CREDIT CARD FRAUD DETECTION SYSTEM

Nilesh Kumar, Pranjali Singh,
Sushmita Kumari
School of Computer Science and Engineering
Galgotias University

***Abstract*— It is critical for credit card firms to be able to recognize fraudulent credit card transactions so that customers are not charged for products they did not purchase. Data Mining, in conjunction with Machine Learning, can be used to solve such difficulties. With Credit Card Fraud Detection, this project demonstrates the modelling of a data collection using machine learning. Modeling prior credit card transactions with data from those that turned out to be fraudulent is part of the Credit Card Fraud Detection Problem.**

The model is then used to determine whether or not a new transaction is fraudulent. Our goal is to detect 100 percent of fraudulent transactions while reducing the number of inaccurate fraud classifications. A classic example of classification is credit card fraud detection. We concentrated on analyzing and pre-processing data sets, as well as deploying numerous anomaly detection algorithms, during this process.

I. INTRODUCTION

A credit card is a handy card that carries identity information such as a signature or a photograph and permits the person listed on it to charge goods or services to his account, for which he will be invoiced on a regular basis. Today, automated teller machines (ATMs), store readers, banks, and online internet banking systems all read the information on the card.

They have a one-of-a-kind card number, which is crucial. Its safety is dependent on both the physical security of the plastic card and the confidentiality of the credit card number.

The quantity of credit card transactions is rapidly increasing, which has resulted in a significant increase in fraudulent activity. Theft and fraud performed with a credit card as a fraudulent source of funds in a transaction are referred to as credit card fraud.

To handle this fraud detection problem, statistical methodologies and a variety of data mining algorithms are commonly used. Artificial intelligence, Meta learning, and pattern matching are used in the majority of credit card fraud detection systems. Genetic algorithms are evolutionary algorithms that try to find better solutions to fraud detection. The development of an efficient and secure electronic payment system to determine whether a transaction is fraudulent or not is given top priority. In this paper, we'll look at credit card fraud and how to detect it. A credit card fraud occurs when someone uses another person's card for their own personal usage without the owner's knowledge. When fraudsters use it in such instances, it is used until its entire usable limit is spent. As a result, we want a solution that reduces the overall available credit card limit, which is more vulnerable to fraud. Furthermore, as time passes, a Genetic algorithm creates better answers. The development of an efficient and secure electronic payment system for identifying fraud is given top priority.

II. LITERATURE REVIEW

Fraud is defined as an illegal or criminal deception intended to gain financial or personal gain. It is a purposeful act committed in violation of a law, rule, or policy with the intent of obtaining unlawful financial gain.

Several literatures on anomaly or fraud detection in this domain have previously been published and are open to the public. Data mining applications, automated fraud detection, and adversarial detection are among the strategies used in this domain, according to a comprehensive survey undertaken by Clifton Phua and his colleagues. Unconventional techniques, such as hybrid data mining/complex network classification algorithm, have shown effective on medium-sized online transactions, based on network reconstruction algorithm that allows building representations of the divergence of one instance from a reference group. Fraud detection is a difficult undertaking, and no algorithm can accurately anticipate if a transaction is fraudulent.

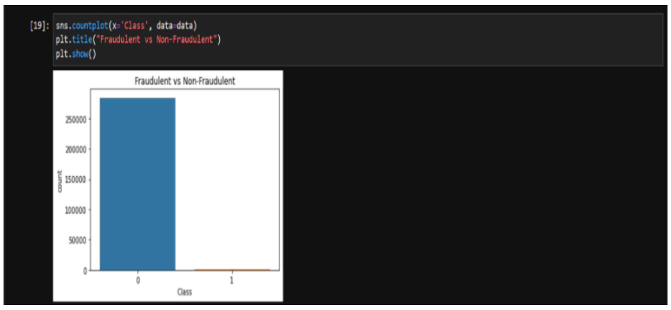
The following are characteristics of a good fraud detection system:

- The frauds should be correctly identified.
- Frauds should be detected quickly.
- A genuine transaction should not be classified as fraud.

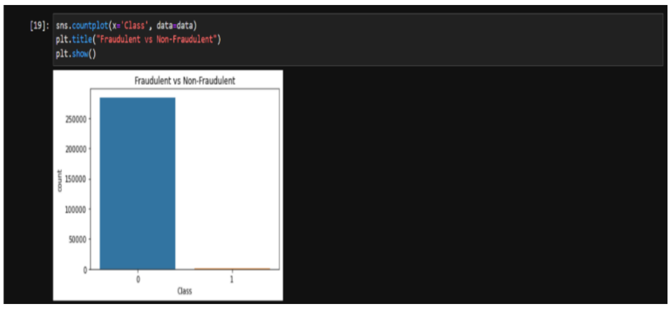
III. PROBLEM IDENTIFICATION

The aim of the project is to predict fraudulent credit card transactions using machine learning models. This is crucial from the bank's as well as customer's perspective. The banks cannot afford to lose their customers' money to fraudsters. Every fraud is a loss to the bank as the bank is responsible for the fraud transactions.

