National College of Ireland

Unleashing the power of Real-Time Data: Personalized Anxiety Interventions

Configuration Manual

MSc Research Project
Data Analytics

Manoj Kumar Periyasamy
Student ID: x22153209

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Manoj Kumar Periyasamy
Student ID:	x22153209
Programme:	MSc. in Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Arjun Chikkankod
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	1225
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:	Manoj Kumar Periyasamy
Date:	14th December 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

Manoj Kumar Periyasamy
x22153209

1 Introduction

Configuration Manual explains in a very concise manner all the details for setting up and configuring a specific system, program, and/or hardware. It usually consists of hardware needs, software installations, setup parameters, problems and solutions. Configuring a manual helps in having a system running perfectly according to expectation through having its set up well. In addition, these logs are important references that can help users who want to carry out troubleshooting or changing system settings.

2 System Configuration

2.1 System Configuration

The success of fatigue detection models relies significantly on the underlying system configuration, encompassing both hardware and software components. A robust system ensures the efficient processing and analysis of data, contributing to the accurate and timely identification of fatigue states. In this section, we outline the key elements of the system configuration employed in our evaluation:

2.2 Hardware Requirments:

The hardware infrastructure comprises the computational backbone responsible for executing the machine learning algorithms and handling the data processing load. In our study, we utilized a system with the following specifications:

These hardware specifications were chosen to provide ample computational power, ensuring efficient model training and evaluation.

2.3 Software Requirments:

The software environment is equally critical, as it dictates the tools, libraries, and frameworks available for implementing and running machine learning algorithms. The software configuration in our study included:

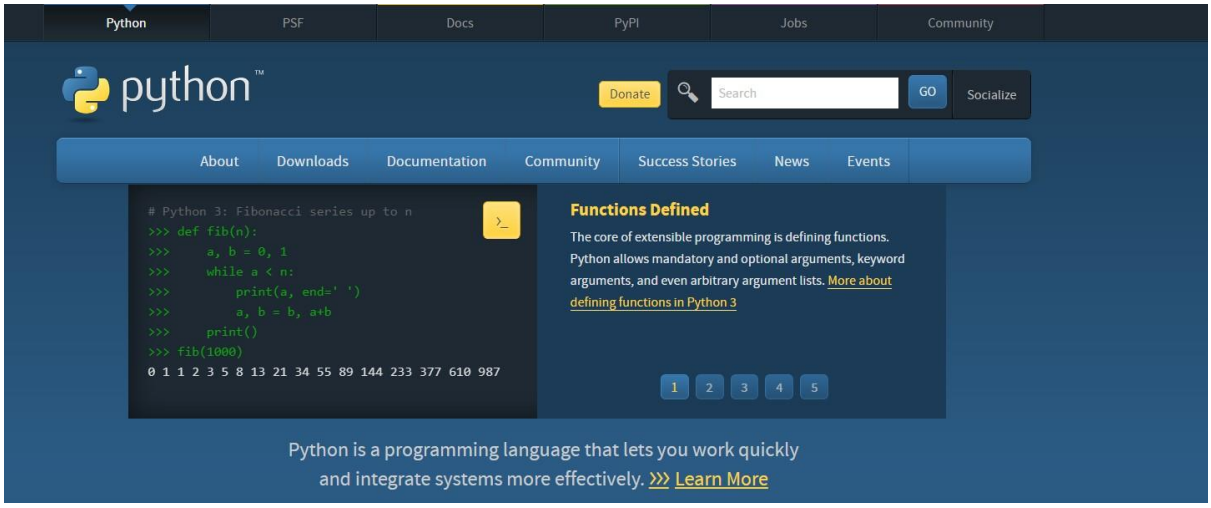
- Jupyter Notebook (Version 6.5.2) or Google Colab
- Python (Version 3.10)
- MS Excel

These software components were carefully chosen to create a cohesive and conducive environment for developing and evaluating fatigue detection models.

3 Installation and Environment Setup

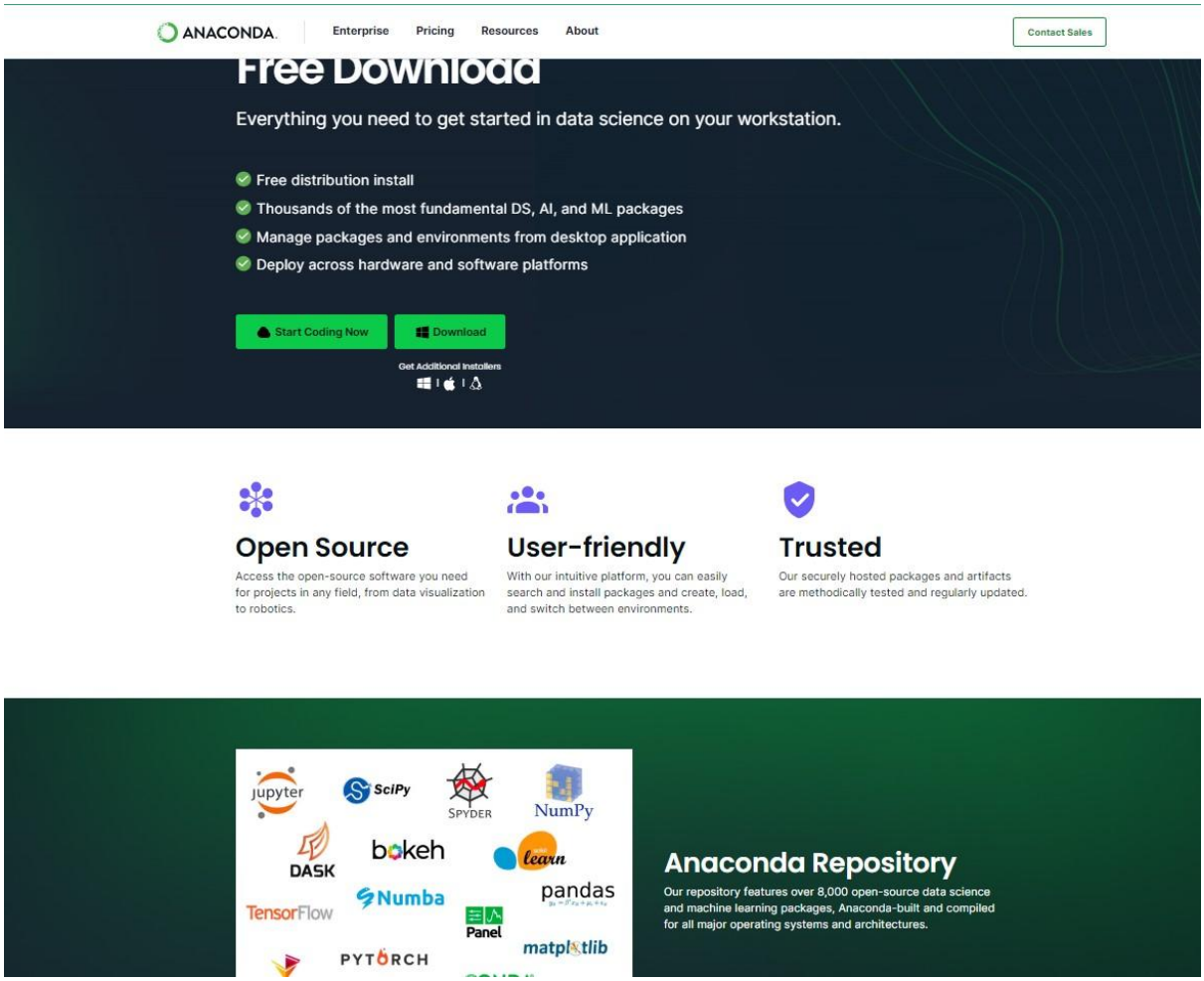
- **Python**

This project made use of a Python package. Since the majority of Deep Learning and Machine Learning Projects are supported by its numerous built-in libraries. With a variety of plots, it makes developing and analysing models easier. Installing the most recent version of Python on the machine is the first prerequisite. The package installer is capable of being downloaded through a web browser from the website reference https://www.python.org/downloads depending on the operating system. Type 'python -version' in the command prompt to confirm Python has been successfully installed from the website, as shown in figure python below.



- **Anaconda**

The anaconda package includes a number of IDE that are helpful for writing code and analyzing outputs from python packages. As seen in the below figure, this package can be obtained and installed from the website https://www.anaconda.com/products/individual. Jupyter notebook and its tasks are launched in browser tabs from the anaconda navigator. Python notebooks are first created and saved in the.ipynb format.



- **Jupyter Notebook**

Using the pip command, the python libraries are installed during the execution of code. Transformers, Scikit-Learn, nltk, Numpy, Pandas, Tensorflow, Matplotlib, googletrans, Seaborn, and heatmap are the necessary libraries for this course of action. In this browser, many different IDEs were available. The model in this project is constructed in Jupyter Notebook.

Command: pip install 'LibraryName'

4 Dataset Details:

The dataset was generated through responses obtained from a distributed survey conducted via Amazon Mechanical Turk between December 3, 2016, and December 5, 2016. Thirty eligible Fitbit users provided consent for the submission of personal tracker data, encompassing minute-level output for physical activity, heart rate, and sleep monitoring. Individual reports can be parsed using either the export session ID (column A) or timestamp (column B). The variability in output reflects the use of various Fitbit tracker types and individual tracking behaviours/preferences.

Dataset Link: https://www.kaggle.com/code/elenamekeshkina/bellabeat-case-study-in-r/input

5 Implementation

5.1 Importing Libraries

The implementation part is explained below in detail on how the project was implemented using Python. Please carry out the instructions step by step. The first step is to preprocess the provided data before we start the implementation. The libraries required for startup are displayed in the below picture.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from IPython.display import Image
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
```

5.2 Data Collection:

```
# Loading datasets
data_dir ='/content/'

relevant_minutes_datasets = [
    data_dir + 'minuteMETsNarrow_merged.csv',
    data_dir + 'minuteStepsNarrow_merged.csv',
    data_dir + 'minuteIntensitiesNarrow_merged.csv',
    data_dir + 'minuteCaloriesNarrow_merged.csv',
    data_dir + 'heartrate_seconds_merged.csv'
]

dfs_minutes = []

dict_rename_date_cols = {
    'Date': 'Time',
    'ActivityMinute':'Time',
    'date': 'Time',
    'Value':'heart_rate'
}

for ds in relevant_minutes_datasets:
    dfs_minutes.append(pd.read_csv(ds).rename(columns=dict_rename_date_cols))

for i_df in range(len(dfs_minutes)):
    dfs_minutes[i_df]['Time'] = pd.to_datetime(dfs_minutes[i_df]['Time'], format=r'%m/%d/%Y %I:%M:%S %p')
```

Added different datasets into single folder

```
# Specify the path where you want to save the CSV file
csv_output_path = '/content/output.csv'

# Save DataFrame to CSV
df_grouped.to_csv('csv_output_path', index=False)
```

Saving Merged file into our local desk

5.3 Data Preprocessing

```
# Aggregating the bins with the min, max and mean values

df_grouped = df_concat.groupby(['Id', 'time_range']).agg(['min', 'max', 'mean'])
df_grouped = (df_grouped.set_axis(df_grouped.columns.map('_'.join), axis=1).
              drop(columns=['Time_max', 'Time_mean']).
              rename(columns={'Time_min':'Time'}))

df_grouped.head()
```

Id	time_range	Time	METs_min	METs_max	METs_mean	Steps_min	Steps_max	Steps_mean	Intensity_min	Intensity_max	Intensity_mea
1503960366	(2016-04-11 23:59:59.999999999, 2016-04-12 00:01:00]	2016-04-12 00:00:00	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.
	(2016-04-12 00:01:00, 2016-04-12 00:02:00]	2016-04-12 00:02:00	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.
	(2016-04-12 00:02:00, 2016-04-12 00:03:00]	2016-04-12 00:03:00	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.
	(2016-04-12 00:03:00, 2016-04-12 00:04:00]	2016-04-12 00:04:00	10.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.
	(2016-04-12 00:04:00, 2016-04-12 00:05:00]	2016-04-12 00:05:00	12.0	12.0	12.0	0.0	0.0	0.0	0.0	0.0	0.

