Configuration Manual

MSc Research Project
Data Analytics

Chukwuemeka Okonji
Student ID: 22103970

School of Computing
National College of Ireland

Supervisor: Dr.Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	CHUKWUEMEKA NWANZE OKONJI
Student ID:	2010
Programme:	MSC IN DATA ANALYTICS **Year:** 2023/2024
Module:	MSC RESEARCH PROJECT
Lecturer:	
Submission Due Date:	
Project Title:	ELECTRICITY PRICE FORECASTING IN THE IRELAND DAY AHEAD MARKET: A MACHINE LEARNING APPROACH
Word Count:	**Page Count:** 12

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: CHUKWUEMEKA OKONJI

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

Chukwuemeka Nwanze Okonji
Student ID: 22103970

1 Introduction

The system setup, software hardware specifications, and activities carried out for the implementation of the Research Project: Electricity Price Forecasting in the Ireland Day Ahead Market: A Machine Learning Approach are detailed in this configuration manual. Section 2 details the hardware configuration while Section 3 details the software configuration. Section 4 describes the data collection and Section 5 details the data preparation and transformation. Section 6 and 7 describes the implementation and results respectively.

2 Hardware Configuration

The Project was conduct on an HP ENVY System with the following configuration:
Device name DESKTOP-NDMMB50
Processor Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Installed RAM16.0 GB (15.8 GB usable)
System type 64-bit operating system, x64-based processor
Pen and touch Pen and touch support with 10 touch points

EditionWindows 11 Pro
Version 22H2
Installed on 4/4/2023
OS build 22621.2861
Experience Windows Feature Experience Pack 1000.22681.1000.0

3 Software Configuration

Python Programming language: this is an open-source language that one of the most widely used. For this project, we would be using the following python library to carry out various tasks:

1. **warnings**: Python built-in library for issuing warning messages.
2. **os**: Python built-in library for interacting with the operating system.
3. **pandas (pd)**: Data manipulation and analysis library.
4. **numpy (np)**: Numerical computing library.
5. **math**: Python built-in library for mathematical operations.
6. **datetime (dt)**: Python built-in library for working with dates and times.
7. **matplotlib.pyplot as plt**: Data visualization library.
8. **seaborn as sns**: Statistical data visualization library based on Matplotlib.
9. **pprint**: Pretty-print data structures.
10. **%matplotlib inline**: IPython magic command for displaying plots inline.

11. **sklearn.metrics**: Metrics for evaluating machine learning models from scikit-learn.
12. **MinMaxScaler**: Feature scaling for machine learning models.
13. **itertools.product**: Efficiently generates Cartesian products.
14. **statsmodels.api as sm**: Provides classes and functions for estimating and testing statistical models.
15. **tensorflow as tf**: Open-source machine learning framework.
16. **tensorflow.keras.models**: Neural network models API for TensorFlow.
17. **tensorflow.keras.layers**: Keras layers for building neural networks.
18. **tensorflow.keras.optimizers**: Optimizers for training Keras models.
19. **tensorflow.keras.losses**: Loss functions for training Keras models.
20. **tensorflow.keras.metrics**: Metrics for evaluating Keras models.
21. **cycle from itertools**: Infinite iterators.
22. **plotly.offline as py**: Plotly library for creating interactive plots.
23. **plotly.graph_objects as go**: Plotly's graph objects for creating figures.
24. **plotly.express as px**: High-level interface for creating various charts with Plotly.
25. **plotly.subplots**: Create a subplot with Plotly.
26. **files from google.colab**: Module for interacting with Google Colab file system.
27. **statsmodels.tsa.seasonal**: Seasonal decomposition tools for time series analysis.
28. **statsmodels.tsa.stattools**: Tools for time series analysis.
29. **keras.models (Sequential)**: Neural network models API for Keras.
30. **keras.layers (LSTM, Dense, Dropout, Conv1D, Input, Flatten)**: Keras layers for building neural networks.
31. **keras.optimizers (Adam)**: Optimizers for training Keras models.
32. **keras.losses (MeanSquaredError, MeanAbsoluteError)**: Loss functions for training Keras models.
33. **keras.metrics (RootMeanSquaredError, MeanAbsolutePercentageError)**: Metrics for evaluating Keras models.
34. **keras.callbacks (ModelCheckpoint, EarlyStopping)**: Callbacks for Keras models.

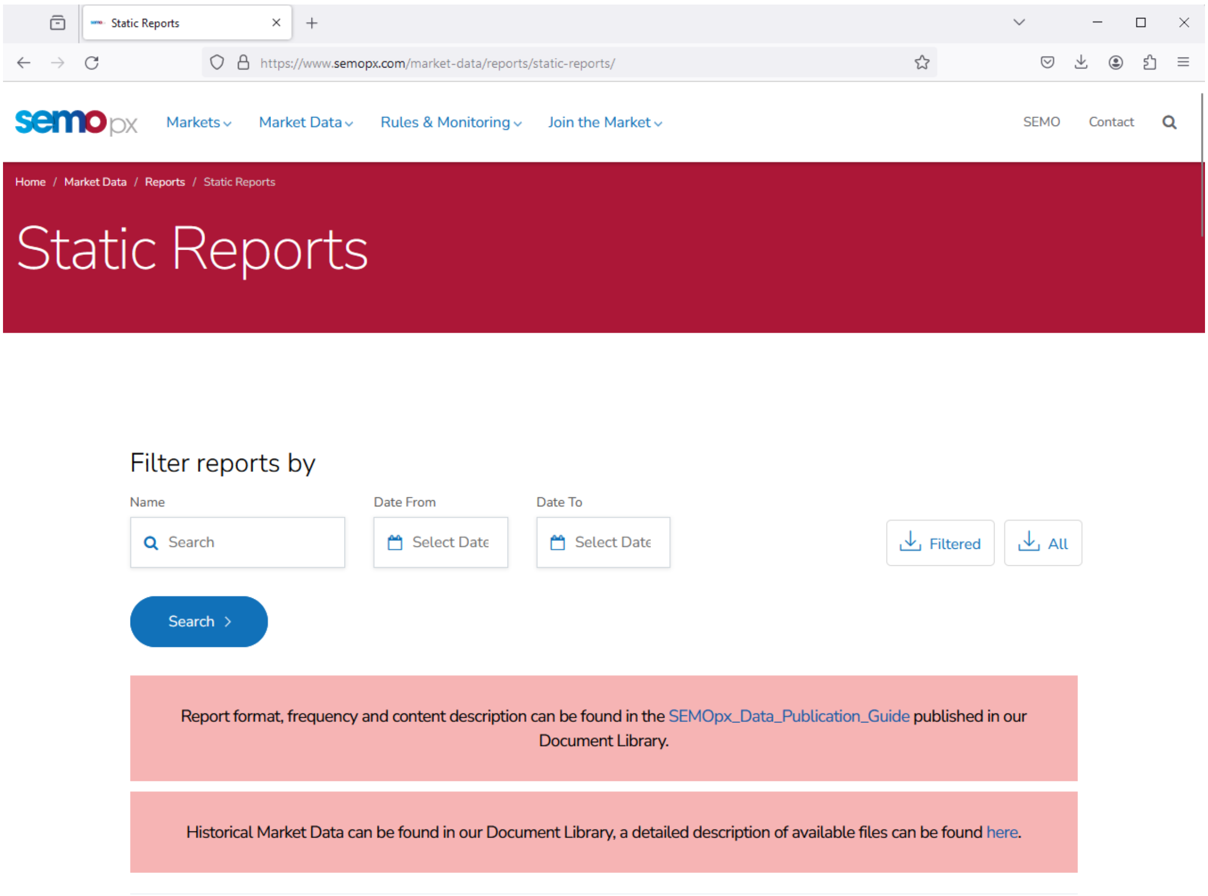
Microsoft Excel: Excel is a spreadsheet application widely used for data cleaning, manipulation, and initial exploration due to its user-friendly interface.

Google Collab: This cloud-based platform offers a collaborative environment for Python scripting, with the added benefits of free access to GPUs and ease of sharing, which enhances the computational capabilities and teamwork.

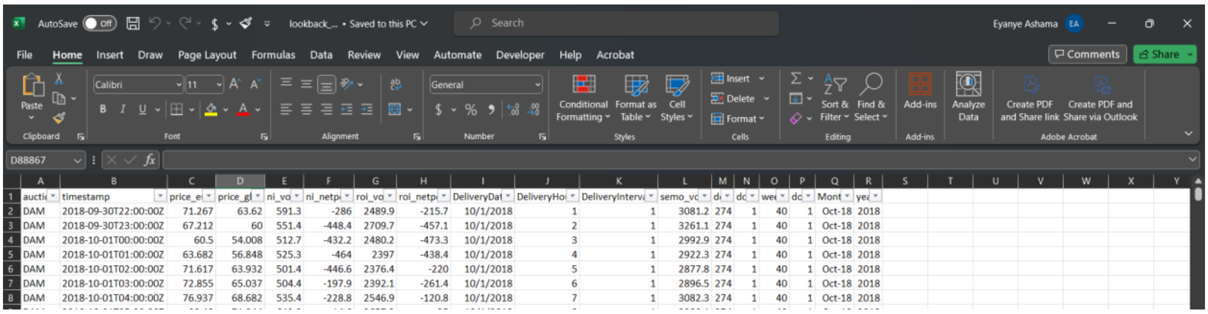
Tableau: Specialized in data visualization, Tableau provides intuitive and interactive dashboards that enable researchers to explore and present data in a visually compelling manner, thereby uncovering patterns and insights that might otherwise remain hidden.

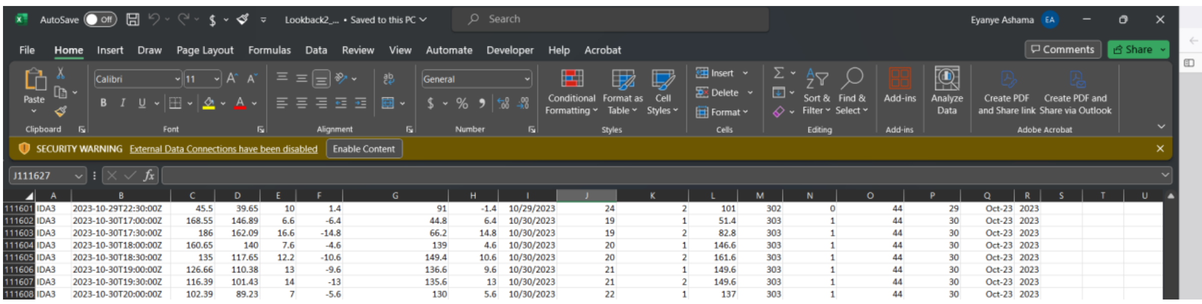
4 Data Collection

The data for this project was gotten from the static report page of the Semepx website. see screenshot below:

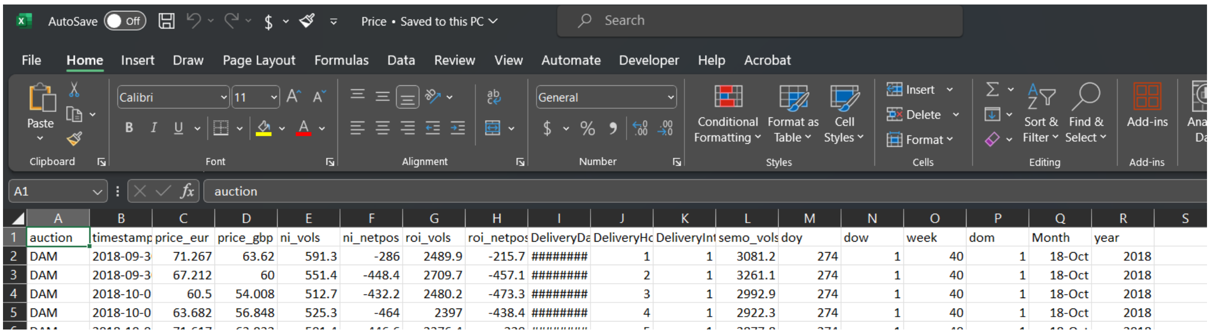


Two data set were downloaded as seen in the screen shot below:





And there combined into one data set with only the ‘DAM’ auction data as seen below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	auction	timestamp	price_eur	price_gbp	ni_vols	ni_netpos	roi_vols	roi_netpos	DeliveryDa	DeliveryHc	DeliveryInt	semo_vols	doy	dow	week	dom	Month	year	
2	DAM	2018-09-3	71.267	63.62	591.3	-286	2489.9	-215.7	########	1	1	3081.2	274	1	40	1	18-Oct	2018	
3	DAM	2018-09-3	67.212	60	551.4	-448.4	2709.7	-457.1	########	2	1	3261.1	274	1	40	1	18-Oct	2018	
4	DAM	2018-10-0	60.5	54.008	512.7	-432.2	2480.2	-473.3	########	3	1	2992.9	274	1	40	1	18-Oct	2018	
5	DAM	2018-10-0	63.682	56.848	525.3	-464	2397	-438.4	########	4	1	2922.3	274	1	40	1	18-Oct	2018	

5 Data Preparation and Transformation

The data preparation steps include the following:
First of all splitting the data

Splitting the Data

```
[25] prediction_hours = 9504 # there are 9,504 hours between 1st October 2022 and 31st October 2023 which is used for testing the model

     # Set Train data to be uplo ( Total data length - prediction_hours )
     df_train = price[:len(price)-prediction_hours].values.reshape(-1,1)


     # Set Test data to be the last prediction_hours (or 1550 days in this case)
     df_test = price[len(price)-prediction_hours:].values.reshape(-1,1)
```

Then normalizing the data:

Min Max Scaling of Data post Train-Test Split

```
[27] scaler_train = MinMaxScaler(feature_range=(0, 1))
     scaled_train = scaler_train.fit_transform(df_train)

     scaler_test = MinMaxScaler(feature_range=(0, 1))
     scaled_test = scaler_test.fit_transform(df_test)
```

Afterwhich, the scaled data is prepared for the data models as seen in the screenshot below:

Data Generation for LSTM

```
def dataset_generator_lstm(dataset, look_back=24):
    """
    Generates input-output pairs for an LSTM dataset.

    Args:
        dataset (numpy.ndarray): The dataset to generate input-output pairs from.
        look_back (int): The number of previous timesteps to use for prediction.

    Returns:
        numpy.ndarray: The input sequences.
        numpy.ndarray: The corresponding output values.
    """
    # A "lookback period" defines the window-size of how many
    # previous timesteps are used in order to predict
    # the subsequent timestep.
    dataX, dataY = [], []

    # Iterate over the dataset, considering a "look_back" window of previous timesteps
    for i in range(len(dataset) - look_back):
        window_size_x = dataset[i:(i + look_back), 0]
        dataX.append(window_size_x)
        dataY.append(dataset[i + look_back, 0]) # this is the label or actual y-value
    return np.array(dataX), np.array(dataY)

trainX, trainY = dataset_generator_lstm(scaled_train)

testX, testY = dataset_generator_lstm(scaled_test)

print("trainX: ", trainX.shape)
print("trainY: ", trainY.shape)
print("testX: ", testX.shape)
print("testY", testY.shape)
```

And then reshaping the train set for the LSTM model

Reshaping the input into a 3D Tensor of [batch_size, timesteps, features]

```
[29] # First check the current shape of trainX and testX
     print(trainX.shape)
     print(testX.shape)

     (35039, 24)
     (9480, 24)
```

```
[30] # And now reshape trainX and testX
     # LSTM input data must be in the form: [batch_size, timesteps, input_dim]

     trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))

     testX = np.reshape(testX, (testX.shape[0], testX.shape[1], 1 ))

     print("Shape of trainX: ", trainX.shape)
     print("Shape of testX: ", testX.shape)

     Shape of trainX:  (35039, 24, 1)
     Shape of testX:  (9480, 24, 1)
```

6 Model Implementation

To run the LSTM model, this is the code for the single LSTM model:

Simple LSTM Model

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from keras.losses import MeanSquaredError, MeanAbsoluteError
from keras.metrics import RootMeanSquaredError, MeanAbsolutePercentageError
from keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import load_model

# Create a sequential model
model = Sequential()

# Adding the first LSTM layer
model.add(LSTM(units=128, input_shape=(trainX.shape[1], trainX.shape[2])))

# Optional: Adding more LSTM layers
#model.add(LSTM(units=64, return_sequences=True))  # Add as many as needed
#model.add(Dropout(0.2))  # Dropout layer after each LSTM layer

# Adding Dense layers
#model.add(Dense(8, 'relu'))
model.add(Dense(1, 'linear'))

# Print the model summary
model.summary()
```

Subsequent model are done by adding layers to the code, for stacked LSTM, see figure below:

Stacked LSTM Model

```
# Create a sequential model
model3 = Sequential()

# Adding the first LSTM layer
model3.add(LSTM(units=128, return_sequences=True, input_shape=(trainX.shape[1], trainX.shape[2])))

# Optional: Adding one more LSTM layer
model3.add(LSTM(units=64, input_shape = (trainX.shape[1], trainX.shape[2])))  # Add as many as needed
model3.add(Dropout(0.2))  # Dropout layer after each LSTM layer

# Adding Dense layers
# model.add(Dense(8, 'relu'))
model3.add(Dense(1, 'linear'))

# Print the model summary
model3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 24, 128)           66560

 lstm_3 (LSTM)               (None, 64)                49408

 dropout (Dropout)           (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 116033 (453.25 KB)
Trainable params: 116033 (453.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

To compare the result of the models, the code belwo is run:

Comparing the LSTM Models

```
[ ] import pandas as pd

    def create_evaluation_dataframe(model_name, evaluation_results):
        """
        Create a DataFrame from model evaluation results.

        Args:
            model_name (str): Name of the model.
            evaluation_results (list): List containing evaluation results.

        Returns:
            pd.DataFrame: DataFrame with model evaluation results.
        """
        columns = ['Model', 'Loss', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolute Percentage Error']
        data = [model_name] + evaluation_results
        result_df = pd.DataFrame([data], columns=columns)
        return result_df

    # Assuming you already have evaluation results for each model
    # evaluation_results1, evaluation_results2, ..., evaluation_results6

    # Create DataFrames for each model's evaluation results
    results1 = create_evaluation_dataframe('LSTM', evaluation_results)
    results2 = create_evaluation_dataframe('LSTM with Dense layer', evaluation_results2)
    results3 = create_evaluation_dataframe('Stacked LSTM - 2 layers', evaluation_results3)
    results4 = create_evaluation_dataframe('Stacked LSTM - 2 layers with Dense Layer', evaluation_results4)
    results5 = create_evaluation_dataframe('Stacked LSTM - 3 layers', evaluation_results5)
    results6 = create_evaluation_dataframe('Stacked LSTM - 4 layers', evaluation_results6)

    # Concatenate the DataFrames to create a single table
    all_LSTM_results = pd.concat([results1, results2, results3, results4, results5, results6], ignore_index=True)

    # Display the table
    all_LSTM_results
```

For MLP implementation, the similar architecture is used but with slight differences, First the input is reshaped for the MLP model

MLP Implementation

```
[32] # Reshape for MLP
     mlp_trainX = np.reshape(trainX, (trainX.shape[0], trainX.shape[1] * trainX.shape[2]))
     mlp_testX = np.reshape(testX, (testX.shape[0], testX.shape[1] * testX.shape[2]))
```

```
[ ] print("Shape of mlp_trainX: ", mlp_trainX.shape)
    print("Shape of mlp_testX: ", mlp_testX.shape)

    Shape of mlp_trainX:  (35039, 24)
    Shape of mlp_testX:  (9480, 24)
```

Then model is then built, see the figure below:

```python
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras.losses import MeanSquaredError, MeanAbsoluteError
from keras.metrics import RootMeanSquaredError, MeanAbsolutePercentageError
from keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.models import load_model
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```python
# Create an MLP model
mlp_model = Sequential()

# Adding Dense layers
mlp_model.add(Dense(64, activation='relu', input_shape=(mlp_trainX.shape[1],)))
mlp_model.add(Dropout(0.2))  # Dropout for regularization

mlp_model.add(Dense(32, activation='relu'))
mlp_model.add(Dropout(0.2))

mlp_model.add(Dense(1, activation='linear'))  # Output layer for regression

# Print the model summary
mlp_model.summary()
```

Comparison for the MLP model is similar to that of the LSTM, just that the variables are changed. See the code below:

Comparing MLP Models

```python
import pandas as pd

def create_mlp_evaluation_dataframe(model_name, evaluation_results):
    """
    Create a DataFrame from MLP model evaluation results.

    Args:
        model_name (str): Name of the MLP model.
        evaluation_results (list): List containing MLP model evaluation results.

    Returns:
        pd.DataFrame: DataFrame with MLP model evaluation results.
    """
    columns = ['MLP Model', 'Loss', 'Root Mean Squared Error', 'Mean Absolute Error', 'Mean Absolute Percentage Error']
    data = [model_name] + evaluation_results
    result_df = pd.DataFrame([data], columns=columns)
    return result_df

# Assuming you already have evaluation results for each MLP model
# mlp_evaluation_results1, mlp_evaluation_results2, ..., mlp_evaluation_results5

# Create DataFrames for each MLP model's evaluation results
mlp_results1 = create_mlp_evaluation_dataframe('MLP Model 1', mlp_evaluation_results)
mlp_results2 = create_mlp_evaluation_dataframe('MLP Model 2', mlp_evaluation_results2)
mlp_results3 = create_mlp_evaluation_dataframe('MLP Model 3', mlp_evaluation_results3)
mlp_results4 = create_mlp_evaluation_dataframe('MLP Model 4', mlp_evaluation_results4)
mlp_results5 = create_mlp_evaluation_dataframe('MLP Model 5', mlp_evaluation_results5)

# Concatenate the DataFrames to create a single table
all_mlp_results = pd.concat([mlp_results1, mlp_results2, mlp_results3, mlp_results4, mlp_results5], ignore_index=True)

# Display the table
all_mlp_results
```

Lastly, for the CNN-LSTM model, the screenshot below shows the code for the first model implementation. Subsequentl model are built by adjusting the parameters of this model.

CNN-LSTM

```
from tensorflow.keras.layers import LSTM, Dense, Dropout, Conv1D, Input, Flatten

# CNN-LSTM Model
model_cnnlstm = Sequential()
model_cnnlstm.add(Conv1D(filters=6, kernel_size=5, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2])))
model_cnnlstm.add(LSTM(64, return_sequences=True, activation='relu'))  # Adjust the number of units
model_cnnlstm.add(LSTM(64, return_sequences=False, activation='relu'))  # Adjust the number of units
model_cnnlstm.add(Dense(1, activation='linear'))


# Print the model summary
model_cnnlstm.summary()
```

7 Model Prediction and Result

After the model is fit, the fiited model is used to make predictions. See the screenshot below to see how the model is loaded and used to make predictions:

```
from tensorflow.keras.models import load_model

# Load the saved model
model3 = load_model(cp3_path, custom_objects={'MeanAbsoluteError': MeanAbsoluteError})

# Generate test predictions
test_predictions3 = model3.predict(testX).flatten()

# Create a DataFrame with actual and predicted values
test_results3 = pd.DataFrame(data={'Test Predictions': test_predictions3, 'Actuals': testY})

test_results3
```