The dataset contains transactions made over a period of two days in September 2013 by European credit cardholders. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. We need to take care of the data imbalance while building the model and come up with the best model by trying various algorithms.



```
[19]: sns.countplot(x='Class', data=data)
      plt.title("Fraudulent vs Non-Fraudulent")
      plt.show()
```



```
[19]: sns.countplot(x='Class', data=data)
      plt.title("Fraudulent vs Non-Fraudulent")
      plt.show()
```

IV. FUNCTIONALITY

Several machine learning and deep learning techniques have been used for the purpose to detect and prevent credit card fraud, i.e., Logistic regression, Naïve Bayes, decision trees, SVM, and random forest. This paper surveys the machine learning techniques used for fraud detection techniques used in credit card transactions by classifying them into suitable categories. A comprehensive comparison of these techniques is also made based on accuracy, precision, recall, etc. In the end, comprehensive insights.

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups such as Yes or No, 0 or 1, etc. Classes can be called as targets/labels or categories. Unlike regression, the output variable of Classification is a category, not a value. Since the Classification algorithm is a Supervised learning technique, hence it takes labelled input data, which means it contains input with the corresponding output.

Classification Algorithms

- Decision Tree Classifier

Decision Tree is a supervised learning technique that may be used to solve both classification and regression problems, however it is most commonly employed to solve classification issues.

Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.

- Logistic Regression

Logistic regression is a supervised learning technique that is used for predicting the categorical dependent variable using a given set of independent variables.

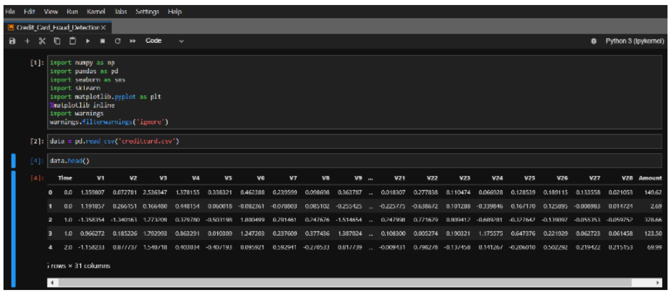
Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can either Yes or No, 0 or 1, etc. But instead of giving exact value as 0 or 1, it gives the probabilistic values which lie between 0 and 1.

- Random Forest Classifier

Random forest is a supervised machine learning algorithm that is commonly used to solve classification and regression problems. It creates decision trees from various samples, using the majority vote for classification and the average for regression. One of the most essential characteristics of the Random Forest Algorithm is that it can handle data sets with both continuous and categorical variables, as in regression and classification. For classification difficulties, it produces superior results.

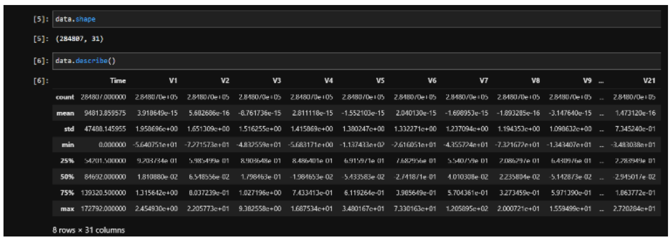
- XGBoost Classifier

XGBoost classifier is a supervised machine learning algorithm that is applied for structured and tabular data. Both types of algorithms fall under the supervised machine learning category. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost is an extreme gradient boost algorithm. And that means it's a big Machine learning algorithm with lots of parts. XGBoost works with large, complicated datasets. XGBoost is an ensemble modelling technique.

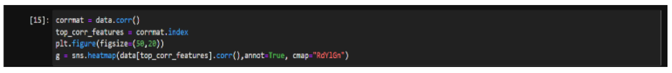


File Edit View Run Kernel Tabs Settings Help
Credit_Card_Fraud_Detection
Code
Python 3 (ipykernel)

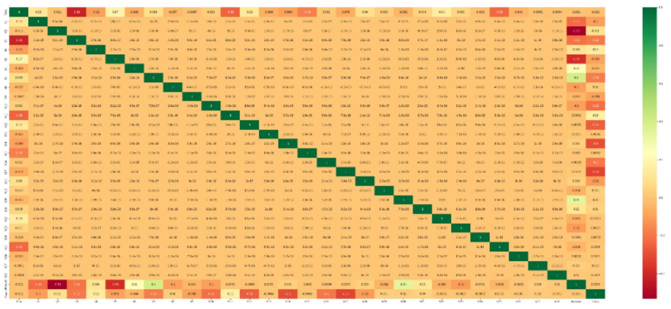
```
[1]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import sklearn
     import matplotlib.pyplot as plt
     %matplotlib inline
     import warnings
     warnings.filterwarnings('ignore')

[2]: data = pd.read_csv('creditcard.csv')

[4]: data.head()

[4]:    Time        V1        V2        V3        V4        V5        V6        V7        V8        V9  ...       V21       V22       V23       V24       V25       V26       V27       V28  Amount
     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  0.128539  0.189115  0.133558  0.021053  149.62
     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  0.167170  0.125895 -0.008983  0.014724    2.69
     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 -0.327642 -0.139097 -0.055353 -0.059752  378.66
     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  0.647376  0.221929  0.062723  0.061458  123.50
     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 -0.206010  0.502292  0.219422  0.215153   69.99

     5 rows × 31 columns
```



