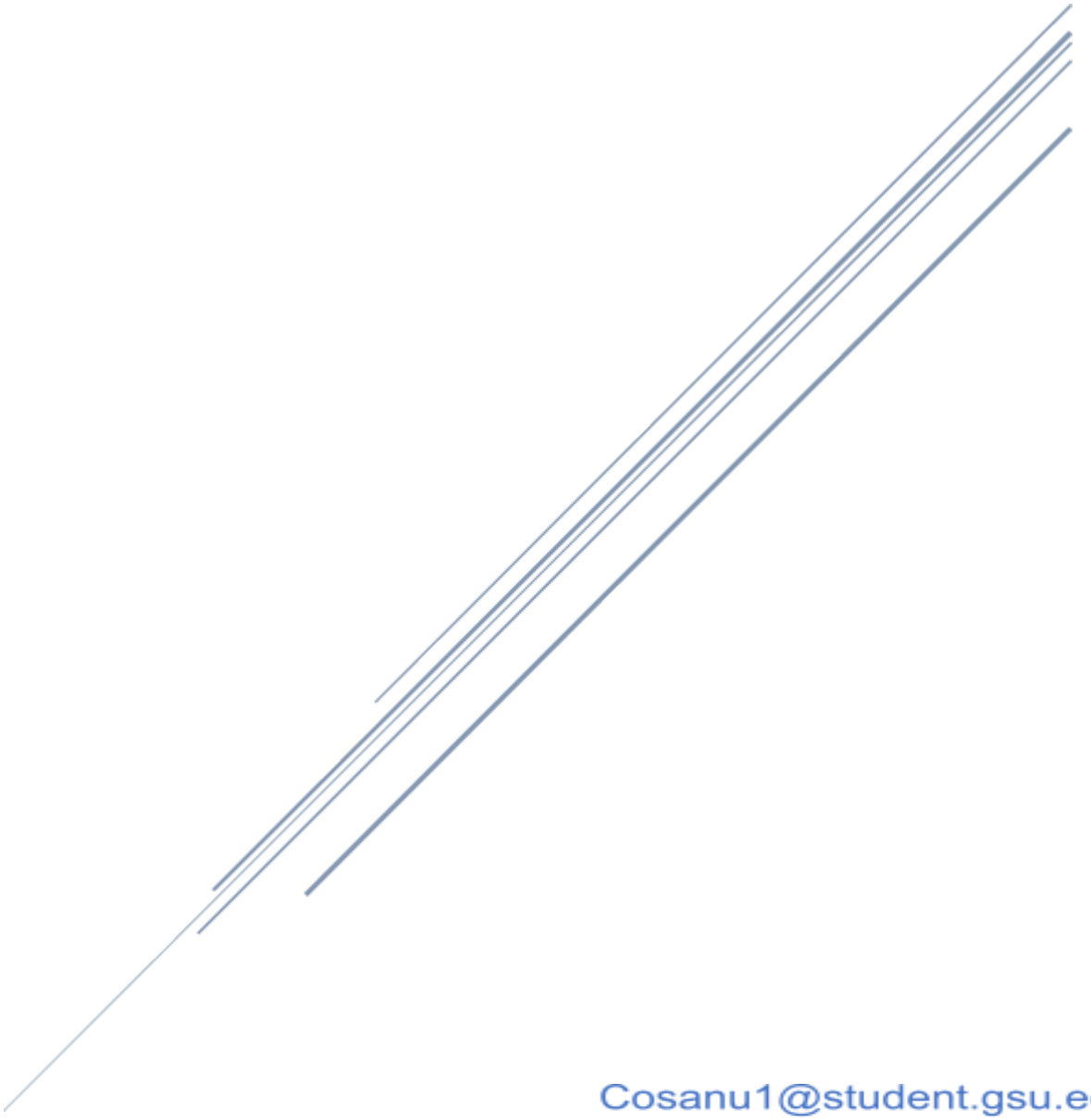
[CREDIT FRAUD DETECTION]

[Dataset detection project with Machine Learning]

By Chukwudi Osanu



Cosanu1@student.gsu.edu
[Georgia State University]

The Case.

