

Configuration Manual

MSc Research Project
Data Analytics

Anish Romario Joseph Ambrose
Student ID: x20190841

School of Computing
National College of Ireland

Supervisor: Abdul Qayum

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name:	Anish Romario Joseph Ambrose
Student ID:	x20190841
Programme:	MSc Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Anish Romario Joseph Ambrose
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	810
Page Count:	18

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:	Anish Romario Joseph Ambrose
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Anish Romario
x20190841@student.ncirl.ie

1 Introduction

Fatigue detection in various contexts, such as transportation and workplace safety, plays a crucial role in mitigating potential risks associated with impaired alertness and cognitive performance. As a response to this imperative, machine learning models have been employed to automate the identification of fatigue-related states, leveraging diverse algorithms to enhance accuracy and reliability.

In this study, we focus on evaluating the performance of different machine learning models for fatigue detection. The models considered include Decision Tree, Feed Forward Neural Network, Deep Learning, K-Nearest Neighbours (KNN), XG Boost and Random Forest. These models are assessed based on key metrics such as Mean Squared Error (MSE), R-Squared, Precision, Recall, F1-score, and Accuracy, providing a comprehensive understanding of their effectiveness in discerning fatigue patterns.

The outcomes of this evaluation not only contribute valuable insights into the strengths and limitations of each model but also aid in informing decisions regarding the adoption of specific fatigue detection methodologies. As we delve into the results, it becomes apparent how these machine learning techniques can be instrumental in advancing the field of fatigue detection, promoting safety, and optimizing performance in scenarios where vigilance is paramount.

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 Configuration:

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:

- Processor: [12th Gen Intel(R) Core(TM) i5-1235U, 1300 Mhz, 10 Core(s), 12 Logical Processor(s)]
- Installed RAM: [16GB]
- Graphics Processing Unit (GPU): [Intel® Iris® Xe Graphics]

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

2.3 Software Configuration:

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:

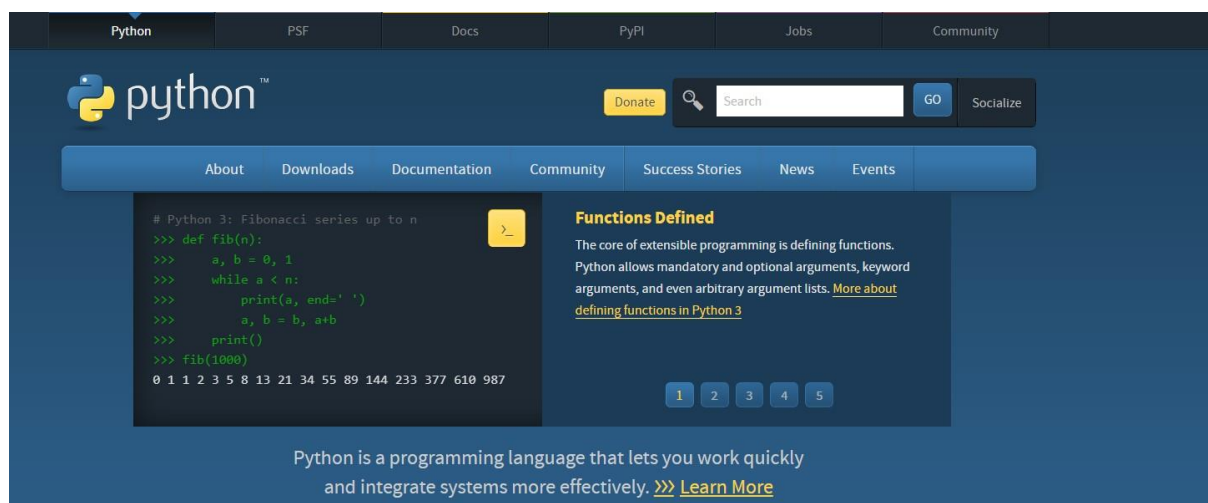
- Operating System: [windows 11]
- Programming Language: [Python]
- Machine Learning Libraries: [Scikit-Learn, TensorFlow]
- Data Processing Tools: [Specify Tools for Data Preprocessing]
- Model Evaluation and Analysis: [Scikit-Learn Metrics]