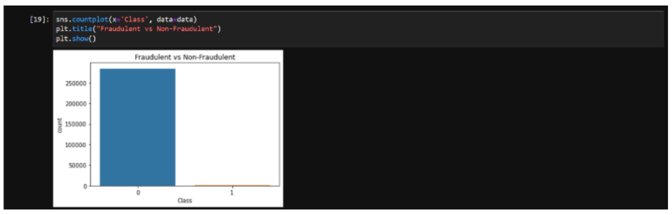
```
[5]: data.shape

[5]: (284807, 31)

[6]: data.describe()

[6]:
            Time           V1            V2            V3            V4            V5            V6            V7            V8            V9  ...           V21
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  ...  2.848070e+05
 mean   94813.859575  3.918649e-15  5.682686e-16 -8.761736e-15  2.811118e-15 -1.552103e-15  2.040130e-15 -1.698953e-15 -1.893285e-16 -3.147640e-15  ...  1.473120e-16
  std   47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00  ...  7.345240e-01
  min       0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01  ... -3.483038e+01
  25%   54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01  ... -2.283949e-01
  50%   84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02 -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02  ... -2.945017e-02
  75%  139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01  ...  1.863772e-01
  max  172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01  ...  2.720284e+01

8 rows × 31 columns
```



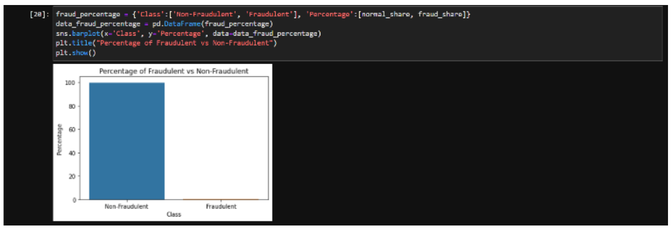
```
[15]: cormat = data.corr()
      top_corr_features = cormat.index
      plt.figure(figsize=(50,20))
      g = sns.heatmap(data[top_corr_features].corr(),annot=True, cmap="RdYlGn")
```



Some of our predictors appear to be related to the class variable, as can be shown. Despite this, there appear to be few substantial relationships for such a large number of variables.

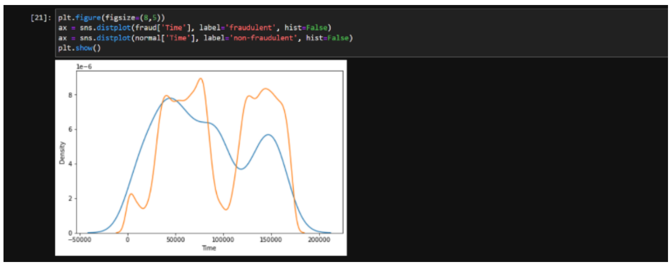


```
[19]: sns.countplot(x='Class', data=data)
      plt.title("Fraudulent vs Non-Fraudulent")
      plt.show()
```



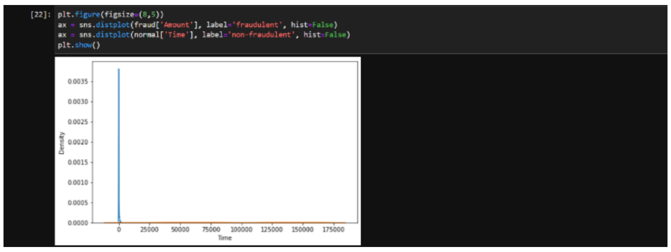
```
[20]: fraud_percentage = {'Class':['Non-Fraudulent', 'Fraudulent'], 'Percentage':[normal_share, fraud_share]}
      data_fraud_percentage = pd.DataFrame(fraud_percentage)
      sns.barplot(x='Class', y='Percentage', data=data_fraud_percentage)
      plt.title("Percentage of Fraudulent vs Non-Fraudulent")
      plt.show()
```

Distribution of classes with time :- We came to know that there is no significant relation of classes to time.