A credit Card Company has reached out and hired a team to create a model that detects potential cases that are suspectable of credit card fraud and to ensure that those charges do not go through. A dataset was provided and organized into multiple categories, including credit card numbers, jobs, customers' names, time, location, and of course, if the transaction was fraud or not. The ultimate goal is to build a classification model that best distinguishes fraud and non-fraud transactions.

The Breakdown.

The classification model was chosen because it can predict a binary variable and give us a simplified yes or no answer/ grouped numbers in one section or another. The steps in creating the models include the following:

1. Importing the dataset and then importing packages (will be mentioned throughout the report and cited at the end) for the given dataset.
2. Cleaned the data by dropping the null value rows, resetting the index, and then dropping columns deemed unusable.
3. I have created a simple visualization of different graphs to see transactions and fraud tendencies.
4. I broke down the cases and scaled the numeric tables.
5. Used the newly scaled data within our packages to feature and split the data.
6. Built and evaluated three types of classification models
7. Created a confusion matrix model and tested it with a test dataset.

The Data.

This project primarily consisted of pandas to work with the dataset, NumPy to handle arrays, and to scale; scikit-learn helped split the data and build our models. Finally, we used matplotlib and seaborn for data visualization purposes. Below will be a screen grab of the following information presented:

```
[1]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt#visualization
     import seaborn as sns #visualization
     from sklearn.model_selection import train_test_split #splitting data
     from sklearn.neighbors import KNeighborsClassifier #Classification model
     from sklearn.linear_model import LogisticRegression #Classification model
     from sklearn.preprocessing import StandardScaler #scaling data
     from sklearn.ensemble import RandomForestClassifier #Classification model
     from sklearn.metrics import accuracy_score #Scoring the model
     from sklearn.metrics import f1_score #Scoring the model
     from sklearn.metrics import precision_score, recall_score #Scoring the model
     from sklearn.metrics import matthews_corrcoef #Scoring the model
     from sklearn.metrics import confusion_matrix #creates matrix model
```

After the dataset was retrieved from Kaggle, referred to as Credit Card Transaction (Dataset is here) and imported through pandas function. It contained over 22 Columns with an object and numeric values; however, this project only took advantage of the numeric values, the category, and state columns

Implementation:

```
[2]  FraudDF = pd.read_csv("fraudTrain.csv")
```

S

```
[3]  FraudDF.head(5)
```

Output:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	.
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	

5 rows × 23 columns

lat	long	city_pop	job	dob	trans_num	unix_time	merch_lat	merch_long	is_fraud
36.0788	-81.1781	3495	Psychologist, counselling	1988-03-09	0b242abb623afc578575680df30655b9	1.325376e+09	36.011293	-82.048315	0.0
48.8878	-118.2105	149	Special educational needs teacher	1978-06-21	1f76529f8574734946361c461b024d99	1.325376e+09	49.159047	-118.186462	0.0
42.1808	-112.2620	4154	Nature conservation officer	1962-01-19	a1a22d70485983eac12b5b88dad1cf95	1.325376e+09	43.150704	-112.154481	0.0
46.2306	-112.1138	1939	Patent attorney	1967-01-12	6b849c168bdad6f867558c3793159a81	1.325376e+09	47.034331	-112.561071	0.0
38.4207	-79.4629	99	Dance movement psychotherapist	1986-03-28	a41d7549acf90789359a9aa5346dcb46	1.325376e+09	38.674999	-78.632459	0.0

The next step was to clear and drop unnecessary data and create some basic data visualizations to get a good idea of what the dataset consisted of. A few graphs were created just to see what kind of data was in our set, any consistencies to look out for, where the transactions were coming from, and any other valuable information that could help evaluate at the end. Below is the cleaned function and then some of the graphs used:

S

Implementation:

```
[4]  FraudDF = FraudDF.replace([np.inf, -np.inf], np.nan)
     FraudDF = FraudDF.dropna()
     FraudDF = FraudDF.reset_index()
```

