


Statement and Confirmation of Own Work

***A signed copy of this form must be submitted with every assignment.
If the statement is missing your work may not be marked.***

Student Declaration

I confirm the following details:

Student Name:	Johnson Kenisha Corera
Student ID Number:	c23020001@cicra.edu.lk
Qualification:	Bachelor in Cyber Security
Unit:	SIT325 Advanced Network Security
Centre:	CICRA Campus
Word Count:	594
<p>I have read and understood both <i>Deakin Academic Misconduct Policy</i> and the <i>Referencing and Bibliographies</i> document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly.</p> <p>I confirm that I have not exceeded the stipulated word limit by more than 10%.</p> <p>I confirm that this is my own work and that I have not colluded or plagiarized any part of it.</p>	
Candidate Signature:	J. 
Date:	03/12/2024

Task 9.2C
Table of Content

Physical layer security.....	05
DAE.....	06
Method.....	07
Output.....	08
Overview.....	08

Table of Figures

Figure 01.....	09
Figure 02.....	10
Figure 03.....	11
Figure 04.....	12
Figure 05.....	13
Figure 06.....	14

Table of Acronyms

Acronyms	Meaning
SNR	Single-to-noise ratio
DAEs	denoising autoencoder

Physical layer security

- The newly emerging security field called physical layer security depends on transmission environment features to enhance system security. Secure physical layer security differs from conventional cryptographic approaches to protect physical layer information transmissions through the use of transmission media attributes.

Fundamental concepts

- The maximum data speed which achieves secure transmission through a communication channel defines secrecy capacity. The security capacity analysis takes into account all potential wiretapping methods along with the channel transmission capabilities.
- Your code uses auto encoders to assess the entire dataset but its main objective focuses on detecting anomalies and pays no attention to physical layer security aspects. The regulations may impact the development and dependability of these systems particularly for data transmission through unstable networks.
- The Signal-to-noise ratio (SNR) when improved through various methods helps enhance communication channel security and reduces its reliance. Spread spectrum combined with multiple antennas function as methods which enhance the signal to noise ratio known as SNR.
- A wiretap channel describes the circumstances where an outsider tries to overhear a conversation. The goal of physical layer security is to convert received signals into unintelligible content for unwarranted listeners.

DAE

- Neural networks perform anomaly detection as their primary function under the name denoising auto encoders (DAEs). Unsupervised learning utilizes these systems to perform its operations. Training DAEs results in learning to reconstruct input data from noisy data versions. The primary basis for this work demonstrates the poor reformatting capability of typical data to recreate anomalies or outliers.

Method

- Testing control occurs through auto encoder reconstruction calculations that lead to measurement of reconstruction errors. A data point becomes anomalous where the error exceeds a predetermined threshold under a decided inclination.
- The training process in Phase of Training uses standard data to extract underlying patterns and correlations from the data. During its operation the technique works to minimize reconstruction differences between input data and output data.

In the code that was supplied;

