Configuration Manual

MSc Research Project
Programme Name

Nwabuogoh Anne Alu
Student ID: x22115871

School of Computing
National College of Ireland

Supervisor: Brian Byrne

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Nwabuogoh Anne Alu
Student ID:	x22115871
Programme:	MSc Fintech **Year:** 2023
Module:	MSc Research Project
Lecturer:	Brian Byrne
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	1322 **Page Count:** ……13…………………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……………………………………………………………………………………

Date: ……………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nwabuogoh Anne Alu
Student ID: x22115871

1 Overview

This guide outlines the implementation details, including system specifications, required software, tools, and environmental prerequisites, essential for conducting the research project focused on evaluating machine learning algorithms in payment card fraud detection. The primary objective of this documentation is to elucidate the technical execution of the project, ensuring experiment reproducibility and facilitating a comprehensive understanding of the undertaken work.

2 System Specification

Spec Name	Spec Value
Operating System	Microsoft Windows 11 Pro
RAM	16.0 GB
Processor	11^{th} Gen Intel(R) Core(TM) i7-1195G7 @ 2.90GHz, 2918 Mhz, Quad Core
Disk Space	500 GB SSD
System Type	Dell XPS 13 9310

Table 1: system requirements

3 Software Tools

The project code implementation was carried out using Google Colab, which is a cloud-based IDE and the programming language of choice was Python. Table 2 contains the details of the development environment.

Spec Name	Spec Value
Operating System	Linux
RAM	16.0 GB

Processor	Intel(R) Xeon(R) CPU @ 2.20GHz, Dual Core
Disk Space	107 GB
Runtime Programming Language	Python 3.10.12
Browser	Google Chrome, version 114.0.5735.134

Table 2: software requirements

4 Data Source

The dataset used for this research was gotten from Kaggle, it was a synthetic one, generated using a simulator developed by Brandon Harris. Due to computational resource restraints, only a sample of the data was used, the link to the complete dataset is here. The data was then loaded into Colab using the Pandas package as shown in figure 1.

```
# read data from file into pandas dataframe
credit_card_data = pd.read_csv('credit_card_data.csv')
```

Figure 1: Reading data into Pandas Data-freame

5 Software Libraries

To carry out this research, several Python libraries needed to be installed and imported into Colab, some the packages include SKlearn, Numpy, imblearn, and Pandas. The breakdown of all the libraries used can be seen in figure 2.

```
import os
# libraries for creating dataframes and arrays
import numpy as np
import pandas as pd

# library for splitting the data into test and train dataframes
from sklearn.model_selection import train_test_split

# libraries for resampling class imbalance
from imblearn.combine import SMOTETomek
from imblearn.combine import SMOTEENN

# libraries for feature encoding, feature engineering
# and scaling features
from sklearn.datasets import make_classification
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import PolynomialFeatures
import category_encoders as ce

# libraries for model building and evaluation
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
import lightgbm as lgb
from lightgbm import LGBMClassifier
import xgboost as xgb
from sklearn import linear_model
from keras.models import Sequential
```

```
from keras.layers import LSTM, Dense
from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score, recall_score, precision_score, f1_score, matthews_corrcoef, balanced_accuracy_score
from sklearn.metrics import make_scorer, confusion_matrix, ConfusionMatrixDisplay, roc_auc_score, roc_curve, auc, precision_recall_curve
from sklearn.metrics import roc_auc_score as ras

# libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
import folium
from folium.plugins import HeatMap
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import plotly.express as px
from scipy.stats import randint, uniform
```

Figure 2: Installed Python Packages

6 Data Preprocessing

This section details the steps carried out prior to the model implementation, after reading the data, the data was then cleaned and transformed.