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

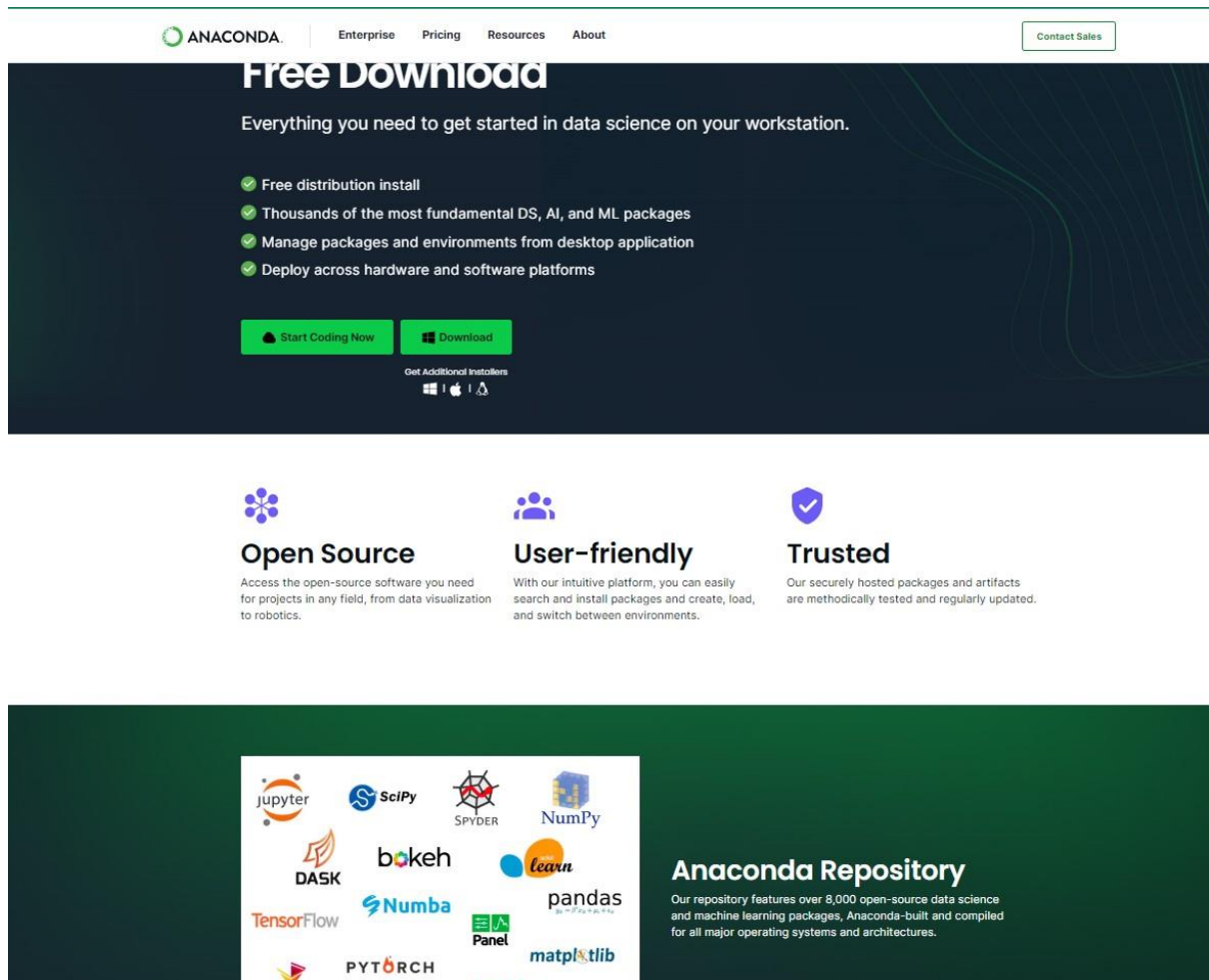
• Python

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 Plotly 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 Data Collection

To address the objectives of the study, a comprehensive and diverse dataset will be collected from various workplace environments. The dataset will include information on physiological measures, work hours, environmental conditions, and employee self-reported fatigue levels. Data sources may include wearables, sensors, employee surveys, and workplace records. This approach ensures a holistic representation of workplace conditions and allows the model to learn patterns from multiple dimensions, contributing to a robust fatigue detection system.

