National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Jeni Keziah Alfrida
Student ID: 23401044

Data Analytics
National College of Ireland

Supervisor: Hamilton Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Jeni Keziah Alfrida

Student ID: 23401044

Programme: MSc Data Analytics **Year:** 2025

Module: Research Project

Lecturer: Hamilton Niculescu

Submission Due Date: 11/12/25

Project Title: FusionGuard: Integrating LSTM-GRU with Temporal Anomaly Detection for Robust Supply Chain Fraud Identification

Word Count: 1613 **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: Jeni Keziah Alfrida

Date: 11/12/25

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

Jeni Keziah Alfrida
Student ID: 23401044

1. Introduction:

One of the most urgent issues of the modern financial ecosystem is credit card fraud as millions of payments are made every second worldwide. Real time fraud must be detected to save both the consumers and financial institutions a lot of money that is lost. The proposed project aims at creating an effective and valid credit card fraud detector based on machine learning and deep learning algorithms (Kharat et al., 2023). The main goal is to develop predictive models with the ability to effectively categorize transactions as either valid or fraudulent, with very imbalanced data.

This project used the dataset along with Kaggle which presented anonymized transactions of credit cards in Europe. The data cleaning, addressing the issue of class imbalance with the help of Synthetic Minority Over-sampling Technique (SMOTE) and the training of several models, such as Logistic Regression and a hybrid LSTM-GRU neural network, make up the workflow. The stages of the project including preprocessing and feature scaling, model training and evaluation were all developed to achieve maximum detection accuracy and minimal false positives. This project does not only illustrate the practical use of advanced analytics in financial security but also presents a reproducible framework that can be customized to the use of fraud detection systems in real-life applications of banks and e-commerce settings. The document describes the full configuration model of the credit card fraud detection system. It contains hardware and software specifications, the environment settings, as well as sequential implementation steps to be followed to replicate and confirm the findings in an effective manner.

The project also uses RandomUnderSampler in addition to SMOTE to create a second balanced training set. This provides the evaluations of both oversampling and undersampling strategies. The LSTMGRU hybrid model and the Logistic Regression are trained on the four versions of balanced data, resulting in four models, respectively. Accuracy, precision, recall, F1-score, ROC curves and AUC are used to compare their results.

2. Environment Used:

1.1 Hardware Required:

This project had its experiments performed on a personal computer of medium computational resources that are adequate to perform machine learning and deep learning tasks.

The following hardware configuration was used during the implementation:

- **System:** Any Windows/Mac/Linux compatible machine
- **Operating System:** Windows 11
- **Processor:** Intel(R) Core i5-1135G7 CPU 2.40GHz or equivalent
- **RAM:** Minimum 8 GB (recommended 16 GB for deep learning models)
- **Storage:** At least 10 GB of free disk space for dataset storage and model outputs

- **GPU:** (Optional) an integrated or dedicated NVIDIA/AMD GPU can be utilized to accelerate training of neural networks

This setup optimally allowed the seamless running of the Jupyter Notebook environment, as well as the fast processing of data and the effective training of the model in the case of the Logistic Regression and LSTM-GRU models.

1.2 Setting up Jupyter Notebook:

Jupyter Notebook was utilized to implement the project as the development environment because of its interactive capabilities and the availability of data science libraries in Python.

1. **Install Python:** Download and install the most recent and stable version of Python (3.8 or later) on the official site.
2. **Install Jupyter Notebook:** On the terminal or command prompt, run the command 'pip install notebook'.
3. **Install Dependencies:** Install all the dependencies by running
'Pip install pandas numpy scikit-learn tensorflow imbalanced-learn matplotlib seaborn joblib'.
4. **Launch the Notebook:** It is possible to start the interactive workspace with the following command: jupyter notebook.

Such an arrangement guarantees a repeatable environment in which the pre-processing of data, model construction and testing can be performed without any issues.

3. Implementation:

In this section, a description of the step-by-step process of implementing the Credit Card Fraud Detection project, such as data preprocessing, balancing, model building and evaluation, is presented. All the experiments were conducted on Python and Jupyter Notebook and the following sections are the direct code blocks of the project notebook.

1.3 Import Required Libraries:

The initial step transports all the required Python packages required in the project. These are data manipulation libraries (pandas, numpy), visualization (matplotlib, seaborn), machine learning (scikit-learn, imblearn) and deep learning modules (tensorflow). The libraries are used to back a given phase of data processing or model development.

Others are RandomUnderSampler, which is imbalanced-learn to undersample, joblib to serialize models, TensorFlow layer (GRU, BatchNormalization and Dropout), scikit-learn to draw ROC curves, compute AUC scoring and classification reports and visualize confusion matrices.

```
# ==============================
# 1. Import Required Libraries
# ==============================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, f1_score, recall_score, precision_score,
    confusion_matrix, classification_report, ConfusionMatrixDisplay,
    roc_auc_score, roc_curve, precision_recall_curve, auc
)

# Handling Imbalance
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from collections import Counter

# Deep Learning
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, GRU, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

# Saving Models
import joblib
```

1.4 Load Dataset:

The creditcard.csv (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud) dataset is loaded into a panda Data Frame. There is exploration of the data with the help of such methods as, head, info and describe to familiarize oneself with the data and find out whether some entries are missing or repeated. This data consists of anonymized attributes based on PCA conversions and one target variable called 'Class'.

Duplicate rows are also verified in the dataset and they are eliminated before outlier data processing. To check the data integrity, a summary of the missing values, the overall shape and feature types is checked.

```
# ================================================
# 2. Load Dataset
# ================================================
df = pd.read_csv('creditcard.csv')
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739

5 rows × 31 columns

1.5 Remove Outliers:

Interquartile Range (IQR) is used to exclude the outliers in the column of amounts to make sure that the model is stable. Scaling, sampling and model training are ensured to eliminate outliers to guarantee steady behavior of models and homogenous data distributions.

```
# ================================================
# 3. Remove Outliers (IQR Method on 'Amount')
# ================================================
Q1 = df['Amount'].quantile(0.25)
Q3 = df['Amount'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

df = df[(df['Amount'] >= lower) & (df['Amount'] <= upper)]
print("Shape after outlier removal:", df.shape)
```

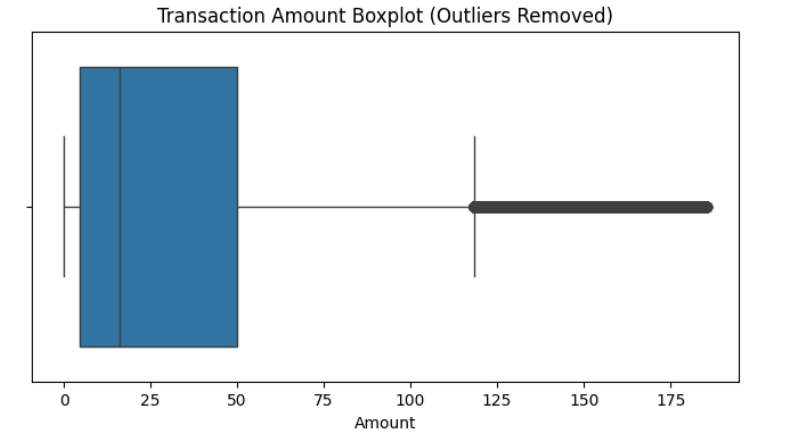
```
Shape after outlier removal: (252041, 31)
```

1.6 Data Visualization:

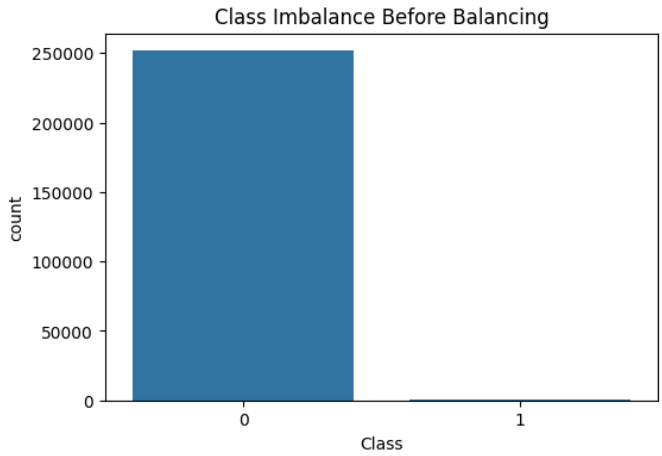
This part describes the visualization of the data to know the distribution and imbalance of the data. Boxplots help to see how the transaction amounts are distributed and countplots demonstrate the extreme asymmetry of fraudulent and non-fraudulent transactions prior to using SMOTE.

Additional visualizations include histograms of selected PCA components (V1–V5) after undersampling and a correlation heatmap generated on the balanced training set to better understand feature relationships.