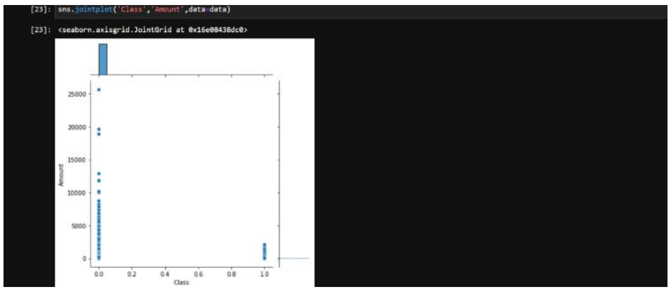


```
[21]: plt.figure(figsize=(8,5))
      ax = sns.distplot(fraud['Time'], label='fraudulent', hist=False)
      ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
      plt.show()
```

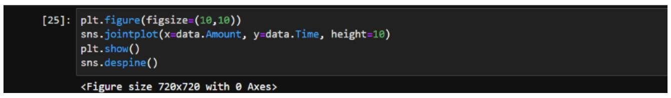
Distribution of classes with amount and time :- We came to know that fraud is maximum for only lower amount whereas for higher amount fraud is very less.



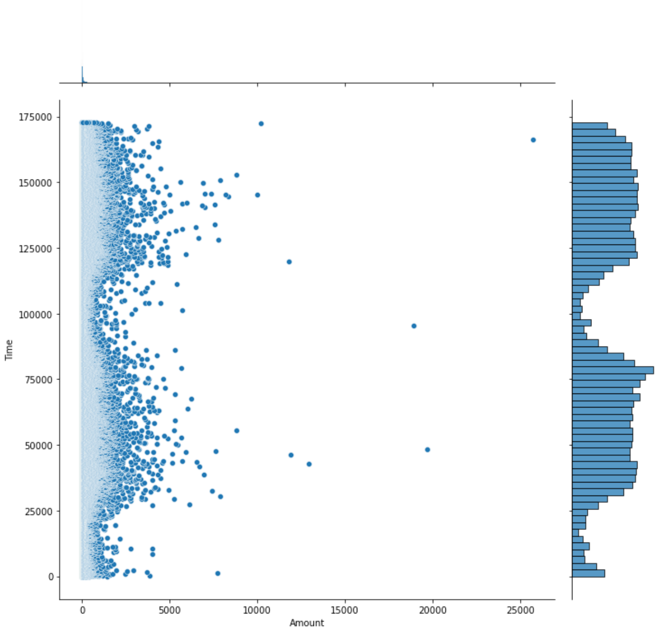
```
[22]: plt.figure(figsize=(8,5))
      ax = sns.distplot(fraud['Amount'], label='fraudulent', hist=False)
      ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
      plt.show()
```

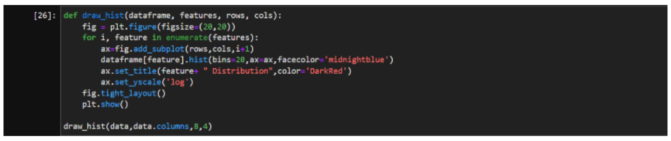


```
[23]: sns.jointplot('Class','Amount',data=data)

[23]: <seaborn.axisgrid.JointGrid at 0x16e08438dc0>
```

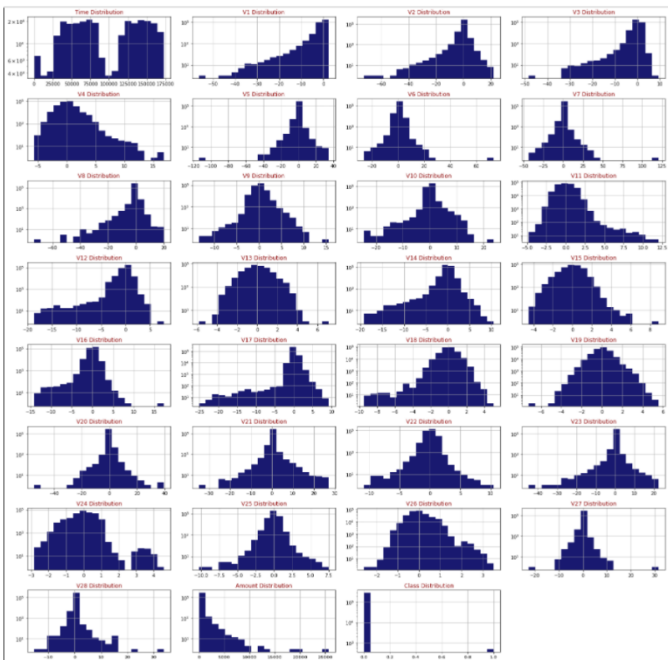


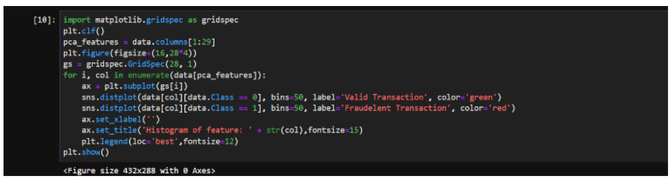
```
[25]: plt.figure(figsize=(10,10))
      sns.jointplot(x=data.Amount, y=data.Time, height=10)
      plt.show()
      sns.despine()

      <Figure size 720x720 with 0 Axes>
```





