National College of Ireland

Configuration Manual

MSc Research Project
Msc Data Analytics

Sabarinathan Chandramohan
Student ID: x22213333

School of Computing
National College of Ireland

Supervisor:**Teerath Kumar Menghwar**

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sabarinathan Chandramohan
Student ID:	x22213333
Programme:	Msc Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	**Teerath Kumar Menghwar**
Submission Due Date:	16/09/2024
Project Title:	Configuration Manual
Word Count:	1327
Page Count:	14

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:	Sabarinathan Chandramohan
Date:	16th September 2024

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

Sabarinathan Chandramohan
x22213333

1 Introduction

This configuration document provides a detail guide manual for setting up needed environment, tools and necessary machine learning libraries needed to re-run the implemented codes of Optimization of wave power energy generation system and wave height forecasting, this manual provides a detail step by step guide to install the prerequisites hardware and software requirements and helps the users to configure it into their system and execute the code to obtain the similar results, this research uses the main and primary tools like Docker, PostgreSQL, ETL-Dagster, Python, and Jupyter Notebook.

2 Environment

Environment setup is needed and main process in executing the code, in this section a detail steps guide is described for the Docker, ETL-Dagster, PostgreSQL, necessary machine learning libraries and Juypter notebook installation and setup, follow all the steps correctly and configure the tools as it's a crucial for executing the code, by this it can sure that the code runs without any lags and hitch and able to reproduce the research findings.

2.1 Hardware Required

- **Operating System**-Windows 10 or Ubuntu Linux
- **Processor**-Multi-core processor with Intel i7 or higher and AMD equivalent
- **Graphics Card**-GPU with CUDA support, NVIDIA GeForce RTX series for accelerated model training speed
- **RAM**-Minimum of 16GB but 32GB recommended for large datasets
- **Storage**-SSD with at least 1TB capacity for fast data access and efficient storage

3 Tools Required and setup

The detail step by step guide for installing the necessary tools for this research implementation are discussing below.

3.1 Docker setup

Docker is used as the container for maintain the Environments, this research is also uses Docker instance to held the different environments, the Docker should be installed correctly and configured with PostgreSQL, for installation of docker, download and install from the official website, and follow the instruction given in the url and as per your operating system needs install the Docker.[1]

3.2 Creating A PostgreSQL Docker Instance

```
version: '3'
services:
  database:
    image: "postgres"
    ports:
      - "5432:5432"
    env_file:
      - postgresql.env
    volumes:
      - dbdata:/var/lib/postgresql/data/
volumes:
  dbdata:
```

Figure 1: docker-compose yml file

```
POSTGRES_USER=dap
POSTGRES_PASSWORD=dap
POSTGRES_DB=postgres
```

Figure 2: postgresql.env file

After the Docker is installed and configured before creating a instance with PostgreSQL, you have to install the **sqlalchemy, psycopg2 and seaborn** modules as this is the prerequisite, as this research uses the PostgreSQL as its Data warehousing and this can be installed with the simple **pip install** command in the command prompt, open

[1]Docker installation: `https://docs.docker.com/desktop/install/windows-install/`

the command prompt either windows 'cmd' or 'PowerShell' and navigate to the directory where the python scripts are installed, then install these module with **pip install sqlalchemy**,**pip install psycopg2** and **pip install seaborn**, this will install the latest version of these models.

For creating a PostgreSQL Docker Instance, need to create a new folder and switch new folder directory and create a two new files named **docker-compse.yml** and **postgresql.env** and add the contents into file as given in the image 1 and 2 respectively, then open the command prompt and navigate into the newly created directory and run the command given in figure 3, this will create a Docker instance, now open the Docket desktop and check for the instance creation.

```
docker-compose up -d
```

Figure 3: Docker instance command

3.3 PostgreSQL setup

Figure 4: Server registration1

To setup the PostgreSQL need to download and install latest version of PGAdmin into the system [2], once after the installation open the PGAdmin, now need to create a new server for this right click on the **Servers** and click **Create** and **Server group** from the pop-up menu, give the name of newly created server as **Servers**. now right click the newly created server and click **Register** and **Server** and enter the details as shown in