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 matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
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
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Import & load the data in a data frame

```
df = pd.read_csv("/content/all-samples.csv")
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51741 entries, 0 to 51740
Data columns (total 61 columns):
#   Column                Non-Null Count  Dtype
---  -
0   MEAN                   51741 non-null  float64
1   MAX                    51741 non-null  float64
2   MIN                    51741 non-null  float64
3   RANGE                  51741 non-null  float64
4   KURT                   51741 non-null  float64
5   SKEW                   51741 non-null  float64
6   MEAN_1ST_GRAD          51741 non-null  float64
7   STD_1ST_GRAD           51741 non-null  float64
8   MEAN_2ND_GRAD          51741 non-null  float64
9   STD_2ND_GRAD           51741 non-null  float64
10  ALSC                   51741 non-null  float64
11  INSC                   51741 non-null  float64
12  APSC                   51741 non-null  float64
```

5.2 Data Preprocessing and Data Selection

5.2.1 Data Preprocessing

The preprocessing on the given data containing the excel file is performed as shown in the figure 5 below, explains the statistical Analysis, correlation, and spreading the regression points.

```
# Get summary statistics of the numeric columns
print(df.describe())
```

	MEAN	MAX	MIN	RANGE	KURT
count	51741.000000	51741.000000	51741.000000	51741.000000	51741.000000
mean	0.010503	0.033205	0.003650	0.029555	3.828619
std	0.021491	0.063043	0.007027	0.056090	4.540064
min	0.000075	0.000129	0.000031	0.000098	-4.764876
25%	0.002261	0.006965	0.000761	0.006076	0.685914
50%	0.004200	0.013511	0.001436	0.012103	2.517461
75%	0.009008	0.031064	0.003229	0.026463	5.688255
max	0.251007	0.569164	0.065985	0.512189	27.527457

	SKEW	MEAN_1ST_GRAD	STD_1ST_GRAD	MEAN_2ND_GRAD	STD_2ND_GRAD
count	51741.000000	51741.000000	51741.000000	51741.000000	51741.000000
mean	1.735443	-0.000757	0.006278	0.000043	0.005170
std	0.849005	0.008941	0.014559	0.007591	0.012289
min	-1.853285	-0.134964	0.000000	-0.153228	0.000000
25%	1.144333	-0.000114	0.001101	-0.000116	0.000944
50%	1.636892	-0.000002	0.002016	0.000006	0.001765
75%	2.241458	0.000096	0.004572	0.000157	0.003907
max	4.987227	0.157541	0.187414	0.111371	0.161786

	MIN_ONSET_LOG	MIN_ONSET_SQRT	MAX_ONSET_LOG	MAX_ONSET_SQRT
count	51741.000000	51741.000000	51741.000000	51741.000000
mean	1.061553	1.455315	3.668287	6.459491
std	0.540830	1.041590	0.588332	2.055178
min	0.530743	0.836777	0.907857	1.216143
25%	0.785657	1.092633	3.192997	4.833356
50%	0.913357	1.221751	3.641075	6.093670
75%	1.105587	1.421617	4.066938	7.574822
max	5.673982	17.035020	5.673982	17.035020