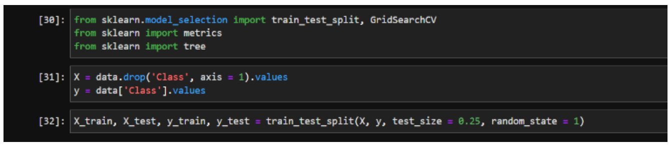
```
[26]: def draw_hist(dataframe, features, rows, cols):
          fig = plt.figure(figsize=(20,20))
          for i, feature in enumerate(features):
              ax=fig.add_subplot(rows,cols,i+1)
              dataframe[feature].hist(bins=20,ax=ax,facecolor='midnightblue')
              ax.set_title(feature+ " Distribution",color='DarkRed')
              ax.set_yscale('log')
          fig.tight_layout()
          plt.show()

      draw_hist(data,data.columns,8,4)
```





```
[10]: import matplotlib.gridspec as gridspec
      plt.clf()
      pca_features = data.columns[1:29]
      plt.figure(figsize=(16,28*4))
      gs = gridspec.GridSpec(28, 1)
      for i, col in enumerate(data[pca_features]):
          ax = plt.subplot(gs[i])
          sns.distplot(data[col][data.Class == 0], bins=50, label='Valid Transaction', color='green')
          sns.distplot(data[col][data.Class == 1], bins=50, label='Fraudelent Transaction', color='red')
          ax.set_xlabel('')
          ax.set_title('Histogram of feature: ' + str(col),fontsize=15)
          plt.legend(loc='best',fontsize=12)
      plt.show()

      <Figure size 432x288 with 0 Axes>
```

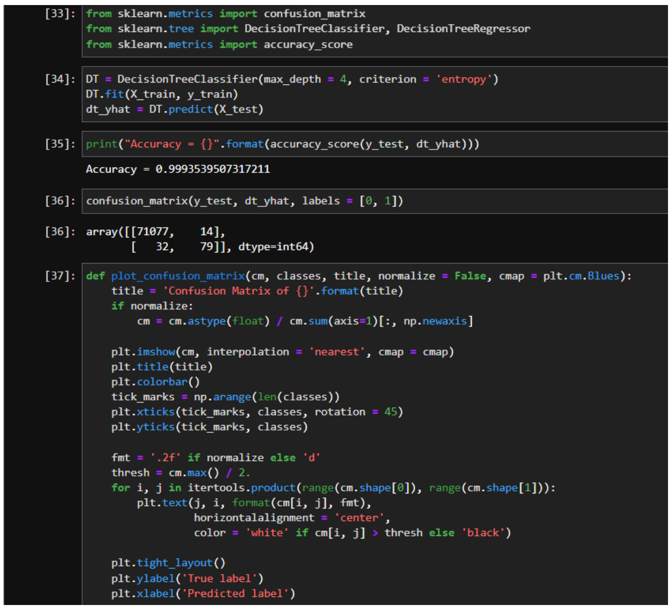
Split & Train Model



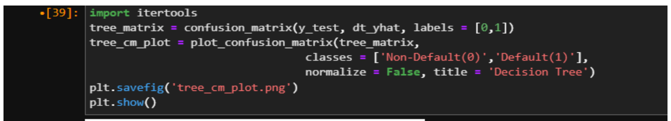
```
[30]: from sklearn.model_selection import train_test_split, GridSearchCV
      from sklearn import metrics
      from sklearn import tree

[31]: X = data.drop('Class', axis = 1).values
      y = data['Class'].values

[32]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

Confusion Matrices

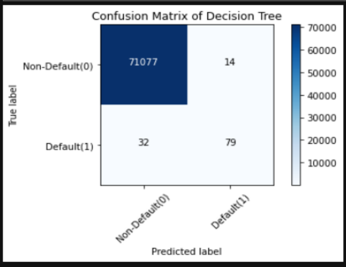
- DecisionTreeClassifier

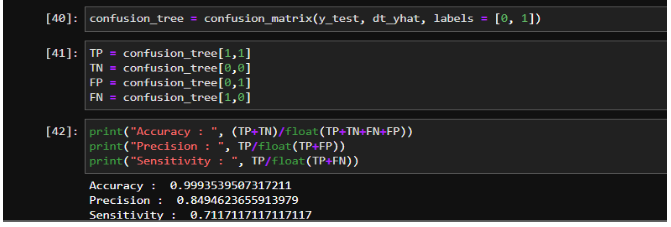


