# Configuration Manual

MSc Research Project
MSc Cybersecurity


CHIJIOKE FRANKLIN EMEJURU
Student ID: X21114382

School of Computing
National College of Ireland


Supervisor: JOEL ALEBURU

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

| | |
|---|---|
| **Student Name:** | Chijioke Franklin Emejuru |
| **Student ID:** | X21114382 |
| **Programme:** | MSc Cybersecurity **Year:** 2024 |
| **Module:** | MSc Research Project |
| **Lecturer:** | Joel Aleburu |
| **Submission Due Date:** | 12-12-2024 |
| **Project Title:** | Optimizing Fraudulent Transaction Detection In E-Commerce: A Comparative Analysis of Machine Learning And Deep Learning Algorithms With Time And CPU Performance Tracking. |
| **Word Count:** | **994 Page Count: 11** |

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:** Chijioke Franklin Emejuru

**Date:** 12-12-2024

**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): | |

**PROJECT MANUAL CONFIGURATION**

# OPTIMISING FRAUDULENT TRANSACTION DETECTION IN E-COMMERCE: A COMPARATIVE ANALYSIS OF MACHINE LEARNING AND DEEP LEARNING ALGORITHMS WITH TIME AND CPU PERFORMANCE TRACKING:

**CHIJIOKE FRANKLIN EMEJURU**
**X21114382**

## PRODUCT OVERVIEW:

This research aimed to improve online fraud detection in e-commerce using machine learning and deep learning models. As online shopping grows, cybercriminals are getting smarter, and old fraud detection methods are no longer effective. The study compared seven machine learning models and found that Random Forest, Xgboost, and Gradient Boosting performed best in detecting fraudulent transactions. The research highlighted the importance of data balancing and model selection for effective fraud detection. Future work should focus on using advanced techniques, real-time data, and increasing transparency to improve detection rates and make online payments safer.

## MATERIALS AND TOOLS UTILIZED

- Hardware: NVIDIA RTX 4090 for deep learning
- Software and Libraries: Python v3.11 for system language. Pandas and numpy for data manipulation, matplotlib and seaborn for visualization, tensorflow, a deep learning framework for building neural networks.

**Code Implementation:**

**IMPORTING THE REQUIRED LIBRARIES:**

- Here I imported the libraries that I used throughout the whole Project.

File Edit View Run Kernel Settings Help — Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

### IMPORTING THE REQUIRED LIBRARIES

```python
[3]:
import pandas as pd # Importing pandas library and aliasing it as pd
import numpy as np # Importing numpy library and aliasing it as np
import time # tracks time for a certain operation
import psutil # tracks cpu usage
import threading # allows continous monitoring
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split, cross_val_score
import matplotlib.pyplot as plt # Importing pyplot module from matplotlib library and aliasing it as plt
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import RandomOverSampler # the oversampler
from sklearn.linear_model import LogisticRegression # logistic regression
from sklearn.ensemble import RandomForestClassifier # random forest
from xgboost import XGBClassifier # extreme gradient boost
from sklearn.ensemble import GradientBoostingClassifier # gradient boost
from sklearn.tree import DecisionTreeClassifier # decision trees
from sklearn.svm import SVC # support vector classifier
import seaborn as sns # Importing seaborn library and aliasing it as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, precision_recall_curve, roc_curve, auc, confusion_mat
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)   # Setting pandas option to display all columns in DataFrame
plt.style.use('ggplot')  # Setting plot style to 'ggplot' from matplotlib
```

**READING THE DATASET:**

- I Read the dataset and visualise the dataset.

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)

```python
plt.style.use('ggplot')  # Setting plot style to 'ggplot' from matplotlib
```

### READING THE DATASET

```python
[5]:
# Reading data from 'metaverse_transactions_dataset' into data DataFrame
data = pd.read_csv("metaverse_transactions_dataset.csv")
```

```python
[6]:
# Displaying the DataFrame
data.head()
```

[6]:

| | timestamp | hour_of_day | sending_address | receiving_address | amount | transaction_type | location_region | ip |
|---|---|---|---|---|---|---|---|---|
| 0 | 2022-04-11 12:47:27 | 12 | 0x9d32d0bf2c00f41ce7ca01b66e174cc4dcb0c1da | 0x39f82e1c09bc6d7baccc1e79e5621ff812f50572 | 796.949206 | transfer | Europe | |
| 1 | 2022-06-14 19:12:46 | 19 | 0xd6e251c23cbf52dbd472f079147873e655d8096f | 0x51e8fbe24f124e0e30a614e14401b9bbfed5384c | 0.010000 | purchase | South America | |
| 2 | 2022-01-18 16:26:59 | 16 | 0x2e0925b922fed01f6a85d213ae2718f54b8ca305 | 0x52c7911879f783d590af45bda0c0ef2b8536706f | 778.197390 | purchase | Asia | |
| 3 | 2022-06-15 09:20:04 | 9 | 0x93efefc25fcaf31d7695f28018d7a11ece55457f | 0x8ac3b7bd531b3a833032f07d4e47c7af6ea7bace | 300.838358 | transfer | South America | |
| 4 | 2022-02-18 14:35:30 | 14 | 0xad3b8de45d63f5cce28aef9a82cf30c397c6ceb9 | 0x6fdc047c2391615b3facd79b4588c7e9106e49f2 | 775.569344 | sale | Africa | |

**DATA PREPROCESSING:**

- I looked at the unique values because One needs to understand the dataset very well to know which column needs to be encoded, which column needs to be dropped, which column has numerical values, and which column has String values like this here.

LOOKING AT THE UNIQUE VALUES TO KNOW WHICH IS IMPORTANT IN THE PREDICTIVE MODELLING.

```
[9]: print(f"Number of Unique IDS: {data['anomaly'].unique()}")
Number of Unique IDS: ['low_risk' 'moderate_risk' 'high_risk']

[10]: print(f"Number of Unique IDS: {data['login_frequency'].unique()}")
Number of Unique IDS: [3 5 8 6 4 1 2 7]

[11]: print(f"Number of Unique IDS: {data['age_group'].unique()}")
Number of Unique IDS: ['established' 'veteran' 'new']

[12]: print(f"Number of Unique IDS: {data['purchase_pattern'].unique()}")
Number of Unique IDS: ['focused' 'high_value' 'random']

[13]: print(f"Number of Unique IDS: {data['location_region'].unique()}")
Number of Unique IDS: ['Europe' 'South America' 'Asia' 'Africa' 'North America']

[14]: print(f"Number of Unique IDS: {data['transaction_type'].unique()}")
Number of Unique IDS: ['transfer' 'purchase' 'sale' 'phishing' 'scam']
```

## EXPLORATORY DATA ANALYSIS:

- This is the Exploratory Data analysis and the analysis here. Which is self-explanatory.

### EXPLORATORY DATA ANALYSIS

TRANSACTION TYPE AND ANOMALY RELATED

```
[17]: plt.figure(figsize=(15, 10))
ax = sns.countplot(x="transaction_type", hue="anomaly", data=data)
for i in ax.patches:
    ax.annotate(f"{i.get_height()}", (i.get_x() + i.get_width() / 2., i.get_height()), ha="center", va="center", fontsize=11,
    color = "black", xytext = (0, 5), textcoords = "offset points")
    ax.tick_params(axis="both", which="major", labelsize=12)
plt.show()
```


## DROPPING OFF REDUNDANT COLUMNS:

- This is the Drop off Redundant column just like this timestamp, sending address, receiving address.

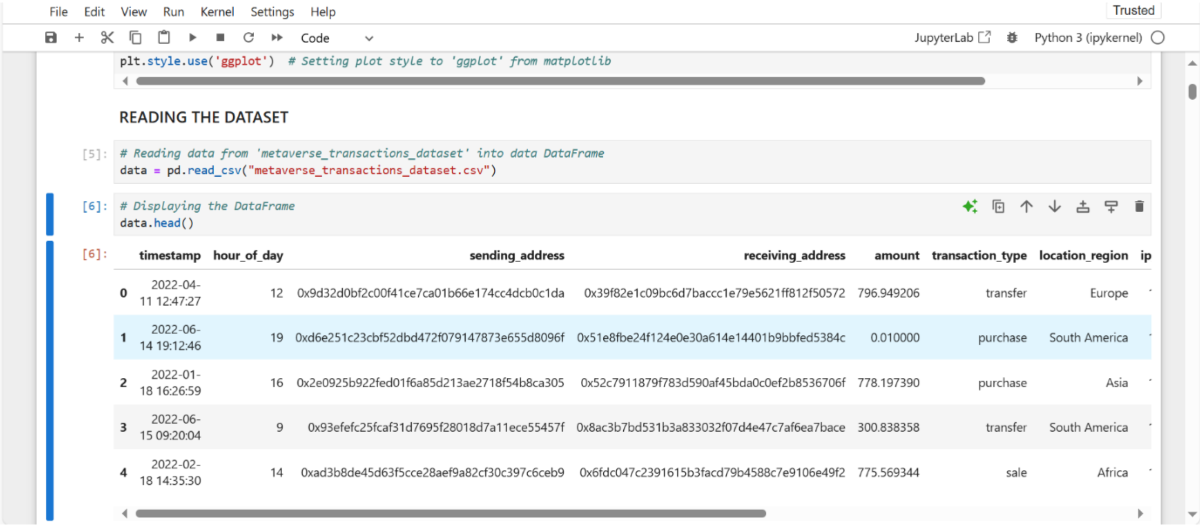
### DROPPING OFF REDUNDANT COLUMNS ¶

```
[29]: # list of columns to drop
COLUMNS_TO_DROP = ["timestamp", "sending_address", "receiving_address", "ip_prefix"]

[30]: # Drop columns
data = data.drop(COLUMNS_TO_DROP, axis=1)

[31]: # looking at new dataframe
data.head()
```