```
# ================================================
# 4. Visualization
# ================================================
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['Amount'])
plt.title("Transaction Amount Boxplot (Outliers Removed)")
plt.show()
```



```
plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df)
plt.title("Class Imbalance Before Balancing")
plt.show()
```



1.7 Handling Class Imbalance:

Pre-model training involves two balancing strategies, which are applied to the training part only:

(a) Random Undersampling (RUS)

RandomUnderSampler is used to reduce the majority class. Following undersampling, feature distributions and counts of classes are plotted and a correlation heatmap is created of the balanced training data.

(b) Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE creates synthetic minor samples, which creates a second balanced training set. The Logistic Regression and the LSTM-GRU hybrid model are individually trained on the undersampled and SMOTE dataset.

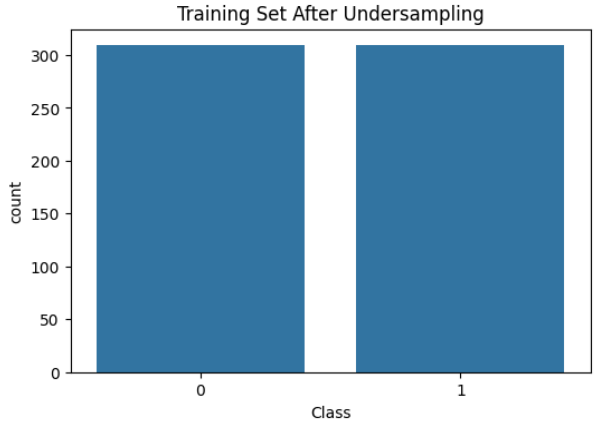
```
# ================================================
# 5. Train test split BEFORE balancing
# ================================================
X_us = df.drop('Class', axis=1)
y_us = df['Class']

scaler_us = StandardScaler()
X_us_scaled = scaler_us.fit_transform(X_us)

X_us_train, X_us_test, y_us_train, y_us_test = train_test_split(
    X_us_scaled, y_us,
    test_size=0.2,
    random_state=42,
    stratify=y_us
)

print("Original training set class distribution:", Counter(y_us_train))
print("Original test set class distribution:", Counter(y_us_test))
```

```
Original training set class distribution: Counter({0: 201323, 1: 309})
Original test set class distribution: Counter({0: 50332, 1: 77})
```

```
# ================================================
# 6. UNDERSAMPLING ON TRAINING SET ONLY
# ================================================
rus = RandomUnderSampler(random_state=42)
X_us_train_sampled, y_us_train_sampled = rus.fit_resample(X_us_train, y_us_train)

print("Training set class distribution AFTER undersampling:", Counter(y_us_train_sampled))
```

```
Training set class distribution AFTER undersampling: Counter({0: 309, 1: 309})
```

```
df_us_train_sampled = pd.DataFrame(X_us_train_sampled, columns=X_us.columns)
df_us_train_sampled['Class'] = y_us_train_sampled

plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df_us_train_sampled)
plt.title("Training Set After Undersampling")
plt.show()
```



1.8 Logistic Regression Model:

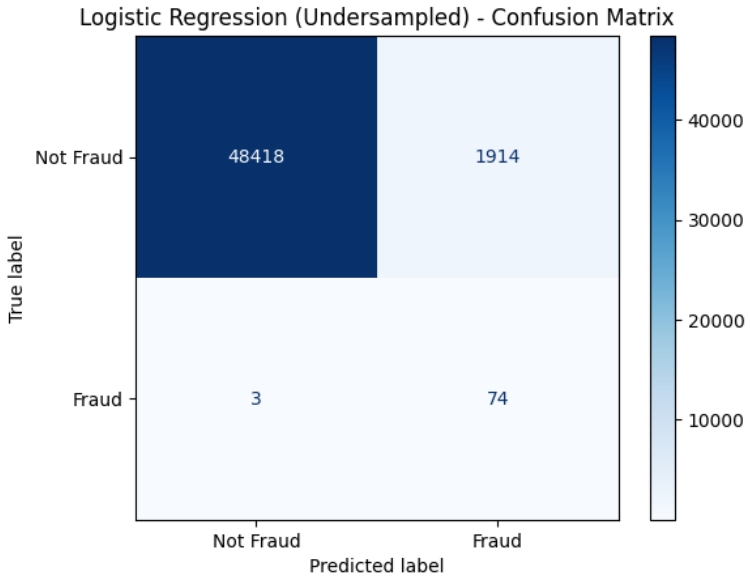
The baseline classifier is a Logistic Regression model that is trained. The balanced dataset is divided into training and testing sets and the model is tested based on the metrics of accuracy, precision, recall and F1-score. The trained model is saved to be used again too.

The Logistic Regression model is trained in two versions with undersampled training data and the SMOTE-balanced data. The models are assessed based on accuracy, precision, recall, F1-score, visualization of the confusion matrices and ROC-AUC curves. The two models are stored in joblib as they can be reused.

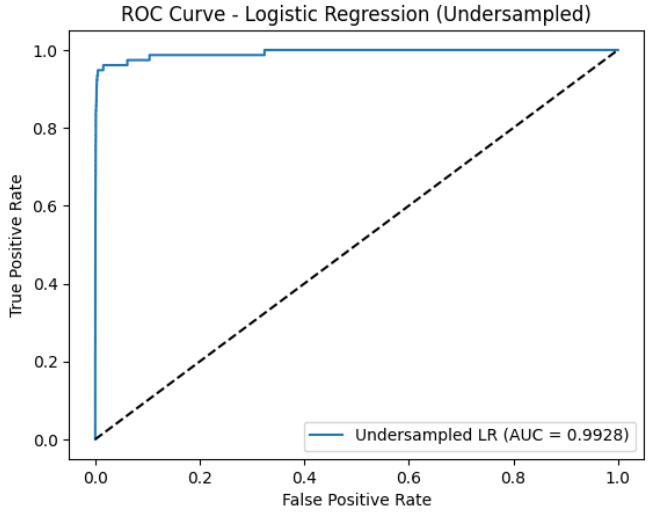
```
# ================================================
# 8. Logistic Regression Model
# ================================================
lr_us = LogisticRegression(max_iter=1000)
lr_us.fit(X_us_train_sampled, y_us_train_sampled)

y_us_proba_lr = lr_us.predict_proba(X_us_test)[:, 1]
y_us_pred_lr = lr_us.predict(X_us_test)

print("\n===== LOGISTIC REGRESSION (UNDERSAMPLED) =====")
print("Accuracy:", accuracy_score(y_us_test, y_us_pred_lr))
print("Precision:", precision_score(y_us_test, y_us_pred_lr, zero_division=0))
print("Recall:", recall_score(y_us_test, y_us_pred_lr, zero_division=0))
print("F1 Score:", f1_score(y_us_test, y_us_pred_lr, zero_division=0))
print("ROC AUC:", roc_auc_score(y_us_test, y_us_proba_lr))
print(classification_report(y_us_test, y_us_pred_lr))
```

```
===== LOGISTIC REGRESSION (UNDERSAMPLED) =====
Accuracy: 0.9619710765934655
Precision: 0.03722334004024145
Recall: 0.961038961038961
F1 Score: 0.07167070217917676
ROC AUC: 0.9928211738988183
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     50332
           1       0.04      0.96      0.07        77

    accuracy                           0.96     50409
   macro avg       0.52      0.96      0.53     50409
weighted avg       1.00      0.96      0.98     50409
```

