493    24]
 [   82    84]]
```



5D- Logistic Regression Accuracy

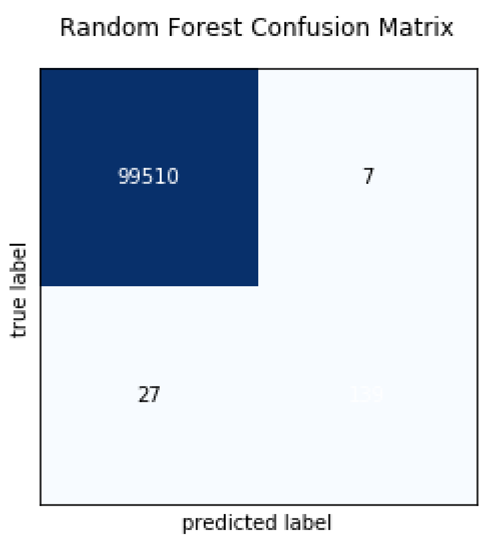
```
The Logistic Regression Accuracy Score is 0.9989366291142923
```

Note: While Logistic Regression Accuracy score is high, its Confusion Matrix is showing incorrect (false positives+false negatives) predictions. We should try other models like Random Forest to compare.

6- Starting Random Forest Analysis

6A- Random Forest Confusion Matrix

```
[[99510     7]
 [   27   139]]
```



6B- Random Forest Accuracy

The Random Forest Accuracy Score is 0.9996589187725089

Note: While Accuracy score of both Random Forest and Logistic Regression are high, Random Forest Confusion Matrix is showing more correct predictions and less incorrect (false positives+false negatives) predictions than Logistics Regression confusion matrix

In [ ]: