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
I have read and understood both *Deakin Academic Misconduct Policy* and the *Referencing and Bibliographies* document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly. I confirm that I have not exceeded the stipulated word limit by more than 10%. I confirm that this is my own work and that I have not colluded or plagiarized any part of it.	
Candidate Signature:	J. Kc.
Date:	03/12/2024

Task 9.2C
Table of Content

Table of Figures

Table of Acronyms

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

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

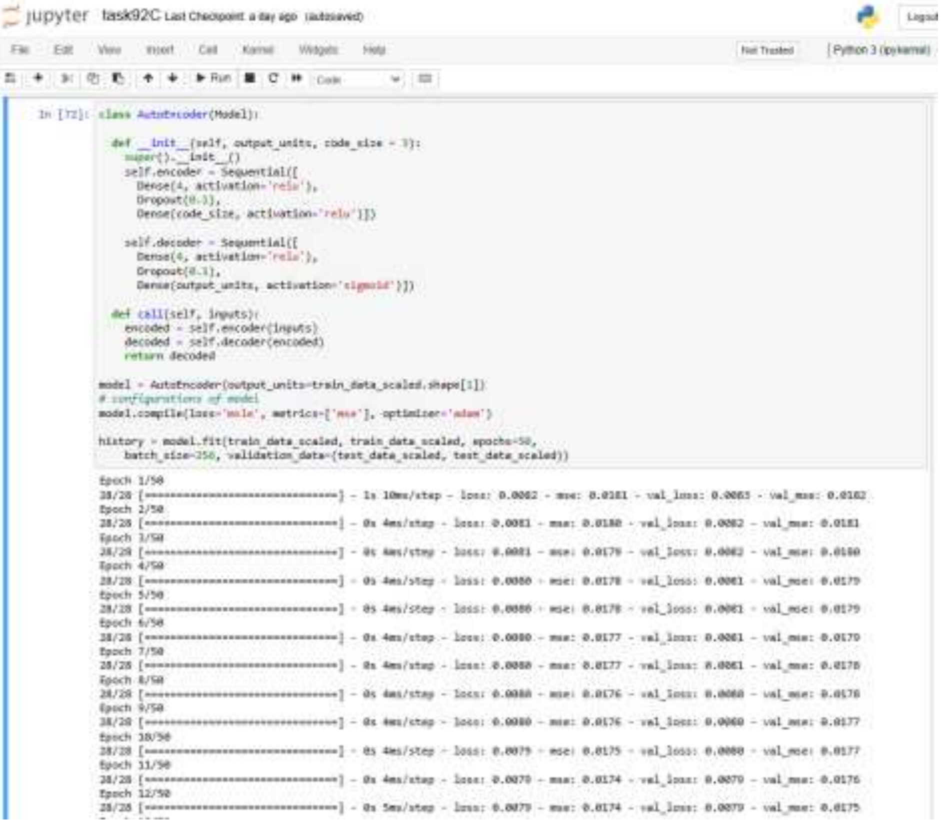
- The method applies auto encoders to data collections for anomaly detection purposes. The model pathophysiology can be identified through its reconstruction mistakes since the model undergoes training for sample reconstruction tasks. The methodology uses the following steps to accomplish its objective 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.