```
cm_us = confusion_matrix(y_us_test, y_us_pred_lr)
ConfusionMatrixDisplay(cm_us, display_labels=["Not Fraud", "Fraud"]).plot(cmap="Blues")
plt.title("Logistic Regression (Undersampled) - Confusion Matrix")
plt.show()
```



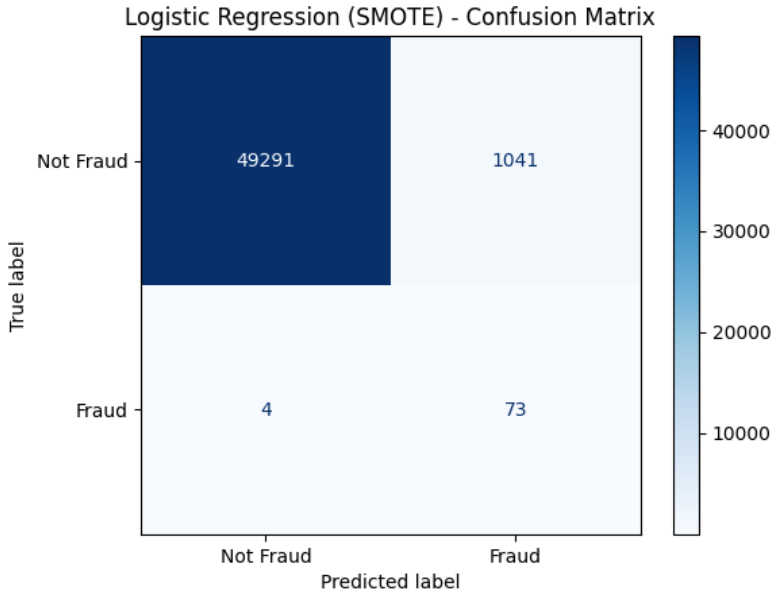

```
# ROC Curve - Undersampled LR
fpr_us_lr, tpr_us_lr, _ = roc_curve(y_us_test, y_us_proba_lr)
auc_us_lr = roc_auc_score(y_us_test, y_us_proba_lr)
plt.plot(fpr_us_lr, tpr_us_lr, label=f"Undersampled LR (AUC = {auc_us_lr:.4f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve - Logistic Regression (Undersampled)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



