LOGISTIC REGRESSION

PDAN8412 Task 2

OCTOBER 25, 2022

SEAN TEUCHERT
ST10041890

Contents

1. Introduction:

The dataset that has been selected to build a Logistic Regression model consists of normal and fraudulent transactions. The dataset contains 6.36 million records that were generated during a research study to simulate normal and fraudulent transactions over a 30-day period. The simulator used to generate the records is called Paysim. During the COVID19 pandemic majority of businesses were forced to move its business operations to a digital platform. This has resulted in many businesses becoming vulnerable to online fraud due to lacking effective preventative measures. A large business may experience hundreds or thousands of transactions during a single day. It can be very time consuming to identify fraudulent transactions manually and human error can result in fraudulent transactions being missed. This has resulted in a problem of business's lacking the tools to combat fraud. To solve this problem, a Machine learning model can be implemented to analyze the transactions and distinguish between normal transactions and fraudulent transactions. This will allow a business to automate the process of identify potential fraudulent transactions which can then be investigated further by an employee. Implementing a machine learning model to automate this process will increase efficiency and provide the business with a tool to combat fraud.

Author of the Dataset: Vardhan Siramdasu

Dataset link:

https://www.kaggle.com/datasets/vardhansiramdasu/fraudulent-transactions-prediction

Paysim Research Paper link:
https://www.researchgate.net/publication/313138956_PAYSIM_A_FINANCIAL_MOBILE_MONEY_SIMULATOR_FOR_FRAUD_DETECTION

Please note:

A Jupyter Notebook was used to develop the initial code, but I used Visual Studio Code to write the finalized code. The file types are the same therefore the code can be run in Jupyter Notebook or Visual Studio Code.

2. What is Logistic Regression?

To improve our understanding of Logistic Regression, it is important to first understand what classification is. Classification is the process of predicting the class of given data points, for example: email spam detection (Asiri, 2018). Classification algorithms maps the mapping function from input variables to discrete output variables. Logistic Regression is a supervised classification algorithm used to assign observations to a discrete set of classes (Pant, 2019). Logistic Regression uses the Sigmoid function to map any real value into another value between 0 and 1. Logistic Regression can be used to solve classification tasks and provide a binary output.

3. Why is the Dataset appropriate for Logistic Regression?

Logistic Regression is used to solve classification problems that require a binary output. The selected dataset consists of numerical features, categorical features, and a binary output. The categorical features can be converted into numerical features using One-hot encoding. This provides the model with enough data and features to make accurate predictions. The dataset is appropriate for Logistic Regression because the features can be converted to similar data types and a binary output is provided in the dataset to whether the transaction is fraudulent or not. The binary output (0 or 1) is provided in the isFraud column. This will allow the Logistic Regression model to perform supervised learning and map the mapping function from the input variables to the output variable. The dataset is unbalanced and only consists of around 8000 fraudulent transactions but SMOTE can be used to solve this problem and increase the model’s accuracy. The dataset meets the requirements for using Logistic Regression for this classification task therefore, it is an appropriate dataset.

4. What is the question that the analysis will answer?

The purpose of this analysis is to answer the following questions:

- Is it possible to identify a fraudulent transaction only using the transaction information?
- Can a Machine Learning model be used to categorize the transactions based on whether the transaction is fraudulent or not?

5. Types of analysis performed:

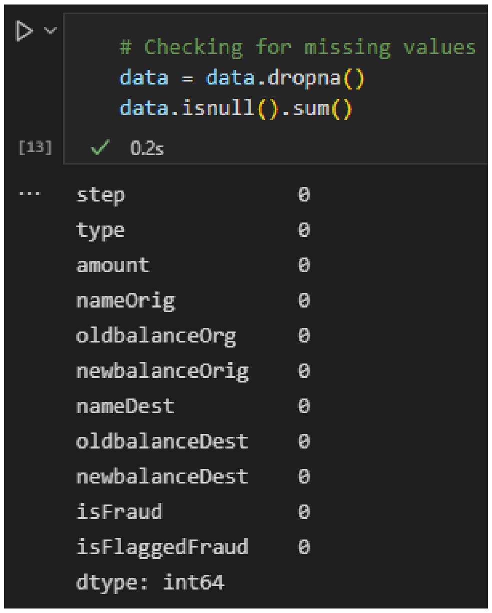
- Descriptive Analysis – Visualize the dataset
- Predictive Analysis – Logistic Regression and Random Forest Classifier has been used to build a model to predict fraudulent transactions.

6. Exploratory Data Analysis and Data Wrangling:

To improve the data analyst understanding of the data set, Exploratory Data Analysis (EDA) was performed. A pie chart was used to determine the distribution of the transaction types. This information can be used to determine what the most common transaction types are. A count plot was used to determine the number of transactions per transaction type. This information can be used to identify what transaction types are most prone to fraud. A 2nd pie chart was used to determine the distribution of fraudulent transactions. This information can be used to determine whether the dataset is balanced or if SMOTE is required. Heatmaps was used to determine the correlation between features. This improves the data analyst's understanding of what features may have a higher weighting during the model's decision.

After performing EDA, it is important to perform Data wrangling or Feature Engineering. The reason for performing EDA before feature engineering is to ensure that the categorical features are accurately represented in the visualization e.g. It can become confusing which dummy variables belong to which categorical values.

Figure 1 illustrates removing of nulls:



```
# Checking for missing values
data = data.dropna()
data.isnull().sum()
```

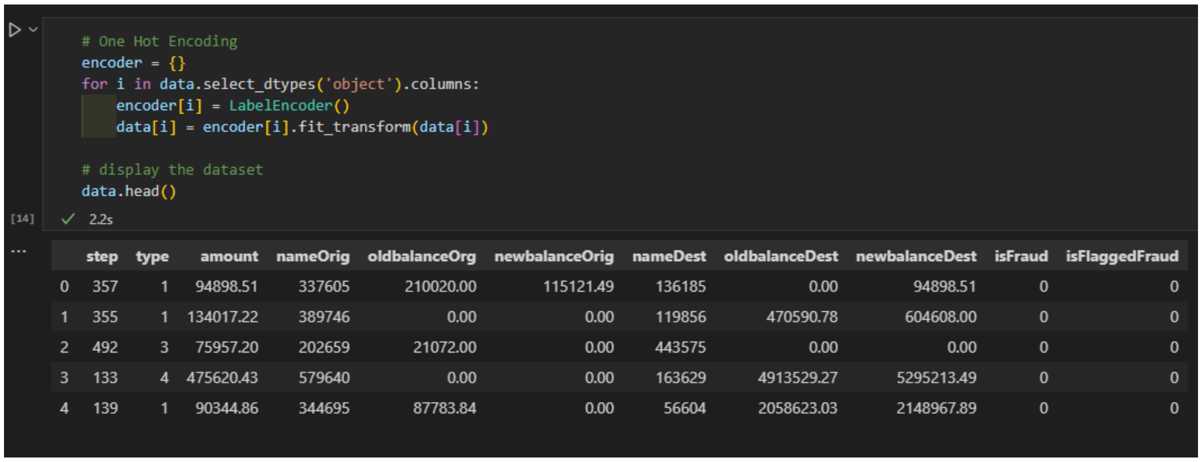
```
[13]  ✓ 0.2s
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Figure 1: Remove Null Method

The first step of Feature Engineering is to remove null values. This will improve the model's accuracy by eliminating any potential outliers.

Figure 2 illustrates the One-Hot Encoding method:



```
# One Hot Encoding
encoder = {}
for i in data.select_dtypes('object').columns:
    encoder[i] = LabelEncoder()
    data[i] = encoder[i].fit_transform(data[i])

# display the dataset
data.head()
```

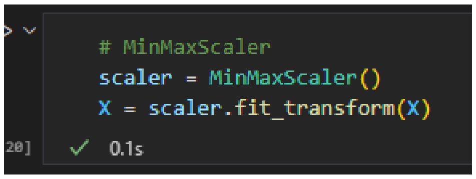
```
[14]  ✓ 2.2s
```

```
   step  type     amount  nameOrig  oldbalanceOrg  newbalanceOrig  nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0   357     1   94898.51    337605      210020.00       115121.49    136185            0.00        94898.51        0               0
1   355     1  134017.22    389746           0.00            0.00    119856       470590.78       604608.00        0               0
2   492     3   75957.20    202659       21072.00            0.00    443575            0.00            0.00        0               0
3   133     4  475620.43    579640           0.00            0.00    163629      4913529.27      5295213.49        0               0
4   139     1   90344.86    344695       87783.84            0.00     56604      2058623.03      2148967.89        0               0
```

Figure 2: One-hot encoding method

The next step of Feature engineering is to One-hot encode all the categorical features. This will increase the models and reduce the training time because Logistic Regression using the Sigmoid Function which works better with numerical values.

Figure 3 illustrates the MinMaxScaler method:



```
# MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
20]  ✓ 0.1s
```

Figure 3: MinMaxScaler method

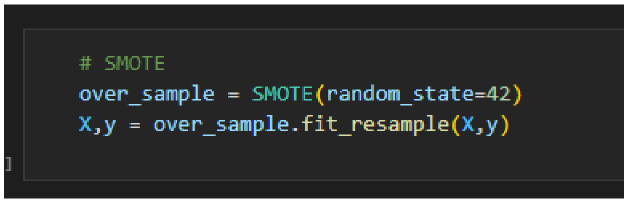
The last step of Feature Engineering is to reduce the dimensionality of the dataset by using the MinMaxScaler. This will improve the accuracy of the model and reduce the processing power required.

After performing EDA, the Data analyst should have a good understanding of the domain problem and has the required knowledge to build the machine learning model. After performing Data wrangling, the dataset is cleaned and appropriate for a Logistic Regression model.

7. Overfitting and Underfitting:

Overfitting refers to a model that models the training data too well or has been over trained with a large amount of data. This can become a problem because the model will not be able to correctly analyze a new dataset. Underfitting refers to a model that cannot capture the underlying trend of the data, this is a result of using the incorrect model to analyze the data or by not providing enough test data.

Figure 4 illustrates the SMOTE method:



```
# SMOTE
over_sample = SMOTE(random_state=42)
X,y = over_sample.fit_resample(X,y)
```

Figure 4: SMOTE method

To combat overfitting, I have implemented a method called SMOTE from the Imblearn library, is a statistical technique that is used to increase the number of cases in your dataset in a balanced way. In context to my dataset, the dataset was unbalanced and consisted of 635 000 non-fraudulent transactions and only 8000 fraudulent transactions. SMOTE balances the dataset to 635 000 non-fraudulent and fraudulent transactions. To combat underfitting, I have provided the model with a large volume of data.

8. Table and Graph Explanation:

Figure 5 illustrates the distribution of transaction types:

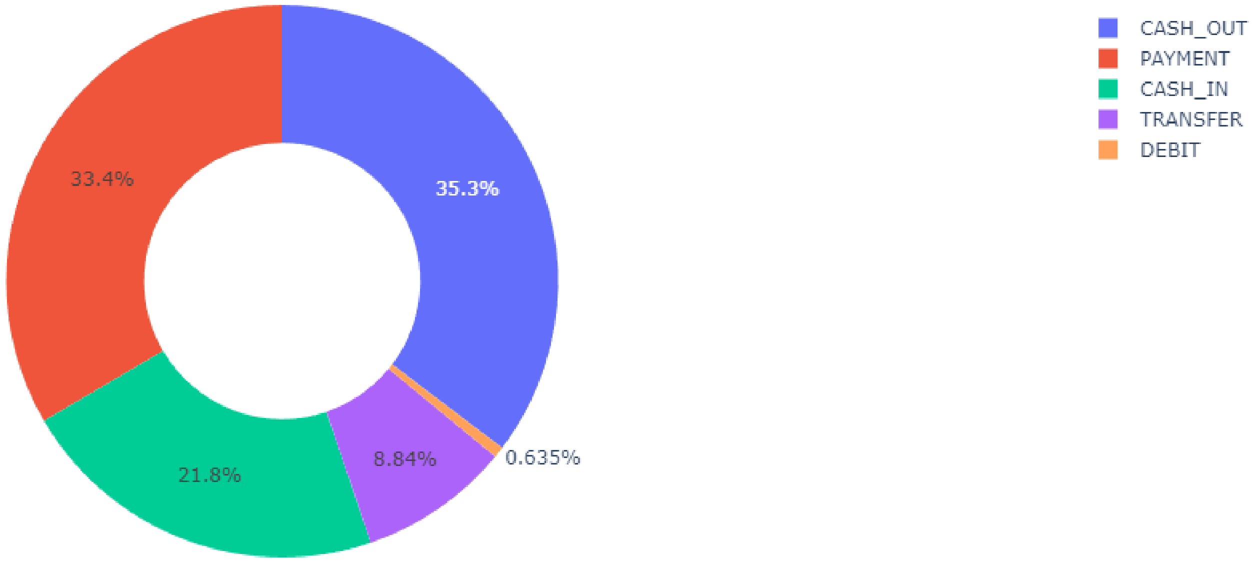


Figure 5: Distribution of Transaction Type Pie Chart

This pie chart conveys the information about the spread of the feature called transaction type. The legend refers to the type of transaction and the corresponding percentage is displayed on the pie chart. This indicates what the most recorded transaction types are in the dataset. This information can help the data analyst identify what transactions are commonly used.

Figure 6 illustrates the number of transactions per type and whether it is fraudulent or not:

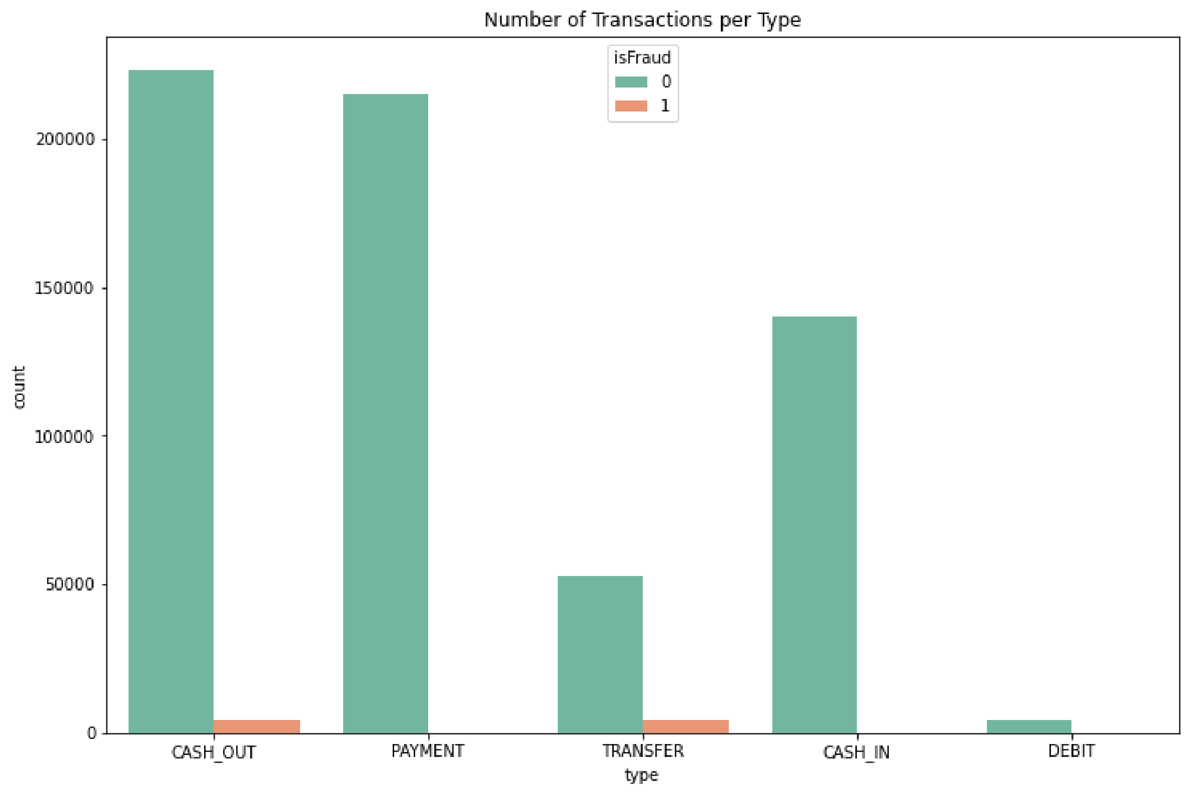


Figure 6: Number of Transactions per Type Bar Chart

The bar chart illustrates two important pieces of information. It firstly illustrates the distribution of transactions in a numerical form which cannot be interpreted via the pie chart. The second important information it visualizes is the distribution of fraudulent transactions per transaction type. This information is important because it will help the data analyst identify any patterns or trends in the transaction types. From the bar chart, we can interpret that the most common transaction type is CASH_OUT, followed by PAYMENT, CASH_IN, TRANSFER and DEBIT. We can also interpret that only CASH_OUT and TRANSFER contain fraudulent transactions.

Figure 7 illustrates the distribution of fraudulent transactions:

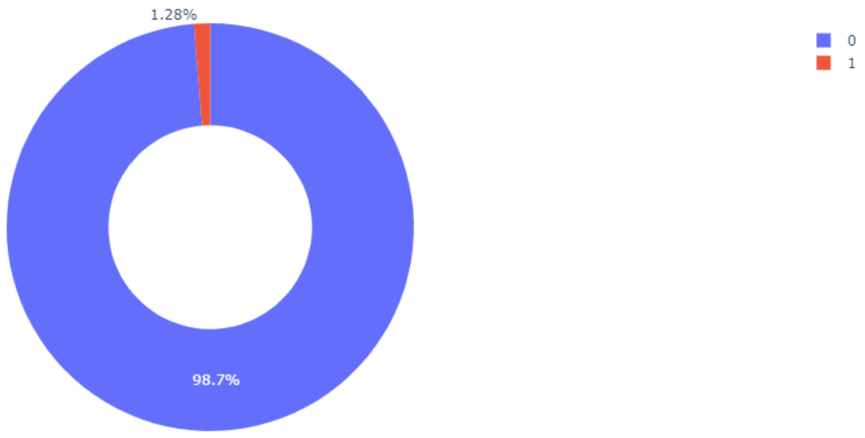


Figure 7: Distribution of Fraud Pie Chart

The pie chart conveys the information about the total number of non-fraudulent and fraudulent transactions in the dataset. The legend refers to whether the transaction was fraudulent or not, the 0 represents the non-fraudulent transactions and 1 represents the fraudulent transaction. This information is important because it indicates the sample sizes and whether the dataset has balanced sample sizes. The dataset consists of 98,7% non-fraudulent transactions and 1,28% fraudulent transactions, this means that the dataset is unbalance. This information tells the data analyst whether SMOTE is needed to balance the samples to combat overfitting.