jupyter task92C Last Checkpoint: a day ago (autosaved)

```
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_30dB.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



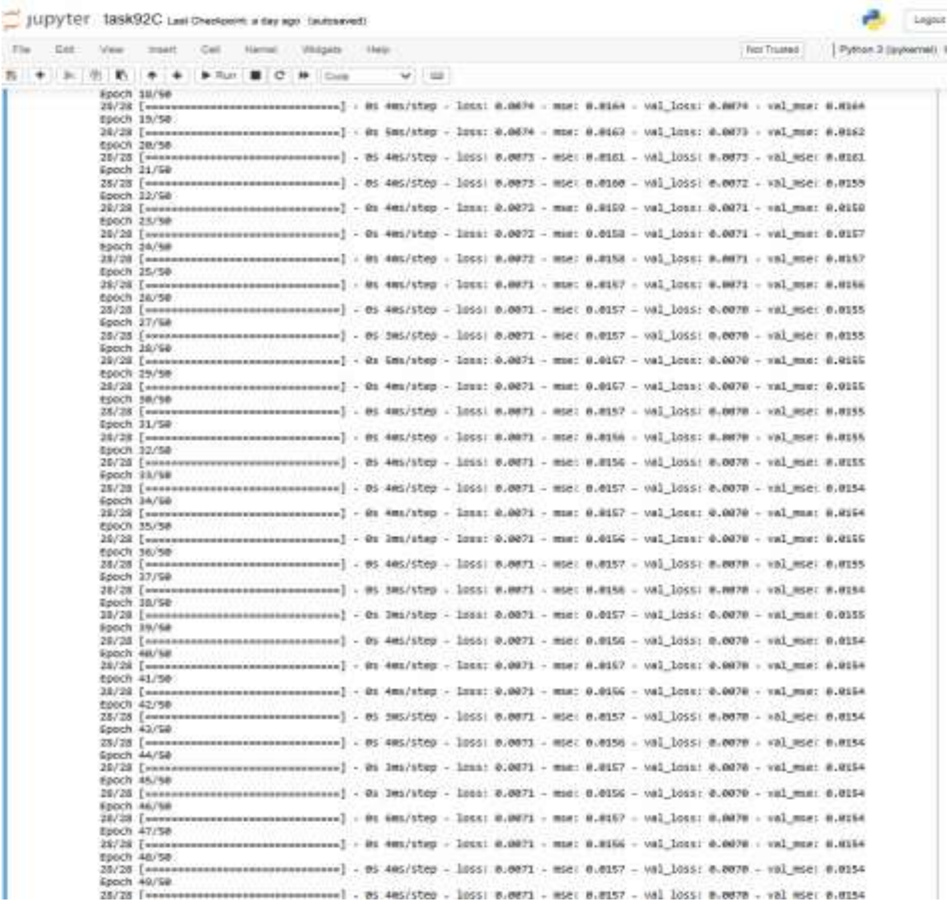
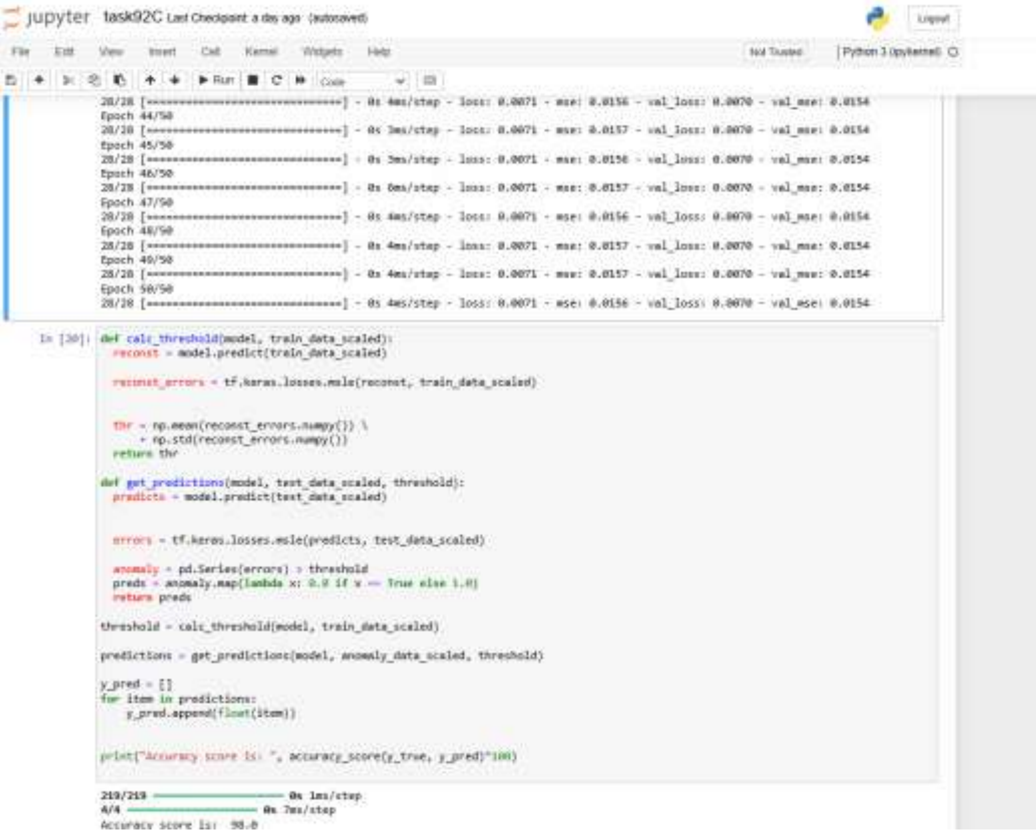


Figure 02: Running