```
[33]: from sklearn.metrics import confusion_matrix
      from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
      from sklearn.metrics import accuracy_score

[34]: DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
      DT.fit(X_train, y_train)
      dt_yhat = DT.predict(X_test)

[35]: print("Accuracy = {}".format(accuracy_score(y_test, dt_yhat)))

      Accuracy = 0.9993539507317211

[36]: confusion_matrix(y_test, dt_yhat, labels = [0, 1])

[36]: array([[71077,    14],
             [   32,    79]], dtype=int64)

[37]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

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

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



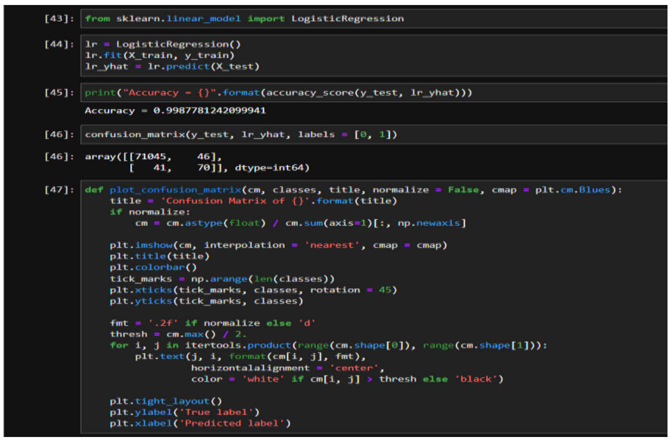
```
•[39]: import itertools
       tree_matrix = confusion_matrix(y_test, dt_yhat, labels = [0,1])
       tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                 classes = ['Non-Default(0)','Default(1)'],
                                 normalize = False, title = 'Decision Tree')
       plt.savefig('tree_cm_plot.png')
       plt.show()
```



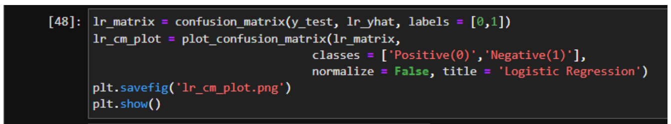


```
[40]: confusion_tree = confusion_matrix(y_test, dt_yhat, labels = [0, 1])

[41]: TP = confusion_tree[1,1]
      TN = confusion_tree[0,0]
      FP = confusion_tree[0,1]
      FN = confusion_tree[1,0]

[42]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9993539507317211
      Precision :  0.8494623655913979
      Sensitivity :  0.7117117117117117
```

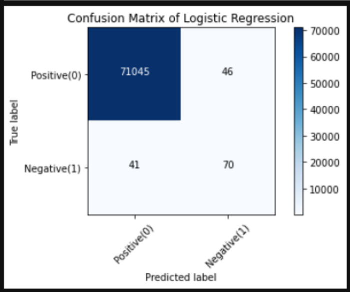
- Logistic Regression

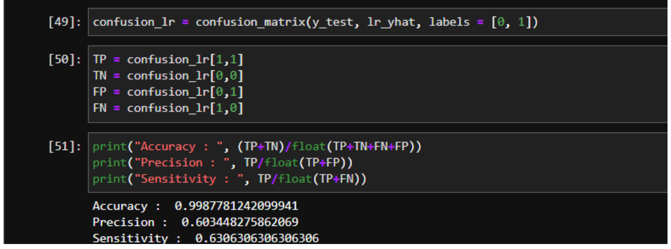


```
[43]: from sklearn.linear_model import LogisticRegression

[44]: lr = LogisticRegression()
      lr.fit(X_train, y_train)
      lr_yhat = lr.predict(X_test)

[45]: print("Accuracy = {}".format(accuracy_score(y_test, lr_yhat)))

      Accuracy = 0.9987781242099941

[46]: confusion_matrix(y_test, lr_yhat, labels = [0, 1])

[46]: array([[71045,    46],
             [   41,    70]], dtype=int64)

[47]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

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

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



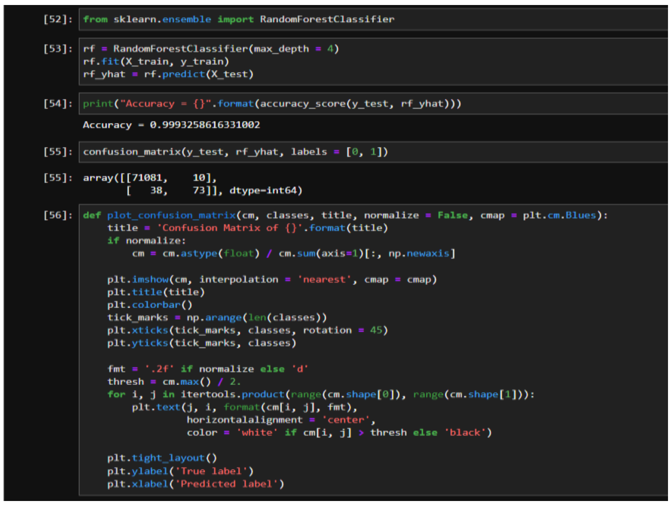
```
[48]: lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0,1])
      lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                 classes = ['Positive(0)','Negative(1)'],
                                 normalize = False, title = 'Logistic Regression')
      plt.savefig('lr_cm_plot.png')
      plt.show()
```



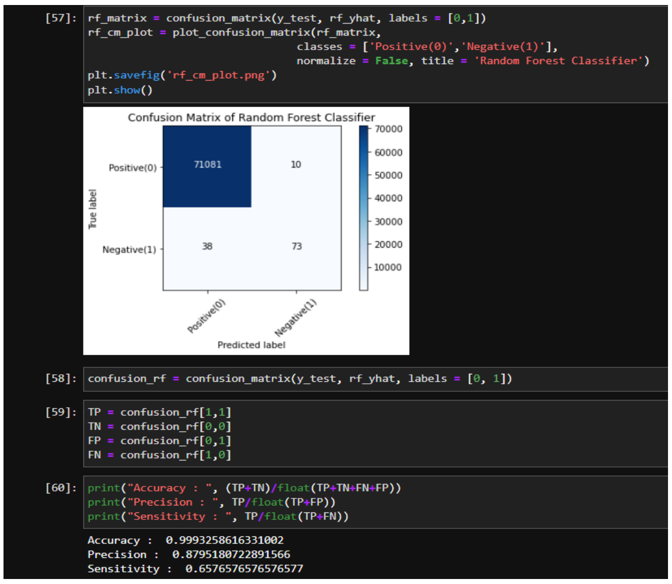