[31]:

| | hour_of_day | amount | transaction_type | location_region | login_frequency | session_duration | purchase_pattern | age_group | risk_score | anomaly |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 12 | 796.949206 | transfer | Europe | 3 | 48 | focused | established | 18.75 | low_risk |
| 1 | 19 | 0.010000 | purchase | South America | 5 | 61 | focused | established | 25.00 | low_risk |
| 2 | 16 | 778.197390 | purchase | Asia | 3 | 74 | focused | established | 31.25 | low_risk |
| 3 | 9 | 300.838358 | transfer | South America | 8 | 111 | high_value | veteran | 36.75 | low_risk |
| 4 | 14 | 775.569344 | sale | Africa | 6 | 100 | high_value | veteran | 62.50 | moderate_risk |

- If you look at the Read data set, it has a timestamp, sending address, receiving address and amount. You will see that they are useless for the training because I am not doing Blockchain, so I don't need all those things here and that's why they were dropped here.



File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
plt.style.use('ggplot')  # Setting plot style to 'ggplot' from matplotlib
```

READING THE DATASET

[5]:
```
# Reading data from 'metaverse_transactions_dataset' into data DataFrame
data = pd.read_csv("metaverse_transactions_dataset.csv")
```

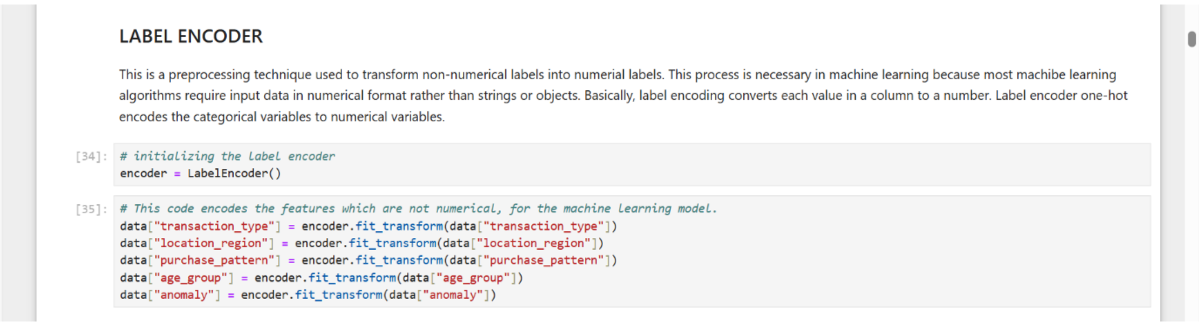
[6]:
```
# Displaying the DataFrame
data.head()
```

[6]:

| | timestamp | hour_of_day | sending_address | receiving_address | amount | transaction_type | location_region | ip |
|---|---|---|---|---|---|---|---|---|
| 0 | 2022-04-11 12:47:27 | 12 | 0x9d32d0bf2c00f41ce7ca01b66e174cc4dcb0c1da | 0x39f82e1c09bc6d7baccc1e79e5621ff812f50572 | 796.949206 | transfer | Europe | |
| 1 | 2022-06-14 19:12:46 | 19 | 0xd6e251c23cbf52dbd472f079147873e655d8096f | 0x51e8fbe24f124e0e30a614e14401b9bbfed5384c | 0.010000 | purchase | South America | |
| 2 | 2022-01-18 16:26:59 | 16 | 0x2e0925b922fed01f6a85d213ae2718f54b8ca305 | 0x52c7911879f783d590af45bda0c0ef2b8536706f | 778.197390 | purchase | Asia | |
| 3 | 2022-06-15 09:20:04 | 9 | 0x93efefc25fcaf31d7695f28018d7a11ece55457f | 0x8ac3b7bd531b3a833032f07d4e47c7af6ea7bace | 300.838358 | transfer | South America | |
| 4 | 2022-02-18 14:35:30 | 14 | 0xad3b8de45d63f5cce28aef9a82cf30c397c6ceb9 | 0x6fdc047c2391615b3facd79b4588c7e9106e49f2 | 775.569344 | sale | Africa | |

## LABEL ENCODER:

- After dropping them I have to label and encode the dataset which is what I did here. Label encoder is just like converting your numinal variables to your categorical variables so that they will have a numeric 1 and 0 format because that is what the machine learning needs. The machine learning cannot understand something like Transfer, or Purchase. It's just 1 and 0 it understands.



LABEL ENCODER

This is a preprocessing technique used to transform non-numerical labels into numerial labels. This process is necessary in machine learning because most machibe learning algorithms require input data in numerical format rather than strings or objects. Basically, label encoding converts each value in a column to a number. Label encoder one-hot encodes the categorical variables to numerical variables.

[34]:
```
# initializing the Label encoder
encoder = LabelEncoder()
```

[35]:
```
# This code encodes the features which are not numerical, for the machine learning model.
data["transaction_type"] = encoder.fit_transform(data["transaction_type"])
data["location_region"] = encoder.fit_transform(data["location_region"])
data["purchase_pattern"] = encoder.fit_transform(data["purchase_pattern"])
data["age_group"] = encoder.fit_transform(data["age_group"])
data["anomaly"] = encoder.fit_transform(data["anomaly"])
```

## CHECKING FOR MISSING VALUES:

- This is the missing Value checking, and you will see that there are no missing values in the dataset.

```python
data["anomaly"] = encoder.fit_transform(data["anomaly"])
```

CHECKING FOR MISSING VALUES

```python
# Counting missing values in each column
missing_values_count = data.isnull().sum()

# Calculating the proportion of missing values for each column
missing_values_proportion= data.isnull().sum() / len(data)

# Combining count and proportion into one DataFrame for a clean summary
missing_values_summary = pd.DataFrame({
    'Missing Values': missing_values_count,
    'Proportion': missing_values_proportion
})
# Displaying the summary table
print(missing_values_summary)
```

```
                  Missing Values  Proportion
hour_of_day                    0         0.0
amount                         0         0.0
transaction_type               0         0.0
location_region                0         0.0
login_frequency                0         0.0
session_duration               0         0.0
purchase_pattern               0         0.0
age_group                      0         0.0
risk_score                     0         0.0
```

## PLOTTING THE CORRELATION MATRIX:

- I plotted the correlational Matrix to check the correlation between the features in the dataset.

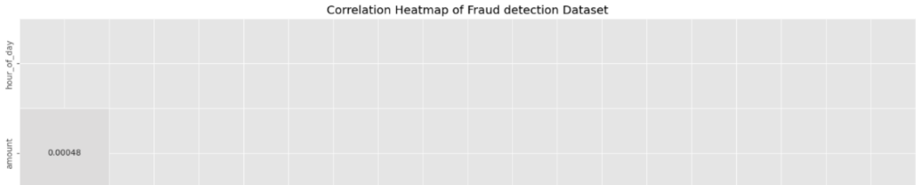
PLOTTING THE CORRELATION MATRIX

A correlation matrix is a table showing correlation coefficients between features in a data. It ranges between -1 and 1. If two features have a high correlation it implies that as one variable increases, the other variable increases.