1. Check datatypes of columns.

```
credit_card_data.dtypes

trans_date_trans_time      object
cc_num                      int64
merchant                   object
category                   object
amt                       float64
first                      object
last                       object
gender                     object
street                     object
city                       object
state                      object
zip                         int64
lat                       float64
long                      float64
city_pop                    int64
job                        object
dob                        object
trans_num                  object
unix_time                   int64
merch_lat                 float64
merch_long                float64
isFraud                     int64
dtype: object
```

2. Check for missing data and duplicates.

```python
# Check for duplicate records
duplicates = credit_card_data.duplicated()

# Count the number of duplicate records
num_duplicates = duplicates.sum()

# Print the number of duplicate records
print(f"Number of duplicate records: {num_duplicates}")

print("--------------------------------------------------")

# check if there are missing values in the dataset

missing_values = credit_card_data.isnull().sum()

# Print the count of missing values for each column
print("Missing Values:")
print(missing_values)

sns.heatmap(credit_card_data.isnull(), cmap='viridis')
```

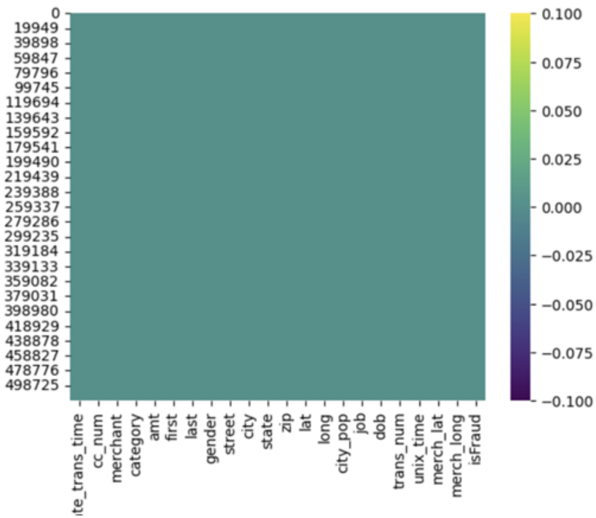

Figure 3: Checking for missing data.

3. Transform the datatypes of columns for feature engineering. The features 'dob', 'trans_date_trans_time' and 'unix_time' were converted to datetime columns and the 'gender' column was converted to Boolean values of 0 and 1. Also, the columns 'zip' and 'cc_num' were converted to string objects as shown in figure 4.

```
# Convert numerical columns to string
nominal_columns = ['cc_num', 'zip']
credit_card_data[nominal_columns] = credit_card_data[nominal_columns].astype(str)

# Convert 'trans_date_trans_time' to datetime variable
credit_card_data['trans_date_trans_time'] = pd.to_datetime(credit_card_data['trans_date_trans_time'])

# Convert 'trans_date_trans_time' to datetime
credit_card_data['dob'] = pd.to_datetime(credit_card_data['dob'])

# Convert 'unix_time' to datetime
credit_card_data['unix_time'] = pd.to_datetime(credit_card_data['unix_time'], unit='s')

# Map gender values to numerical values
gender_mapping = {'M': 0, 'F': 1}
credit_card_data['gender'] = credit_card_data['gender'].map(gender_mapping)

# categorical_cols = credit_card_data.select_dtypes(include='object').columns
# encoder = LabelEncoder()
# credit_card_data[categorical_cols] = credit_card_data[categorical_cols].apply(encoder.fit_transform)

# check the data types after transformation
credit_card_data.dtypes
```

```
trans_date_trans_time    datetime64[ns]
cc_num                           object
merchant                         object
category                         object
amt                             float64
first                            object
last                             object
gender                           object
street                           object
city                             object
state                            object
zip                              object
lat                             float64
long                            float64
city_pop                          int64
job                              object
dob                      datetime64[ns]
trans_num                        object
unix_time                datetime64[ns]
merch_lat                       float64
merch_long                      float64
isFraud                           int64
dtype: object
```

Figure 4: Data transformation

7 Data Exploration

This section contains the steps carried out for exploratory data analysis (EDA). The relationship among the variables was explored and their relationship with the target variable (`isFraud`).