```
[49]: confusion_lr = confusion_matrix(y_test, lr_yhat, labels = [0, 1])

[50]: TP = confusion_lr[1,1]
      TN = confusion_lr[0,0]
      FP = confusion_lr[0,1]
      FN = confusion_lr[1,0]

[51]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9987781242099941
      Precision :  0.603448275862069
      Sensitivity :  0.6306306306306306
```

- RandomForest Classifier



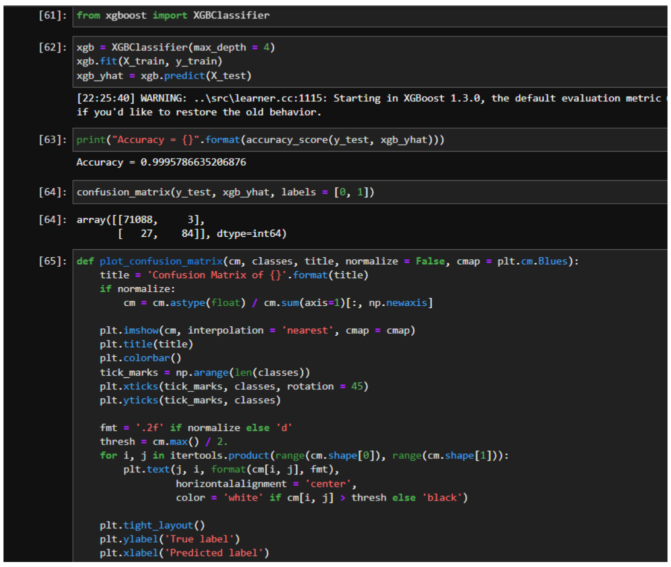
```
[52]: from sklearn.ensemble import RandomForestClassifier

[53]: rf = RandomForestClassifier(max_depth = 4)
      rf.fit(X_train, y_train)
      rf_yhat = rf.predict(X_test)

[54]: print("Accuracy = {}".format(accuracy_score(y_test, rf_yhat)))

      Accuracy = 0.9993258616331002

[55]: confusion_matrix(y_test, rf_yhat, labels = [0, 1])

[55]: array([[71081,    10],
             [   38,    73]], dtype=int64)