Figure 8 illustrates a heatmap of the dataset variables before One-hot encoding:

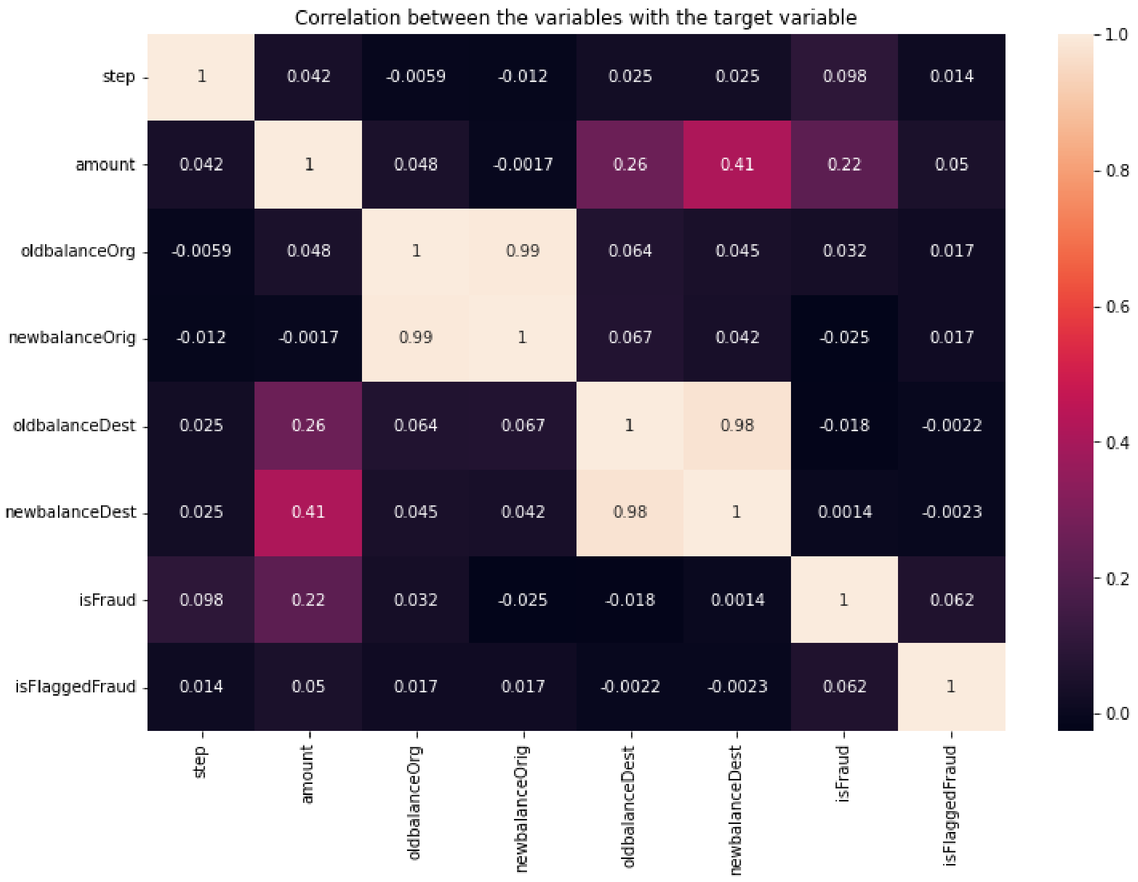


Figure 8: Feature Heatmap before One-hot Encoding

The heatmap illustrates the correlations between the numerical features. This provides the data analyst with a visual representation of the potential relationships between features in the dataset. This is important information because it can help the data analyst identify what features have the potential to increase or decrease the model's performance.

Figure 9 illustrates a heatmap of the data variables after One-hot encoding:

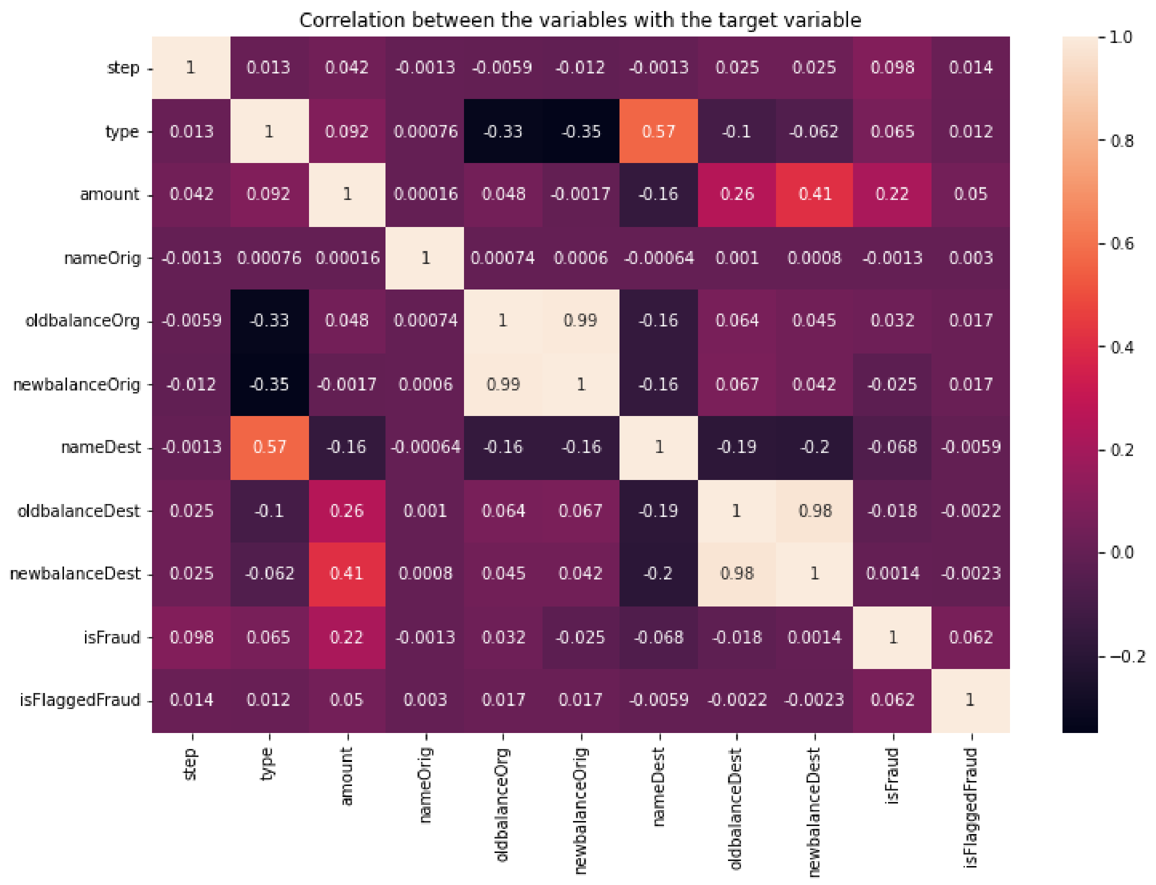


Figure 9: Feature Heatmap after One-hot Encoding

As discussed in the section above, the heatmap provides the data analyst with a visual representation of the correlation and relationships between features in the dataset. The data analyst has generated two heat maps, a heatmap before one-hot encoding and after one-hot encoding. This helps the data analyst improve their understanding of the dataset by first focusing on the numerical features and then looking at the dataset as a whole. It is important to note that the features with a negative correlation may affect the accuracy of the model negatively. The data analyst tested each feature when building the model and decided to still include the features with a negative correlation due to the accuracy being higher with them included. This is due to the nature of the problem being a fraudulent problem for example: the feature nameDest has a negative correlation with isFraud but this information is important. A trend may occur where the fraudulent transactions are transferred to the same account. The model will be able to identify this trend if the feature is included in the dataset.

Figure 10 illustrates Learning rate vs Accuracy scatterplot:

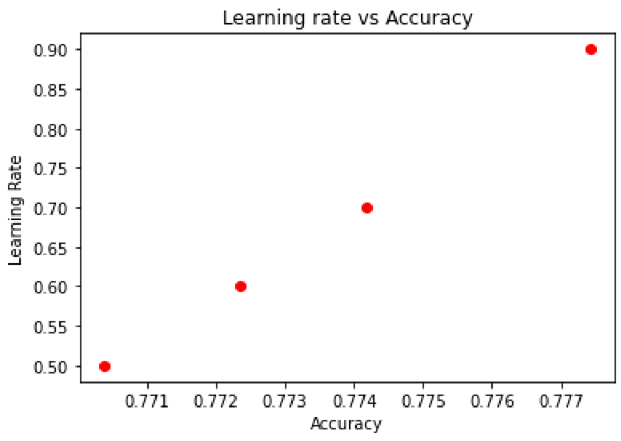


Figure 10: Learning rate vs Accuracy Scatterplot

The scatterplot conveys information about how learning rate affects the model's accuracy. The X-axis refers to the accuracy score and the Y-axis refers to the learning rate. Four different learning rates were tested, and the scatterplot helps the data analyst to visually see which learning rate had the highest impact on accuracy. This information is important to know when tuning the model's hyperparameters. We can identify a trend using the scatterplot, as learning rate increases the model's accuracy also increases.

Figure 11 illustrates the Logistic Regression from Scratch confusion matrix:

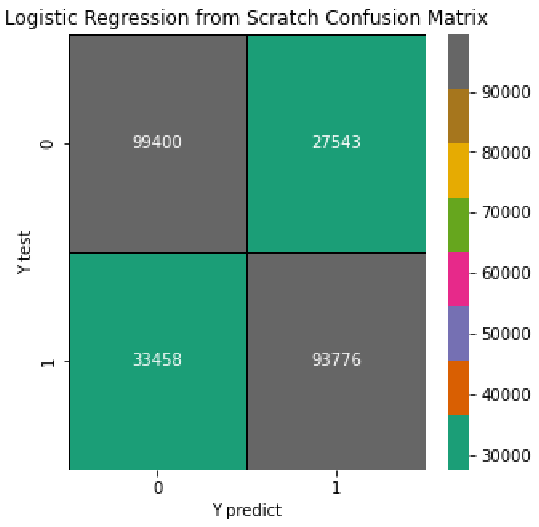


Figure 11: Logistic Regression from Scratch Confusion Matrix

