National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Aafreen Shan Asmath
Student ID: x23231335

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aafreen Shan Asmath		
Student ID:	X23231335		
Programme:	MSc Data Analytics	**Year:**	2024 - 25
Module:	MSc Research Project		
Lecturer:	Prof. Christian Horn		
Submission Due Date:	29/01/2025		
Project Title:	Enhancing Credit Card Fraud Detection Accuracy by Optimisation of Anomaly Detection Algorithms and Resampling Techniques		
Word Count:	809	**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: Aafreen Shan Asmath

Date: 29/01/2025

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

Aafreen Shan Asmath
Student ID: x23231335

1 Introduction

This manual explains the configuration necessary for replicating the experimental setup to reproduce the findings of the research project "Optimizing Anomaly Detection Algorithms and Resampling Techniques for More Accurate Detection of Credit Card Fraud." The investigation studies the performance of anomaly detection models like Isolation Forest and Local Outlier Factor combined with resampling techniques like SMOTE, SMOTE-ENN, and Random Undersampling. In this manual, software, hardware, and technical definitions are introduced to maintain experimental consistency in their reappearance. This document truly serves a great purpose in replicating the experimental environment, testing, and reproducing the findings and results.

2 Development Environment

2.1 Hardware Specifications

- **Processor:** 11th Gen Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- **RAM:** 8GB
- **GPU:** NVIDIA RTX 3050

2.2 Software Specifications

- **Operating System:** Windows 11 (any version compatible with Python can be used)
- **Programming Language:** Python 3.11.5

```
3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]
```

Figure 1: Python Version

- **Integrated Development Environment (IDE):** Jupyter Notebook 6.5.7 or higher version

2.3 Required Libraries and Frameworks

These libraries in Python are essential for performing the implementations. They could be installed as follows by typing in a command line.

- NumPy: Numerical computation
- pandas: Manipulating and analyzing data
- matplotlib: Data visualization
- seaborn: Statistical data visualization
- scikit-learn: Packages with a collection of machine learning algorithms and utilities
- imbalanced-learn: This library can be used to run various functions that help in various resampling techniques like SMOTE and SMOTE-ENN.
- SciPy: Scientific computing
- warnings: Managing the runtime warnings

```
import numpy as np
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import MinMaxScaler, StandardScaler,RobustScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,confusion_matrix, roc_auc_score, roc_curve
import warnings
```

Figure 2: Libraries Used

3 Data Source

This research uses an open-source dataset from Kaggle:

Dataset Name: Credit Card Fraud Detection Dataset

Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data

The number of records: 284,807

Features: 31 (Time, Amount, and anonymized features from V1 to V28).

Target Variable: Class (0 for non-fraud, 1 for fraud).

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055

Figure 3: Data frame

4 Project Code Files

One Jupyter Notebook Code File is used in this project, and it's named as in the below image.

The code file in the below image contains data pre-processing, model building and evaluation.

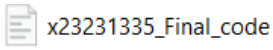

Figure 4: Jupyter Notebook Files

5 Data Preparation

5.1 Extracting Data:

Extracting the Data from the CSV file

Load Dataset

```
In [2]: cred = pd.read_csv("E:\Research_Project\creditcard.csv")

In [3]: cred.shape
Out[3]: (284807, 31)

In [4]: cred.head(7)
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.23279
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.75013

7 rows × 31 columns

Figure 5: Loading the dataset

5.2 Data Pre-Processing:

Data pre-processing consists of data cleaning, data transformation, and applying techniques like SMOTE to address class imbalance.