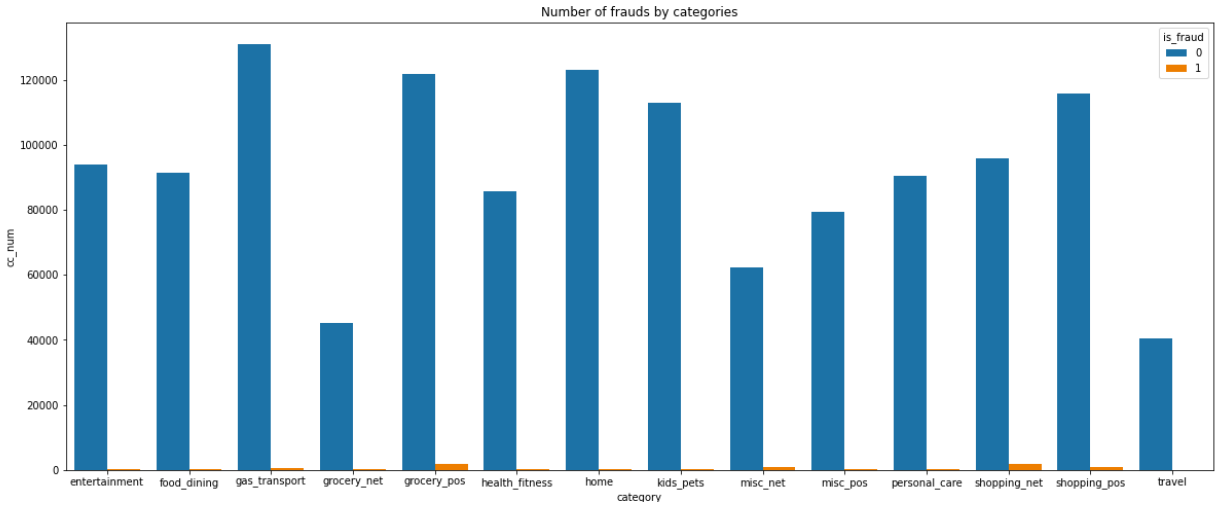
```
[5]  FraudDF.drop(['index','Unnamed: 0'],axis='columns',inplace=True)
```

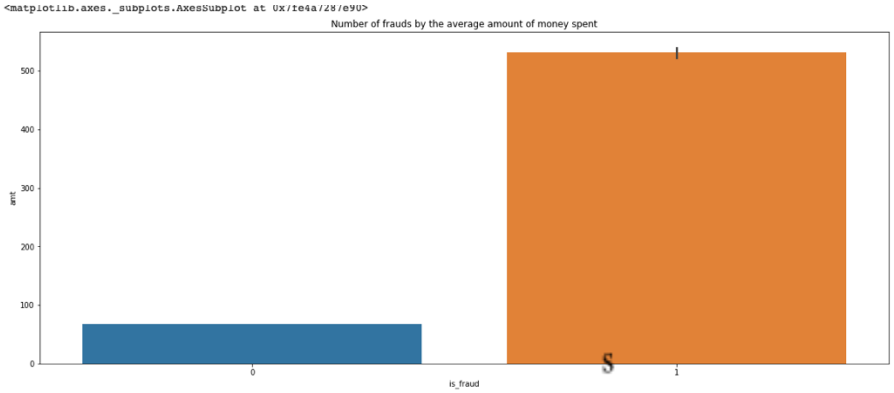
Implementation:

```
#gives the number of frauds by categories
plt.figure(figsize=(20,8))
plt.title('Number of frauds by categories')
sns.barplot(x="category", y='cc_num' ,hue="is_fraud", data=FraudDF.groupby(['category','is_fraud']).agg({'cc_num':'count'}).reset_index())
```

```
✓ [8]  #Average money spent on fraud is way higher
8s     plt.figure(figsize=(20,8))
       plt.title('Number of frauds by the average amount of money spent')
       sns.barplot(x="is_fraud", y='amt', data=FraudDF)
```

Output:





This doesn't cover every graph used, but a good indicator that 1. Groceries and online shopping are the most relevant categories with potential fraud cases. 2. The average amount of money spent between non-fraud and fraud cases show that fraud purchases are substantially more expensive than non-fraud. Finally, before getting into the more technical side of things, the data was separated into total cases, non-fraud cases, and fraud cases. Below is the data after completing the task:

Implementation:

```
[12] #does a count of all the cases in the DF
     cases = len(FraudDF['is_fraud'])
     #does a count of non fraud cases
     nonfraud_count = len(FraudDF[FraudDF['is_fraud'] == 0])
     #count of fraud cases
     fraud_count = len(FraudDF[FraudDF['is_fraud'] == 1])
     #gives the percentage
     fraud_percentage = round(fraud_count/nonfraud_count*100, 2)
```

Output:

```
There are: 1296675 Cases
There are: 1289169 Non fraud cases found
There are: 7506 Fraud cases found
0.58 % is the percentage of fraud cases
```

As stated from the graphics, there are 1,296,675 Cases, and out of these cases, only 7506 are considered fraud. So with only .58 percent of the total dataset being a fraud, this dataset is exceptionally imbalanced and must be dealt with carefully and consistently while creating the classification models. From here on out, we will only use the relevant numeric columns and drop the following: cc_num, name, street, trans_num, city, zip, job, dob, unix_time. Also, we will normalize the following data: category, gender, state

Train Test splits

To create the models first, the data must be featured and data split. However, before doing this, the data must be normalized and scaled due to the wide range of data in the set and the string columns relevant to the dataset. It will make this more accessible and accurate to formalize the models. Below will be the dataset being normalized and scaled. (Each numeric column was mounted, so for the sake of an example, only one column will be shown.):

Implementation:

```
def mapping(data,feature):
    featureMap=dict()
    count=0
    for i in sorted(data[feature].unique(),reverse=True):
        featureMap[i]=count
        count=count+1
    data[feature]=data[feature].map(featureMap)
    return data
```

Output:

gender	state
1	23
1	3
0	37
0	24
0	5

Implementation:

```
#scaled it due to the varriables being too big
sc = StandardScaler()
amount = FraudDF['amt'].values
FraudDF['amt'] = sc.fit_transform(amount.reshape(-1, 1))
```

Output:

amt
-0.407826
0.230039
0.934149

Output:

	category	amt	gender	state	lat	long	city_pop	merch_lat	merch_long	is_fraud
0	-0.452853	-0.407826	0.909206	-0.022519	-0.484420	0.657620	-0.282589	-0.494354	0.593864	0
1	0.569266	0.230039	0.909206	-1.418098	2.039120	-2.033870	-0.293670	2.078699	-2.030341	0
2	1.591384	0.934149	-1.099861	0.954386	0.717754	-1.601537	-0.280406	0.902849	-1.592323	0

After scaling the data, two variables were created, X is considered the independent variable, and the dependent variable Y was defined and split into a training set and testing set. X was consistent with every numeric value except the column is_fraud, and Y was constant of only the column is_fraud. Once implemented, there should be four different functions to call for: X_train, X_test, y_train, and y_test. After splitting it with the train_test_split algorithm, it is good to use for classification modeling. The below graphic is how it was put into python as an output of a sample:

Implementation:

```
[18] X = FraudDF.drop(['trans_date_trans_time','merchant','category','first','last','gender','street','city','state','job','dob','trans_num','is_fraud'], axis = 1).values
     y = FraudDF['is_fraud'].values
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

Output:

```
X_train samples :  [[-0.31416446 -0.02969782  1.87862754 ...  0.39917188  1.44809085
  -2.15024602]
 [ 3.34664219 -0.42834805 -0.05018631 ...  0.82559781  0.16655098
   0.33975716]
 [-0.31700897 -0.14241273 -1.44131802 ...  0.7132644   0.64254505
   1.20125905]
 ...
 [-0.31873468 -0.07616855  0.99100579 ...  0.75767238 -0.98298082
  -0.41884021]
 [-0.31846895 -0.25044937  0.19905874 ... -1.4085208   1.07017525
   0.0825144 ]
 [-0.31604773 -0.06425458  1.0365935  ... -0.86105056 -1.23346742
  -0.42697711]]
```

The Modeling.

Due to the constraints placed on this specific project, it was decided to use three different classification models for this dataset. K-Nearest Neighbors, Logistic Regression, and Random Forrest Classifier. A brief explanation of the three will be added with images of their function and how it was used on python.

1. K-Nearest Neighbors Classifier is an algorithm that stores all the cases and then reclassifies them based on similarity
    a. **Implementation:**