A confusion matrix visually conveys information about how accurate the model can make predictions. It is important to make predictions using a test dataset that was kept separate during training to ensure that the model is not overfitting. After making predictions using a test dataset, the confusion matrix outputs how many predictions were correct and how many were incorrect. After analyzing the confusion matrix, we can interpret that the model was able to correctly predict 93 776 fraudulent transactions and 99 400 non-fraudulent transactions. The model incorrectly predicted 33 458 fraudulent transactions as non-fraudulent transactions (referred to as false-negative) and 27 543 non-fraudulent transactions as fraudulent transactions (referred to as false-positives). This information can be used to evaluate the model's performance and its ability to make accurate predictions using new data that wasn't used during training.

Figure 12 illustrates the Logistic Regression using Sklearn confusion matrix:

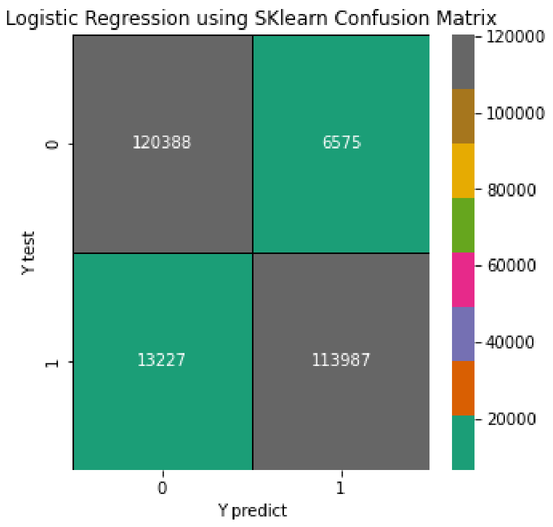


Figure 12: Logistic Regression using SKlearn Confusion Matrix

After analyzing the confusion matrix, we can interpret that the model was able to correctly predict 113 987 fraudulent transactions and 120 338 non-fraudulent transactions. The model incorrectly predicted 13 227 fraudulent transactions as non-fraudulent transactions (referred to as false-negative) and 6575 non-fraudulent transactions as fraudulent transactions (referred to as false-positives). This information can be used to evaluate the model's performance and its ability to make accurate predictions using new data that wasn't used during training.

Figure 13 illustrates the Random Forest Classifier confusion matrix:

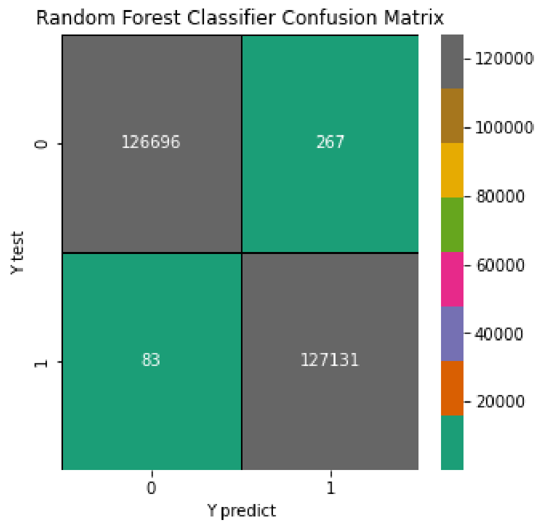


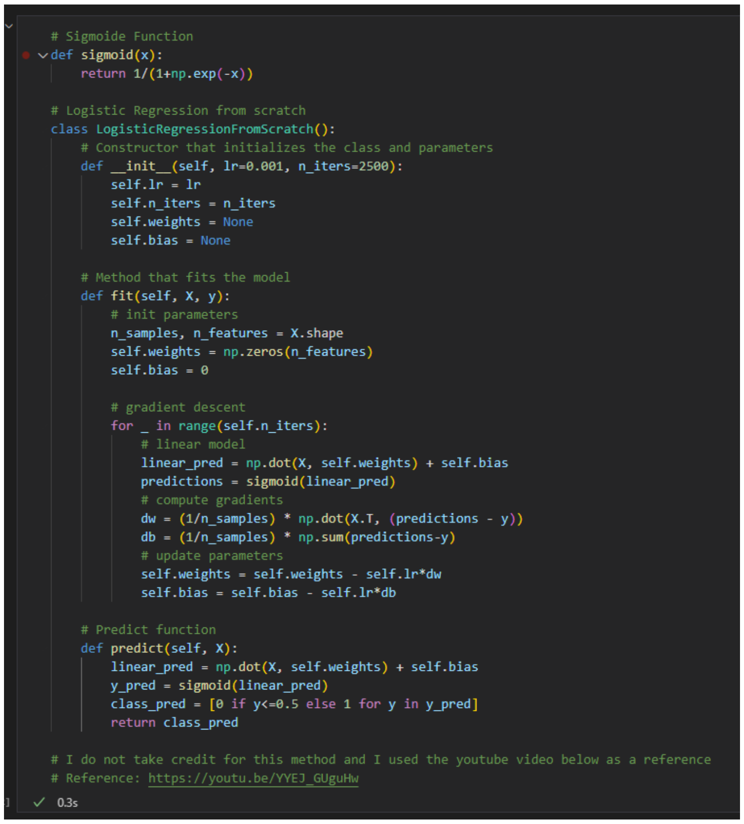
Figure 13: Random Forest Classifier Confusion Matrix

After analyzing the confusion matrix, we can interpret that the model was able to correctly predict 127 131 fraudulent transactions and 126 696 non-fraudulent transactions. The model incorrectly predicted 83 fraudulent transactions as non-fraudulent transactions (referred to as false-negative) and 267 non-fraudulent transactions as fraudulent transactions (referred to as false-positives). This information can be used to evaluate the model's performance and its ability to make accurate predictions using new data that wasn't used during training.

9. Tuning the model's Hyperparameters:

A Logistic Regression model can be tuned by using GridsearchCV but a Logistic Regression from Scratch model cannot. This means that the data analyst must manually tune the hyperparameters of a Logistic Regression from Scratch model via trial and error.

Figure 14 illustrates the Logistic Regression from Scratch method:

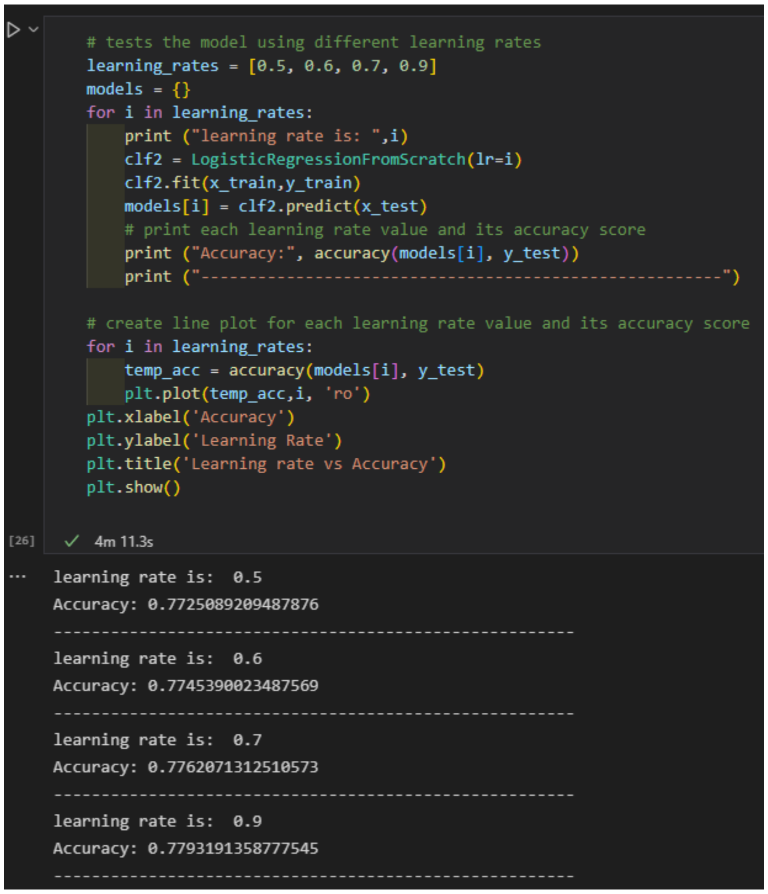