```
In [8]: print(cred.duplicated().sum())
```

```
1081
```

```
In [9]: duplicates = cred.duplicated()
        # Get the indices of duplicated rows
        duplicate_indices = cred[duplicates].index
        print("Indices of duplicated rows:", duplicate_indices)
```

```
Indices of duplicated rows: Int64Index([    33,     35,    113,    114,    115,    221,    223,   1178,
                1180,   1382,
               ...
               282210, 282211, 282212, 282213, 282985, 282987, 283483, 283485,
               284191, 284193],
              dtype='int64', length=1081)
```

```
In [10]: cred= cred.drop_duplicates()
```

```
In [11]: cred.shape # after removing 1081 row
```

```
Out[11]: (283726, 31)
```

Figure 6: Data Cleaning

Step-2: Handling Class Imbalance with SMOTE ,SMOTE- ENN and Random Undersampling

```
In [33]: from imblearn.over_sampling import SMOTE
         from imblearn.combine import SMOTEENN
         from imblearn.under_sampling import RandomUnderSampler

         # Define feature matrix X and target vector y
         X = cred.drop(columns=["Class"])  # Assuming "Class" is the target column
         y = cred["Class"]

         # Apply SMOTE
         smote = SMOTE(random_state=50)
         X_smote, y_smote = smote.fit_resample(X, y)
         print("Shape after SMOTE resampling:", X_smote.shape, y_smote.shape)

         # Apply SMOTE-ENN
         smote_enn = SMOTEENN(random_state=50)
         X_smote_enn, y_smote_enn = smote_enn.fit_resample(X, y)
         print("Shape after SMOTE-ENN resampling:", X_smote_enn.shape, y_smote_enn.shape)

         # Apply Random Undersampling
         random_under_sampler = RandomUnderSampler(random_state=50)
         X_under, y_under = random_under_sampler.fit_resample(X, y)
         print("Shape after Random Undersampling:", X_under.shape, y_under.shape)
```

```
Shape after SMOTE resampling: (566506, 32) (566506,)
Shape after SMOTE-ENN resampling: (565859, 32) (565859,)
Shape after Random Undersampling: (946, 32) (946,)
```

Figure 7: Handling class imbalance using SMOTE and SMOTE-ENN

