Configuration Manual

MSc Research Project
Data Analytics and Programing

Damilare Kolawole
Student ID: x21235571

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Damilare Kolawole		
Student ID:	X21235571		
Programme:	Data Analytics and Programing	**Year:**	2023
Module:	MSc Research Project		
Lecturer:	Vikas Tomer		
Submission Due Date:	14-12-2023		
Project Title:	Credit Card Fraud Detection: A Hybrid Approach		
Word Count:	563	**Page Count:**	16

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: *Damilare Kolawole*

Date: 14-12-2023

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

Damilare Kolawole
X21235571

1 Overview and Design Flow

Credit card fraud is a prevalent issue that researchers have continuously looked into, for improved model performance, an hybrid approach is proposed. The figure below is the flowchart for the system design.

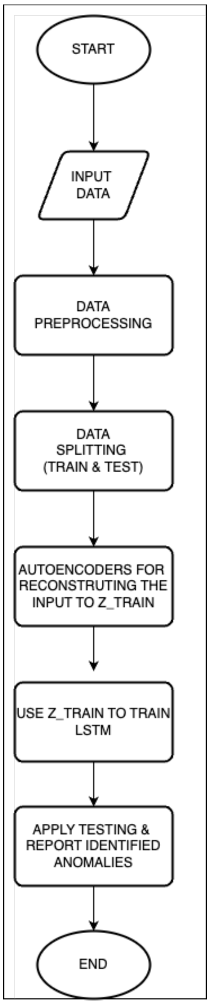


figure 1: Flowchart of the Process Design

2 System Requirement

- Processor: Apple M1
- Memory RAM: 8GB

- Operating System: macOS Sonoma 14.1.1 (23B81)
- Storage: 500GB

3 Software Requirement

- Miniconda
- Jupyter Notebook
- Python
- Terminal

4 Software Installation

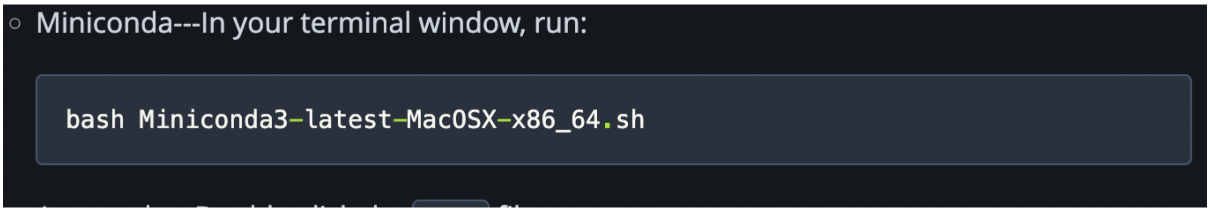


figure 2: Command to Install Miniconda

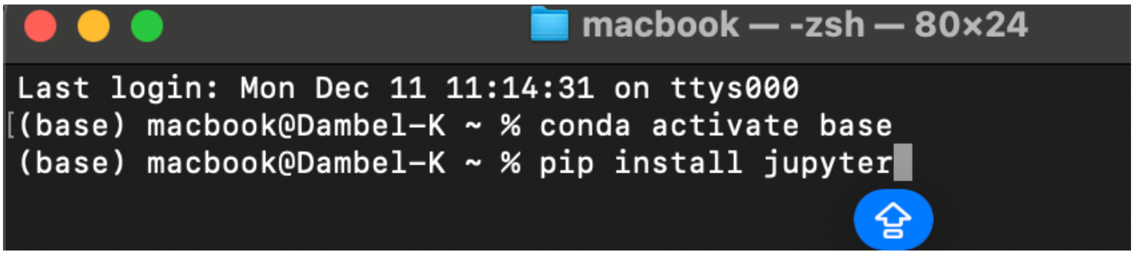


figure 3: Install Jupyter Notebook

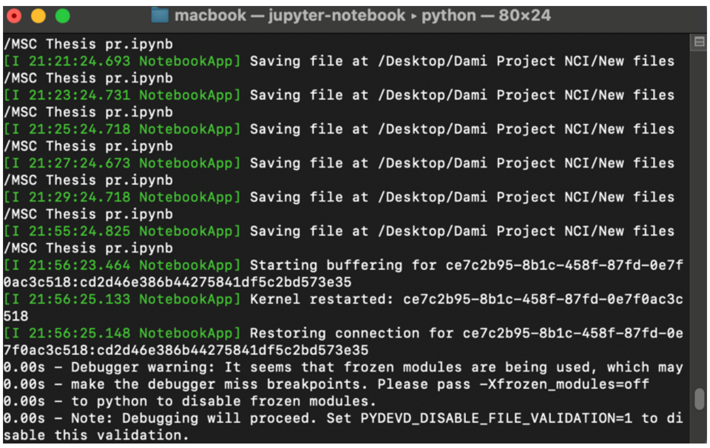


figure 4: Jupyter Notebook Running

5 Implementation

To implement we made use of python libraries such as:

- NumPy
- Pandas
- Matplotlib
- Seaborn Sklearn, etc

5.1 AE+LSTM

```
[1]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Dense
import pandas as pd
import numpy as np
from scipy import stats
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_curve
from sklearn.metrics import recall_score, classification_report, auc, roc_curve
from sklearn.metrics import precision_recall_fscore_support, f1_score
from sklearn.preprocessing import StandardScaler
from pylab import rcParams
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers

import warnings
warnings.filterwarnings('ignore')

print('Imported successfully')
```

figure 5: Python Libraries

```
: data.head(n=10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.26
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.08
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.80
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.85
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.06

10 rows × 31 columns

```
: # Check for normal transactions and fraudulent ones
counts = data.Class.value_counts()
normal = counts[0]
fraudulent = counts[1]
perc_normal = (normal/(normal+fraudulent))*100
perc_fraudulent = (fraudulent/(normal+fraudulent))*100
print('There were {} non-fraudulent transactions ({:.3f}%) and {}
```

```
There were 284315 non-fraudulent transactions (99.827%) and 492 f
raudulent transactions (0.173%).
```

```
From the above result the dataset is highly imbalanced.
Below is a visual representation
Time is given in seconds, that would be a feautre that would be
added later (mins and hours).
```

figure 6: Exploratory Data Analysis

We then checked for correlations after we had explored the data.

In [6]:
```python
#finding correlation between columns and plotting heatmap