1. Figure 5 shows a chart of the distribution of fraudulent transactions by shopping categories.

```
custom_colors = ['#f7baad', '#FFB933']
trans_label = ['gas_transport','misc_pos','shopping_pos','grocery_pos',
               'misc_net','shopping_net','grocery_net','entertainment',
               'food_dining','health_fitness','travel','personal_care',
               'kids_pets','home']

# Plotting pie chart for fraudulent transactions
fig = px.pie(fraudulent_transactions,
             values=quantity_fraud,
             names=trans_label,
             hole=0.4,
             title="Distribution of Transaction Type for Fraudulent Transactions"
             #color_discrete_sequence=custom_colors)
fig.show()
```

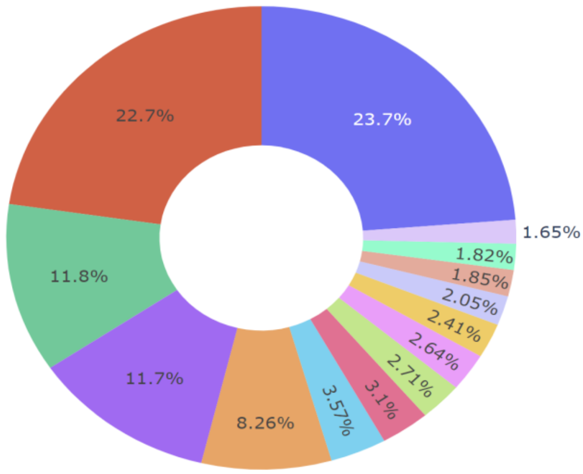

Figure 5: Code and Result of fraudulent transactions vs shopping category

2. The pattern of fraudulent transactions by amount was also explored as seen in figure 6 below.

```
# plot the distribution of fraudulent transactions by amount
ax=sns.histplot(x='amt',data=credit_card_data[credit_card_data.amt<=1000],
                hue='isFraud',stat='percent',multiple='dodge',
                common_norm=False,bins=25)
ax.set_ylabel('Percentage in Each Transaction Class')
ax.set_xlabel('Transaction Amount in USD')
plt.legend(title='Class', labels=['Fraud', 'Not Fraud'])
```

```
<matplotlib.legend.Legend at 0x7c4f6c88e7d0>
```

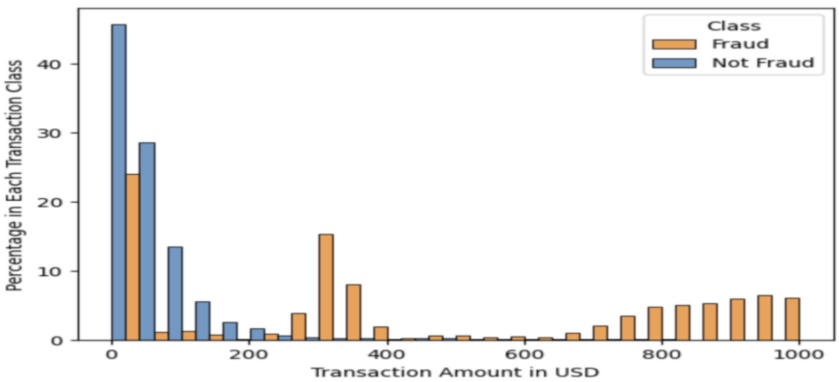

Figure 6: Distribution of fraudulent transactions by amount

3. Figure 7 depicts how the card holders gender affect the fraudulent transactions.

```
# Fraudulent transactions volume by gender
ax=sns.histplot(x='gender',data=credit_card_data, hue='isFraud',stat='percent',
                multiple='dodge',common_norm=False)
ax.set_ylabel('Percentage')
ax.set_xlabel('Credit Card Holder Gender')
plt.legend(title='CLass', labels=['Fraud', 'Not Fraud'])
```

```
<matplotlib.legend.Legend at 0x7c4fcc934b50>
```