```
28/28 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 44/50
28/28 [==============================] - 0s 3ms/step - loss: 0.0071 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 45/50
28/28 [==============================] - 0s 3ms/step - loss: 0.0071 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 46/50
28/28 [==============================] - 0s 6ms/step - loss: 0.0071 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 47/50
28/28 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 48/50
28/28 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 49/50
28/28 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0157 - val_loss: 0.0070 - val_mse: 0.0154
Epoch 50/50
28/28 [==============================] - 0s 4ms/step - loss: 0.0071 - mse: 0.0156 - val_loss: 0.0070 - val_mse: 0.0154

In [30]: def calc_threshold(model, train_data_scaled):
  reconst = model.predict(train_data_scaled)

  reconst_errors = tf.keras.losses.msle(reconst, train_data_scaled)


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

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


  errors = tf.keras.losses.msle(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)

219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
Accuracy score is:  98.0
```

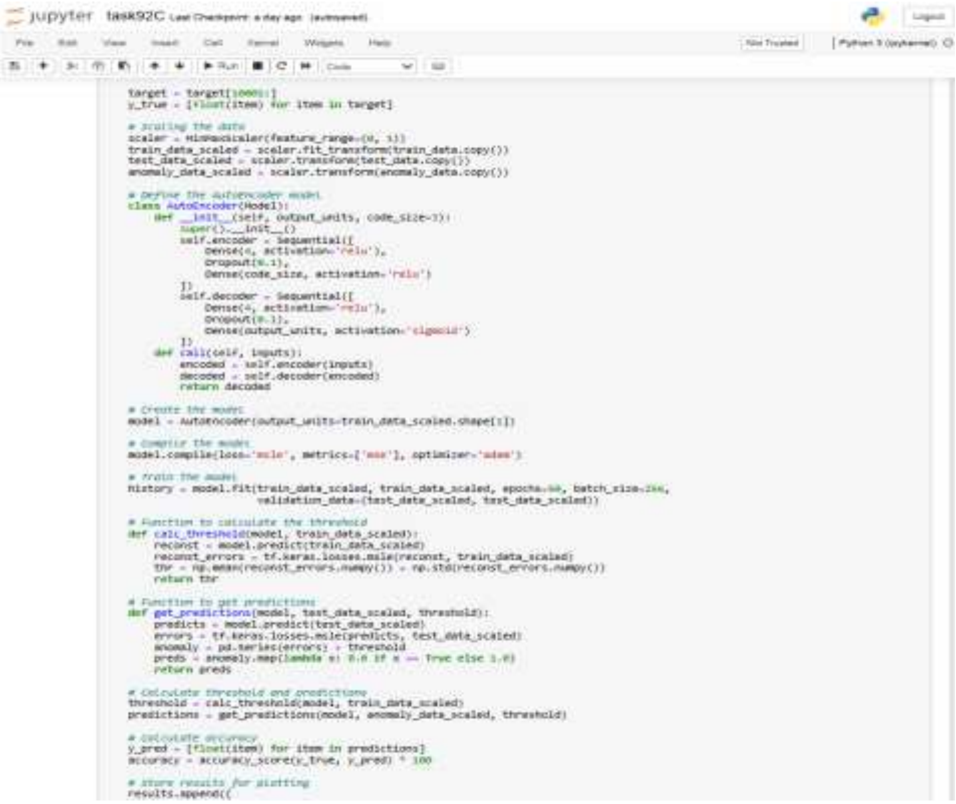
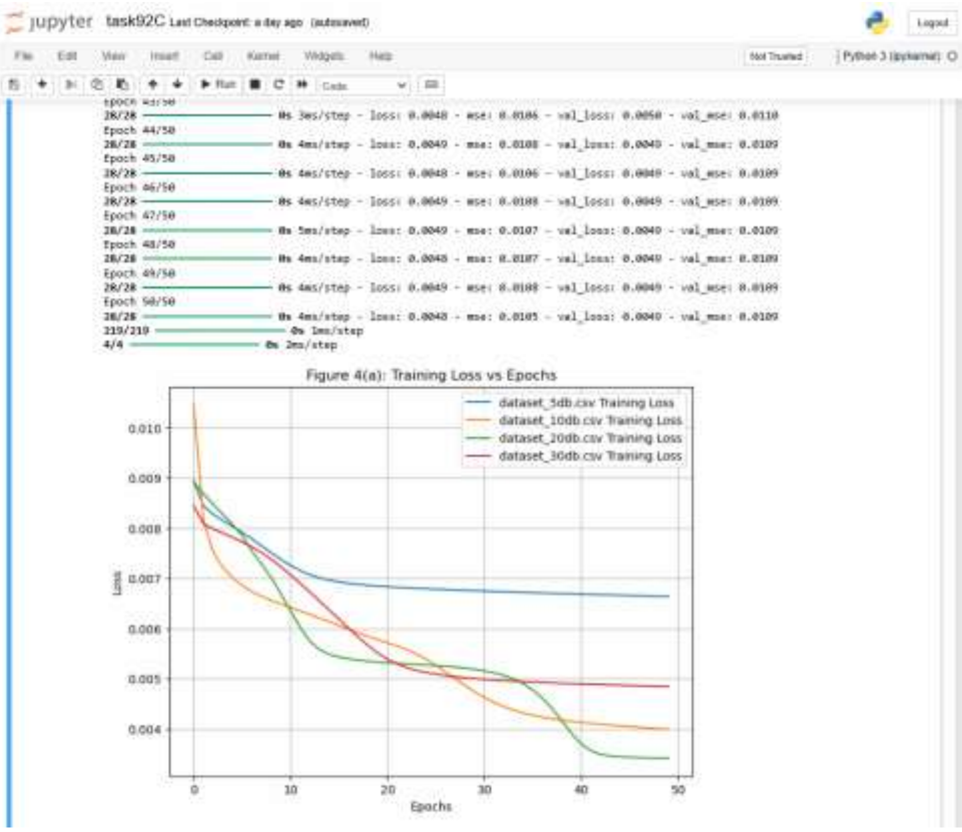