```
In [29]: Q1 = cred['Amount'].quantile(0.25)
         Q3 = cred['Amount'].quantile(0.75)
         IQR = Q3 - Q1
         lower_bound = Q1 - 1.5 * IQR
         upper_bound = Q3 + 1.5 * IQR
         df_clean = cred[(cred['Amount'] >= lower_bound) & (cred['Amount'] <= upper_bound)]
```

Figure 8: Outlier detection and removal

5.3 Exploratory Data Analysis and Feature Engineering

This section is about the exploratory data analysis of credit card fraud detection. This section involves using various graphs to analyse the dataset.

Box Plots for Amount and Time

In [19]:
```
# Create box plots for 'Amount' and 'Time'
fig, ax = plt.subplots(1, 2, figsize=(15, 8))
sns.boxplot(y='Amount', data=cred, ax=ax[0])
ax[0].set_title('Box Plot for Transaction Amount')
sns.boxplot(y='Time', data=cred, ax=ax[1])
ax[1].set_title('Box Plot for Transaction Time')
```

Out[19]: Text(0.5, 1.0, 'Box Plot for Transaction Time')

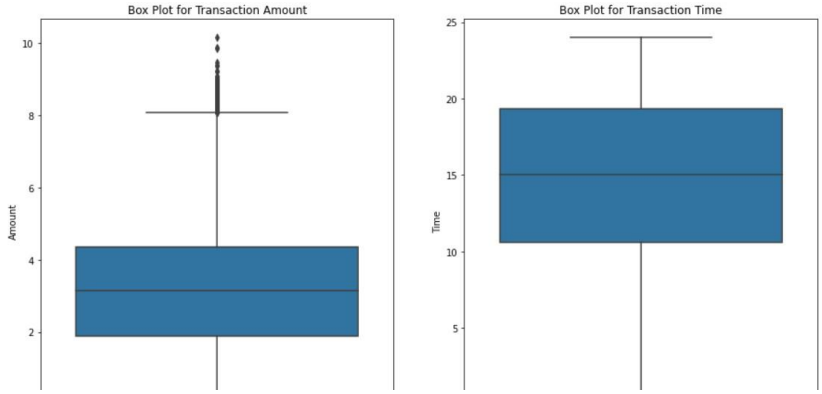

Figure 9: Box plots of transaction time and amount.

Fraud vs Non-Fraud Amount Distribution

In [22]:
```
fraud = cred[cred['Class'] == 1]
# Pot the 'Amount' distribution for Fraud transactions
plt.figure(figsize=(8, 5))
fraud['Amount'].hist(alpha=0.7, color='red', bins=50)
plt.title('Fraud Amount Distribution')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.legend(['Fraud'])
plt.show()
```

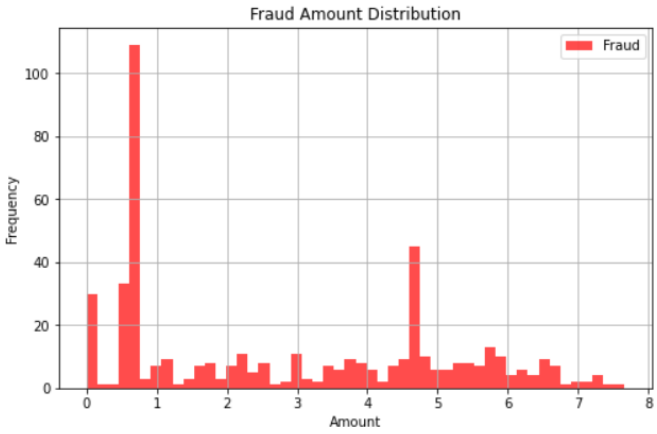

Figure 10: Fraud vs Non-Fraud Amount Distribution

Correlation Matrix

In [27]:
```
corr_matrix = cred.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=False, cmap='RdPu', linewidths=0.5)
plt.title("Correlation Matrix of Features")
```

Out[27]: Text(0.5, 1.0, 'Correlation Matrix of Features')

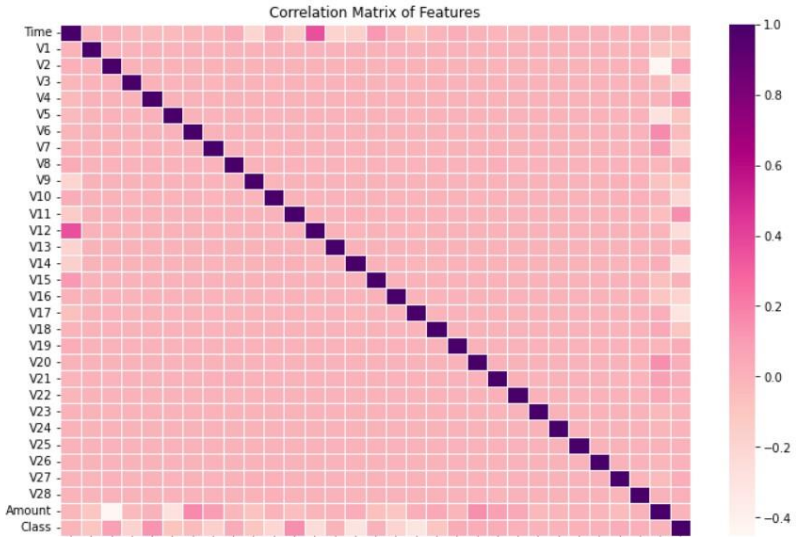

Figure 11: Correlation Matrix

In [28]:
```
# Set up the subplots
f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 6))

v14_fraud_dist = cred['V14'].loc[cred['Class'] == 1].values
sns.histplot(v14_fraud_dist, ax=ax1, kde=True, color='#FB8861', stat='density')
ax1.set_title('V14 Distribution \n (Fraudulent Transactions)', fontsize=14)