[2]PGAdmin Download: `https://www.pgadmin.org/download/`

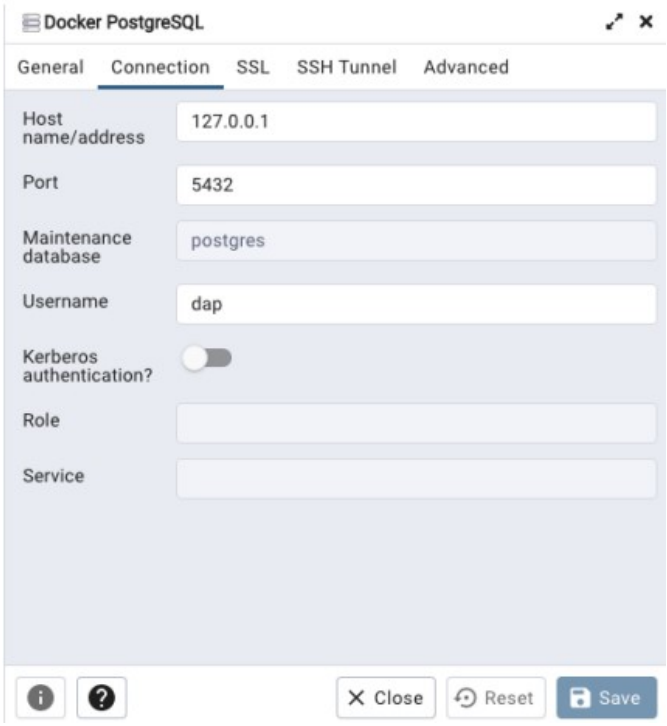


Figure 5: Server registration2

the image 4 and image 5, after this configurations of the server in the PGAdmin, create a schema, database and table in name **test**.

3.4 ETL-Dagster Environment

Dagster has been used in this research for the pipeline automating ETL process, here the data are Extracted, Transformed and Loaded into PostgreSQL as it has been used as the Data warehousing, for the detail and correct Dagster configuration follow these steps,

1. Install **Dagster** and **Dagit** using the pip install, as the **Dagit** is the web-based interface for the Dagster ETL, for this follow this command in the **Anaconda command prompt – pip install dagster dagit**.

2. For setting up the ETL pipeline, create a new Dagster project for Extract, Transform and Load into PostgreSQL, by using this command **dagit -f your_pipeline.py** can be able to run the Dagster pipeline.

3. After that access the **Dagit** through the browser to monitor and the manager the ETL process.

3.5 Python and Jupyter Notebook Setup

This research was coded in the Python as it's a main programming language and widely used for the machine learning models as with the jupyter Notebook as the environment development.
Ensure the Python 3.12 version is installed and configured, if not it can be downloaded

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import psycopg2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from kerastuner.tuners import RandomSearch
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from kerastuner.tuners import RandomSearch
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