```
target = target[1000:]
y_true = [float(item) for item in target]

# scaling 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=3):
        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

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

# Compile the model
model.compile(loss='msle', 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):
    reconst = model.predict(train_data_scaled)
    reconst_errors = tf.keras.losses.msle(reconst, train_data_scaled)
    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.msle(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

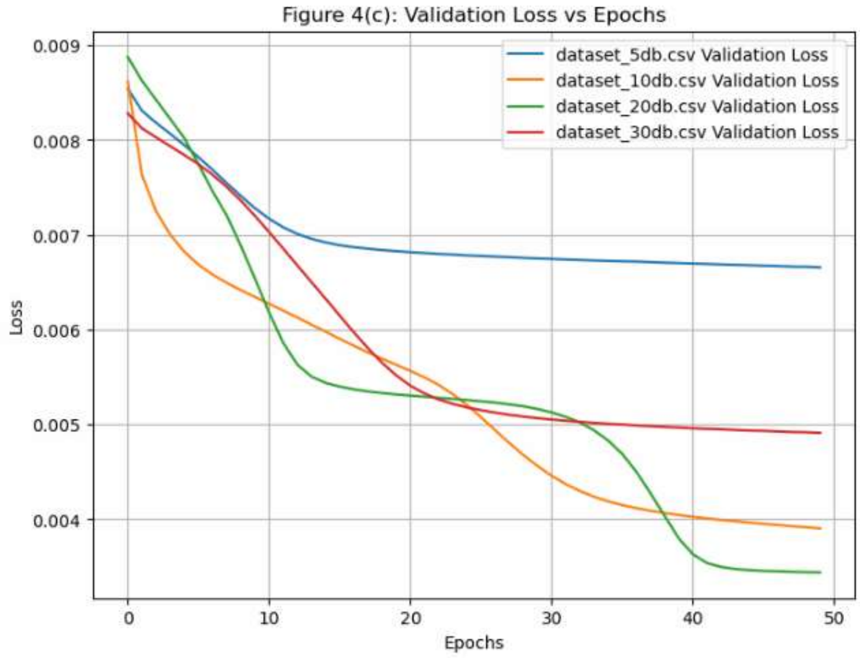


Figure 04: Output

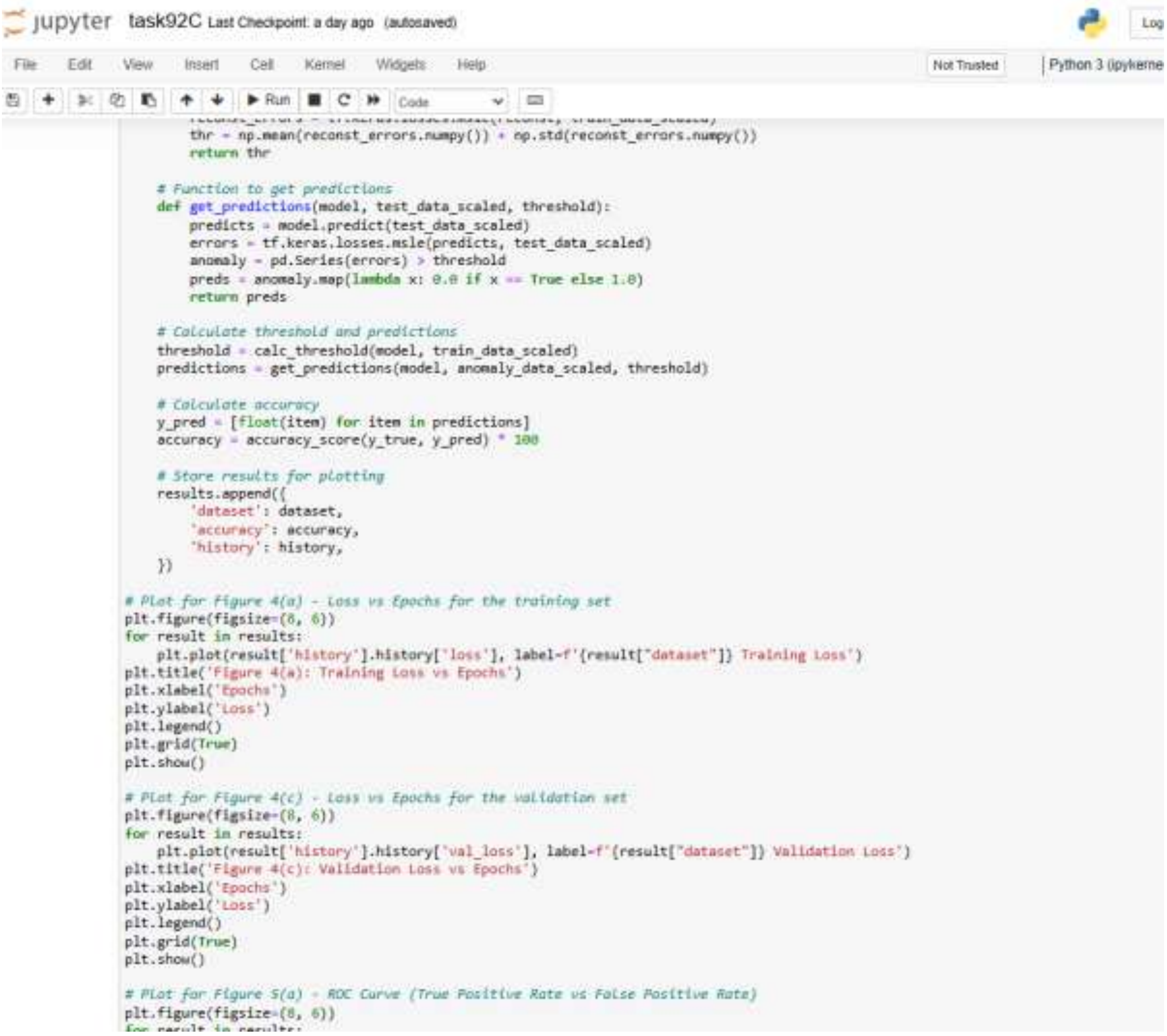


Figure 05: code to execute

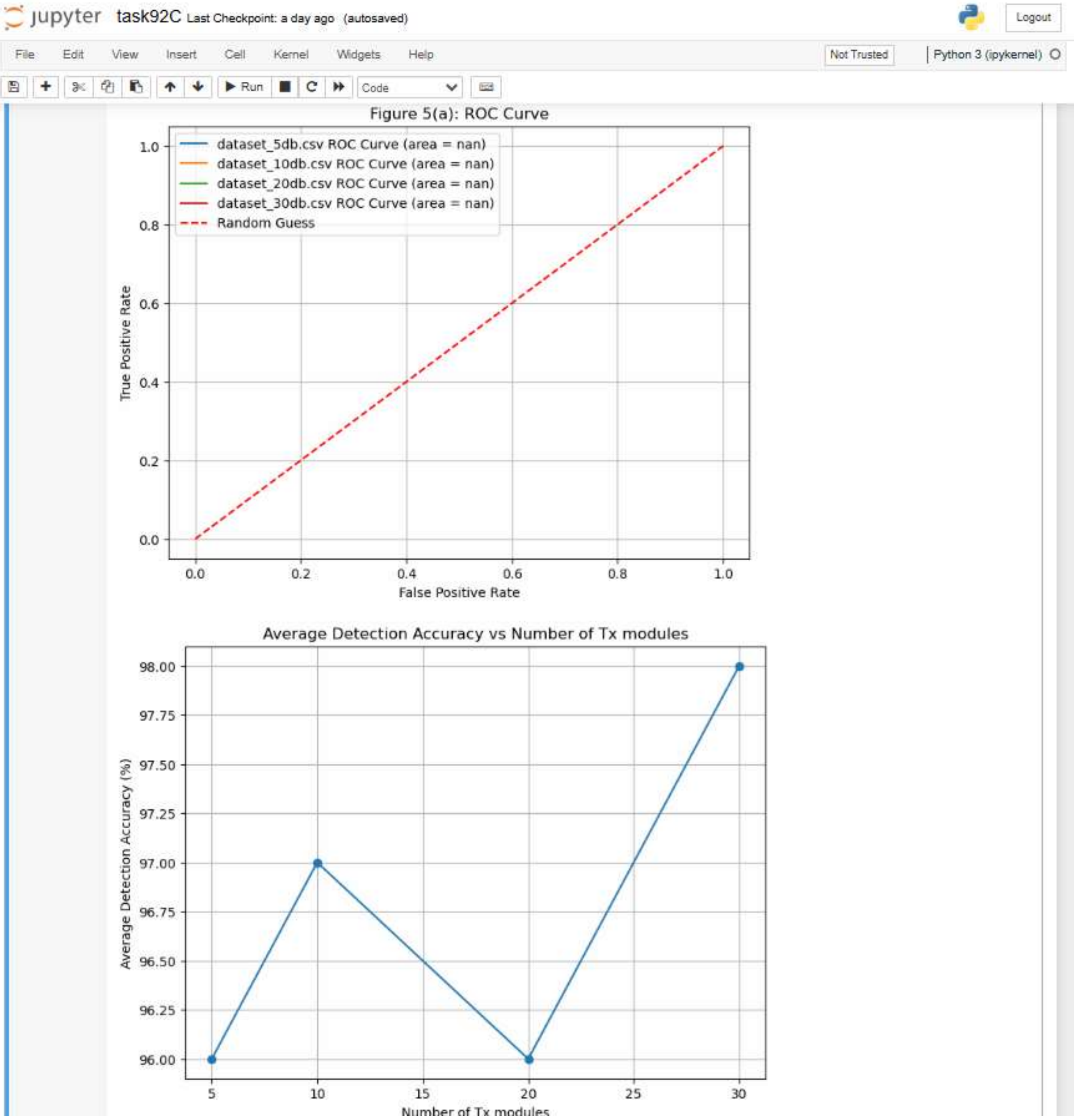


Figure 06: Output