Figure 7: Gender vs isFraud

4. An exploration of fraudulent transactions by age can be seen in figure 8.

```
ax=sns.kdeplot(x='age',data=credit_card_data, hue='isFraud', common_norm=False)
ax.set_xlabel('Credit Card Holder Age')
ax.set_ylabel('Density')
plt.xticks(np.arange(0,110,5), rotation=90)
plt.title('Age Distribution in Fraudulent vs Non-Fraudulent Transactions')
plt.legend(title='Class', labels=['Fraud', 'Not Fraud'])
```

```
<matplotlib.legend.Legend at 0x7c4fcb6a69b0>
```

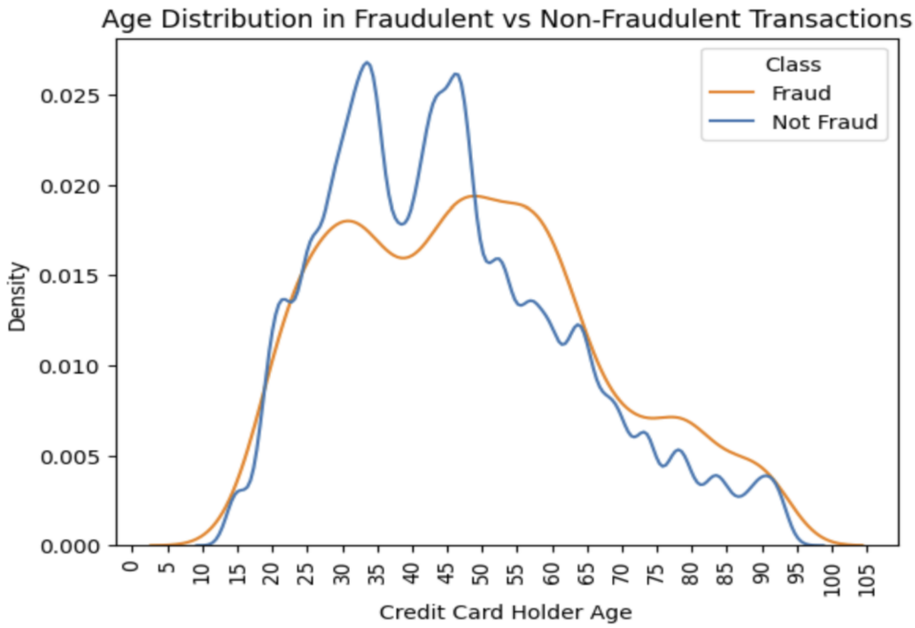

Figure 8: Exploration of fraudulent transactions by age.

8 Feature Engineering

Feature engineering was carried out on the dataset, to create new variables that may enhance the performance of the models. The variables generated include ‘age, ‘hour’, ‘week_day’, ‘month’, ‘cust_merc_lat_dist’ and ‘cust_merc_long_dist’.

```
# create an 'age' variable using the 'trans_date_trans_time' and 'dob' variable
credit_card_data['age'] = (credit_card_data['trans_date_trans_time'] - credit_card_data['dob']).dt.days // 365
credit_card_data['hour'] = pd.to_datetime(credit_card_data['trans_date_trans_time']).dt.hour
credit_card_data['week_day'] = pd.to_datetime(credit_card_data['trans_date_trans_time']).dt.dayofweek
credit_card_data['month'] = pd.to_datetime(credit_card_data['trans_date_trans_time']).dt.month
credit_card_data['cust_merch_lat_dist'] = abs(round(credit_card_data['merch_lat']-credit_card_data['lat'],3))
credit_card_data['cust_merch_long_dist'] = abs(round(credit_card_data['merch_long']-credit_card_data['long'],3))
```

Figure 9: Feature Engineering

9 Feature Selection

Based on the EDA and feature engineering carried out, the final features selected for the model building was determined and can be seen in figure 10.

```
credit_card_data_to_use = credit_card_data[['category', 'amt', 'gender', 'state',
                                            'city_pop', 'isFraud', 'age', 'hour',
                                            'week_day', 'month', 'cust_merch_lat_dist',
                                            'cust_merch_long_dist']]
```

Figure 10: feature selection.