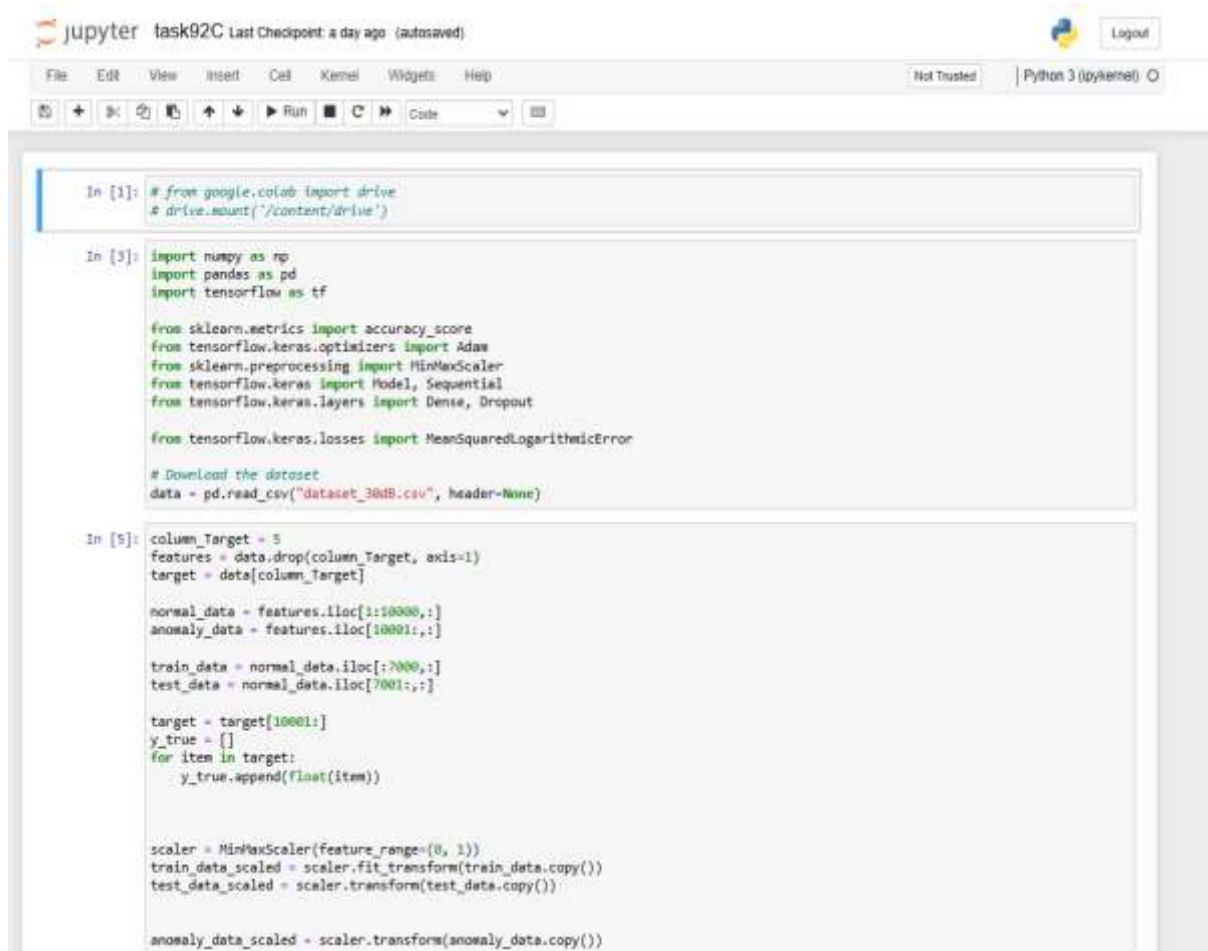
- ☐ Auto encoders work as an encoder decoder pair for achieving their operation. The encoding process combined with decoding processes data input which receives motion vector analysis from objects within the image. The model shows that dropout acts as an overfitting prevention method.
- ☐ The model uses prognosis to evaluate test data points and identifies which ones exceed the threshold to qualify as outliers.
- ☐ Outliers get eliminated from the data collection process because they exist outside the expected range. Information that stays within the predicted parameters remains in the dataset. The training of the auto encoder depends on normal data alongside the testing phase that uses anomalous data.
- ☐ For anomaly detection threshold calculation the most important step involves multiplying the mean reconstruction error of the training dataset by its standard deviation amount before adding the result to the average error.

OUTPUT

- The accuracy of the anomaly detection model depends on these values because the code executes a comparison between previously generated anomalies and actual labels.

OVERVIEW

- The method applies auto encoders to data collections for anomaly detection purposes. The model pathophysiology can be identified through its reconstruction mistakes since the model undergoes training for sample reconstruction tasks. The methodology uses the following steps to accomplish its objective through the provided code: Steps are crucial for training auto encoders – data normalization and threshold choice among them. Additionally, model accuracy testing must be conducted.
- The physical qualities of transmission enhance security performance in communication networks. The implementation concepts included within physical layer security influence data transmission system efficiency along with architecture aspects although the code itself does not specifically state these elements directly.