```python
# initialize the correlation matrix
corr = data.corr()
```

```python
# Create heatmap
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(30, 20))
sns.heatmap(corr, mask=mask, vmax=0.3, center=0, square=True, linewidths=0.5,cbar_kws={"shrink":0.5}, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap of Fraud detection Dataset")
plt.show()
```

Correlation Heatmap of Fraud detection Dataset




## VISUALIZING THE DISTRIBUTION OF THE LABEL OR TARGET VARIABLE:

- I potted a target because I have to check whether the dataset is balanced or not, and since it is not balanced, I have to treat it using the balancing technique that randomly samples our target variables based on the minority and majority classes. From this place, the majority class is 80.8%


VISUALIZING THE DISTRIBUTION OF THE LABEL OR TARGET VARIABLE

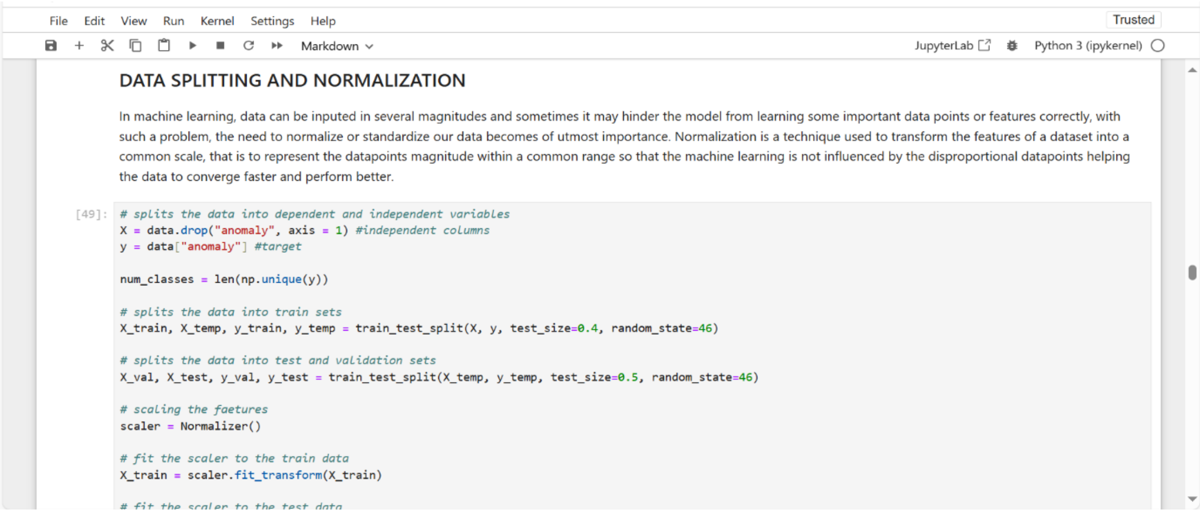
```
[43]: # initialize the value counts of the labels
label_counts = data["anomaly"].value_counts()
```

```
[44]: # create plot
plt.figure(figsize=(5,5))
plt.pie(label_counts, labels=label_counts.index, autopct="%1.1f%%", startangle=140)
plt.title("Label distribution")
plt.axis("equal")
plt.show()
```



## DATA SPLITTING AND NORMALIZATION:

- After splitting the dataset, you will see the trained text split and I normalised it using normaliser because I tested using standard scaler, Min Max scaler, and robust scaler. I noticed that all of them gave an overfitting value, so I now transformed this test and validation set using the scaler that I did.

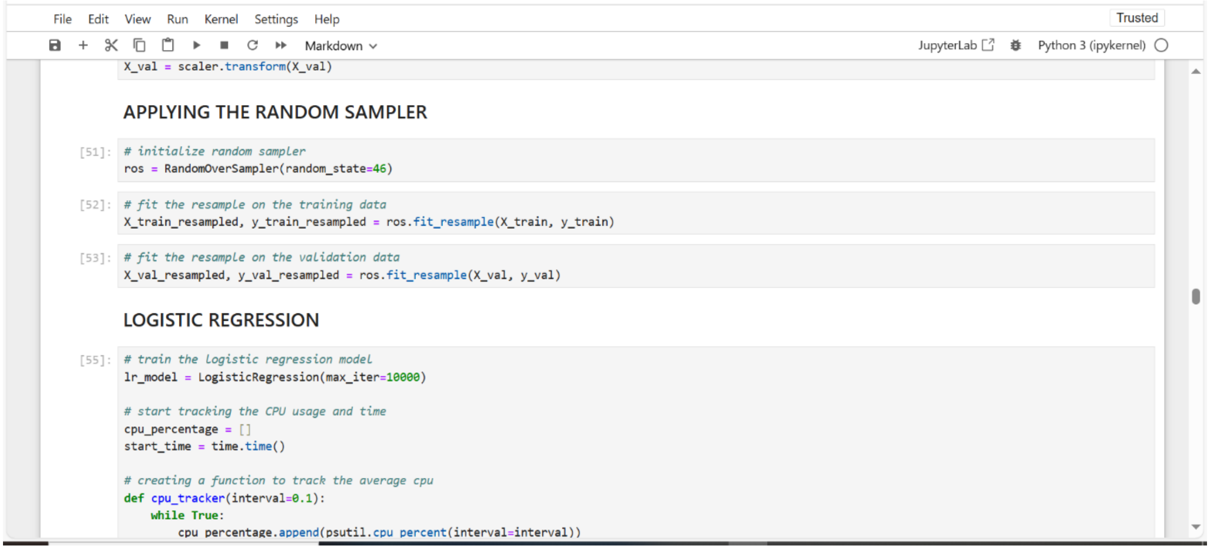

DATA SPLITTING AND NORMALIZATION

In machine learning, data can be inputed in several magnitudes and sometimes it may hinder the model from learning some important data points or features correctly, with such a problem, the need to normalize or standardize our data becomes of utmost importance. Normalization is a technique used to transform the features of a dataset into a common scale, that is to represent the datapoints magnitude within a common range so that the machine learning is not influenced by the disproportional datapoints helping the data to converge faster and perform better.