10 Split Data into Train and Test Dataframes

The section contains the steps carried out to split the data into test and training sets to be used for training and testing the models.

```
# Split the dataset into input features (X) and target variable (y)
y = credit_card_data_to_use['isFraud']
X = credit_card_data_to_use.drop(['isFraud'], axis=1)

X.info()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

Figure 11: Split Data.

11 Class Imbalance

```
smote_enn = SMOTEENN(random_state=42)
X_resampled, y_resampled = smote_enn.fit_resample(X_train, y_train)
```

```
Class distribution before SMOTE-ENN:
0    412514
1      2422
Name: isFraud, dtype: int64
Class distribution after SMOTE-ENN:
1    409117
0    401058
Name: isFraud, dtype: int64
```

Figure 12: handling class imbalance

To handle the class imbalance on the target class, the hybrid technique SMOTE-ENN was used as shown in figure 12. The distributions of fraudulent transaction before the resampling and after the resampling can also be seen in figure 12. Balancing was applied solely to the training data, as it forms the basis for model construction. Balancing the test data is unnecessary, as the test data's role is to emulate the model's performance in a real-world scenario, where imbalanced credit card fraud datasets are prevalent.

12 Helper Methods

To avoid repetition of code, some helper methods were created to help with generating model evaluation results.

```
# this method was created to plot the Precision-Recall
# Receiver Operating Characteristics Graph for the models evaluation
def plot_pr_roc_curve(recall, precision, name):
  # calculate the no skill line as the proportion of the positive class
  no_skill = len(y_test[y_test==1]) / len(y)
  # plot the no skill precision-recall curve
  pyplot.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
  # plot the model precision-recall curve
  pyplot.title("PR ROC curve plot")
  pyplot.plot(recall, precision, marker='.', label=name)
  # axis labels
  pyplot.xlabel('Recall')
  pyplot.ylabel('Precision')
  # show the legend
  pyplot.legend()
  # show the plot
  pyplot.show()

# this method displays the evaluation results of the models
# the accuracy, classification report, recall, MCC and
# other metrics are displayed
def model_evaluation(test, pred):
  print("model accuracy: \n", accuracy_score(test, pred))
  print("classification report: \n", classification_report(test,pred))
  print("-------------------------------------------------------------")
  print("Recall:", recall_score(test, pred))
  print("Precision:", precision_score(test, pred))
  print("F1 Score:", f1_score(test, pred))
  print("MCC:", matthews_corrcoef(test, pred))
  print("Geometric Mean:", balanced_accuracy_score(test, pred))
```

```
# this method computes the confusion matrix of the models
def display_confusion_matrix(test, pred):
  cm = confusion_matrix(test, pred)
  cmd = ConfusionMatrixDisplay(cm, display_labels=['Non-Fraudulent','Fraudulent'])
  cmd.plot()

# this method displays a chart of the most important feature
#used in training the model
def display_important_features(model, name):
  # Get important feature from the trained model
  feature_names = X_test.columns.tolist()
  if hasattr(model, 'feature_importances_'):
        # For tree-based models, use feature_importances_
        features = model.feature_importances_
  else:
      # For linear models like logistic regression, use coefficient magnitudes
      features = np.abs(model.coef_[0])

  # Get the indices of features sorted by their importance in descending order
  sorted_feature_indices = np.argsort(features)[::-1]

  # plot the features in a bar chart
  plt.figure(figsize=(10, 6))
  plt.bar(range(len(features)), features[sorted_feature_indices], align='center')
```

Figure 13: Helper Methods

13 Model Implementation and Evaluation

The models were implemented using the Keras and Sklearn Python libraries, and the results were computed using sklearn.metrics library. The following sections highlight each of the models' implementation.

13.1 Logistic Regression

Figure 14 and 15 show the code snippet for hyperparameter tuning and model building for the logistic regression model.