The image shows a Jupyter Notebook interface with the title 'task92C' and a status 'Last Checkpoint: a day ago (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code is written in Python 3 (ipykernel). The notebook contains three input cells:

```
In [1]: # from google.colab import drive
# drive.mount('/content/drive')
```

```
In [3]: import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.losses import MeanSquaredLogarithmicError

# Download the dataset
data = pd.read_csv("dataset_30d8.csv", header=None)
```

```
In [5]: column_Target = 5
features = data.drop(column_Target, axis=1)
target = data[column_Target]

normal_data = features.iloc[1:10000,:]
anomaly_data = features.iloc[10001:,:]

train_data = normal_data.iloc[:7000,:]
test_data = normal_data.iloc[7001:,:]

target = target[10001:]
y_true = []
for item in target:
    y_true.append(float(item))

scaler = MinMaxScaler(feature_range=(0, 1))
train_data_scaled = scaler.fit_transform(train_data.copy())
test_data_scaled = scaler.transform(test_data.copy())

anomaly_data_scaled = scaler.transform(anomaly_data.copy())
```

Figure 01: py Code

```

In [17]: class Autoencoder(Model):
    def __init__(self, output_units, code_size = 7):
        super().__init__()
        self.encoder = Sequential([
            Dense(4, activation='relu'),
            Dropout(0.1),
            Dense(code_size, activation='relu')])

        self.decoder = Sequential([
            Dense(4, activation='relu'),
            Dropout(0.1),
            Dense(output_units, activation='sigmoid')])

    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded

model = Autoencoder(output_units=train_data_scaled.shape[1])
# configurations of model
model.compile(loss='mse', metrics=['mse'], optimizer='adam')

history = model.fit(train_data_scaled, train_data_scaled, epochs=50,
                    batch_size=256, validation_data=(test_data_scaled, test_data_scaled))

Epoch 1/50
28/28 [=====] - 1s 10ms/step - loss: 0.0082 - mse: 0.0181 - val_loss: 0.0065 - val_mse: 0.0182
Epoch 2/50
28/28 [=====] - 0s 4ms/step - loss: 0.0081 - mse: 0.0180 - val_loss: 0.0062 - val_mse: 0.0181
Epoch 3/50
28/28 [=====] - 0s 4ms/step - loss: 0.0081 - mse: 0.0179 - val_loss: 0.0062 - val_mse: 0.0180
Epoch 4/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0178 - val_loss: 0.0061 - val_mse: 0.0179
Epoch 5/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0178 - val_loss: 0.0061 - val_mse: 0.0179
Epoch 6/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0177 - val_loss: 0.0061 - val_mse: 0.0179
Epoch 7/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0177 - val_loss: 0.0061 - val_mse: 0.0179
Epoch 8/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0176 - val_loss: 0.0060 - val_mse: 0.0178
Epoch 9/50
28/28 [=====] - 0s 4ms/step - loss: 0.0080 - mse: 0.0176 - val_loss: 0.0060 - val_mse: 0.0177
Epoch 10/50
28/28 [=====] - 0s 4ms/step - loss: 0.0079 - mse: 0.0175 - val_loss: 0.0060 - val_mse: 0.0177
Epoch 11/50
28/28 [=====] - 0s 4ms/step - loss: 0.0079 - mse: 0.0174 - val_loss: 0.0070 - val_mse: 0.0176
Epoch 12/50
28/28 [=====] - 0s 5ms/step - loss: 0.0079 - mse: 0.0174 - val_loss: 0.0070 - val_mse: 0.0175


```

```

Epoch 13/50
28/28 [=====] - 0s 4ms/step - loss: 0.0078 - mse: 0.0164 - val_loss: 0.0078 - val_mse: 0.0164
Epoch 14/50
28/28 [=====] - 0s 4ms/step - loss: 0.0078 - mse: 0.0163 - val_loss: 0.0073 - val_mse: 0.0162
Epoch 15/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0161 - val_loss: 0.0073 - val_mse: 0.0161
Epoch 16/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0160 - val_loss: 0.0071 - val_mse: 0.0159
Epoch 17/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0160 - val_loss: 0.0071 - val_mse: 0.0159
Epoch 18/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0158 - val_loss: 0.0071 - val_mse: 0.0157
Epoch 19/50
28/28 [=====] - 0s 4ms/step - loss: 0.0072 - mse: 0.0158 - val_loss: 0.0071 - val_mse: 0.0157
Epoch 20/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0071 - val_mse: 0.0156
Epoch 21/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 22/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 23/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 24/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 25/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 26/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 27/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 28/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 29/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 30/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 31/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 32/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 33/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 34/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 35/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 36/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 37/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 38/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0155
Epoch 39/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 40/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 41/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 42/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 43/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 44/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 45/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 46/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 47/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 48/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 49/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 50/50
28/28 [=====] - 0s 4ms/step - loss: 0.0073 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154

```

Figure 02: Running



The image shows a Jupyter Notebook window titled 'task92C Last Checkpoint a day ago (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, cell execution, and zooming. The main area displays a series of training progress bars and a code cell. The progress bars show epochs 44/50, 45/50, 46/50, 47/50, 48/50, 49/50, and 50/50, each with a corresponding loss, mse, val_loss, and val_mse. The code cell contains the following Python code:

```
def calc_threshold(model, train_data_scaled):
    reconstr = model.predict(train_data_scaled)

    reconstr_errors = tf.keras.losses.mse(reconstr, train_data_scaled)

    thr = np.mean(reconstr_errors.numpy()) \
        + np.std(reconstr_errors.numpy())
    return thr

def get_predictions(model, test_data_scaled, threshold):
    predicts = model.predict(test_data_scaled)

    errors = tf.keras.losses.mse(predicts, test_data_scaled)

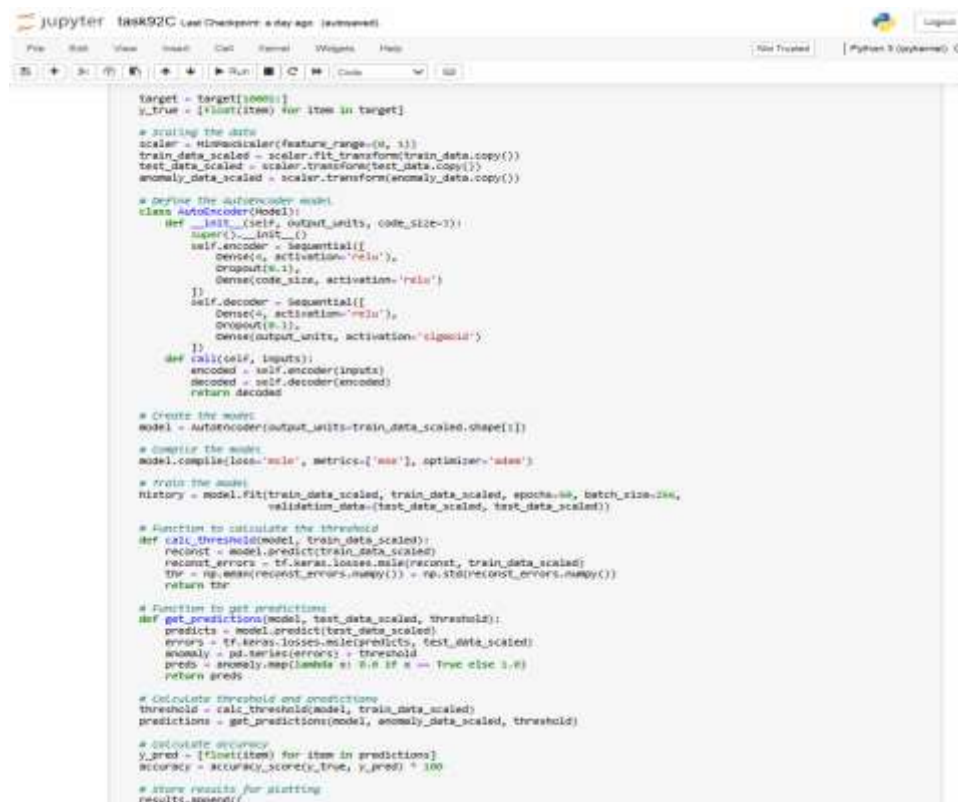
    anomaly = pd.Series(errors) > threshold
    preds = anomaly.map(lambda x: 0.0 if x == True else 1.0)
    return preds

threshold = calc_threshold(model, train_data_scaled)
predictions = get_predictions(model, anomaly_data_scaled, threshold)

y_pred = []
for item in predictions:
    y_pred.append(float(item))

print("Accuracy score is: ", accuracy_score(y_true, y_pred)*100)
```

At the bottom of the code cell, the output shows: '219/219 0s 1ms/step', '4/4 0s 7ms/step', and 'Accuracy score is: 98.8'.