```
# Removing all features min/max column

df_grouped = df_grouped.drop(columns=[
'METs_min', 'METs_max',
'Steps_min', 'Steps_max',
'Intensity_min','Intensity_max',
'Calories_min', 'Calories_max',
'heart_rate_max', 'heart_rate_min'
]).rename(columns={
c:c.replace('_mean', '') for c in df_grouped.columns
})

df_grouped.head()
```

Id	time_range	Time	METs	Steps	Intensity	Calories	heart_rate
1503960366	(2016-04-11 23:59:59.999999999, 2016-04-12 00:01:00]	2016-04-12 00:00:00	10.0	0.0	0.0	0.7865	NaN
	(2016-04-12 00:01:00, 2016-04-12 00:02:00]	2016-04-12 00:02:00	10.0	0.0	0.0	0.7865	NaN
	(2016-04-12 00:02:00, 2016-04-12 00:03:00]	2016-04-12 00:03:00	10.0	0.0	0.0	0.7865	NaN
	(2016-04-12 00:03:00, 2016-04-12 00:04:00]	2016-04-12 00:04:00	10.0	0.0	0.0	0.7865	NaN
	(2016-04-12 00:04:00, 2016-04-12 00:05:00]	2016-04-12 00:05:00	12.0	0.0	0.0	0.9438	NaN

5.4 Checking Percentage of Missing Values:

Percentage of each column with NaN values

```
]: pd.DataFrame(df_grouped.isna().sum() / len(df_grouped)).T.style.format('{:.2%}')
```

]:

	Time	METs_min	METs_max	METs_mean	Steps_min	Steps_max	Steps_mean	Intensity_min	Intensity_max	Intensity_mean	Calories_min	Calories_max
0	9.06%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%	9.08%

5.5 Model building:

The model building part of the configuration manual describes how to create and customize models inside the product or system. This part is critical for users who wish to make use of the setting up process and customize it to their specific requirements.

5.5.1 Machine Learning Models:

In building our Anxiety prediction model, we preprocess the dataset, split it into training and testing sets, and employ Machine Learning model for training. Evaluation metrics like Mean Squared Error and R-squared gauge the model's accuracy for Linear Regression, Decision Tree, Random Forest and Gradient Boost.

```
# Assuming df is your preprocessed DataFrame
X = df_grouped.drop(['Calories'], axis=1)  # Features excluding 'Calories' and 'Time'
y = df_grouped['Calories']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize predictions vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Calories')
plt.ylabel('Predicted Calories')
plt.title('Actual vs Predicted Calories')
plt.show()
```

```
# Model 1: Decision Tree Regressor
model_dt = DecisionTreeRegressor(random_state=42)
model_dt.fit(X_train, y_train)
y_pred_dt = model_dt.predict(X_test)
```

```
# Model 2: Random Forest Regressor
model_rf = RandomForestRegressor(random_state=42)
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)
```

```
# Model 3: Gradient Boosting Regressor
model_gb = GradientBoostingRegressor(random_state=42)
model_gb.fit(X_train, y_train)
y_pred_gb = model_gb.predict(X_test)
```

5.5.2 Deep Learning Models:

In building our model for Deep Learning, we begin by standardizing features using the StandardScaler. Utilize the StandardScaler to standardize the features.Build a neural network model using the Sequential API from Kera.Configure layers with a ReLU activation function and a linear activation function for the output layer. Compile the neural network model using the Adam optimizer and mean squared error loss function. Early stopping is implemented to prevent overfitting, and training is executed over 100 epochs with a batch size of 32, leveraging validation data for model evaluation.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Build the neural network model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
```

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Set up early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
# Train the model
history = model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_data=(X_test_scaled, y_test), cal
```

```
# Make predictions on the test set
y_pred = model.predict(X_test_scaled).flatten()
```

```
13669/13669 [==============================] - 17s 1ms/step
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
```

Long Short-Term Memory (LSTM) architecture

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Reshape data for LSTM (samples, time steps, features)
X_train_reshaped = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_reshaped = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))
```

```
# Build the LSTM model
model_lstm = Sequential()
model_lstm.add(LSTM(64, activation='relu', input_shape=(1, X_train_scaled.shape[1])))
model_lstm.add(Dense(1, activation='linear'))
```

```
# Compile the model
model_lstm.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Set up early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
# Train the model
history_lstm = model_lstm.fit(X_train_reshaped, y_train, epochs=100, batch_size=32, validation_data=(X_test_reshaped
```

```
# Make predictions on the test set
y_pred_lstm = model_lstm.predict(X_test_reshaped).flatten()
```

```
13669/13669 [==============================] - 17s 1ms/step
```

```
# Evaluate the LSTM model
mse_lstm = mean_squared_error(y_test, y_pred_lstm)
r2_lstm = r2_score(y_test, y_pred_lstm)

print(f'LSTM Mean Squared Error: {mse_lstm}')
print(f'LSTM R-squared: {r2_lstm}')
```

5.6 Model Evaluation:

```
# Plot training and validation loss over epochs
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

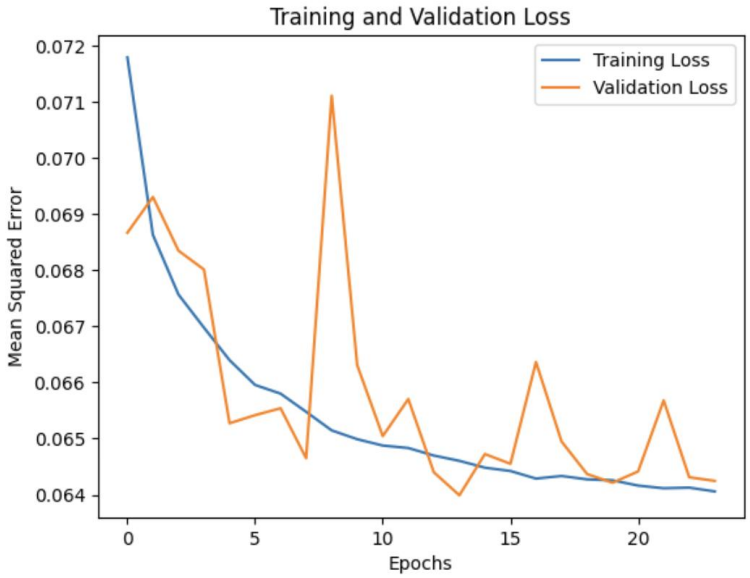


Figure represnt the Model Evaluation for Feedforward Neural Network.

```
# Plot training and validation loss over epochs for LSTM
plt.plot(history_lstm.history['loss'], label='Training Loss (LSTM)')
plt.plot(history_lstm.history['val_loss'], label='Validation Loss (LSTM)')
plt.xlabel('Epochs')
plt.ylabel('Mean Squared Error')
plt.title('Training and Validation Loss (LSTM)')
plt.legend()
plt.show()
```

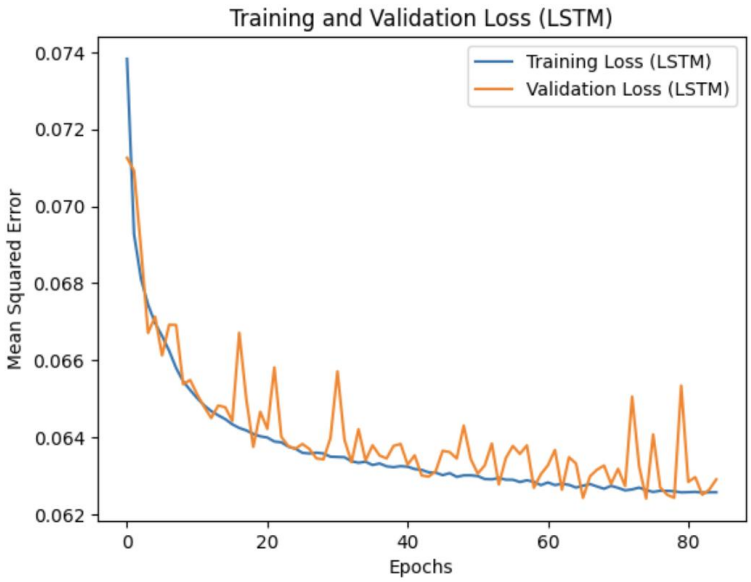


Figure represnt the Model Evaluation for Long Short Term Memory.