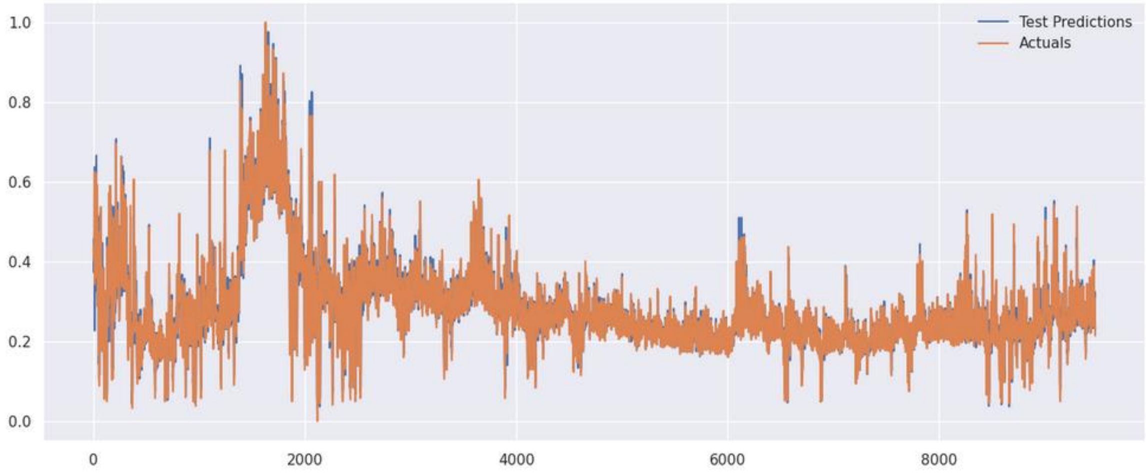
```
297/297 [==============================] - 3s 10ms/step
```

	Test Predictions	Actuals
0	0.457118	0.440211
1	0.394943	0.412228
2	0.372195	0.425483
3	0.397302	0.425483
4	0.397495	0.408398
...	...	...
9475	0.306274	0.294843
9476	0.284524	0.257696
9477	0.240468	0.248744
9478	0.243460	0.235112
9479	0.229418	0.214247

9480 rows × 2 columns

A plot of the actual vs residual is done with code snippet below:

```
[ ] # Plot the test predictions and actual values
    plt.plot(test_results3['Test Predictions'], label='Test Predictions')
    plt.plot(test_results3['Actuals'], label='Actuals')
    plt.legend()
    plt.show()
```



Afterwards, the train and test is returned to it orignal values by inverse transformation;

```
# Inverse transform the normalized values to get the actual values
test_predictions_actual3 = scaler_test.inverse_transform(test_predictions3.reshape(-1, 1)).flatten()
test_actual_actual3 = scaler_test.inverse_transform(testY.reshape(-1, 1)).flatten()

# Create a DataFrame with actual values
test_results_actual3 = pd.DataFrame(data={'Test Predictions': test_predictions_actual3, 'Actuals': test_actual_actual3})

test_results_actual3
```

And then finally, the predicted data is evaluated, see screeshot below:

```
[ ] # Evaluate the model on the test set
    evaluation_results3 = model3.evaluate(testX, testY, verbose=1)

    # Print the evaluation results
    print(f"Loss: {evaluation_results3[0]}")
    print(f"Root Mean Squared Error: {evaluation_results3[1]}")
    print(f"Mean Absolute Error: {evaluation_results3[2]}")
    print(f"Mean Absolute Percentage Error: {evaluation_results3[3]}")

    297/297 [==============================] - 4s 12ms/step - loss: 7.1257e-04 - root_mean_squared_error: 0.0267 - mean_absolute_error: 0.0166 - mean_absolute_percentage_error: 12350.4375
    Loss: 0.0007125685806386173
    Root Mean Squared Error: 0.026693981140851974
    Mean Absolute Error: 0.016591276973485947
    Mean Absolute Percentage Error: 12350.4375
```