	STD_ONSET_YEO_JON	MEAN_ONSET_LOG	MEAN_ONSET_SQRT	NasaTLX
count	51741.000000	51741.000000	51741.000000	51741.000000
mean	1.061553	1.455315	3.668287	6.459491
std	0.540830	1.041590	0.588332	2.055178
min	0.530743	0.836777	0.907857	1.216143
25%	0.785657	1.092633	3.192997	4.833356
50%	0.913357	1.221751	3.641075	6.093670
75%	1.105587	1.421617	4.066938	7.574822
max	5.673982	17.035020	5.673982	17.035020


```
: # Check for missing values
print(df.isnull().sum())
```

MEAN	0
MAX	0
MIN	0
RANGE	0
KURT	0
MEAN_ONSET_LOG	0
MEAN_ONSET_SQRT	0
NasaTLX	0
subject_id	0
condition	0
Length: 61, dtype: int64	


```
: # Data Visualization
# Example: Plot histograms for numeric columns
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
for col in numeric_columns:
    plt.figure()
    sns.histplot(df[col])
    plt.title(f'Histogram of {col}')
    plt.show()
```

```

: # Example: Create a scatter plot
plt.scatter(df['MEAN'], df['MAX'])
plt.xlabel('MEAN')
plt.ylabel('MAX')
plt.title('Scatter Plot between MEAN and MAX')
plt.show()

```

```

]: ## Create an interaction feature between 'MEAN' and 'MAX'
df['MEAN_MAX_INTERACTION'] = df['MEAN'] * df['MAX']

```

6 Model Building and Model Evaluation

In our thesis I have build Regression and classification models. Namely, Decision Tree, Random Forest, linear regression, KNN, Grid Search and XG Boost. And below this I have attached all the screenshot.

Decision Tree Regressor

```

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Features for prediction
X = df[['MEAN', 'MAX', 'MIN', 'RANGE', 'KURT', 'SKEW']]

# Target variable
y = df['NasaTLX']

# 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)

# Train a Decision Tree Regressor model
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

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

# Evaluate the model
# 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}')

```

```

Mean Squared Error: 33.83170429558331
R-squared: 0.8437776694428303

```

Hyperparameter Tuning with Grid Search CV


```

from sklearn.model_selection import GridSearchCV

# Define hyperparameters to search
param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(DecisionTreeRegressor(random_state=42), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Train a Decision Tree Regressor with the best hyperparameters
best_model = DecisionTreeRegressor(random_state=42, **best_params)
best_model.fit(X_train, y_train)

# Make predictions and evaluate the best model
y_pred_best = best_model.predict(X_test)
mse_best = mean_squared_error(y_test, y_pred_best)
r2_best = r2_score(y_test, y_pred_best)
print(f'Optimized Mean Squared Error: {mse_best}')
print(f'Optimized R-squared: {r2_best}')

```

Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
Optimized Mean Squared Error: 33.13462653496709
Optimized R-squared: 0.846996517402472

Random Forest and Gradient Boost Regressor

```

[ ] from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

# Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f'Random Forest Mean Squared Error: {mse_rf}')
print(f'Random Forest R-squared: {r2_rf}')

```

Random Forest Mean Squared Error: 27.847374045877274
Random Forest R-squared: 0.8714110990290215

```

: from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create the GridSearchCV object
grid_search = GridSearchCV(
    GradientBoostingRegressor(random_state=42),
    param_grid,
    cv=5, # You can adjust the number of folds for cross-validation
    scoring='neg_mean_squared_error', # Use negative MSE for GridSearchCV
    n_jobs=-1 # Use all available CPU cores
)

# Fit the model to the data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print(f'Best Hyperparameters: {best_params}')

# Get the best model
best_gb_model = grid_search.best_estimator_


```

```

# Predict on the test set
y_pred_gb = best_gb_model.predict(X_test)

# Evaluate the model
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)
print(f'Gradient Boosting Mean Squared Error: {mse_gb}')
print(f'Gradient Boosting R-squared: {r2_gb}')

```

 Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 150}
 Gradient Boosting Mean Squared Error: 35.11189732514119
 Gradient Boosting R-squared: 0.8378662102714788

Linear Regression

```

▶ from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split your dataset 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)

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

# Create a Linear Regression model
lr_model = LinearRegression()

# Fit the model to the standardized training data
lr_model.fit(X_train_scaled, y_train)

# Make predictions on the standardized test data
y_pred_lr = lr_model.predict(X_test_scaled)

```

```

# Calculate Mean Squared Error (MSE) and R-squared (R2) scores
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

# Print the results
print(f'Linear Regression Mean Squared Error: {mse_lr}')
print(f'Linear Regression R-squared: {r2_lr}')

```

```

| Linear Regression Mean Squared Error: 215.7005580621789
  Linear Regression R-squared: 0.003974390750546952

```

Now we will be creating a dataframe from the results dictionary by sorting the dataframe by Mean Squared Error (ascending order, lower is better)

```

import pandas as pd

# Create a dictionary to store the results
results = {
    "Model": ["Linear Regression", "Random Forest", "Gradient Boosting", "Optimized Decision Tree", "Decision Tree"],
    "Mean Squared Error (MSE)": [mse_lr, mse_rf, mse_gb, mse_best, mse],
}

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame(results)

# Sort the DataFrame by MSE (ascending order, lower is better)
results_df = results_df.sort_values(by="Mean Squared Error (MSE)")

# Reset the index for a clean ranking
results_df.reset_index(drop=True, inplace=True)

# Display the comparison table
print(results_df)

```

	Model	Mean Squared Error (MSE)
0	Random Forest	27.847374
1	Optimized Decision Tree	33.134627
2	Decision Tree	33.831704
3	Gradient Boosting	35.111897
4	Linear Regression	215.700558

Now again we will be creating a dataframe from the results dictionary by sorting the dataframe by R2 (descending order, higher is better)

```

import pandas as pd

# Create a dictionary to store the results
results = {
    "Model": ["Linear Regression", "Random Forest", "Gradient Boosting", "Optimized Decision Tree", "Decision Tree"],
    "R-squared (R2)": [r2_lr, r2_rf, r2_gb, r2_best, r2],
}

# Create a DataFrame from the results dictionary
results_df = pd.DataFrame(results)

# Sort the DataFrame by R2 (descending order, higher is better)
results_df = results_df.sort_values(by="R-squared (R2)", ascending=False)

# Reset the index for a clean ranking
results_df.reset_index(drop=True, inplace=True)

# Display the comparison table
print(results_df)

```

	Model	R-squared (R2)
0	Random Forest	0.871411
1	Optimized Decision Tree	0.846997
2	Decision Tree	0.843778
3	Gradient Boosting	0.837866
4	Linear Regression	0.003974

Now the models for Classification will be applied where the target variable will be **‘condition’**

```

# Features for prediction
X = df[['MEAN', 'MAX', 'MIN', 'RANGE', 'KURT', 'SKEW']]

df = df.dropna(subset=["condition"])

from sklearn.model_selection import train_test_split
# Target variable
y = df['condition']

# 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)

```

Random Forest Classifier is now performed

```

# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define hyperparameters and their possible values for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Create a classifier with the best hyperparameters
best_rf_classifier = RandomForestClassifier(**best_params)

# Train the model with the training data
best_rf_classifier.fit(X_train, y_train)

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

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))

```

```

Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Accuracy: 0.965148489338401

```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	5194
1	0.94	0.95	0.95	5119
2	0.96	0.95	0.95	5210
accuracy			0.97	15523
macro avg	0.97	0.97	0.97	15523
weighted avg	0.97	0.97	0.97	15523

Now XG Boost model and its respective results will be displayed

✶ xgboost

```
[ ] from sklearn.model_selection import train_test_split, GridSearchCV
    from xgboost import XGBClassifier
    from sklearn.metrics import accuracy_score, classification_report

[ ] # Create the XGBoost classifier
    xgb_classifier = XGBClassifier()

    # Define hyperparameters and their possible values for tuning
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 4, 5],
        'learning_rate': [0.01, 0.1, 0.2],
    }

    # Initialize GridSearchCV for hyperparameter tuning
    grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the model to find the best hyperparameters
grid_search.fit(X_train, y_train)


# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Create a classifier with the best hyperparameters
best_xgb_classifier = XGBClassifier(**best_params)

# Train the model with the training data
best_xgb_classifier.fit(X_train, y_train)

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

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

 Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 300}
Accuracy: 0.9645687045029956

	precision	recall	f1-score	support
0	0.99	0.99	0.99	5194
1	0.95	0.95	0.95	5119
2	0.96	0.95	0.95	5210
accuracy			0.96	15523
macro avg	0.96	0.96	0.96	15523
weighted avg	0.96	0.96	0.96	15523

Now Decision Tree Classifier and its associated results will be displayed

▼ Decision Tree

```
[ ] from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.model_selection import train_test_split

    # Create the Decision Tree classifier
    dt_classifier = DecisionTreeClassifier(random_state=42)

    # Train the model with the training data
    dt_classifier.fit(X_train, y_train)

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

    # Evaluate the model's performance
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
```

```
# Display classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9571603427172583
      precision    recall  f1-score   support

0         0.99      0.99      0.99       5194
1         0.92      0.95      0.94       5119
2         0.95      0.93      0.94       5210

 accuracy                   0.96       15523
macro avg                   0.96       15523
weighted avg                0.96       15523
```

Now KNN and its associated results will be displayed

▼ KNN model

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score, classification_report
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler

    # Create a scaler
    scaler = StandardScaler()

    # Fit and transform on the entire dataset
    X_scaled = scaler.fit_transform(X)

    # Split the scaled data into training and testing sets
    X_train_scaled, X_test_scaled, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

```
# Experiment with different values of k
for k in range(3, 11):
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train_scaled, y_train)
    y_pred = knn_classifier.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy with k={k}: {accuracy}')
```

```
Accuracy with k=3: 0.9080718933195903
Accuracy with k=4: 0.9007923726083875
Accuracy with k=5: 0.8920956000773046
Accuracy with k=6: 0.8854602847387747
Accuracy with k=7: 0.8795335953101849
Accuracy with k=8: 0.8730271210461895
Accuracy with k=9: 0.8669071700057979
Accuracy with k=10: 0.8586613412355859
```

Classification Report for KNN

```
from sklearn.metrics import classification_report

# Assuming you have already trained and predicted with your KNN classifier
knn_classifier.fit(X_train_scaled, y_train)
y_pred = knn_classifier.predict(X_test_scaled)

# Display classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

    0               0.87         0.85         0.86         5194
    1               0.82         0.85         0.84         5119
    2               0.88         0.87         0.87         5210

 accuracy                   0.86         0.86         0.86         15523
 macro avg                0.86         0.86         0.86         15523
 weighted avg             0.86         0.86         0.86         15523
```

Now Deep Learning and its associated results will be displayed

Deep Learning

```
[ ] import pandas as pd
import numpy as np
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
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Assuming df is your preprocessed DataFrame
X = df.drop(['condition'], axis=1)
y = df['condition'] # 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=0.2, random_state=42)
```



```

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

# Reshape data for LSTM (samples, time steps, features)
X_train_resaped = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_resaped = 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_resaped, y_train, epochs=100, batch_size=32, validation_data=(X_test_re

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

# 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}')

# 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()

```

There are 100 epochs for Deep Learning and I have added only one with its MSE and R2 as the complete epochs will be present in the code file