corr = data.corr()
plt.figure(figsize=(12,10))
heat = sns.heatmap(data=corr)
plt.title('Heatmap of Correlation')
```

Out[6]:
```
Text(0.5, 1.0, 'Heatmap of Correlation')
```

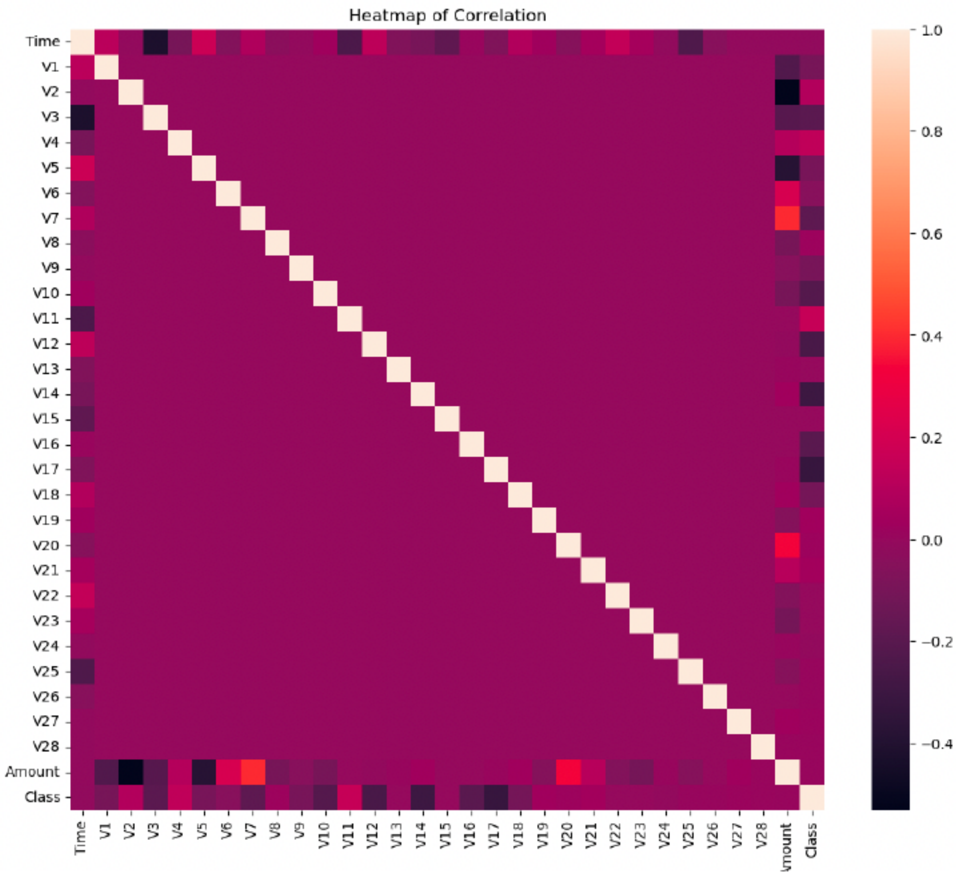


figure 7: Checking for Correlation

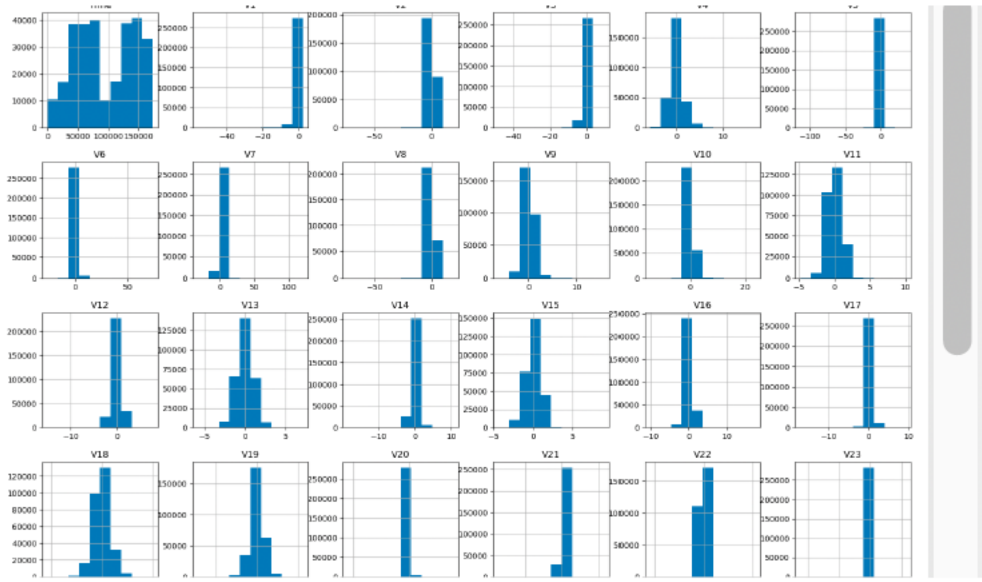


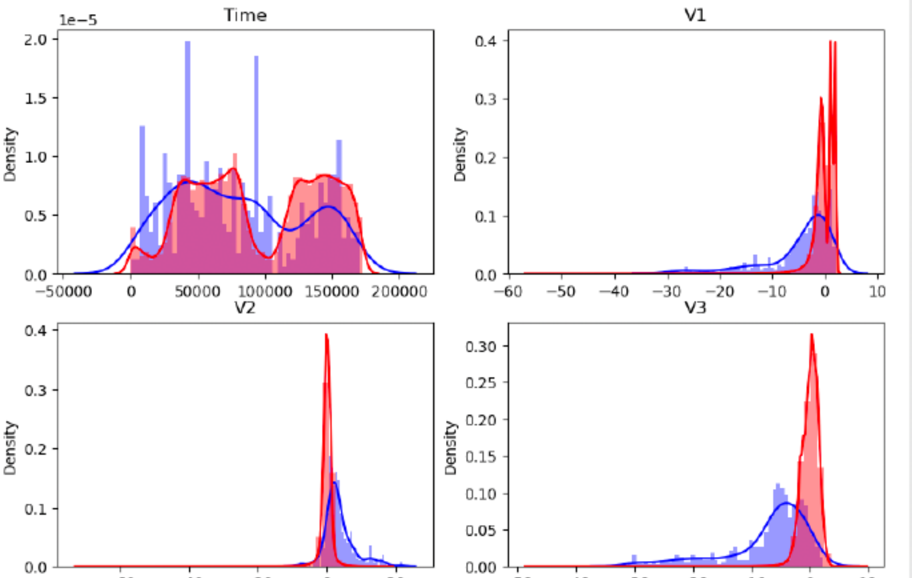
figure 8: Plot for Normal Transactions

Ater this overview, next is feature selection. A plot was made for that to see significant features and to also know which to drop.

```
In [9]: # transform the dataset
        from imblearn.over_sampling import SMOTE
        oversample = SMOTE()
        X_r, y = oversample.fit_resample(X, tr_data['Class'])
        # summarize the new class distribution
        counter = Counter(y)
        print(counter)
        # scatter plot of examples by class label
        for label, _ in counter.items():
            row_ix = where(y == label)[0]

        Counter({0: 284315, 1: 284315})
```

figure 9: SMOTE for Handling Imbalances

```
[12]: #Feature Selection via distribution graphs

import matplotlib.gridspec as gridspec # to do the grid of plots #
columns  = data.iloc[:,data.columns  != 'Class'].columns
frauds = data.Class == 1
normals = data.Class == 0
grid = gridspec.GridSpec(17, 2)
plt.figure(figsize=(10,15*4))

for n, col in enumerate(data[columns]):
    ax = plt.subplot(grid[n])
    sns.distplot(data[col][frauds], bins = 50, color='b') #Will re
    sns.distplot(data[col][normals], bins = 50, color='r') #Will r
    ax.set_ylabel('Density')
    ax.set_title(str(col))
    ax.set_xlabel('')
plt.show()
```




```
[13]: # some features would be dropped because they have almost the same
data_features = data.drop(['V15','V17','V24','V27','Time_hour','Ti
```

figure 10: Feature Selection

The next is splitting the data for train and test purposes.

Data Spliting

Desktop/Dami Project NCI/New files /MSC Thesis pr.ipynb

MSC Thesis pr - Jupyter Notebook

```
from sklearn.model_selection import train_test_split

data_training, data_testing = train_test_split(data_features,test_

#data_testing.Class.value_counts()
```

```
from sklearn.model_selection import train_test_split

# Assuming data_features contains your feature columns and 'Class'
X = data_features.drop('Class', axis=1)  # Features
y = data_features['Class']  # Target variable

# Splitting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_siz
```

```
train_data,validation_data,train_label, validation_label = train_t
                                                random_state = 42)
```

figure 11: Data Splitting

[18]:
```
# Scaling the data using min max scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data_scaled = scaler.fit(train_data)
train_data_normalised = data_scaled.transform(train_data)
validation_data_normalised = data_scaled.transform(validation_data
```

[19]:
```
test_data = data_testing.loc[:, data_testing.columns != 'Class']
test_label = data_testing.Class
test_data_normalised = data_scaled.transform(test_data)
```

[20]:
```
#test_data.shape
```

[21]:
```
# changing the labels with boolean
train_label,validation_label, test_label = train_label.astype(bool

# now  lets seperate the normal and fraud data out of training dat
normal_train_data = train_data_normalised[~train_label] # normal t
normal_test_data = test_data_normalised[~test_label] # normal tran
normal_validation_data = validation_data_normalised[~validation_la
```

[22]:
```
print(len(normal_train_data))
print(len(normal_test_data))
print(len(normal_validation_data))
```

```
181961
56864
45490
```

ıotebooks/Desktop/Dami Project NCI/New files /MSC Thesis pr.ipynb 10/.

16 MSC Thesis pr - Jupyter Notebook

[23]:
```
fraud_train_data = train_data_normalised[train_label]
fraud_test_data = test_data_normalised[test_label]
fraud_validation_data = validation_data_normalised[validation_labe
```

figure 12: Standardizing the data

```
!7]: import tensorflow
from tensorflow.keras.layers import Dense,LSTM
from tensorflow.keras.models import Model
from tensorflow.keras import models,layers,activations,losses,opti
from tensorflow.keras.callbacks import EarlyStopping
n_features = len(train_data.columns)
encoder = models.Sequential(name='encoder')
encoder.add(layer=layers.Dense(units=200, activation=activations.r
encoder.add(layers.Dropout(0.1))
encoder.add(layer=layers.Dense(units=100, activation=activations.r
encoder.add(layer=layers.Dense(units=5, activation=activations.rel

decoder = models.Sequential(name='decoder')
decoder.add(layer=layers.Dense(units=100, activation=activations.r
decoder.add(layer=layers.Dense(units=200, activation=activations.r
decoder.add(layers.Dropout(0.1))
decoder.add(layer=layers.Dense(units=n_features, activation=activa

autoencoder = models.Sequential([encoder, decoder])

autoencoder.compile(
    loss=losses.MSE,
    optimizer=optimizers.Adam(),
    metrics=[metrics.mean_squared_error])
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimiz
ers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras
optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
```

figure 13: Model Building Autoencoder

Integrating AE+LSTM

8]:
```
# Extract encoded representations (latent space)
encoded_train_data = autoencoder.predict(x_train)
encoded_test_data = autoencoder.predict(x_test)
```

```
7121/7121 [==============================] - 3s 360us/step
1781/1781 [==============================] - 1s 318us/step
```


0]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Reshape data for LSTM (assuming a sequence length of 10)
sequence_length = 10

def create_sequences(data, sequence_length):
    sequences = []
    for i in range(len(data) - sequence_length + 1):
        sequences.append(data[i:i+sequence_length])
    return np.array(sequences)

# Create sequences for LSTM input
train_sequences = create_sequences(encoded_train_data, sequence_le
test_sequences = create_sequences(encoded_test_data, sequence_leng

# LSTM model
lstm_model = Sequential([
    LSTM(units=64, input_shape=(train_sequences.shape[1], train_se
    # Add more LSTM layers or Dense layers if needed
    Dense(units=1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', m

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(patience=3, restore_best_weights=Tr

# Train LSTM on sequences
lstm_model.fit(train_sequences, y_train[sequence_length-1:], epoch

# Evaluate the model
score = lstm_model.evaluate(test_sequences, y_test[sequence_length
print("Test Accuracy:", score[1])
```