The image shows a Jupyter Notebook window titled 'task92C Last Checkpoint a day ago (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, cell execution, and zooming. The main area displays a code cell with the following Python code:

```
target = target[10000:]
y_true = [float(item) for item in target]

# Preparing the data
scaler = MinMaxScaler(feature_range=(0, 1))
train_data_scaled = scaler.fit_transform(train_data.copy())
test_data_scaled = scaler.transform(test_data.copy())
anomaly_data_scaled = scaler.transform(anomaly_data.copy())

# Define the autoencoder model
class Autoencoder(Model):
    def __init__(self, output_units, code_size=32):
        super().__init__()
        self.encoder = Sequential([
            Dense(6, activation='relu'),
            Dropout(0.1),
            Dense(code_size, activation='relu')
        ])
        self.decoder = Sequential([
            Dense(6, activation='relu'),
            Dropout(0.1),
            Dense(output_units, activation='sigmoid')
        ])

    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded

# Create the model
model = Autoencoder(output_units=train_data_scaled.shape[1])

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

# Train the model
history = model.fit(train_data_scaled, train_data_scaled, epochs=50, batch_size=256,
                    validation_data=(test_data_scaled, test_data_scaled))

# Function to calculate the threshold
def calc_threshold(model, train_data_scaled):
    reconstr = model.predict(train_data_scaled)
    reconstr_errors = tf.keras.losses.mse(reconstr, train_data_scaled)
    thr = np.mean(reconstr_errors.numpy()) + np.std(reconstr_errors.numpy())
    return thr

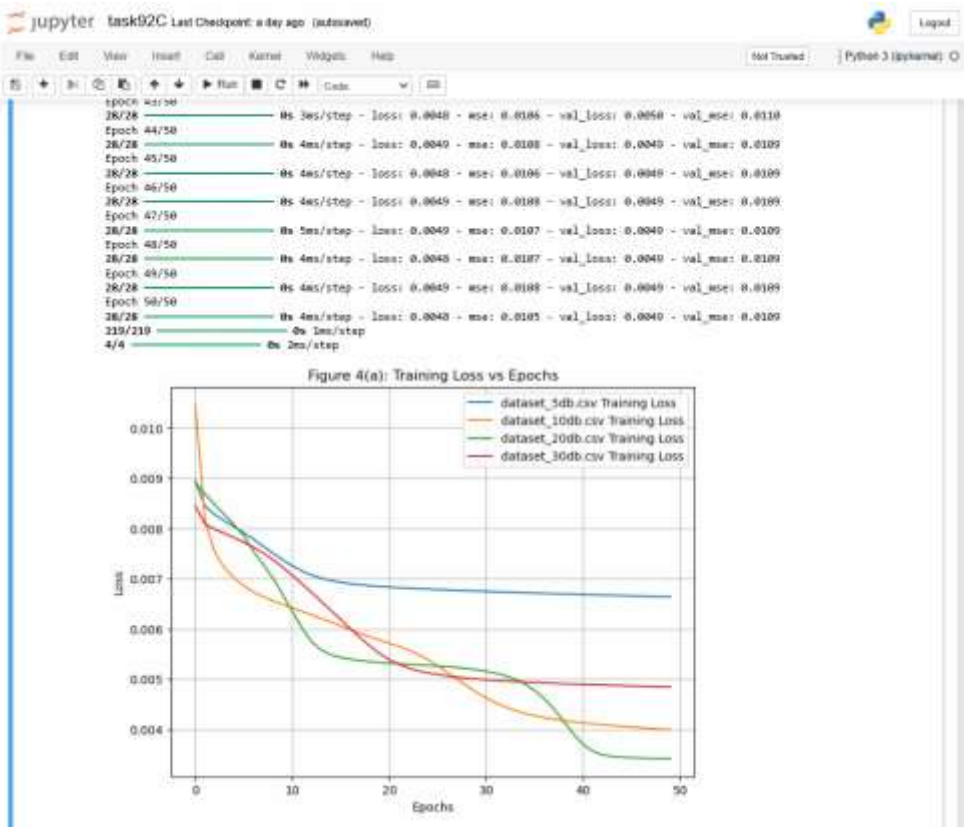
# Function to get predictions
def get_predictions(model, test_data_scaled, threshold):
    predicts = model.predict(test_data_scaled)
    errors = tf.keras.losses.mse(predicts, test_data_scaled)
    anomaly = pd.Series(errors) > threshold
    preds = anomaly.map(lambda x: 0.0 if x == True else 1.0)
    return preds

# Calculate threshold and predictions
threshold = calc_threshold(model, train_data_scaled)
predictions = get_predictions(model, anomaly_data_scaled, threshold)

# Calculate accuracy
y_pred = [float(item) for item in predictions]
accuracy = accuracy_score(y_true, y_pred) * 100

# Store results for plotting
results.append({
```