```
# Define the hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga'],
    'class_weight': [None, 'balanced'],
    'max_iter': [10, 20]
}

# Create a LogisticRegression model
logreg = LogisticRegression()

random_search = RandomizedSearchCV(logreg, param_distributions=param_grid, n_iter=10,
                                   scoring=make_scorer(auc, greater_is_better=True,
                                   needs_proba=True, roc_curve=precision_recall_curve),
                                   cv=3, random_state=42, n_jobs=-1)

# Perform hyperparameter tuning using GridSearchCV
#grid_search = GridSearchCV(logreg, param_grid, cv=3, scoring=make_scorer(auc, greater_
random_search.fit(X_resampled, y_resampled)

# Get the best parameters
best_params = random_search.best_params_
```

Figure 14: hyperparameter tuning for Logistic Regression

```
# Initialize the Logistic Regression classifier
logreg_classifier = LogisticRegression(
    C=best_params['C'],
    penalty=best_params['penalty'],
    solver=best_params['solver'],
    class_weight=best_params['class_weight'],
    max_iter=1000
)

# Train the classifier on the training data
logreg_classifier.fit(X_resampled, y_resampled)

# Make predictions on the test data
y_pred = logreg_classifier.predict(X_test)
```

Figure 15: Optimized Logistic Regression model

13.2 Random Forest

The random forest model was built with default parameters, as shown in figure 16.

```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf_classifier.fit(X_resampled, y_resampled)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)
```

Figure 16: Random forest model

13.3 LightGBM

Prior to training the model, randomized 3-fold cross validation was performed to optimize the hyperparameters (figure 14) after which the selected hyperparameters were applied to the model.

```
# Define the parameter distribution for hyperparameter tuning
param_grid = {
    'n_estimators': [10, 50, 100, 200, 300, 500, 1000],
    'max_samples': [0.3, 0.5, 1.0],
    'max_features': [0.3, 0.5, 1.0]
}

# create lightGBM classifier
lgb_model = LGBMClassifier()

# Perform RandomizedSearchCV for hyperparameter tuning
random_search = RandomizedSearchCV(
    estimator=lgb_model,
    param_distributions=param_grid,
    n_iter=10,  # Number of parameter settings that are sampled
    scoring=make_scorer(auc, greater_is_better=True, needs_proba=True,
                        roc_curve=precision_recall_curve),  # Use a sui
    cv=3,  # Number of cross-validation folds
    verbose=1,
    random_state=42,
    n_jobs=-1  # Number of CPU cores to use (-1 uses all available core
)
random_search.fit(X_resampled, y_resampled)

# Print the best hyperparameters
print("Best parameters found:", random_search.best_params_)
```

Figure 17: hyperparameter tuning for LGBM

```
gb_model_tuned = LGBMClassifier(n_estimators=500, max_depth=6, max_samples=1.0,
                                max_features=1.0, learning_rate=0.2, subsample=0.5,
                                num_leaves=4272, colsample_by_tree=1)

# Train the model on the training data
gb_model_tuned.fit(X_resampled, y_resampled)

# Make predictions on the test data
y_pred = gb_model_tuned.predict(X_test)
```

Figure 18: Optimized LGBM model

13.4 XGBoost

The model's hyperparameters were manually selected and used to train the model, the figure below shows the code used to build the model.

```
# Create the XGBClassifier with your desired hyperparameters
xgb_model = XGBClassifier(n_estimators=500, max_depth=6, learning_rate=0.3,
            subsample=0.75, min_child_weight=1, colsample_bytree=0.5, gamma=0.2)

# Train the model on the training data
xgb_model.fit(X_resampled, y_resampled)

# Make predictions on the test data
y_pred = xgb_model.predict(X_test)
```

Figure 19: Extreme Gradient Boosting Model

13.5 Deep Learning Models (Multilayer Perceptron and LSTM)

```
# Convert the data and labels to numpy arrays
X_mlp = np.array(X_resampled)
y_mlp = np.array(y_resampled)

# Build the fully connected neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_mlp.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
batch_size = 32
epochs = 10
model.fit(X_mlp, y_mlp, batch_size=batch_size, epochs=epochs, verbose=1)