```python
# Sigmoide Function
def sigmoid(x):
    return 1/(1+np.exp(-x))

# Logistic Regression from scratch
class LogisticRegressionFromScratch():
    # Constructor that initializes the class and parameters
    def __init__(self, lr=0.001, n_iters=2500):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    # Method that fits the model
    def fit(self, X, y):
        # init parameters
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):
            # linear model
            linear_pred = np.dot(X, self.weights) + self.bias
            predictions = sigmoid(linear_pred)
            # compute gradients
            dw = (1/n_samples) * np.dot(X.T, (predictions - y))
            db = (1/n_samples) * np.sum(predictions-y)
            # update parameters
            self.weights = self.weights - self.lr*dw
            self.bias = self.bias - self.lr*db

    # Predict function
    def predict(self, X):
        linear_pred = np.dot(X, self.weights) + self.bias
        y_pred = sigmoid(linear_pred)
        class_pred = [0 if y<=0.5 else 1 for y in y_pred]
        return class_pred

# I do not take credit for this method and I used the youtube video below as a reference
# Reference: https://youtu.be/YYEJ_GUguHw
```

Figure 14: Logistic Regression from Scratch

The Logistic Regression from Scratch method has two hyperparameters that can be tuned: Learning rate (lr) and number of iterations (n_iters). The default hyper parameters that were tested were lr=0.01 and n_iters=2500, the model performed decently with an accuracy of 76%.

Figure 14 illustrates the hyperparameter tuning for Logistic Regression from Scratch:



```
# tests the model using different learning rates
learning_rates = [0.5, 0.6, 0.7, 0.9]
models = {}
for i in learning_rates:
    print ("learning rate is: ",i)
    clf2 = LogisticRegressionFromScratch(lr=i)
    clf2.fit(x_train,y_train)
    models[i] = clf2.predict(x_test)
    # print each learning rate value and its accuracy score
    print ("Accuracy:", accuracy(models[i], y_test))
    print ("-------------------------------------------------------")

# create line plot for each learning rate value and its accuracy score
for i in learning_rates:
    temp_acc = accuracy(models[i], y_test)
    plt.plot(temp_acc,i, 'ro')
plt.xlabel('Accuracy')
plt.ylabel('Learning Rate')
plt.title('Learning rate vs Accuracy')
plt.show()
```

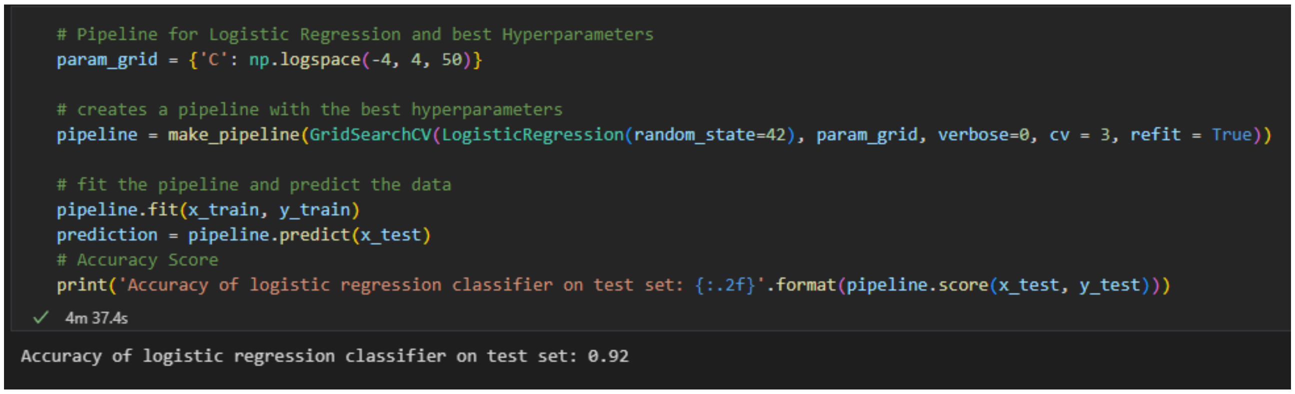
```
[26]  ✓ 4m 11.3s
```

```
learning rate is:  0.5
Accuracy: 0.7725089209487876
-------------------------------------------------------
learning rate is:  0.6
Accuracy: 0.7745390023487569
-------------------------------------------------------
learning rate is:  0.7
Accuracy: 0.7762071312510573
-------------------------------------------------------
learning rate is:  0.9
Accuracy: 0.7793191358777545
-------------------------------------------------------
```

Figure 15: Learning rate tuning method

The data analyst has implemented a method to identify the best learning rate by using a for loop to fit the model and make predictions using different learning rates. Through trial and error, the data analyst identified that as the learning rate increases the accuracy increases but it is important to note that setting the learning rate to anything value above one will create noise in the model. The data analyst did not use any special method to test the best number of iterations, it was done manually through trial and error.

Figure 15 illustrates the hyperparameter tuning for Logistic Regression using Sklearn:



```
# Pipeline for Logistic Regression and best Hyperparameters
param_grid = {'C': np.logspace(-4, 4, 50)}

# creates a pipeline with the best hyperparameters
pipeline = make_pipeline(GridSearchCV(LogisticRegression(random_state=42), param_grid, verbose=0, cv = 3, refit = True))

# fit the pipeline and predict the data
pipeline.fit(x_train, y_train)
prediction = pipeline.predict(x_test)
# Accuracy Score
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(pipeline.score(x_test, y_test)))
```