```
[49]: # splits the data into dependent and independent variables
X = data.drop("anomaly", axis = 1) #independent columns
y = data["anomaly"] #target

num_classes = len(np.unique(y))

# splits the data into train sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=46)

# splits the data into test and validation sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=46)

# scaling the faetures
scaler = Normalizer()

# fit the scaler to the train data
X_train = scaler.fit_transform(X_train)

# fit the scaler to the test data
```

## APPLYING THE RANDOM SAMPLER:

- This is a random Sampler that sampled the imbalanced data to make it balance. After balancing the data, the next is to train the models. This is straightforward because all you have to do is call the machine learning algorithm and fix the train and test set. This CPU and time are optional but I like using them in the machine learning project, so you can calculate the CPU and Time that was expended when training the models.



File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

```
X_val = scaler.transform(X_val)
```

APPLYING THE RANDOM SAMPLER

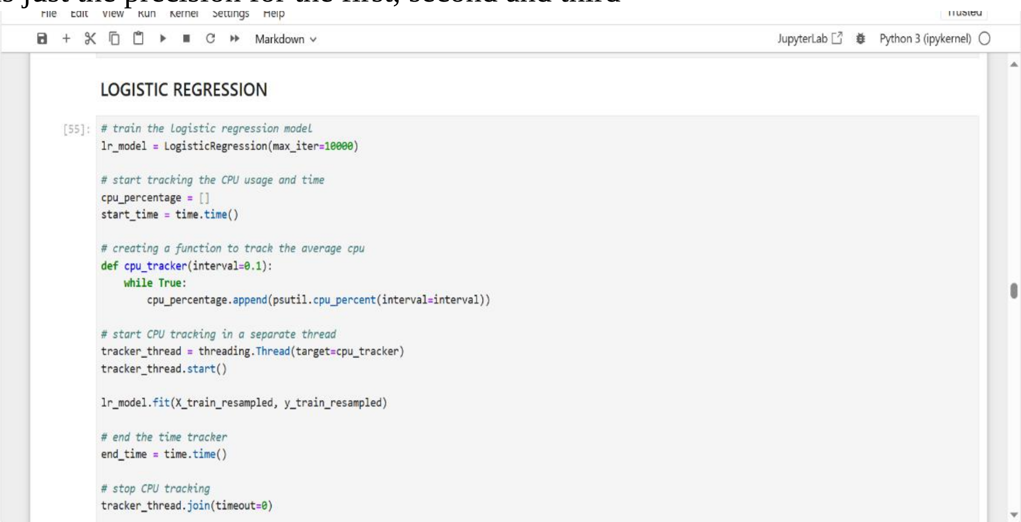
```
[51]: # initialize random sampler
ros = RandomOverSampler(random_state=46)

[52]: # fit the resample on the training data
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

[53]: # fit the resample on the validation data
X_val_resampled, y_val_resampled = ros.fit_resample(X_val, y_val)
```

LOGISTIC REGRESSION

```
[55]: # train the logistic regression model
lr_model = LogisticRegression(max_iter=10000)

# start tracking the CPU usage and time
cpu_percentage = []
start_time = time.time()

# creating a function to track the average cpu
def cpu_tracker(interval=0.1):
    while True:
        cpu_percentage.append(psutil.cpu_percent(interval=interval))
```

## LOGISTIC REGRESSION:

- Here you can see the Logistic regression, Accuracy, Precision, F1 score and Recall. This is just the precision for the first, second and third class.



Markdown JupyterLab Python 3 (ipykernel)

LOGISTIC REGRESSION

```
[55]: # train the logistic regression model
lr_model = LogisticRegression(max_iter=10000)

# start tracking the CPU usage and time
cpu_percentage = []
start_time = time.time()

# creating a function to track the average cpu
def cpu_tracker(interval=0.1):
    while True:
        cpu_percentage.append(psutil.cpu_percent(interval=interval))

# start CPU tracking in a separate thread
tracker_thread = threading.Thread(target=cpu_tracker)
tracker_thread.start()

lr_model.fit(X_train_resampled, y_train_resampled)

# end the time tracker
end_time = time.time()

# stop CPU tracking
tracker_thread.join(timeout=0)
```

## RANDOM FOREST CLASSIFIER:

- The same goes for the Random Forest, the Gradient boosting is straightforward. You call your model, you fit your train and test. This is the accuracy, precision, recall and F1 score.