v12_fraud_dist = cred['V12'].loc[cred['Class'] == 1].values
sns.histplot(v12_fraud_dist, ax=ax2, kde=True, color='#56F9BB', stat='density')
ax2.set_title('V12 Distribution \n (Fraudulent Transactions)', fontsize=14)

v10_fraud_dist = cred['V10'].loc[cred['Class'] == 1].values
sns.histplot(v10_fraud_dist, ax=ax3, kde=True, color='#C5B3F9', stat='density')
ax3.set_title('V10 Distribution \n (Fraudulent Transactions)', fontsize=14)
plt.tight_layout()
```

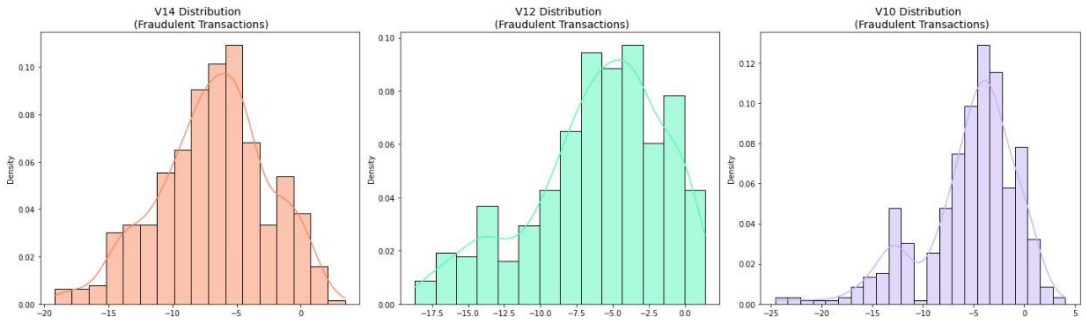

Figure 12: Histogram Plots of independent and target variables.

In [31]:
```
# Select the features you want to normalize
features = ['V10','V11', 'V12', 'V13', 'V14','V16','V17']
scaler = MinMaxScaler()
cred[features] = scaler.fit_transform(cred[features])
```

In [32]:
```
cred.shape
```

Out[32]: (283726, 33)

Figure 13: Applying Normalization

6 Model Building and Evaluation

This section contains splitting the data before training the model, implementing the model. It also contains implementation of anomaly detection models.

Step -3: Train-Test Split

In [34]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets for each resampling technique

# For SMOTE
X_train_smote, X_test_smote, y_train_smote, y_test_smote = train_test_split(
    X_smote, y_smote, test_size=0.3, random_state=50
)

# For Non-SMOTE
X_train_non_smote, X_test_non_smote, y_train_non_smote, y_test_non_smote = train_test_split(
    X, y, test_size=0.3, random_state=50
)

# For SMOTE-ENN
X_train_smote_enn, X_test_smote_enn, y_train_smote_enn, y_test_smote_enn = train_test_split(
    X_smote_enn, y_smote_enn, test_size=0.3, random_state=50
)

# For Random Undersampling
X_train_under, X_test_under, y_train_under, y_test_under = train_test_split(
    X_under, y_under, test_size=0.3, random_state=50
)
```