Figure 03: py code



Epochs

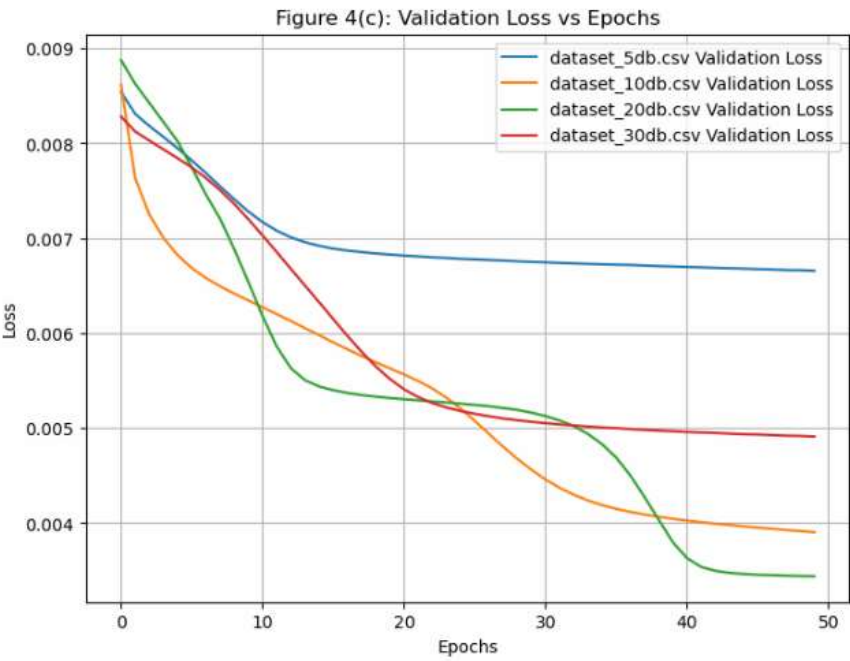
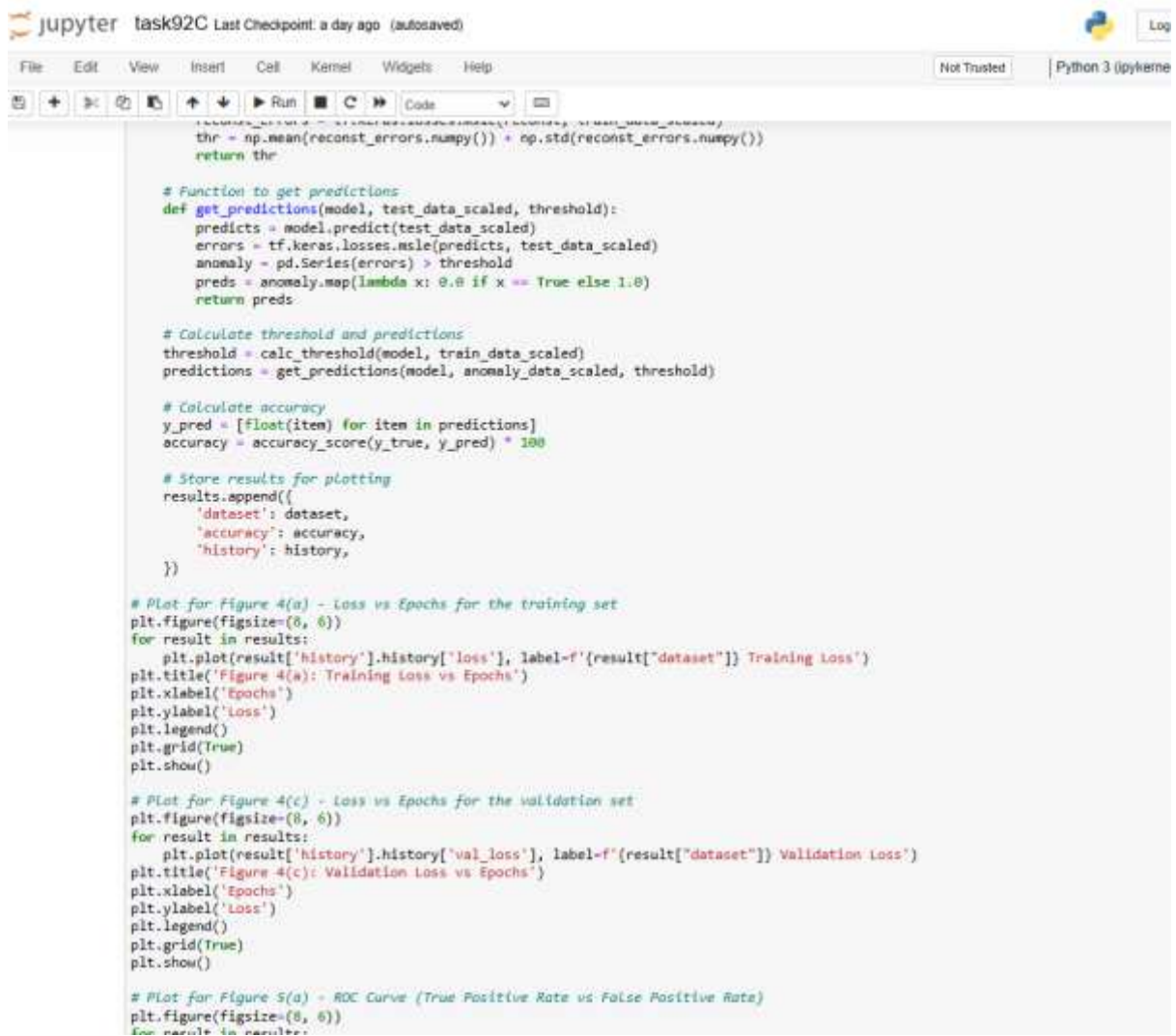