Figure 6: Necessary Libraries of Phase 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import psycopg2
from psycopg2 import sql
from statsmodels.tsa.statespace.sarimax import SARIMAX
from itertools import product
import warnings
warnings.filterwarnings('ignore')
```

Figure 7: Necessary Libraries of Phase 2

directly from the official website [3] and following the installation guide it can be easily installed. For the accessing the juypter notebook use the **Anaconda Navigator** from the official website [4] and do the installation, after the installation open the **Anaconda Navigator** and there in application list install the **juypter notebook**, after installing open the juypter notebook it will open up in the default browser from there we can open this research implementation code files, these are necessary libraries used in the both phase of the research, since this proposed research was executed in two phases 1st phase is Wave Energy Power Generation Optimization using MLP-Multilayer Perceptron its used and needed libraries are given in the image 6 and 2nd phase is Wave Height Forecasting and Predicting the Future Wave Height using ARIMA/SARIMA its libraries are given in the image 7, install those using pip install commands in command prompt or directly in the juypter notebook itself, after successful installation can able to proceed with the implementation and run the code without any errors.

[3]python Download: `https://www.python.org/downloads`
[4]Anaconda Download: `https://www.anaconda.com/download`

4 Implementations

After this initial setup is completed, check all the setup is done correctly, if not follow all the steps again by re-installing again, once after this for the implementation open the codes in two separate juypter note book, since this research is splits into two phases, these are the two code notebooks **Wave_power_energy_output_optimization.ipynb** and **Waveheight_Forecast_SligoArea.ipynb**, after that put these two code file in a separate folder **my_etl_pipeline.py** and **repository.py**, now open the docker desktop and click run of the PostgreSQL instance shown in image, after that open PGAdmin and click refresh and connect with the server.

4.1 Collecting the Data

The data are collected from the Ireland government websites like [5], [6] and these data are stored in the one drive for easy access into it and can able to download the data in csv file from [7].

4.2 ETL- Data Preprocessing

Now open the anaconda command prompt and give this command **conda activate dagster_env** after that navigate to the folder where these are created **my_etl_pipeline.py and repository.py**, then give this command **dagit -f repository.py** for initiate ETL-Dagster in pipeline and open this url [8] for access Dagit, once after opening the Dagit you can see the ETL architecture for this research show as in the image 8, after which

Figure 8: ETL Architecture

[5]Dataset source: `https://www.marine.ie`

[6]Dataset source: `https://data.gov.ie`

[7]Dataset: `https://studentncirl-my.sharepoint.com/:u:/g/personal/x22213333_student_ncirl_ie/EdAWcHB1fGlLm3SZOdyDRZcBjhD0RIAX4Ne1T2ePOGPItA?e=0sKaGh`

[8]Dagit: `http://localhost:3000`

navigate into Launch pad of the Dagit and click Launch run as show in the image 9, after

Figure 9: Dagster-Launch Run

which you can able to see the pipeline ETL process and upon successful completion the data will be loaded into PostgreSQL, in the PGAdmin now open and check the test table where the columns are added as shown in the image 10.

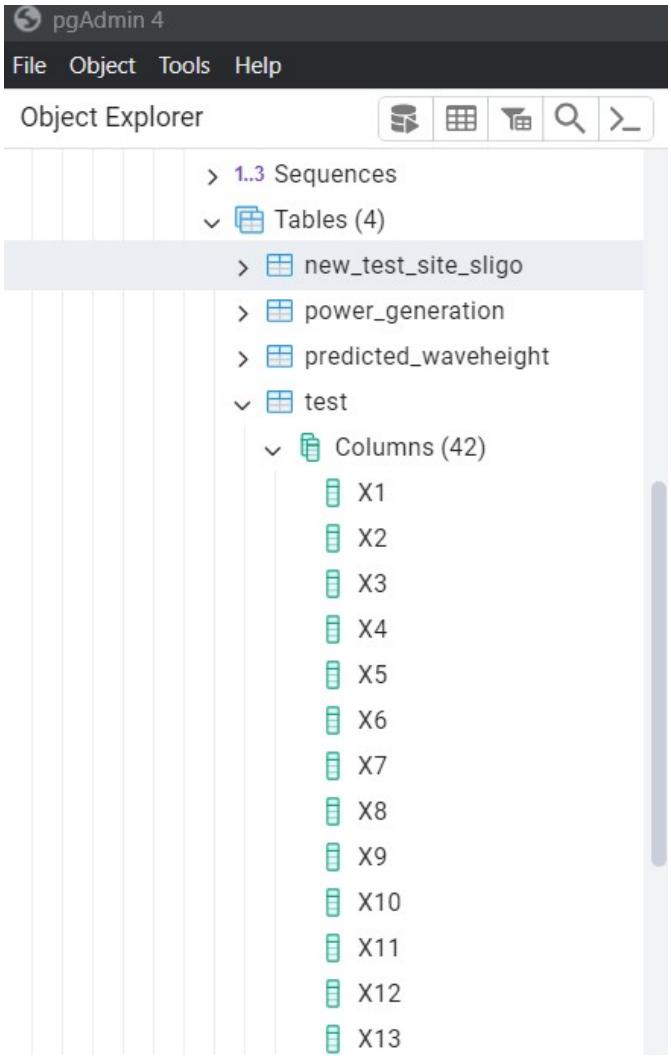


Figure 10: Data loaded into PostgreSQL

In the Data preprocessing the data are Transformed by eliminating the duplicate and null values, data and time transformation with also column transformation, after this the data are loaded into PostgreSQL as the Data Warehouse.

4.3 EDA & Data Preparation

Now once the data are downloaded and both juypter notebook files have been open separately for the code implementation, as this research has splits into two phases.

4.3.1 Phase 1 Wave power generation optimization

In the **Wave_power_energy_output_optimization.ipynb** file run the EDA- Exploratory Data Analysis cell as show in the image 11, where the data are cleaned and analysis for modelling.

```
connection_string = "postgresql://dap:dap@127.0.0.1:5432/Wave energy output"
conn = psycopg2.connect(connection_string)
query = """
SELECT "X1", "X2", "X3", "X4", "X5", "X6", "X7", "X8", "X9", "X10", "X11", "X12", "X13", "X14", "X15", "X16",
       "Y1", "Y2", "Y3", "Y4", "Y5", "Y6", "Y7", "Y8", "Y9", "Y10", "Y11", "Y12", "Y13", "Y14", "Y15", "Y16",
       "Power_Output", "wave_height"
FROM test
"""
df = pd.read_sql_query(query, con=conn)
conn.close()

df
```

Figure 11: EDA-Phase1

Then run the visualization cell as given in the image 12, this is as part of the EDA of phase 1.