[56]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

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

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



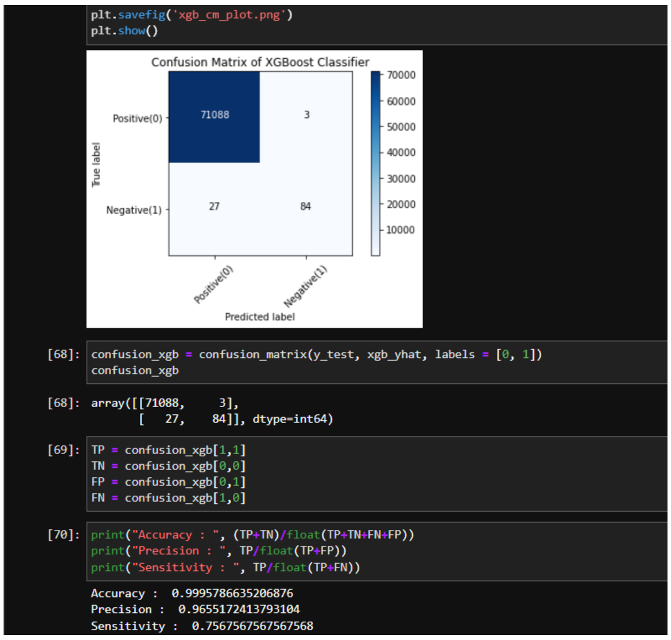
```
[57]: rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0,1])
      rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                         classes = ['Positive(0)','Negative(1)'],
                                         normalize = False, title = 'Random Forest Classifier')
      plt.savefig('rf_cm_plot.png')
      plt.show()
```

```
[58]: confusion_rf = confusion_matrix(y_test, rf_yhat, labels = [0, 1])

[59]: TP = confusion_rf[1,1]
      TN = confusion_rf[0,0]
      FP = confusion_rf[0,1]
      FN = confusion_rf[1,0]

[60]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9993258616331002
      Precision :  0.8795180722891566
      Sensitivity :  0.6576576576576577
```

- XGBoost Classifier



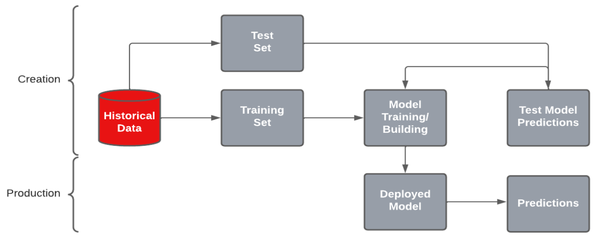
```
[61]: from xgboost import XGBClassifier

[62]: xgb = XGBClassifier(max_depth = 4)
      xgb.fit(X_train, y_train)
      xgb_yhat = xgb.predict(X_test)

      [22:25:40] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
      if you'd like to restore the old behavior.

[63]: print("Accuracy = {}".format(accuracy_score(y_test, xgb_yhat)))

      Accuracy = 0.9995786635206876

[64]: confusion_matrix(y_test, xgb_yhat, labels = [0, 1])

[64]: array([[71088,     3],
             [   27,    84]], dtype=int64)

[65]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

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

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

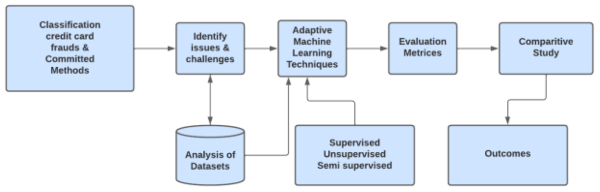


```
      plt.savefig('xgb_cm_plot.png')
      plt.show()
```

```
[68]: confusion_xgb = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
      confusion_xgb

[68]: array([[71088,     3],
             [   27,    84]], dtype=int64)

[69]: TP = confusion_xgb[1,1]
      TN = confusion_xgb[0,0]
      FP = confusion_xgb[0,1]
      FN = confusion_xgb[1,0]

[70]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9995786635206876
      Precision :  0.9655172413793104
      Sensitivity :  0.7567567567567568
```

BLOCK DIAGRAM & DESCRIPTION




Classification credit card frauds & Committed Methods
Identify issues & challenges
Adaptive Machine Learning Techniques
Evaluation Metrices
Comparitive Study
Analysis of Datasets
Supervised Unsupervised Semi supervised
Outcomes

V. RESULTS

We checked for the accuracy scores after fitting out training data into the model and we received an accuracy of 99.95 percent which is better that the accuracy received from other classification algorithms.

Algorithms	Accuracy	Precision	Sensitivity
Decision Tree Classifier	99.935%	84.946%	71.171%
Logistic Regression	99.877%	60.344%	63.063%
Random Forest	99.932%	87.951%	65.765%
XG Boost Classifier	99.957%	96.551%	75.675%

VI. CONCLUSION AND FUTURE SCOPE

Without a question, credit card fraud is a form of criminal deception. This article outlined the most frequent types of fraud, as well as how to detect them, and examined recent research findings in the field. This paper also includes a detailed explanation of how machine learning can be used to improve fraud detection findings, as well as the algorithm, pseudocode, explanation, and experimentation results. While the method achieves a precision of over 99.6%, when only a tenth of the data set is taken into account, it only achieves a precision of 28%. When the complete dataset is given into the system, however, the precision increases to 33%. Due to the massive imbalance, such a high percentage of accuracy is to be expected.

VII. REFERENCES

[1] **https://en.wikipedia.org/wiki/Logistic_regression**

[2] **https://stackoverflow.com/questions/22721060/matplotlib-unexpected-gridspec- behavior**

[3] **https://www.sciencedirect.com/science/article/pii/S187705092030065X**

[4] **https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive- models/notebook**