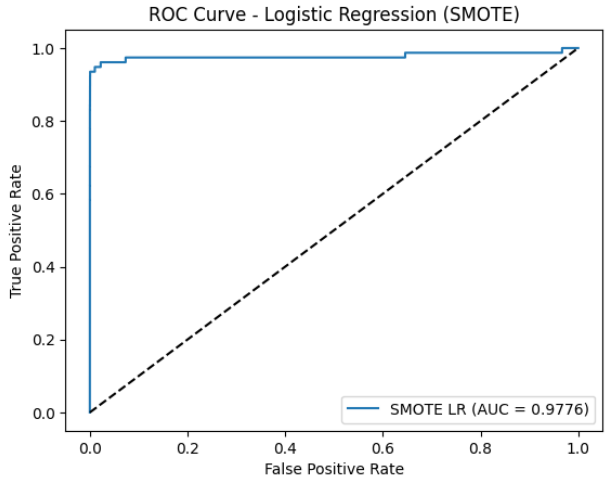
```python
# ================================================
# 3. Logistic Regression (SMOTE)
# ================================================
lr_sm = LogisticRegression(max_iter=1000)
lr_sm.fit(X_sm_train_expanded, y_sm_train_expanded)

y_sm_proba_lr = lr_sm.predict_proba(X_sm_test)[:, 1]
y_sm_pred_lr = lr_sm.predict(X_sm_test)

print("\n===== LOGISTIC REGRESSION (SMOTE) =====")
print("Accuracy:", accuracy_score(y_sm_test, y_sm_pred_lr))
print("Precision:", precision_score(y_sm_test, y_sm_pred_lr, zero_division=0))
print("Recall:", recall_score(y_sm_test, y_sm_pred_lr, zero_division=0))
print("F1 Score:", f1_score(y_sm_test, y_sm_pred_lr, zero_division=0))
print("ROC AUC:", roc_auc_score(y_sm_test, y_sm_proba_lr))
print(classification_report(y_sm_test, y_sm_pred_lr))
```

```
===== LOGISTIC REGRESSION (SMOTE) =====
Accuracy: 0.979269574877502
Precision: 0.06552962298025135
Recall: 0.948051948051948
F1 Score: 0.12258606213266163
ROC AUC: 0.9776035178363718
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     50332
           1       0.07      0.95      0.12        77

    accuracy                           0.98     50409
   macro avg       0.53      0.96      0.56     50409
weighted avg       1.00      0.98      0.99     50409
```

```python
cm_lr_sm = confusion_matrix(y_sm_test, y_sm_pred_lr)
ConfusionMatrixDisplay(cm_lr_sm, display_labels=["Not Fraud", "Fraud"]).plot(cmap="Blues")
plt.title("Logistic Regression (SMOTE) - Confusion Matrix")
plt.show()
```



```python
# ROC Curve - SMOTE LR
fpr_sm_lr, tpr_sm_lr, _ = roc_curve(y_sm_test, y_sm_proba_lr)
auc_sm_lr = roc_auc_score(y_sm_test, y_sm_proba_lr)
plt.plot(fpr_sm_lr, tpr_sm_lr, label=f"SMOTE LR (AUC = {auc_sm_lr:.4f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve - Logistic Regression (SMOTE)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



1.9 LSTM–GRU Hybrid Neural Network:

The deep-learning model is a combination of LSTM with GRU, where BatchNormalization and Dropout are used to minimize overfitting.

Two neural networks are trained:

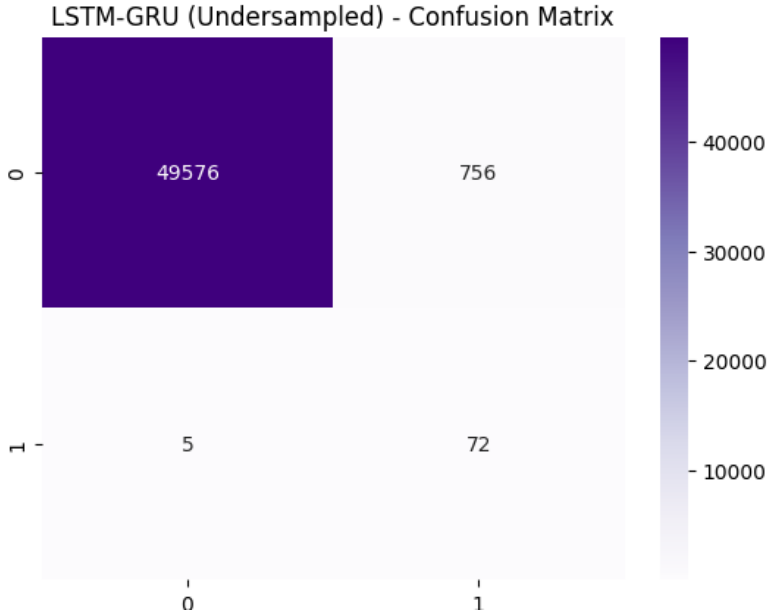
- GRU-LSTM on undersampled data.
- LMST-GRU on SMOTE-balanced data.

Both models are compared based on accuracy, precision, recall, F1-score and confusion matrices and ROC-AUC curves. All the models are stored in the form of keras.

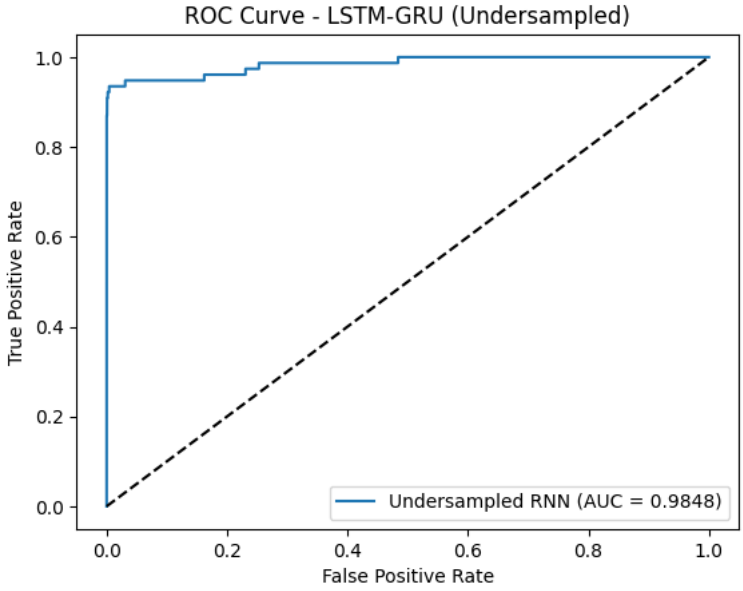
```python
print("\n===== LSTM-GRU (UNDERSAMPLED) =====")
print("Accuracy:", accuracy_score(y_us_test, y_us_pred_rnn))
print("Precision:", precision_score(y_us_test, y_us_pred_rnn, zero_division=0))
print("Recall:", recall_score(y_us_test, y_us_pred_rnn, zero_division=0))
print("F1 Score:", f1_score(y_us_test, y_us_pred_rnn, zero_division=0))
print("ROC AUC:", roc_auc_score(y_us_test, y_us_proba_rnn))
print(classification_report(y_us_test, y_us_pred_rnn))
```