```
[22] kn = KNeighborsClassifier()
     kn.fit(X_train,y_train)
     kn_yhat = kn.predict(X_test)
```

2. A logistic Regression Classifier is used to predict the probability of a binary event occurring
    a. **Implementation:**

```
[23] lr = LogisticRegression()
     lr.fit(X_train, y_train)
     lr_yhat = lr.predict(X_test)
```

3. Random Forrest Classifier contains several decisions on various subsets and takes the average to improve a prediction accuracy
    a. **Implementation:**

```
[47] rf = RandomForestClassifier()
     rf.fit(X_train, y_train)
     rf_yhat = rf.predict(X_test)
```

The Evaluation.

Five metrics were chosen that were best fit to give an accurate score and reasonable evaluation of the dataset. The five metrics below are Accuracy, Precision, Recall, F1-Score, and the Matthews correlation Coefficient. Each one is explained to what its purpose is, and then at the end, implementation of each with their scores for the three classifiers.

1. Accuracy score
    I. Accuracy is the number of correctly predicted data points out of all the data points.
    II. Accuracy = CorrectPredictions/ TotalPredictions
2. Precision
    I. Precision refers to the number of true positives divided by the total number of optimistic predictions
    II. Precision = TruePositives / (TruePositives + FalsePositives)
3. Recall
    I. Recall is the number of relevant documents retrieved by a search divided by the total number of existing relevant documents
    II. Recall = TruePositives / (TruePositives + FalseNegatives)
4. F1-Score
    I. The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean.
    II. F1 score = 2((precision * recall) / (precision + recall))
5. Matthews Correlation Coefficient
    I. known as the Matthews correlation coefficient (MCC) and used as a measure of the quality of binary (two-class) classifications
    II. divide the covariance by the product of the two variables' standard deviations

These five metrics are a mathematic calculation that should print out a score based on how accurate one of the three classifiers picked should be. Using these metrics placed is the Python implementation, and then the output of the scores received with these algorithms. Here is the performance below:

Random Forest Classifier Implementation:

```
[32] n_outliers = len(is_fraud)
     n_errors = (rf_yhat != y_test).sum()
     print("The model used is Random Forest Classifier")
     acc = accuracy_score(y_test, rf_yhat)
     print("The accuracy is {}".format(acc))

     prec = precision_score(y_test, rf_yhat)
     print("The precision is {}".format(prec))

     rec = recall_score(y_test, rf_yhat)
     print("The recall is {}".format(rec))

     f1 = f1_score(y_test, rf_yhat)
     print("The F1-Score is {}".format(f1))

     MCC = matthews_corrcoef(y_test, rf_yhat)
     print("The Matthews correlation coefficient is {}".format(MCC))
```

The implementation for Kneighbor and Logistic Regression is similar. With the following three outputs, it can grade the score of the five metrics ideally; it would be best to be as close to the number 1 as possible. After a comparison can be made of the three models on which is the most reliable. The scoring output is Below:

Random Forest Classifier Output:

```
The model used is Random Forest Classifier
The accuracy is 0.9972175007480666
The precision is 0.8331099195710456
The recall is 0.6555907172995781
The F1-Score is 0.7337662337662338
The Matthews correlation coefficient is 0.7377060972422131
```

Kneighbor Classifier Output:

```
The model used is Kneighbor Classifier
The accuracy is 0.996134732192159
The precision is 0.7275300778485492
The recall is 0.5421940928270043
The F1-Score is 0.6213357509821699
The Matthews correlation coefficient is0.6262095847966577
```

Logistic Regression Output:

```
The model used is Logistic Regression Classifier
The accuracy is 0.9936236962818776
The precision is 0.0
The recall is 0.0
The F1-Score is 0.0
The Matthews correlation coefficient is-0.0017621143829963564
```

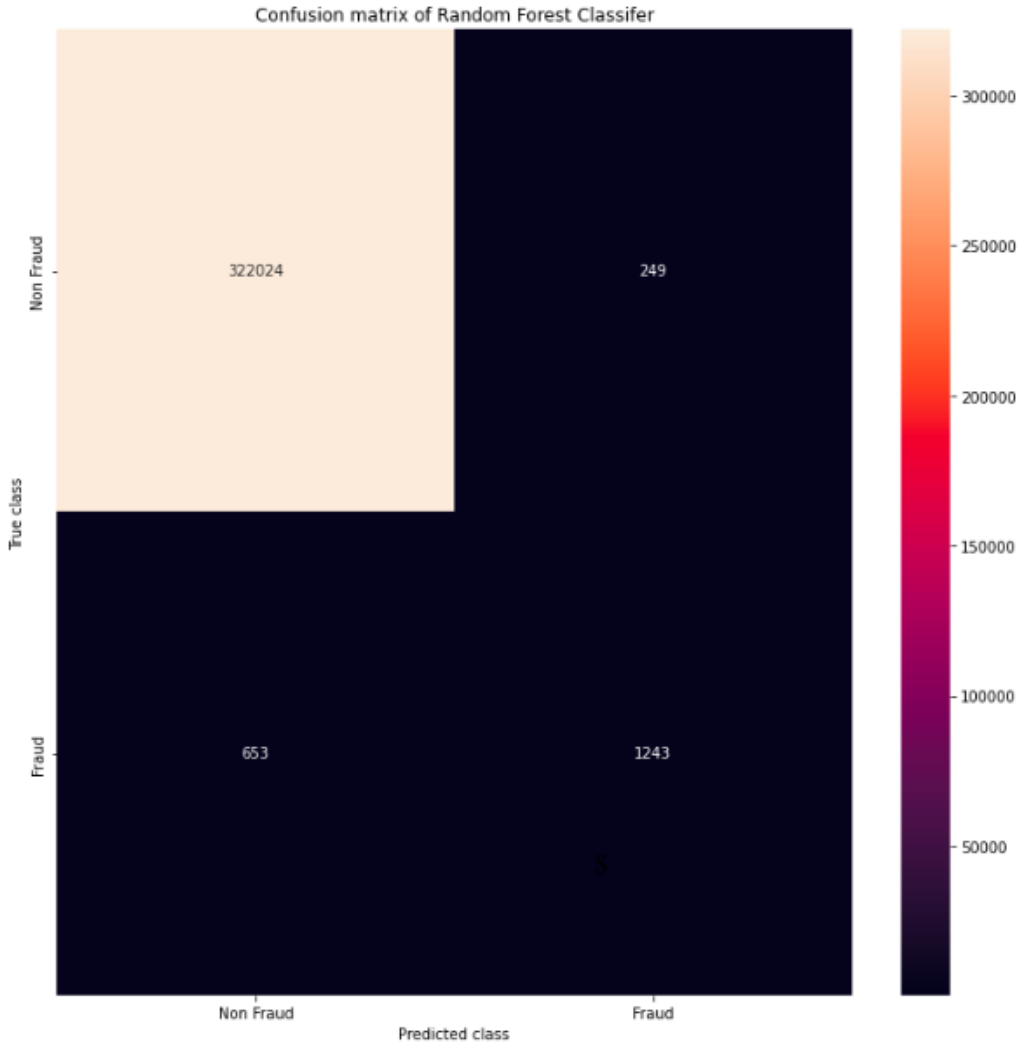
The Matrix.

Finally, the Confusion Matrix will be implemented as the final use of judgment on which classification model to use as the optimal choice. The