```

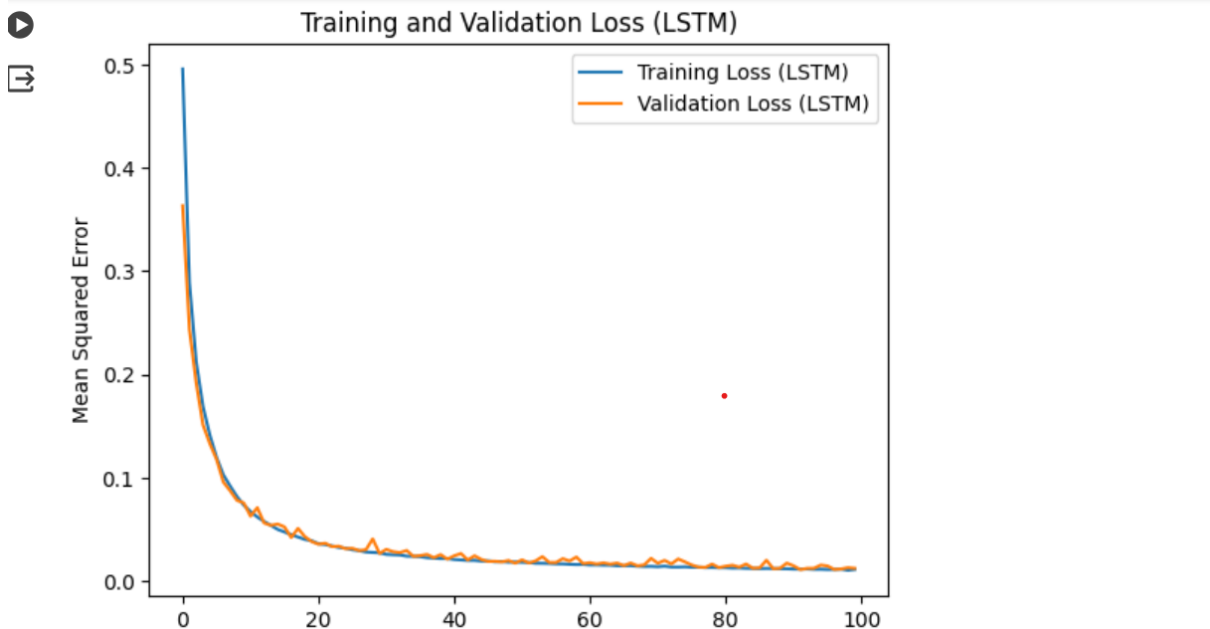
1294/1294 [=====] - 4s 3ms/step - loss: 0.0148 - val_loss: 0.0174
Epoch 66/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0149 - val_loss: 0.0149
Epoch 67/100
1294/1294 [=====] - 6s 5ms/step - loss: 0.0149 - val_loss: 0.0175
Epoch 68/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0144 - val_loss: 0.0148
Epoch 69/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0141 - val_loss: 0.0156
Epoch 70/100
1294/1294 [=====] - 6s 5ms/step - loss: 0.0142 - val_loss: 0.0220
Epoch 71/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0139 - val_loss: 0.0172
Epoch 72/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0144 - val_loss: 0.0200
Epoch 73/100
1294/1294 [=====] - 6s 5ms/step - loss: 0.0136 - val_loss: 0.0166
Epoch 74/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0135 - val_loss: 0.0213
Epoch 75/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0137 - val_loss: 0.0185
Epoch 76/100
1294/1294 [=====] - 6s 4ms/step - loss: 0.0133 - val_loss: 0.0155
Epoch 77/100

```

```

Epoch 99/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0104 - val_loss: 0.0129
Epoch 100/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0112 - val_loss: 0.0123
324/324 [=====] - 1s 2ms/step
LSTM Mean Squared Error: 0.012346963010165694
LSTM R-squared: 0.9815161485805857

```



Now Feed Forward Neural Network and its results will be displayed

✓ feedforward neural network for regression

```

[ ] import pandas as pd
import numpy as np
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