```
===== LSTM-GRU (UNDERSAMPLED) =====
Accuracy: 0.9849034894562478
Precision: 0.08695652173913043
Recall: 0.935064935064935
F1 Score: 0.1591160220994475
ROC AUC: 0.9847666042929495
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     50332
           1       0.09      0.94      0.16        77

    accuracy                           0.98     50409
   macro avg       0.54      0.96      0.58     50409
weighted avg       1.00      0.98      0.99     50409
```

```
cm_us_rnn = confusion_matrix(y_us_test, y_us_pred_rnn)
sns.heatmap(cm_us_rnn, annot=True, fmt='d', cmap='Purples')
plt.title("LSTM-GRU (Undersampled) - Confusion Matrix")
plt.show()
```

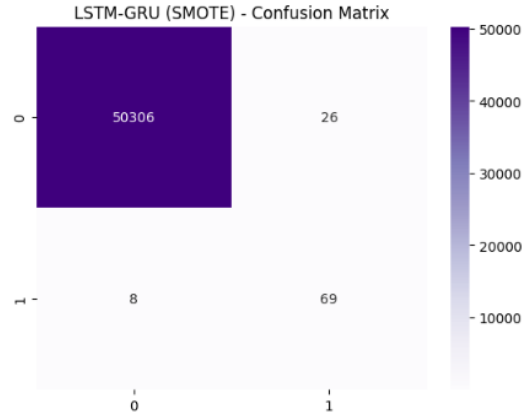



```
# ROC Curve - Undersampled RNN
fpr_us_rnn, tpr_us_rnn, _ = roc_curve(y_us_test, y_us_proba_rnn)
auc_us_rnn = roc_auc_score(y_us_test, y_us_proba_rnn)
plt.plot(fpr_us_rnn, tpr_us_rnn, label=f"Undersampled RNN (AUC = {auc_us_rnn:.4f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve - LSTM-GRU (Undersampled)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

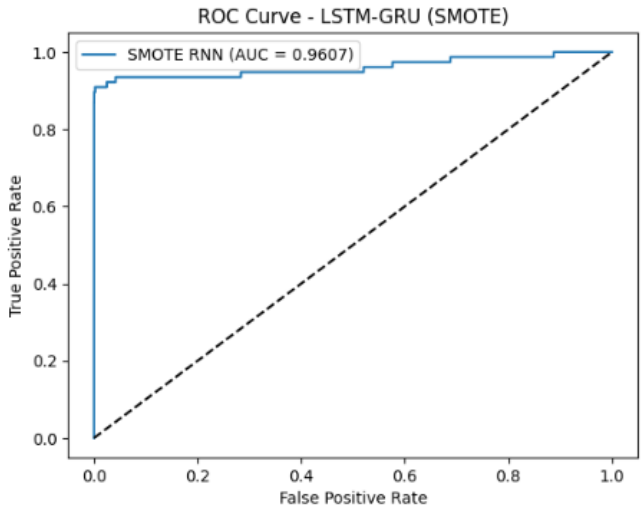


```
print("\n===== LSTM-GRU (SMOTE) =====")
print("Accuracy:", accuracy_score(y_sm_test, y_sm_pred_rnn))
print("Precision:", precision_score(y_sm_test, y_sm_pred_rnn, zero_division=0))
print("Recall:", recall_score(y_sm_test, y_sm_pred_rnn, zero_division=0))
print("F1 Score:", f1_score(y_sm_test, y_sm_pred_rnn, zero_division=0))
print("ROC AUC:", roc_auc_score(y_sm_test, y_sm_proba_rnn))
print(classification_report(y_sm_test, y_sm_pred_rnn))

cm_sm_rnn = confusion_matrix(y_sm_test, y_sm_pred_rnn)
sns.heatmap(cm_sm_rnn, annot=True, fmt='d', cmap='Purples')
plt.title("LSTM-GRU (SMOTE) - Confusion Matrix")
plt.show()
```

```
===== LSTM-GRU (SMOTE) =====
Accuracy: 0.9993255172687417
Precision: 0.7263157894736842
Recall: 0.8961038961038961
F1 Score: 0.8023255813953488
ROC AUC: 0.9606879153589
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     50332
           1       0.73      0.90      0.80        77

    accuracy                           1.00     50409
   macro avg       0.86      0.95      0.90     50409
weighted avg       1.00      1.00      1.00     50409
```




```
# ROC Curve - SMOTE RNN
fpr_sm_rnn, tpr_sm_rnn, _ = roc_curve(y_sm_test, y_sm_proba_rnn)
auc_sm_rnn = roc_auc_score(y_sm_test, y_sm_proba_rnn)
plt.plot(fpr_sm_rnn, tpr_sm_rnn, label=f"SMOTE RNN (AUC = {auc_sm_rnn:.4f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve - LSTM-GRU (SMOTE)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()

model_sm_rnn.save("lstm_gru_smote.keras")
```