```
sns.set(style="whitegrid")

# Plot the distribution of Power Output with custom color
plt.figure(figsize=(15, 8))
sns.histplot(df_clean['Power_Output'], kde=True, color='dodgerblue')
plt.title('Distribution of Power Output', fontsize=16)
plt.xlabel('Power Output', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()

# Plot the distribution of Wave Height with custom color
plt.figure(figsize=(15, 8))
sns.histplot(df_clean['wave_height'], kde=True, color='darkorange')
plt.title('Distribution of Wave Height', fontsize=16)
plt.xlabel('Wave Height', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()

# Scatter plot of Power Output vs Wave Height with custom color
plt.figure(figsize=(15, 8))
sns.scatterplot(x=df_clean['wave_height'], y=df_clean['Power_Output'], color='forestgreen')
plt.title('Power Output vs Wave Height', fontsize=16)
plt.xlabel('Wave Height', fontsize=14)
plt.ylabel('Power Output', fontsize=14)
plt.show()
```

Figure 12: Phase 1 Visulization

Once after the EDA is done now run the cell Data preparation as shown in the image 13, where the feature is selected with the correlated heat map analysis. By these steps now

```
# Set a style for the plot
sns.set(style="white")

# Create a heatmap with customizations
plt.figure(figsize=(18, 14))
sns.heatmap(df_clean.corr(), annot=True, fmt='.2f', cmap='viridis', cbar_kws={'shrink': .8}, linewidths=0.5, linecolor='gray', annot_kws={"size": 12})

# Add title for better context
plt.title('Correlation Heatmap', fontsize=18)
plt.show()
```

Figure 13: Phase 1 Data preparation

the data are cleaned, and feature are selected for the model building.

4.3.2 Phase 2 Forecast the wave height and period

Now in the **Waveheight_Forecast_SligoArea.ipynb** file navigate to EDA- Exploratory Data Analysis and Data preparation cell and run the code as given the image 14, this will clear the data with analysing the dataset for the modelling by removing the outliers the cleaned data is prepared.

```
connection_string = "postgresql://dap:dap@127.0.0.1:5432/Wave energy output"
conn = psycopg2.connect(connection_string)
query = """
SELECT "wave_height", "PeakDirection", "SignificantWaveHeight", "Hmax", "SeaTemperature", "MeanCurSpeed", "MeanCurDirTo", "WavePeriod", "time"
FROM test;
"""
buoy_data = pd.read_sql_query(query, con=conn)
conn.close()

# Convert 'time' column to datetime
buoy_data['time'] = pd.to_datetime(buoy_data['time'])

# Set 'time' as the index
buoy_data.set_index('time', inplace=True)

# Handle missing values by interpolation
buoy_data['SignificantWaveHeight'].interpolate(method='linear', inplace=True)

# Resample the data to daily frequency
daily_data = buoy_data.resample('D').mean()

# Remove outliers
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

for column in ['wave_height', 'SignificantWaveHeight', 'Hmax', 'SeaTemperature', 'MeanCurSpeed', 'MeanCurDirTo', 'WavePeriod']:
    daily_data = remove_outliers(daily_data, column)

daily_data.head(20)
```

Figure 14: Phase 2 EDA and Data prepration

After that execute the visualization cell as given in the image 15, this will plot the timeline series of the data based upon the time.

```
plt.figure(figsize=(12, 6))
plt.plot(daily_data['SignificantWaveHeight'], label='SignificantWaveHeight')
plt.title('Significant Wave Height Over Time')
plt.xlabel('Time')
plt.ylabel('Significant Wave Height')
plt.legend()
plt.show()
```

Figure 15: Phase 2 visulization

To the next cell which is Feature Engineering by running the code given in the image 16 will extract and select the feature for the model building.

```
daily_data['SWH_MA7'] = daily_data['SignificantWaveHeight'].rolling(window=7).mean()
daily_data['SWH_MA30'] = daily_data['SignificantWaveHeight'].rolling(window=30).mean()
daily_data['SWH_STD7'] = daily_data['SignificantWaveHeight'].rolling(window=7).std()
daily_data['SWH_STD30'] = daily_data['SignificantWaveHeight'].rolling(window=30).std()

daily_data.dropna(inplace=True)
```

Figure 16: Phase 2 Feature Engineering

4.4 Modelling Training and Evaluations

Once the data was cleaned, analysed and the feature was selected for the model building, based on the research goal the model was build using MLP in phase 1 and SARIMAX for phase 2.

4.4.1 Phase 1 Wave power generation optimization

In the **Wave_power_energy_output_optimization.ipynb** file navigate to the Model Building – MLP cell and execute the code as given in the image 17, this will take much

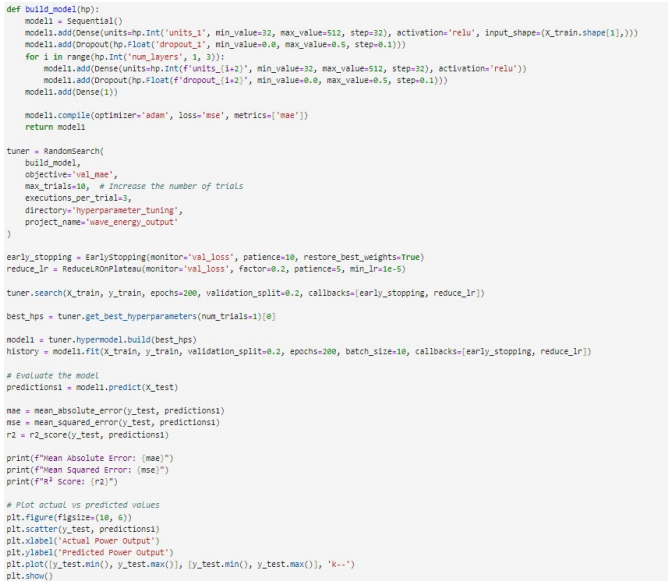
```python
def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units_1', min_value=32, max_value=512, step=32), activation='relu', input_shape=(X_train.shape[1],)))
    model.add(Dropout(hp.Float('dropout_1', min_value=0.0, max_value=0.5, step=0.1)))
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(Dense(units=hp.Int(f'units_{i+2}', min_value=32, max_value=512, step=32), activation='relu'))
        model.add(Dropout(hp.Float(f'dropout_{i+2}', min_value=0.0, max_value=0.5, step=0.1)))
    model.add(Dense(1))

    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

tuner = RandomSearch(
    build_model,
    objective='val_mae',
    max_trials=5,
    executions_per_trial=3,
    directory='hyperparameter_tuning',
    project_name='wave_energy_output'
)

tuner.search(X_train, y_train, epochs=200, validation_split=0.2)

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, y_train, validation_split=0.2, epochs=200, batch_size=10)
```

Figure 17: Phase 1 Model building-MLP.

```python
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Make predictions
predictions = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, predictions)
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions)
plt.xlabel('Actual Power Output')
plt.ylabel('Predicted Power Output')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--')
plt.show()
```

Figure 18: Phase 1 predictions

longer time to execute since its build with 3 layers of neural networks, once the model is trained will all the epoch, now execute the cell predictions as given in the image 18, this



```
def build_model(hp):
    model1 = Sequential()
    model1.add(Dense(units=hp.Int('units_1', min_value=32, max_value=512, step=32), activation='relu', input_shape=(X_train.shape[1],)))
    model1.add(Dropout(hp.Float('dropout_1', min_value=0.0, max_value=0.5, step=0.1)))
    for i in range(hp.Int('num_layers', 1, 3)):
        model1.add(Dense(units=hp.Int(f'units_{i+2}', min_value=32, max_value=512, step=32), activation='relu'))
        model1.add(Dropout(hp.Float(f'dropout_{i+2}', min_value=0.0, max_value=0.5, step=0.1)))
    model1.add(Dense(1))

    model1.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model1

tuner = RandomSearch(
    build_model,
    objective='val_mae',
    max_trials=10,  # Increase the number of trials
    executions_per_trial=3,
    directory='hyperparameter_tuning',
    project_name='wave_energy_output'
)

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-5)

tuner.search(X_train, y_train, epochs=200, validation_split=0.2, callbacks=[early_stopping, reduce_lr])

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

model1 = tuner.hypermodel.build(best_hps)
history = model1.fit(X_train, y_train, validation_split=0.2, epochs=200, batch_size=10, callbacks=[early_stopping, reduce_lr])

# Evaluate the model
predictions1 = model1.predict(X_test)

mae = mean_absolute_error(y_test, predictions1)
mse = mean_squared_error(y_test, predictions1)
r2 = r2_score(y_test, predictions1)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions1)
plt.xlabel('Actual Power Output')
plt.ylabel('Predicted Power Output')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--')
plt.show()
```

Figure 19: Phase 1 Optimization-Hypertunning

will makes the predictions of the trained model with the test data, since this research uses the hyperparameter tunning, execute the cell Optimization – Hypertunning referred in keras (n.d.a) given in the image 19, this will tune the model more for its efficiency,

```
# Residuals Plot
residuals = y_test - predictions1.flatten()
plt.figure(figsize=(10, 6))
plt.scatter(predictions1, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()

# Distribution of Errors
plt.figure(figsize=(10, 6))
sns.histplot(residuals, bins=30, kde=True)
plt.xlabel('Residuals')
plt.title('Distribution of Residuals')
plt.show()

# Predicted vs Actual Distribution
plt.figure(figsize=(10, 6))
sns.histplot(y_test, color='blue', label='Actual', kde=True)
sns.histplot(predictions1.flatten(), color='orange', label='Predicted', kde=True)
plt.xlabel('Power Output')
plt.title('Distribution of Actual and Predicted Values')
plt.legend()
plt.show()
```

Figure 20: Phase 1 Predictions of Optimization

last this optimizations was predicted in the cell Predictions of Optimization given in the image 20, this will gives the over all efficient of the model trained, the results like MSE, MAE and R2 will give the model accuracy in optimization refered from keras (n.d.b).

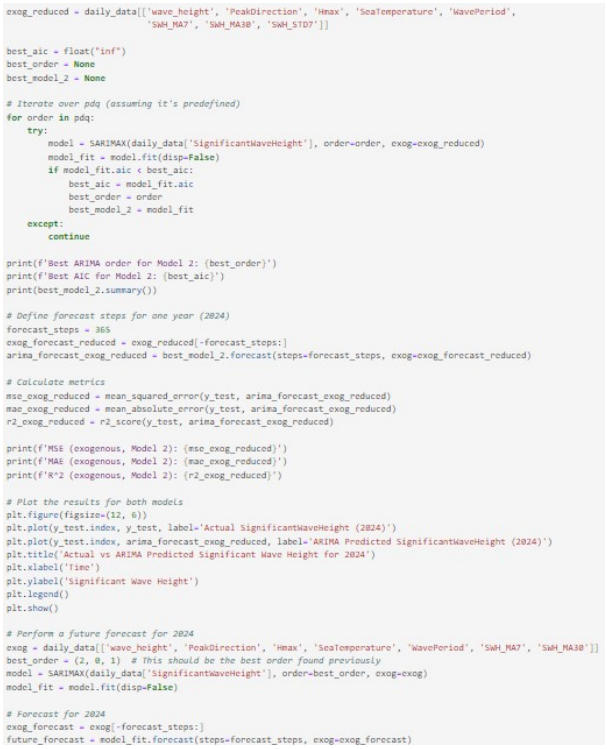
4.4.2 Phase 2 Forecast the wave height and period

In the file **Waveheight_Forecast_SligoArea.ipynb** run the cell Model Building 1 – SARIMAX for the model building as given in the image 21, by this the SARIMAX

```
exog_all = daily_data[['wave_height', 'PeakDirection', 'Hmax', 'SeaTemperature', 'MeanCurSpeed', 'MeanCurDirTo', 'WavePeriod',
                       'SWH_MA7', 'SWH_MA30', 'SWH_STD7', 'SWH_STD30']]

# Parameter Tuning for Model 1
p = d = q = range(0, 3)
pdq = list(product(p, d, q))
best_aic = float("inf")
best_order = None
best_model_1 = None

for order in pdq:
    try:
        model = SARIMAX(daily_data['SignificantWaveHeight'], order=order, exog=exog_all)
        model_fit = model.fit(disp=False)
        if model_fit.aic < best_aic:
            best_aic = model_fit.aic
            best_order = order
            best_model_1 = model_fit
    except:
        continue

print(f'Best ARIMA order for Model 1: {best_order}')
print(f'Best AIC for Model 1: {best_aic}')
print(best_model_1.summary())

forecast_steps = 365
exog_forecast_all = exog_all[-forecast_steps:]
arima_forecast_exog_all = best_model_1.forecast(steps=forecast_steps, exog=exog_forecast_all)

y_test = daily_data['SignificantWaveHeight'][-forecast_steps:]


mse_exog_all = mean_squared_error(y_test, arima_forecast_exog_all)
mae_exog_all = mean_absolute_error(y_test, arima_forecast_exog_all)
r2_exog_all = r2_score(y_test, arima_forecast_exog_all)

print(f'MSE (exogenous, Model 1): {mse_exog_all}')
print(f'MAE (exogenous, Model 1): {mae_exog_all}')
print(f'R^2 (exogenous, Model 1): {r2_exog_all}')
```

Figure 21: Phase 2 Model building 1-SARIMAX

results were generated and check for the p-value, the variable with the p-value more than 0.5 are eliminated in the model 2, now the cell Model Building 2 – SARIMAX as show in the image 22, by this the final fit model is trained for the forecasting, now run the code