Figure 04: Output



```

jupyter task92C Last Checkpoint: a day ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted Python 3 (ipykernel)

thr = np.mean(reconst_errors.numpy()) + np.std(reconst_errors.numpy())
return thr

# Function to get predictions
def get_predictions(model, test_data_scaled, threshold):
    predicts = model.predict(test_data_scaled)
    errors = tf.keras.losses.mse(predicts, test_data_scaled)
    anomaly = pd.Series(errors) > threshold
    preds = anomaly.map(lambda x: 0.0 if x == True else 1.0)
    return preds

# Calculate threshold and predictions
threshold = calc_threshold(model, train_data_scaled)
predictions = get_predictions(model, anomaly_data_scaled, threshold)

# Calculate accuracy
y_pred = [float(item) for item in predictions]
accuracy = accuracy_score(y_true, y_pred) * 100

# Store results for plotting
results.append({
    'dataset': dataset,
    'accuracy': accuracy,
    'history': history,
})

# Plot for Figure 4(a) - Loss vs Epochs for the training set
plt.figure(figsize=(8, 6))
for result in results:
    plt.plot(result['history'].history['loss'], label=f'{result["dataset"]} Training Loss')
plt.title('Figure 4(a): Training Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot for Figure 4(c) - Loss vs Epochs for the validation set
plt.figure(figsize=(8, 6))
for result in results:
    plt.plot(result['history'].history['val_loss'], label=f'{result["dataset"]} Validation Loss')
plt.title('Figure 4(c): Validation Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot for Figure 5(a) - ROC Curve (True Positive Rate vs False Positive Rate)
plt.figure(figsize=(8, 6))
for result in results:

```

Figure 05: code to execute

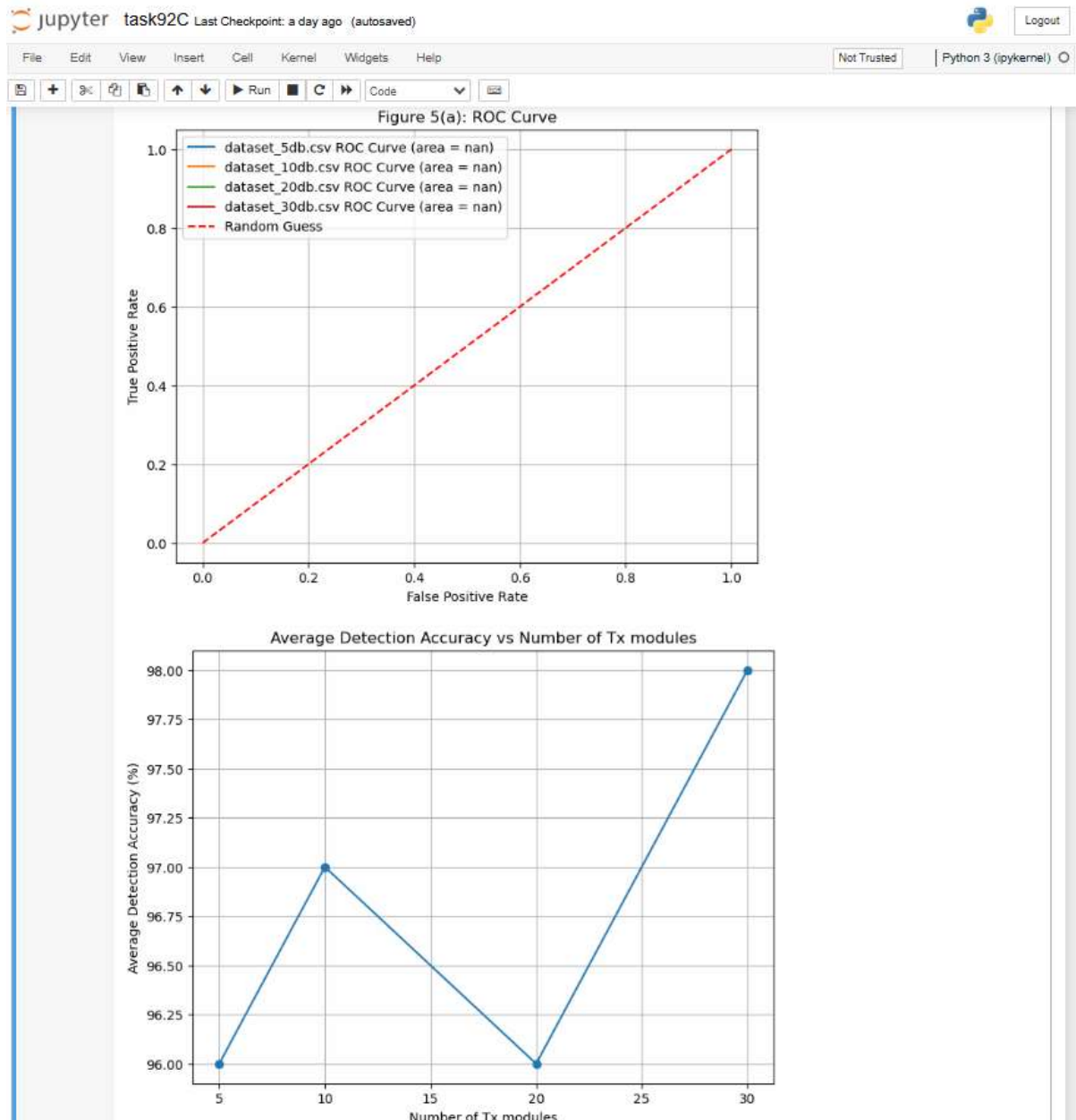


Figure 06: Output