```
✓ 4m 37.4s
```

```
Accuracy of logistic regression classifier on test set: 0.92
```

Figure 16: GridSearchCV method in a pipeline

The last method used to tune the models' hyper parameters is GridSearchCV, as stated earlier this method only works on algorithms that come from a library e.g., Sklearn Logistic Regression. GridsearchCV was used to tune the Logistic Regression using Sklearn model, the model's accuracy increased after tuning the hyperparameters from 86% to 92%. An improvement of 6% can be considered good and it will reduce the model's potential to make incorrect predictions.

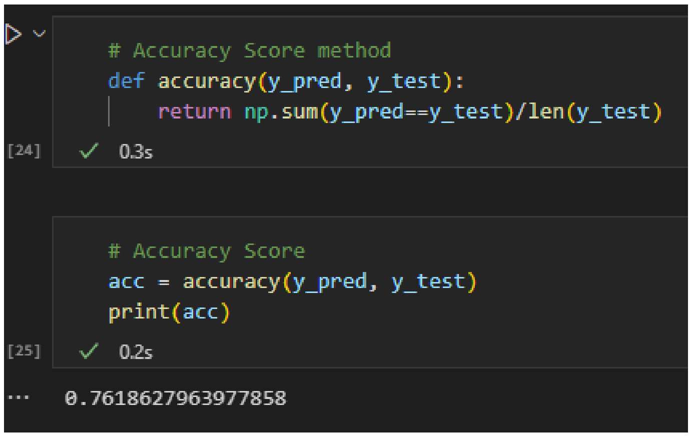
10. Evaluating the Model:

During analysis, the data analyst built three different models using three different machine learning algorithms. Each model was evaluated using the following methods:

- Accuracy Score
- Classification Report
- Confusion Matrix

Please note: this section only explains the steps for one model, but the same steps were applied to all three models. The results are discussed later in section 11.

Figure 16 illustrates the accuracy score method



```
# Accuracy Score method
def accuracy(y_pred, y_test):
    return np.sum(y_pred==y_test)/len(y_test)
```
[24] ✓ 0.3s

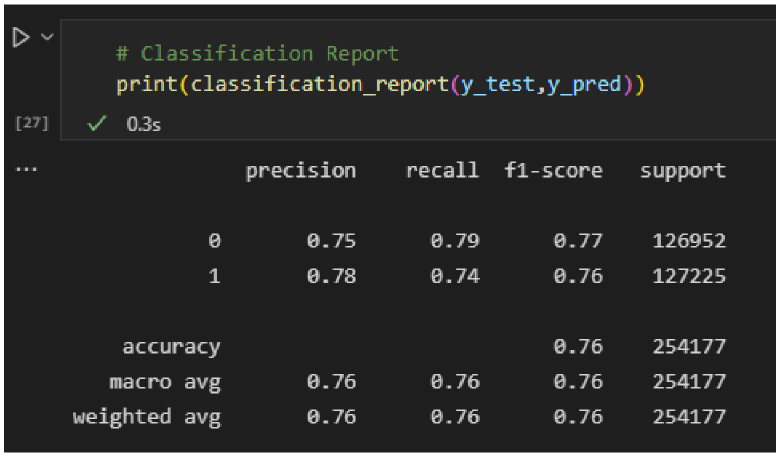
```
# Accuracy Score
acc = accuracy(y_pred, y_test)
print(acc)
```
[25] ✓ 0.2s

```
0.7618627963977858
```

Figure 17: Accuracy score method

The model has been evaluated using a test dataset called y_test which was not included in the model training. This ensures that the model is not evaluating against the data it learnt from and producing a false accuracy. Accuracy refers to how often we can expect our model to correctly predict an outcome. An accuracy score of 0.76 is a good and this means that majority of the time it will make accurate predictions.

Figure 17 illustrates the classification report method:



```
# Classification Report
print(classification_report(y_test,y_pred))
```

```
[27]   ✓ 0.3s
```

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77    126952
           1       0.78      0.74      0.76    127225

    accuracy                           0.76    254177
   macro avg       0.76      0.76      0.76    254177
weighted avg       0.76      0.76      0.76    254177
```

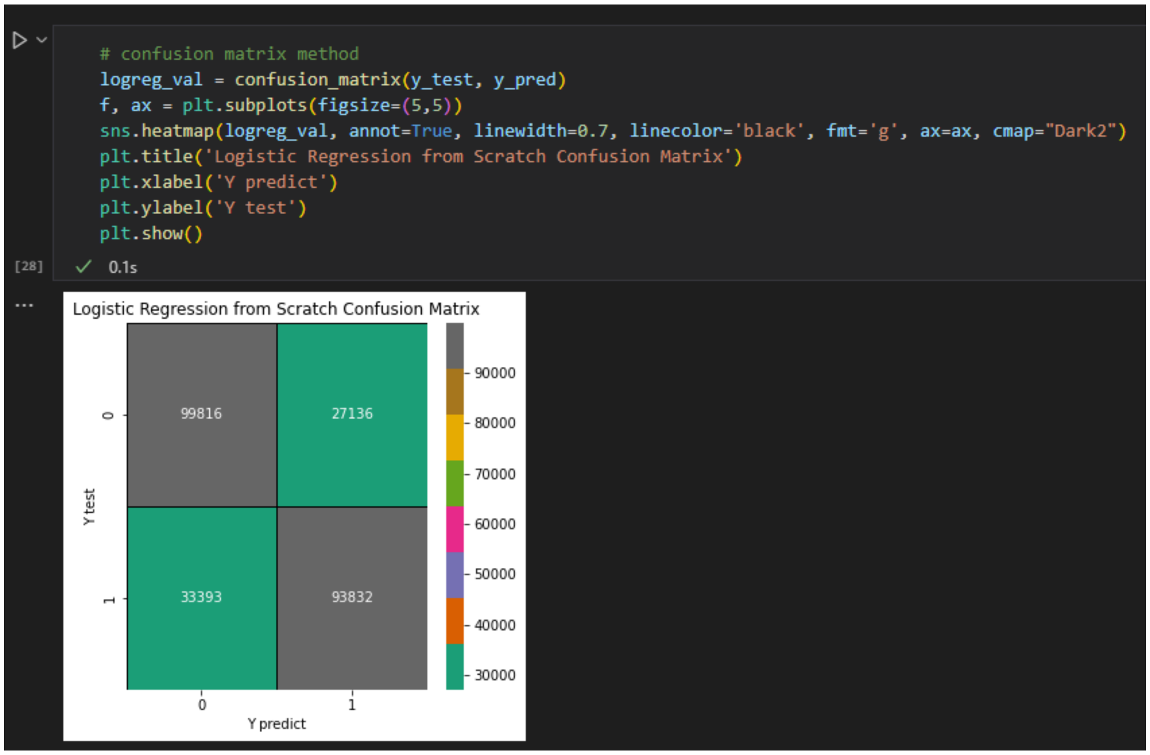
Figure 18: Classification Report

From Sklearn's classification report, we can get the precision, recall and f1_score scores for the model. The precision score measures the proportion of positively predicted labels that are actually correct (Kumar, 2022). A precision score of 0.78 for predicted fraudulent transactions, 0.75 for predicted non-fraudulent transactions is a good score. This means that majority of the positive predicted values are correct.

Recall score represents the model's ability to correctly predict the positives out of actual positives (Kumar, 2022). A recall score of 0.74 for fraudulent transactions, 0.79 for positive reviews is a good score. This tells us that the model is good at identifying actual positive out of all positives that exist within the dataset.