1.10 Comparison:

SMOTE preserves all majority-class data and enriches the minority class with synthetic samples which typically improves the model's sensitivity to fraud. In contrast undersampling balances the dataset by removing majority-class instances, making training faster but potentially reducing accuracy due to information loss.

```
# ================================================
# 5. Summary
# ================================================
print("\n=== FINAL SUMMARY ===")
print("LR (Undersampled) ROC AUC:", roc_auc_score(y_us_test, y_us_proba_lr))
print("LR (SMOTE) ROC AUC:", roc_auc_score(y_sm_test, y_sm_proba_lr))
print("LSTM-GRU (Undersampled) ROC AUC:", roc_auc_score(y_us_test, y_us_proba_rnn))
print("LSTM-GRU (SMOTE) ROC AUC:", roc_auc_score(y_sm_test, y_sm_proba_rnn))

print("\nModels saved successfully.")
```

```
=== FINAL SUMMARY ===
LR (Undersampled) ROC AUC: 0.9928211738988183
LR (SMOTE) ROC AUC: 0.9776035178363718
LSTM-GRU (Undersampled) ROC AUC: 0.9847666042929495
LSTM-GRU (SMOTE) ROC AUC: 0.9606879153589

Models saved successfully.
```

4. Conclusion:

In this project, machine learning and deep learning have been successfully implemented to identify fraudulent transactions in datasets that are highly unbalanced. The systematic preprocessing, scaling and application of SMOTE made data quality and balance of classes appropriate and the models began to work with better reliability. The Logistic Regression model was a strong baseline as it is a model that can be easily interpreted and consistent and the hybrid LSTM -GRU model greatly advanced the detection accuracy and recall since it can capture sequential dependencies on transaction data (Ndama et al., 2024). The comparison analysis ensured that deep learning models are more effective compared to traditional methods in handling intricate, time-dependent patterns. This project provides a template to repeated and scale to the real-world financial systems, which in the end can lead to better prevention of fraud and safe monitoring of transactions. It can be assumed that further work will be related to the deployment of a model in real time and its combination with advanced anomaly detection methods to improve it continuously.

Determining the difference between undersampling and SMOTE demonstrated that oversampling will yield better recalls, which is essential in detecting fraud and undersampling will yield faster training and conservative decision boundaries. The model behavior of the comparison between the sampling strategies is shown by training separate Logistic Regression and LSTMGRU models on each sampling strategy. The work of the future can include putting these models together or working out the sampling ratios dynamically.

5. References:

Wibowo, M. A. A., & Setiadi, D. R. I. M. (2024). Optimized machine learning model for credit card fraud detection using SMOTE-Tomek and feature engineering. Journal of Applied Informatics and Computing, 8(2), 580–588. https://doi.org/10.30871/jaic.v8i2.8732

Kharat, S., Taur, S., Khalate, R., & Kate, K. (2023). Detection of credit card fraud using machine learning and deep learning: A review. International Journal of Progressive Research in Science and Engineering, 4(5), 80–83. https://journal.ijprse.com/index.php/ijprse/article/view/840

Ndama, O., Bensassi, I., & En-Naimi, E. M. (2024). Optimizing credit card fraud detection: A deep learning approach to imbalanced datasets. International Journal of Electrical and Computer Engineering (IJECE), 14(4), 4802–4814. https://doi.org/10.11591/ijece.v14i4.pp4802-4814

Assabil, J. J. (2024, August 6). Credit card Fraud Detection Using Machine Learning Algorithms: A Comparative study of six models. https://ijisae.org/index.php/IJISAE/article/view/7040