# 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))

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

# 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}')

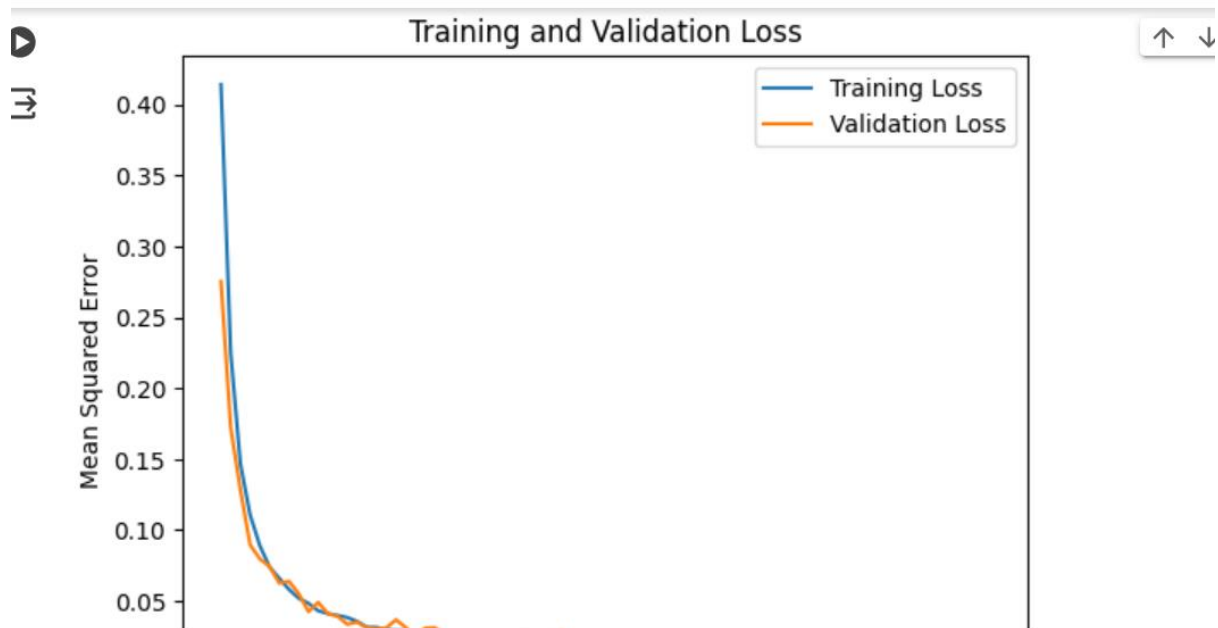
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# 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()
```

Epoch 22/100
1294/1294 [=====] - 3s 2ms/step - loss: 0.0254 - val_loss: 0.0308
Epoch 23/100
1294/1294 [=====] - 3s 3ms/step - loss: 0.0243 - val_loss: 0.0311
Epoch 24/100
1294/1294 [=====] - 4s 3ms/step - loss: 0.0250 - val_loss: 0.0248
Epoch 25/100
1294/1294 [=====] - 3s 3ms/step - loss: 0.0241 - val_loss: 0.0258
Epoch 26/100
.....

There are a total of 80 epochs, I have added two screenshots as the complete epochs result will be present in the code file.

```
1294/1294 [=====] - 3s 2ms/step - loss: 0.0110 - val_loss: 0.0109
Epoch 77/100
1294/1294 [=====] - 3s 2ms/step - loss: 0.0110 - val_loss: 0.0109
Epoch 78/100
1294/1294 [=====] - 5s 4ms/step - loss: 0.0109 - val_loss: 0.0136
Epoch 79/100
1294/1294 [=====] - 3s 3ms/step - loss: 0.0116 - val_loss: 0.0126
Epoch 80/100
1294/1294 [=====] - 3s 3ms/step - loss: 0.0105 - val_loss: 0.0111
324/324 [=====] - 1s 2ms/step
Mean Squared Error: 0.01090480469115277
R-squared: 0.9836751118875916
```



Now the results of Deep Learning and Feed forward neural network will be compared

✓ Comparision Result

```
[ ] import pandas as pd

# Create a DataFrame to store the results
results = pd.DataFrame(index=['Mean Squared Error (MSE)', 'R-squared (R2)'])

# Results for the Feedforward Neural Network
results['Feedforward Neural Network'] = [mse, r2]

# Results for the LSTM model
results['LSTM'] = [mse_lstm, r2_lstm]

# Display the results table
print(results)
```

	Feedforward Neural Network	LSTM
Mean Squared Error (MSE)	0.010905	0.012347
R-squared (R2)	0.983675	0.981516