Figure 14: Splitting the data into train and test

In [36]:
```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# Initialize models
isolation_forest = IsolationForest(random_state=50)
lof = LocalOutlierFactor(n_neighbors=20, novelty=True)

# Train and predict using Isolation Forest on SMOTE, SMOTE-ENN, Non-SMOTE,and Random Undersampling

# Isolation Forest - SMOTE
isolation_forest.fit(X_train_smote)
y_pred_if_smote = isolation_forest.predict(X_test_smote)
y_pred_if_smote = [1 if x == -1 else 0 for x in y_pred_if_smote]  # Converting predictions to 0 and 1

# Isolation Forest - SMOTE-ENN
isolation_forest.fit(X_train_smote_enn)
y_pred_if_smote_enn = isolation_forest.predict(X_test_smote_enn)
y_pred_if_smote_enn = [1 if x == -1 else 0 for x in y_pred_if_smote_enn]

# Isolation Forest - Non-SMOTE
isolation_forest.fit(X_train_non_smote)
y_pred_if_non_smote = isolation_forest.predict(X_test_non_smote)
y_pred_if_non_smote = [1 if x == -1 else 0 for x in y_pred_if_non_smote]  # Converting predictions to 0 and 1

# Isolation Forest - Random Undersampling
isolation_forest.fit(X_train_under)
y_pred_if_under = isolation_forest.predict(X_test_under)
y_pred_if_under = [1 if x == -1 else 0 for x in y_pred_if_under]

# Train and predict using Local Outlier Factor on SMOTE, SMOTE-ENN, Non-SMOTE,and Random Undersampling

# Local Outlier Factor - SMOTE
lof.fit(X_train_smote)
y_pred_lof_smote = lof.predict(X_test_smote)
y_pred_lof_smote = [1 if x == -1 else 0 for x in y_pred_lof_smote]
```

Figure 15: Implementation of Isolation model

```
In [37]: from sklearn.linear_model import LogisticRegression
         from sklearn.preprocessing import StandardScaler

         # Step 1: Scale the data
         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(X)

         # Step 2: Train-test split
         X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=50, stratify=y)

         # Step 3: LogisticRegression - SMOTE
         smote = SMOTE(random_state=50)
         X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

         # Step 4: LogisticRegression - Non-SMOTE
         model_non_smote = LogisticRegression(random_state=42, max_iter=500)
         model_non_smote.fit(X_train, y_train)

         # Predict on the test set
         y_pred_non_smote = model_non_smote.predict(X_test)

         # Step 5:LogisticRegression - SMOTE-ENN
         smote_enn = SMOTEENN(random_state=50)
         X_train_smote_enn, y_train_smote_enn = smote_enn.fit_resample(X_train, y_train)

         model_smote_enn = LogisticRegression(random_state=50, max_iter=500)
         model_smote_enn.fit(X_train_smote_enn, y_train_smote_enn)

         # Predict on the test set
         y_pred_smote_enn = model_smote_enn.predict(X_test)

         # Step 6: Random Undersampling
         undersampler = RandomUnderSampler(random_state=50)
         X_train_undersampled, y_train_undersampled = undersampler.fit_resample(X_train, y_train)

         model undersample = LogisticRegression(random state=50, max iter=500)
```

Figure 16: Logistic Regression Implementation

```
In [38]: from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

         # Evaluation function
         def evaluate_model(y_true, y_pred, model_name, technique):
             print(f"{model_name} - {technique}")
             print(classification_report(y_true, y_pred))
             print("AUC:", roc_auc_score(y_true, y_pred))

             # Confusion matrix
             cm = confusion_matrix(y_true, y_pred)
             plt.figure(figsize=(5, 4))
             sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Fraud', 'Fraud'], yticklabels=['Non-Fraud', 'Fraud'])
             plt.title(f"Confusion Matrix: {model_name} - {technique}")
             plt.xlabel("Predicted")
             plt.ylabel("Actual")
             plt.show()
             print("\n")

         # Isolation Forest Evaluation
         evaluate_model(y_test_smote, y_pred_if_smote, "Isolation Forest", "SMOTE")
         evaluate_model(y_test_smote_enn, y_pred_if_smote_enn, "Isolation Forest", "SMOTE-ENN")
         evaluate_model(y_test_under, y_pred_if_under, "Isolation Forest", "Random Undersampling")
         evaluate_model(y_test_non_smote, y_pred_if_non_smote, "Isolation Forest", "Non-SMOTE")

         #Local Outlier Factor Evaluation
         evaluate_model(y_test_smote, y_pred_lof_smote, "Local Outlier Factor", "SMOTE")
         evaluate_model(y_test_smote_enn, y_pred_lof_smote_enn, "Local Outlier Factor", "SMOTE-ENN")
         evaluate_model(y_test_under, y_pred_lof_under, "Local Outlier Factor", "Random Undersampling")
         evaluate_model(y_test_non_smote, y_pred_lof_non_smote, "Local Outlier Factor", "Non-SMOTE")

         # Logistic Regression models
         evaluate_model(y_test, y_pred_non_smote, "Logistic Regression", "SMOTE")
         evaluate_model(y_test, y_pred_non_smote, "Logistic Regression", "Non-SMOTE")
         evaluate_model(y_test, y_pred_smote_enn, "Logistic Regression", "SMOTE-ENN")
         evaluate_model(y_test, y_pred_undersample, "Logistic Regression", "Random Undersampling")
```