Confusion matrix is a summary of predicted outcomes, and it is split into four classifications with a binary choice split. What is going to be looked for is an extremely high number in the top left square and a good number in the Fraud – Fraud square. Below will be the implementation of one of the models and the outcome of all three:

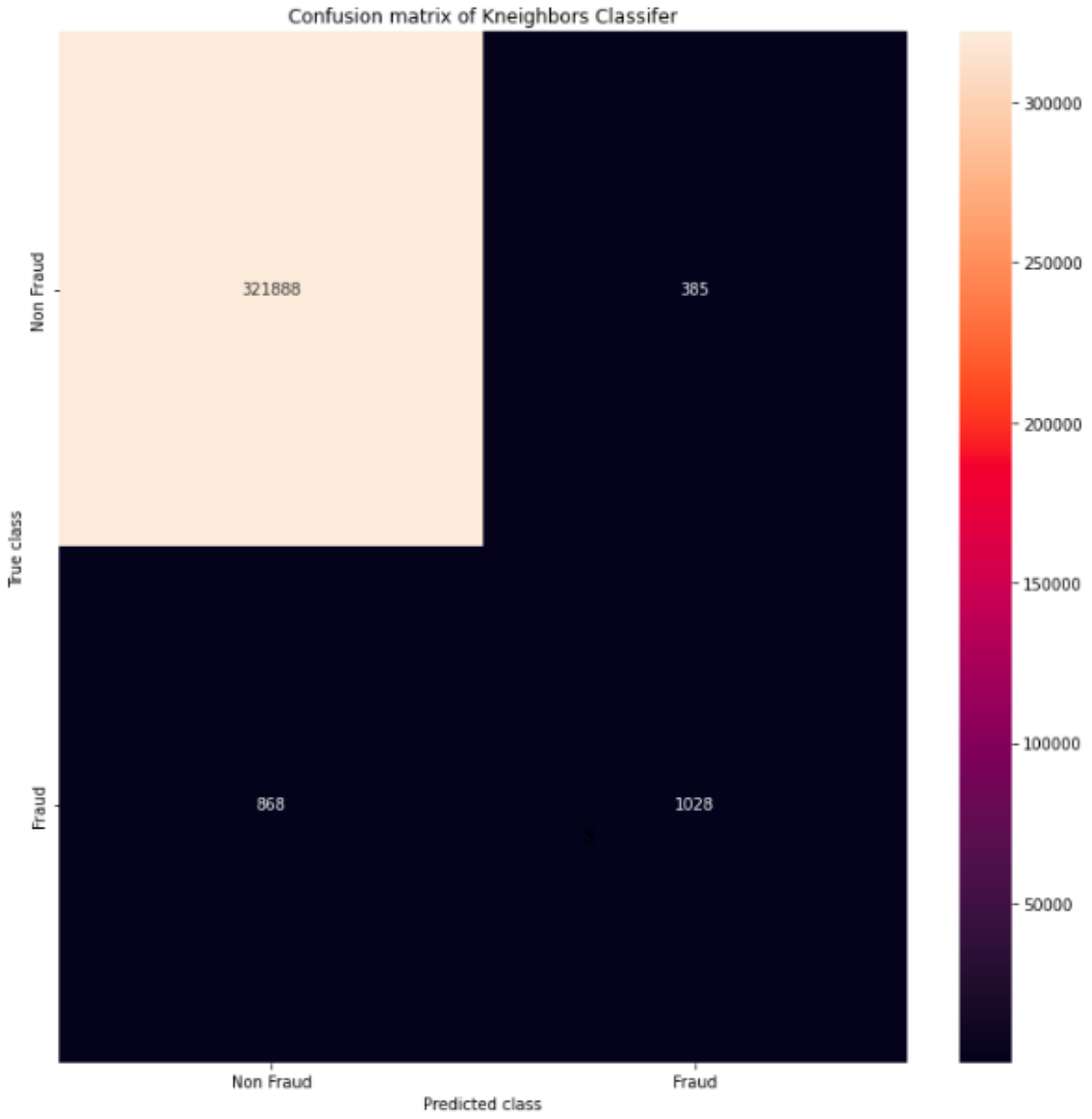
Random Forest Classifier implementation:

```
#Creating a confusion matrix with the RF yhat
LABELS = ['Non Fraud', 'Fraud']
conf_matrix = confusion_matrix(y_test, rf_yhat)
plt.figure(figsize =(12, 12))
sns.heatmap(conf_matrix, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of Random Forest Classifer")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

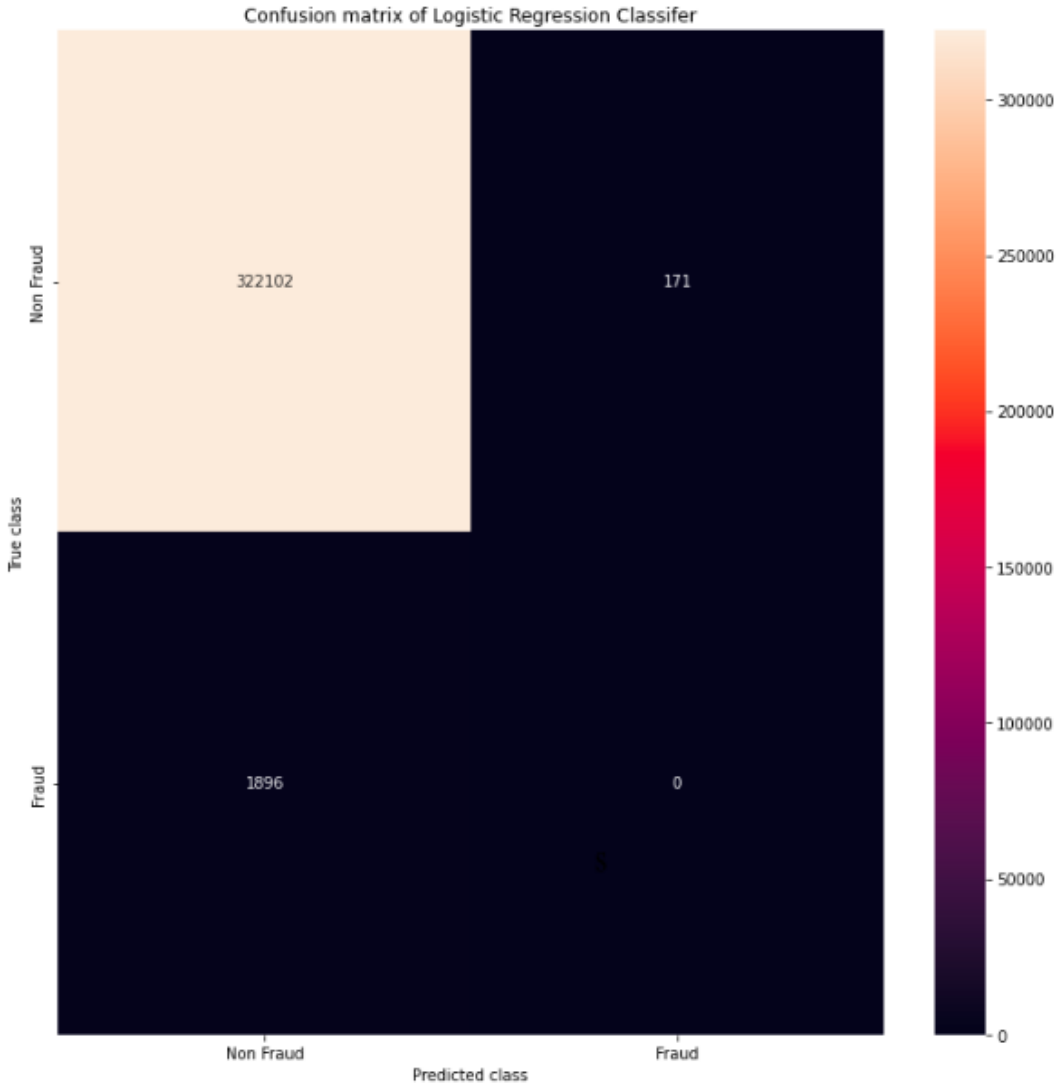
Random Forest Classifier Output:



Kneighbors Classifier Output:



Logistic Regression Output:


Confusion matrix of Logistic Regression Classifer
322102
171
1896
0
Non Fraud
Fraud
True class
Non Fraud
Fraud
Predicted class
300000
250000
200000
150000
100000
50000
0

The Decision

Based on the extensive research and breaking down three different classification models, it would be best to suggest using the Random Forest Classifier model for classifying the fraud set. While KNeighbors had good statistics and was pretty accurate in detecting cases. Overall, the matrix for Random Forest Classifier did a better job creating a distinction between non-fraud and fraud cases. The model that is suggested to stay away from was the Logistics Regression model; across the board, it had poor scores, and it straight up failed in terms of finding any fraud cases. Using a Test Set, an exact project was ran, and Random Forest Classifier was the clear choice for this project. Below will be a copy of the dataset and code.

Fraud Detection Full Code
Kaggle Dataset