```
[56]: print(f"Time taken to train the model: {training_time:.2f} seconds")
      print(f"Average CPU usage during training: {average_CPU_usage:.2f}%")

      Time taken to train the model: 2.59 seconds
      Average CPU usage during training: 58.78%

[57]: # evaluate the model predictions
      y_pred = lr_model.predict(X_test)

[58]: lr_accuracy = accuracy_score(y_test, y_pred)

[59]: print(f"Accuracy: {lr_accuracy * 100:.2f}%")

      Accuracy: 79.19%

[60]: # print the classification report
      print("Classification Report:\n", classification_report(y_test, y_pred, zero_division=1))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.80      1.00      0.89      1277
                 1       0.96      0.78      0.86     12683
                 2       0.34      0.76      0.47      1760

          accuracy                           0.79     15720
         macro avg       0.70      0.85      0.74     15720
      weighted avg       0.88      0.79      0.82     15720
```

- The Same thing for Decision tree, Support Vector and all.

**MULTILAYER PERCEPTRON:**

- For Deep Learning, One needs to familiarize oneself with the just artificial neural network with different layers of neurons like a normal human brain. Here I used tensor flow not Python.

MULTILAYER PERCEPTRON

```
[ ]: import tensorflow as tf # importing the neural network framework with alias tf
     from tensorflow.keras.models import Sequential # activating the linear layers of the neural network
     from tensorflow.keras.layers import Dense # activating the dense and fully connected layer of the neural network

[ ]: # define the neural netwrok model
     model = Sequential([
         Dense(64, input_shape=(X_train.shape[1],), activation = 'relu'),
         Dense(32, activation='relu'),
         Dense(num_classes, activation='softmax') # multi-class classification
     ])

[ ]: # model compilation
     model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

[ ]: # start tracking the CPU usage and time
     cpu_percentage = []
     start_time = time.time()

     # creating a function to track the average cpu
     def cpu_tracker(interval=0.1):
         while True:
             cpu_percentage.append(psutil.cpu_percent(interval=interval))
```

**Model Sequential:**

- Here I created a model, this is the sequential Model.

```
[ ]: # define the neural netwrok model
     model = Sequential([
         Dense(64, input_shape=(X_train.shape[1],), activation = 'relu'),
         Dense(32, activation='relu'),
         Dense(num_classes, activation='softmax') # multi-class classification
     ])
```

## Model Compilation:

- This is the Model compilation using the optimiser and loss function because the last function that was used will make sure that function reduces the error. That is the work of the loss function to reduce the error when training the model.

```
[ ]: # model compilation
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

[ ]: # start tracking the CPU usage and time
cpu_percentage = []
start_time = time.time()

# creating a function to track the average cpu
def cpu_tracker(interval=0.1):
    while True:
        cpu_percentage.append(psutil.cpu_percent(interval=interval))
```

## Categorical Cross-entropy:

I used sparse categorical cross-entropy here because I am using label encoded target. If you are just using a normal binary target, binary cross entropy would have been used.

```
[ ]: # model compilation
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

## The Training Loop:

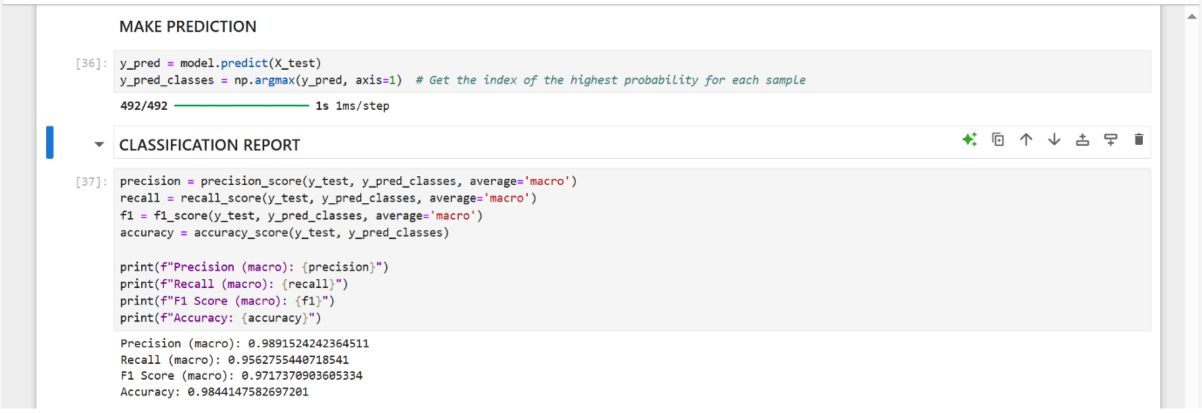
After training, I had accuracy of 98% of training loop. This is the Training Loop.