figure 14: Creating the Hybrid Model

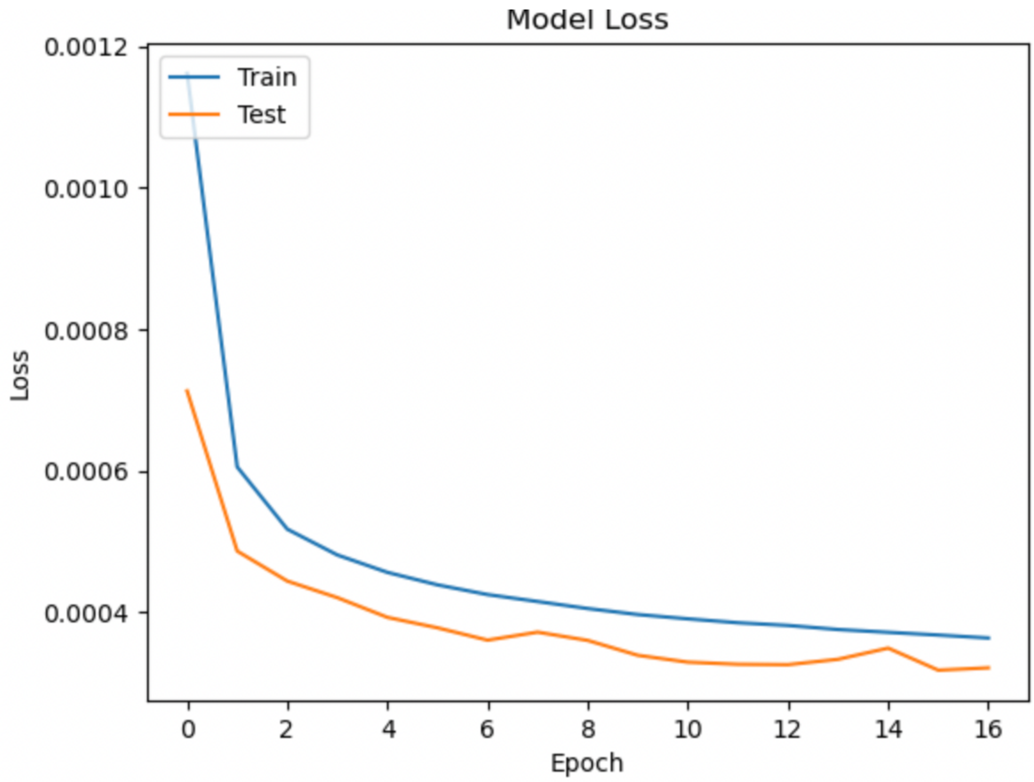


figure 15: Model Loss

```
In [64]: ### classification report
         from sklearn.metrics import classification_report, confusion_matrix
         print(classification_report(true_label, predicted))
         print(confusion_matrix(true_label, predicted))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56864
           1       0.12      0.74      0.20        98

    accuracy                           0.99     56962
   macro avg       0.56      0.87      0.60     56962
weighted avg       1.00      0.99      0.99     56962

[[56320   544]
 [   25    73]]
```

figure 16: Results

5.2 Autoencoders

Following the same process from data loading to exploratory data analysis, handling class imbalance, feature selection up until Model building the process remains the same. The figures below shows how this autoencoder was built and evaluated.

```
#Creating the Model
#Autoencoder Layer Structure and Parameters
nb_epoch = 100 #number of time algorithm runs in training the dataset
batch_size = 128 #number of samples(single row in data) taken for updating the model paraeters
input_dim = train_x.shape[1] #num of columns, 30
encoding_dim = 14
hidden_dim = int(encoding_dim / 2) #e.g 7
learning_rate = 1e-7 #scale of how much model weights should be updated

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="tanh", activity_regularizer=regularizers.l1(learning_rate))(input_layer)
encoder = Dense(hidden_dim, activation="relu")(encoder)
decoder = Dense(hidden_dim, activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

```
#Model Training and Logging
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')
```

```
#create check point callback
cp = ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               save_best_only=True,
                               verbose=0)
```

```
#Tensorboard callback
tb = TensorBoard(log_dir='./logs',
                histogram_freq=0,
                write_graph=True,
                write_images=True)
```

figure 17: Autoencoder Model

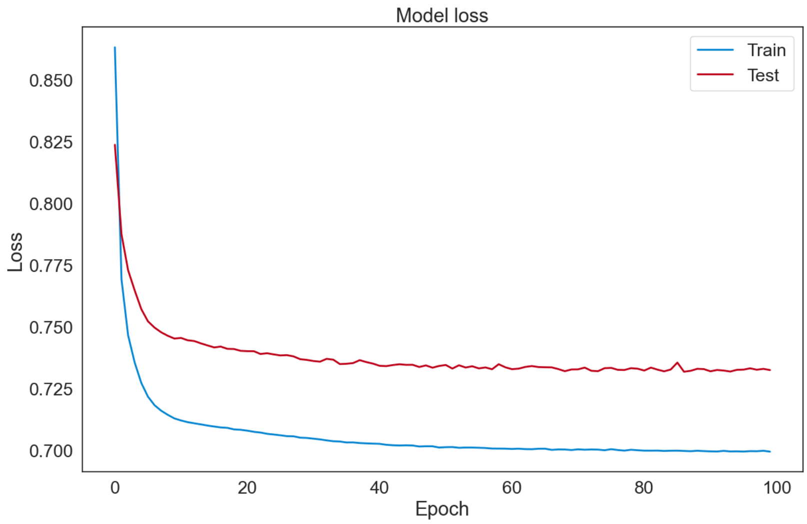


figure 18: Model Loss

```
In [34]: print(classification_report(error_df.True_class, pred_y))

              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56847
           1       0.10      0.63      0.17       115

    accuracy                           0.99     56962
   macro avg       0.55      0.81      0.58     56962
weighted avg       1.00      0.99      0.99     56962
```

figure 19: Results

Conclusion

The implementations of the codes used and pictorial diagram as seen above is for better understanding of the work flow.