Figure 17: Evaluation of Models

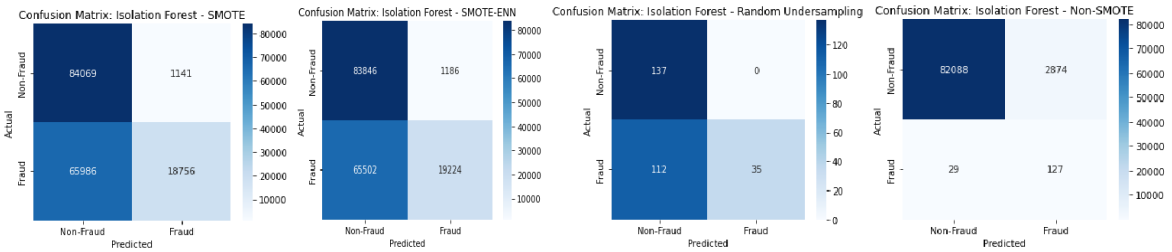

Figure 18: Confusion Matrix of Isolation Forest

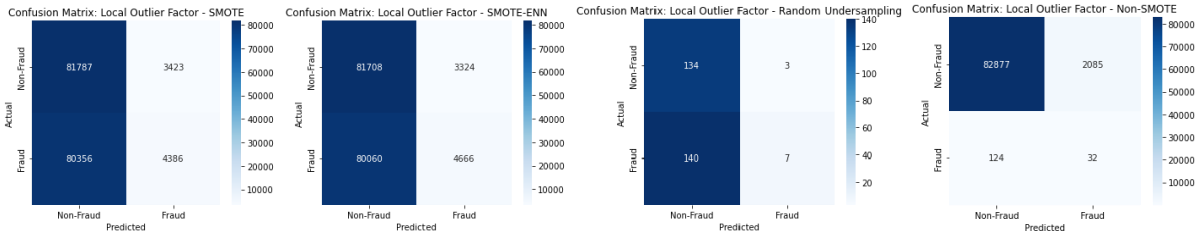

Figure 19: Confusion Matrix of LOF

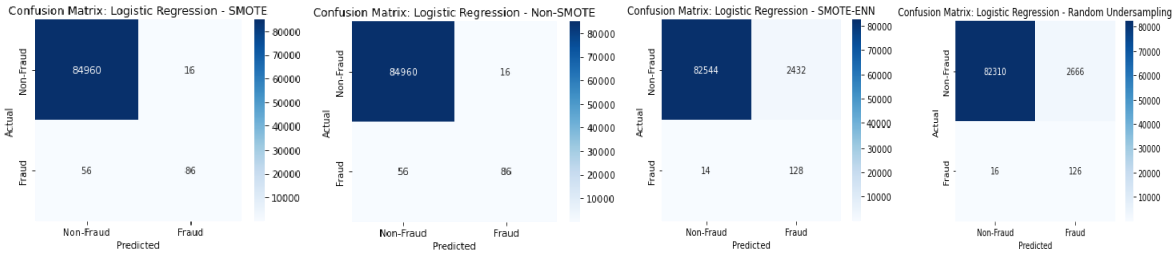

Figure 20: Confusion Matrix of Logistic Regression