F1_score represents the model score as a function of precision and recall score. F1_score is a performance metric that gives equal weight to both the Precision and Recall for measuring its performance in terms of accuracy (Kumar, 2022). A f1_score of 0.76 for fraudulent transactions, 0.77 for non-fraudulent transactions is a good score. This tells us that the model is optimized, and it is providing accurate outputs.

Figure 17 illustrates the confusion matrix method:



```
# confusion matrix method
logreg_val = confusion_matrix(y_test, y_pred)
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(logreg_val, annot=True, linewidth=0.7, linecolor='black', fmt='g', ax=ax, cmap="Dark2")
plt.title('Logistic Regression from Scratch Confusion Matrix')
plt.xlabel('Y predict')
plt.ylabel('Y test')
plt.show()
```

[28] ✓ 0.1s

Figure 19: Confusion Matrix method

We can further investigate the model’s prediction performance visually using a confusion matric. A confusion matrix illustrates how many predicts were correctly and incorrectly predicted. From the confusion matrix we can see, the model only predicted 33 393 fraudulent transactions incorrectly and 27 136 non-fraudulent transactions incorrectly. We can assume the model performed well due to the number of incorrectly predicted values being lower than the number of correctly predicted values.

11. Interpreting the Results:

The data analyst has built three different models to test which algorithm is best suited for the dataset. The Logistic Regression from Scratch after hyperparameter tuning had an accuracy of 77% when the learning rate is 0.5 and the number of iterations is 2500. The hyperparameter tuning only increased this model's accuracy by 1%. The Logistic Regression using Sklearn algorithm outperformed the Logistic Regression from Scratch algorithm, after hyperparameter tuning the model had an accuracy of 92%. The hyperparameter tuning increased this model's accuracy by 6%. The best performing algorithm overall was the Random Forest Classifier. It achieved a perfect accuracy of 100% and the hyperparameters did not need to be tuned. All three models were able to make accurate predictions when testing with new user inputted data. A recommendation is to build the model using a Deep Learning Algorithm. This was not tested during analysis but due to the high performance of the Random Forest Classifier, a deep learning algorithm has the potential to improve the model. This is due to the nature of deep learning algorithms being improved versions of the Random Forest Classifier.

12. Possible more appropriate methods:

During analysis the data analyst tested three different algorithms when building the model: Logistic Regression from Scratch, Logistic Regression using Sklearn Library and Random Forest classifier. After evaluating each algorithm using the same methods, the Random Forest Classifier performed the best with an accuracy of 100%, followed by Logistic Regression using Sklearn with an accuracy of 92%, and in last place was the Logistic Regression from Scratch with an accuracy of 77%. Using the Random Forest Classifier as the machine learning algorithm would be a more appropriate method. Another possible method that may be more appropriate for fraud detection is using a Deep Learning algorithm. A Deep Learning Algorithm was not tested during this analysis but due to the Random Forest Classifier's high accuracy, it is safe to assume a Deep Learning algorithm would perform well.

13. Summary of Code:

The model created follows the structure of Figure 19:

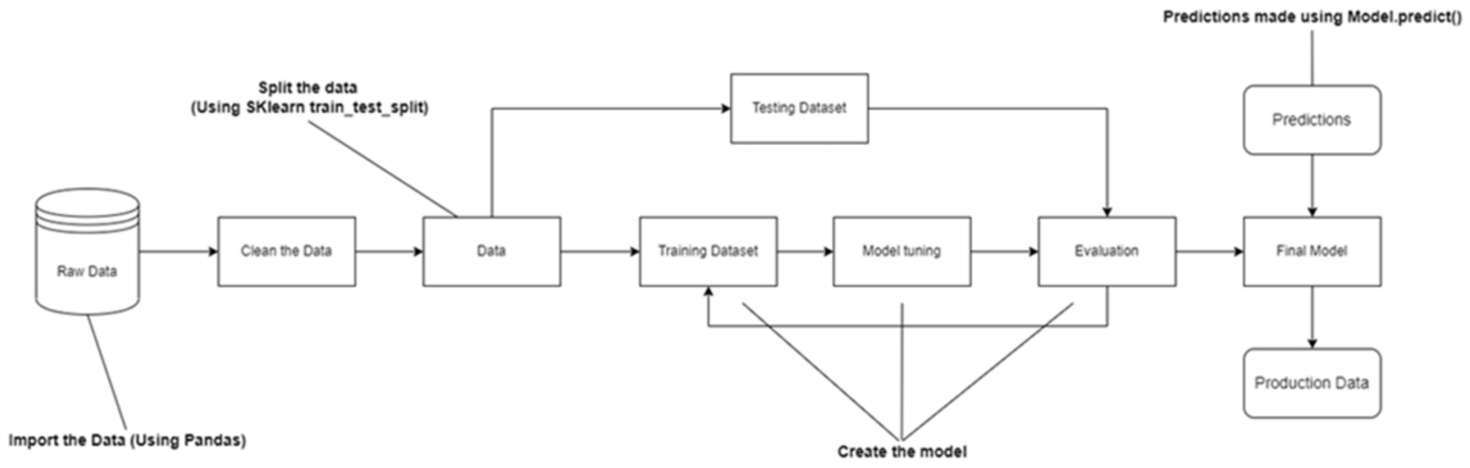


Figure 20: Machine Learning Model diagram

The raw data is first imported using Pandas. The data is visualized using graphs to gain an understanding of any potential trends. The data is then cleaned using Featured engineering to eliminate any null values and one-hot encode. The next step is to split the dataset into training and test data (80% train and 20% test data). The training data is used to train the model and the algorithm selected to train the model is Logistic Regression. After building the model, the hyperparameters are tuned using GridSearchCV. After tuning the hyperparameters, the model is evaluated using the test data and the sklearn libraries (classification report, accuracy score, confusion matrix.

14. Conclusion:

After performing Exploratory Data Analysis, the transaction types most likely to contain fraud is Transfers and Cash_out. This trend can be due to the nature of fraudulent actions, a malicious attacker is likely to transfer the money between accounts and withdraw it in cash to avoid the money becoming traceable to a bank account. The dataset is also unbalanced which means SMOTE was used to balance the samples which increased the model’s accuracy. Three different models were built using three different machine learning algorithms. The best performing algorithm was the Random Forest Classifier with an accuracy of 100%, followed by Logistic Regression using Sklearn library with an accuracy of 92% and Logistic Regression from Scratch with an accuracy of 77%. The model was successfully able to make predictions and differentiate between non-fraudulent and fraudulent transactions. Fraud is a big problem worldwide and as the internet of things continues to grow so will fraudulent transactions. This model can be used by any company that needs a tool to combat fraud. The model can identify fraudulent transactions which can be flagged by the model and then investigated further by an employee. This automates the process of flagging fraudulent transactions and eliminates manual labour from the process. Ultimately increasing efficiency and reduces the loss of profits due to fraud. The outcome achieved by this model is that machine learning algorithms can be used to identify fraud. All the analysis questions are answered by the results produced by the model and the model was successfully able to categorize the transactions.

15. References

Asiri, S., 2018. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623
[Accessed 23 10 2022].

Kumar, A., 2022. *Vital Flux.* [Online]
Available at: https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/
[Accessed 06 06 2022].

Pant, A., 2019. *Towards Data Science.* [Online]
Available at: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148
[Accessed 23 10 2022].