```
exog_reduced = daily_data[['wave_height', 'PeakDirection', 'Hmax', 'SeaTemperature', 'WavePeriod',
                           'SWH_MA7', 'SWH_MA30', 'SWH_STD7']]

best_aic = float("inf")
best_order = None
best_model_2 = None

# Iterate over pdq (assuming it's predefined)
for order in pdq:
    try:
        model = SARIMAX(daily_data['SignificantWaveHeight'], order=order, exog=exog_reduced)
        model_fit = model.fit(disp=False)
        if model_fit.aic < best_aic:
            best_aic = model_fit.aic
            best_order = order
            best_model_2 = model_fit
    except:
        continue

print(f'Best ARIMA order for Model 2: {best_order}')
print(f'Best AIC for Model 2: {best_aic}')
print(best_model_2.summary())

# Define forecast steps for one year (2024)
forecast_steps = 365
exog_forecast_reduced = exog_reduced[-forecast_steps:]
arima_forecast_exog_reduced = best_model_2.forecast(steps=forecast_steps, exog=exog_forecast_reduced)

# Calculate metrics
mse_exog_reduced = mean_squared_error(y_test, arima_forecast_exog_reduced)
mae_exog_reduced = mean_absolute_error(y_test, arima_forecast_exog_reduced)
r2_exog_reduced = r2_score(y_test, arima_forecast_exog_reduced)

print(f'MSE (exogenous, Model 2): {mse_exog_reduced}')
print(f'MAE (exogenous, Model 2): {mae_exog_reduced}')
print(f'R^2 (exogenous, Model 2): {r2_exog_reduced}')

# Plot the results for both models
plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Actual SignificantWaveHeight (2024)')
plt.plot(y_test.index, arima_forecast_exog_reduced, label='ARIMA Predicted SignificantWaveHeight (2024)')
plt.title('Actual vs ARIMA Predicted Significant Wave Height for 2024')
plt.xlabel('Time')
plt.ylabel('Significant Wave Height')
plt.legend()
plt.show()

# Perform a future forecast for 2024
exog = daily_data[['wave_height', 'PeakDirection', 'Hmax', 'SeaTemperature', 'WavePeriod', 'SWH_MA7', 'SWH_MA30']]
best_order = (2, 0, 1)  # This should be the best order found previously
model = SARIMAX(daily_data['SignificantWaveHeight'], order=best_order, exog=exog)
model_fit = model.fit(disp=False)

# Forecast for 2024
exog_forecast = exog[-forecast_steps:]
future_forecast = model_fit.forecast(steps=forecast_steps, exog=exog_forecast)
```

Figure 22: Phase 2 Model building 2- SARIMAX

in the cell Future wave height prediction as given in the image 23, this is a visualization of the forecasting the wave height prediction, and for the final run the last cell Storing the future predicted wave heights into the PostgreSQL as Data Warehousing given in the image 24, this is aimed to store the predicted values into the database for the future use.

```
exog = daily_data[['wave_height', 'PeakDirection', 'Hmax', 'SeaTemperature', 'WavePeriod', 'SWH_MA7', 'SWH_MA30']]
best_order = (2, 0, 1)
model = SARIMAX(daily_data['SignificantWaveHeight'], order=best_order, exog=exog)
model_fit = model.fit(disp=False)

forecast_steps = 365
exog_forecast = exog[-forecast_steps:]
future_forecast = model_fit.forecast(steps=forecast_steps, exog=exog_forecast)

plt.figure(figsize=(12, 6))
plt.plot(daily_data.index, daily_data['SignificantWaveHeight'], label='Actual SignificantWaveHeight')
plt.plot(pd.date_range(start=daily_data.index[-1], periods=forecast_steps, freq='D'), future_forecast, label='Forecasted SignificantWaveHeight')
plt.title('Significant Wave Height Forecast')
plt.xlabel('Time')
plt.ylabel('Significant Wave Height')
plt.legend()
plt.show()
```

Figure 23: Phase 2 Future wave height prediction

```
forecast_dates = pd.date_range(start=daily_data.index[-1], periods=forecast_steps + 1, inclusive='right')
forecast_df = pd.DataFrame({'time': forecast_dates, 'predicted_SignificantWaveHeight': future_forecast})

forecast_df_filtered = forecast_df[forecast_df['time'] >= '2024-01-01']