File Edit View Run Kernel Settings Help    Not Trusted
Markdown    JupyterLab    Python 3 (ipykernel)

```
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9611 - loss: 0.1003 - val_accuracy: 0.9648 - val_loss: 0.0970
Epoch 7/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9655 - loss: 0.0888 - val_accuracy: 0.9649 - val_loss: 0.0860
Epoch 8/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9714 - loss: 0.0780 - val_accuracy: 0.9728 - val_loss: 0.0716
Epoch 9/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9719 - loss: 0.0747 - val_accuracy: 0.9782 - val_loss: 0.0641
Epoch 10/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9724 - loss: 0.0715 - val_accuracy: 0.9743 - val_loss: 0.0643
Epoch 11/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9753 - loss: 0.0664 - val_accuracy: 0.9625 - val_loss: 0.0935
Epoch 12/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9755 - loss: 0.0642 - val_accuracy: 0.9764 - val_loss: 0.0593
Epoch 13/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9781 - loss: 0.0576 - val_accuracy: 0.9778 - val_loss: 0.0555
Epoch 14/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9762 - loss: 0.0596 - val_accuracy: 0.9773 - val_loss: 0.0553
Epoch 15/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9783 - loss: 0.0547 - val_accuracy: 0.9768 - val_loss: 0.0558
Epoch 16/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9798 - loss: 0.0524 - val_accuracy: 0.9770 - val_loss: 0.0563
Epoch 17/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9791 - loss: 0.0544 - val_accuracy: 0.9810 - val_loss: 0.0487
Epoch 18/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9804 - loss: 0.0500 - val_accuracy: 0.9814 - val_loss: 0.0454
Epoch 19/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9800 - loss: 0.0491 - val_accuracy: 0.9849 - val_loss: 0.0393
Epoch 20/20
1474/1474 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9816 - loss: 0.0471 - val_accuracy: 0.9833 - val_loss: 0.0422
```

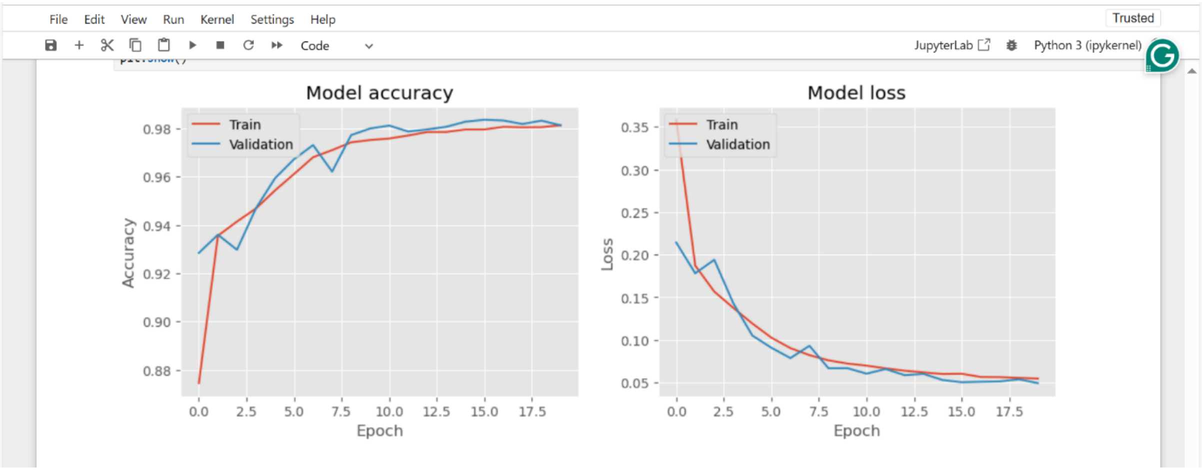
## Classification Report:

Then you make predictions, This is the test set, I used the Y-test and Y Pred to predict. After testing the accuracy I had was 98% just like the train too.




**Model Accuracy And Model Loss:**

This is the plot of the Model accuracy and the model loads because as you are training your model, it has to increase from the blue validation, you can see that the model did not overfit, just as my supervisor thought that it overfitted when he gave me feedback. As the blue is going up, the red is going up as well. The loss is going down as it should be.




**Evaluate The Test Accuracy:**

This is the Final accuracy for the test which Is 98.44% and that is the end of my project.

**References:**

**“Metaverse Financial Transaction Dataset”, [Online]. Available: https://www.kaggle.com/datasets/faizaniftikharjanjua/metaverse-financial-transactions-dataset [Accessed on 23 Oct 2024].**