# Make predictions on the test data
y_pred = model.predict(X_test)
```

Figure 20: MLP model training

```
# Convert the data and labels to numpy arrays
X_mlp = np.array(X_resampled)
y_mlp = np.array(y_resampled)

# Build the fully connected neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_mlp.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
batch_size = 32
epochs = 10
model.fit(X_mlp, y_mlp, batch_size=batch_size, epochs=epochs, verbose=1)

# Make predictions on the test data
y_pred = model.predict(X_test)
```

Figure 21: LSTM model training

The keras package was used to build this model and the hyperparameters were chosen manually, figures 20 and 21 show the code required to build the models.

13.6 Model Evaluation

The model's evaluation implementation were similar, the same metrics were used to assess the performance of each of the models, the metrics computed include Precision Recall, F1-Score, Geometric Mean, MCC and PR-AUC receiver operating characteristics. Figure 22 shows the code used to evaluate the performance of the XGBoost model, with the help of the helper methods described in figure 13.

```
y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred]

# Evaluate the model's performance
model_evaluation(y_test, y_pred_binary)

# Get the probabilities for the positive class
positive_probs = xgb_model.predict_proba(X_test)[:, 1]

# calculate the precision-recall auc score
precision, recall, _ = precision_recall_curve(y_test, positive_probs)
auc_score_pr = auc(recall, precision)
# print the pr-auc score
print('PR AUC: %.3f' % auc_score_pr)

# plot the pr-roc curve
plot_pr_roc_curve(recall, precision, 'XGBoost model')

# display the confusion matrix
display_confusion_matrix(y_test, y_pred_binary)

# determine the most important feature ysed by the model
display_important_features(xgb_model, 'XGBoost model')
```

Figure 22: model evaluation for XGBoost model

the code above computes the classification report, the model accuracy, pr-auc curve and confusion matrix, as seen in figures 23 and 24.

```
model accuracy:
 0.9936857732276785
classification report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    103128
           1       0.48      0.84      0.61       606

    accuracy                           0.99    103734
   macro avg       0.74      0.92      0.80    103734
weighted avg       1.00      0.99      0.99    103734

-------------------------------------------------------
Recall: 0.8448844884488449
Precision: 0.4771668219944082
F1 Score: 0.6098868374032163
MCC: 0.6322867304041073
Geometric Mean: 0.9197223233493934
PR AUC: 0.804
```

Figure 23: classification report for XGBoost model

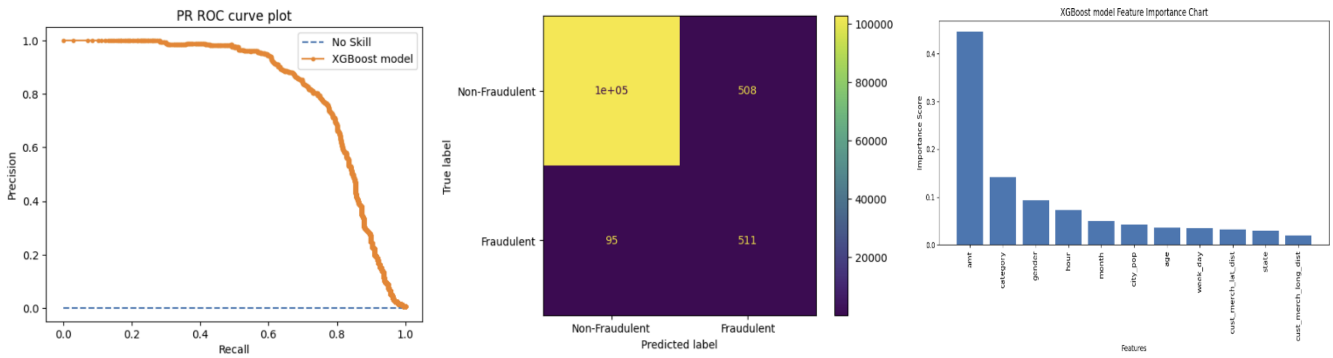

Figure 24: XGBoost model evaluation