# Connect to the PostgreSQL database
connection_string = "postgresql://dap:dap@127.0.0.1:5432/Wave energy output"
conn = psycopg2.connect(connection_string)
cur = conn.cursor()
create_table_query = """
CREATE TABLE IF NOT EXISTS predicted_Waveheight (
    time TIMESTAMP PRIMARY KEY,
    predicted_SignificantWaveHeight FLOAT
)
"""
cur.execute(create_table_query)

insert_query = sql.SQL("""
INSERT INTO predicted_Waveheight (time, predicted_SignificantWaveHeight)
VALUES (%s, %s)
""")

for i, row in forecast_df_filtered.iterrows():
    cur.execute(insert_query, (row['time'], row['predicted_SignificantWaveHeight']))

conn.commit()
cur.close()
conn.close()
```

Figure 24: Phase 2 storing the predicted values into PostgreSQL

5 Conclusion

After executing these steps, the desirable results can be achieved as per the proposed research, for the both phase of this proposed research, from the phase 1 for optimization the wave power generation efficiency results are shown in the image 25 and the image 26 shows the results of the forecasted wave height as for the phase 2 is generated, by

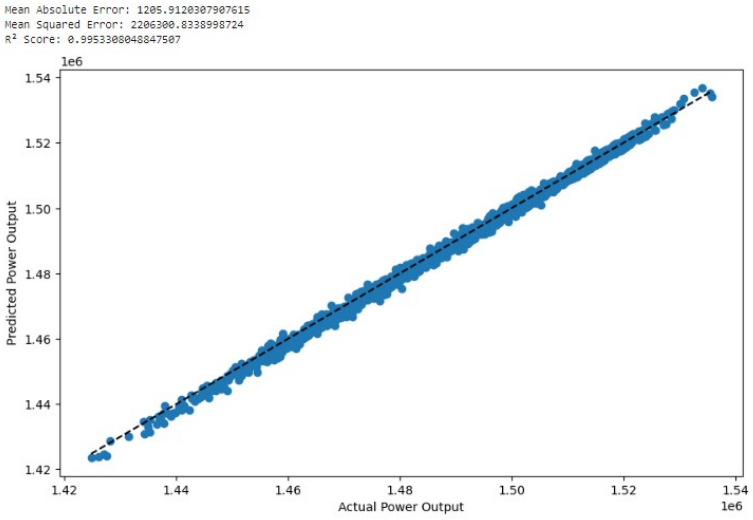


Figure 25: Phase 1 Results

following these steps in this configuration manual, we can able to achieve the desirable outputs,

As this manual provides a detail step by step procedure from setting the environment,

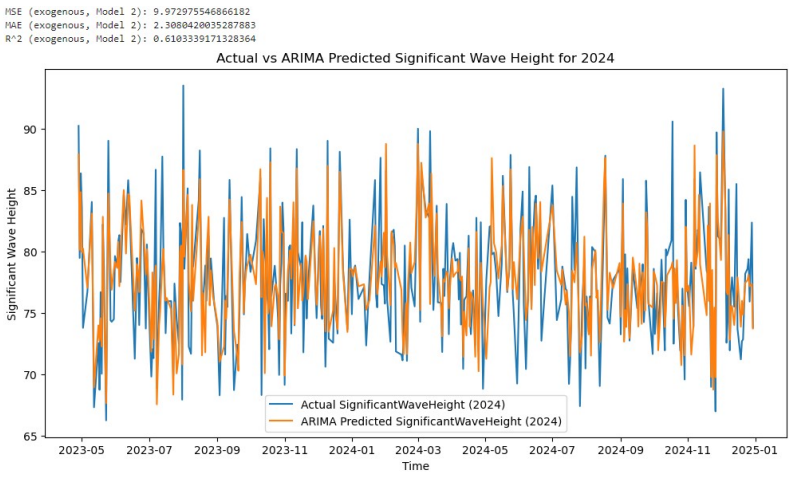


Figure 26: Phase 2 results

configuration the tools and executing the code for modelling of both wave power energy output optimization and wave height forecasting, by following this manual steps will provide a correct replica of the implementation code of this research, this research concludes with this by providing a most accurate in optimizing the efficiency in wave power generation system and forecasting the wave height and period.

References

keras, T. (n.d.a). Keras documentation of early stopping.
URL: *https://keras.io/api/callbacks/early*$_s$*topping/*

keras, T. (n.d.b). Keras documentation of regression metrics.
URL: *https://keras.io/api/metrics/regression*$_m$*etrics/*