Configuration Manual

MSc Research Project
Data Analytics

Samrudhi Hawalli Ramachandra
Student ID:x23242361

School of Computing
National College of Ireland

Supervisor: Teerath Kumar Menghwar

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Samrudhi Hawalli Ramachandra
Student ID:	x23242361
Programme:	MSc.Data Analytics
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Teerath Kumar Menghwar
Submission Due Date:	12/12/2024
Project Title:	ConfigurationManual
Word Count:	430
Page Count:	7

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:	Samrudhi Hawalli Ramachandra
Date:	12th December 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

Samrudhi Hawalli Ramachandra
x23242361

1 Introduction

This manual serves as a guide to document the technical procedures followed during the research. It includes details about the tools, environment, and code configurations used in the study. Additionally, it provides code snippets highlighting specific configurations relevant to the project. The purpose of this manual is to outline the steps required to replicate work and explore extensions of the research.

Section 2 outlines the setup of the environment used to execute the project. Section 3 describes the process of data collection in detail. Section 4 highlights the initial analysis, model implementation, and the results obtained.

2 Environment

The hardware utilized for this project is detailed in Figure 1. It includes an AMD Ryzen 5 5500U processor with Radeon Graphics running at 2.10 GHz, 8.00 GB of installed RAM (7.33 GB usable), and a 64-bit operating system with an x64-based processor architecture.

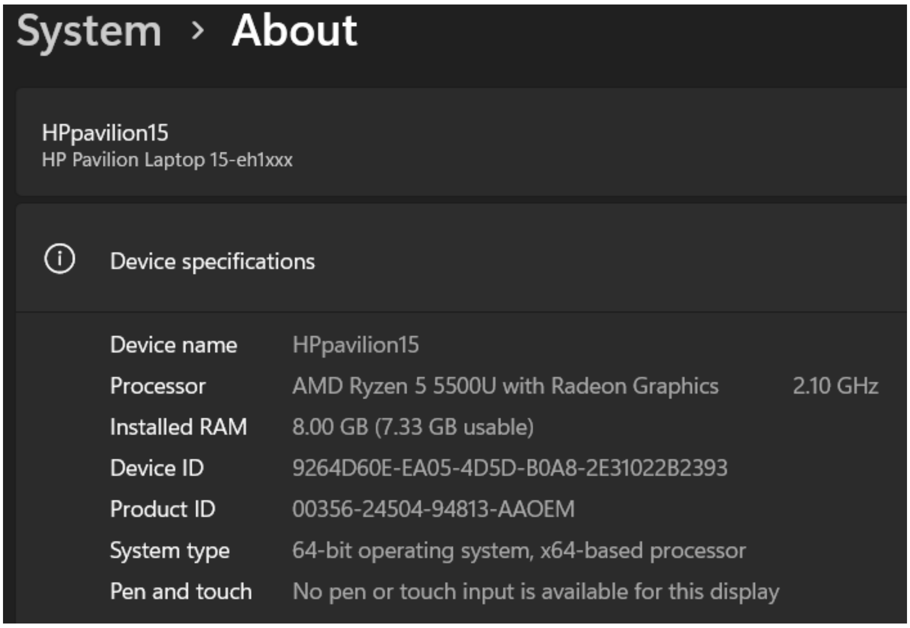

Figure 1: Device specifications

The code was executed using Jupyter Notebook and Python version 3.10.0.

3 Data Collection

The earthquake datasets from Kaggle offer valuable insights into global earthquake activity. These datasets include various parameters, such as time, location (latitude and longitude), depth, magnitude, magnitude type, and other relevant details.

Figure 2: Dataset1

Figure 3: Dataset2

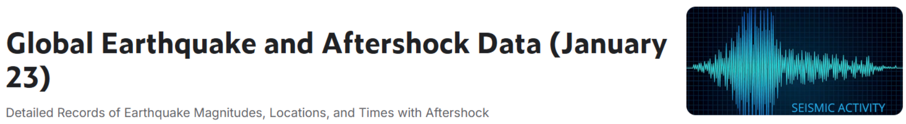

Figure 4: Dataset3

4 Model Implementation

Figure 5 shows the libraries imported into the code.

```python
# import necessary libraries

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import time

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

from sklearn.ensemble import AdaBoostRegressor,RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import BayesianRidge

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

Figure 5: Imported Libraries

Next we load the dataset and check the information of the dataset like total columns and rows. Also check the datatypes of the columns

```
# Load the dataset
dataset1_file_path = 'earthquakes_2023_global.csv'
data1 = pd.read_csv(dataset1_file_path)
```

```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26642 entries, 0 to 26641
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   time             26642 non-null  object
 1   latitude         26642 non-null  float64
 2   longitude        26642 non-null  float64
 3   depth            26642 non-null  float64
 4   mag              26642 non-null  float64
 5   magType          26642 non-null  object
 6   nst              25227 non-null  float64
 7   gap              25225 non-null  float64
 8   dmin             24776 non-null  float64
 9   rms              26642 non-null  float64
 10  net              26642 non-null  object
 11  id               26642 non-null  object
 12  updated          26642 non-null  object
 13  place            25034 non-null  object
 14  type             26642 non-null  object
 15  horizontalError  25093 non-null  float64
 16  depthError       26642 non-null  float64
 17  magError         24970 non-null  float64
 18  magNst           25065 non-null  float64
 19  status           26642 non-null  object
 20  locationSource   26642 non-null  object
 21  magSource        26642 non-null  object
dtypes: float64(12), object(10)
memory usage: 4.5+ MB
```

Figure 6: Load the dataset

Figure 7, shows dropping irrelevant columns handling of missing values by filling empty values by mean/median.

```python
# Drop irrelevant columns
columns_to_drop = ['id', 'net', 'updated', 'magSource', 'locationSource','status']
data_cleaned1 = data1.drop(columns=columns_to_drop)

# Handle missing values
data_cleaned1 = data_cleaned1.fillna(data_cleaned1.median(numeric_only=True))
```

Figure 7: Preprocessing Steps

Figure 8 and 9 shows the detection and removal of outliers

```python
##outliers
outliers_columns = ['mag','depth', 'gap', 'dmin', 'rms', 'horizontalError', 'depthError']

for col in outliers_columns:
    # Create a boxplot for the column 'Values'
    sns.boxplot(y=data_cleaned1[col])

    # Add a title and labels
    plt.title(f'Boxplot for {col} Column')
    plt.ylabel(col)

    # Show the plot
    plt.show()
```

Figure 8: Outliers Detection

```python
def remove_outliers_iqr(data, columns):
    for col in columns:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        data = data[(data[col] >= lower_bound) & (data[col] <= upper_bound)]
    return data

# Columns for outlier removal (choose relevant numerical features)
outlier_columns = ['gap', 'dmin', 'rms', 'horizontalError', 'depthError']

# Remove outliers from the training data
data_cleaned1 = remove_outliers_iqr(data_cleaned1, outlier_columns)
```

Figure 9: Outliers removal

Label encoding is done for categorical columns as a part of feature engineering

```
# Encode categorical variables
label_encoder = LabelEncoder()
data_cleaned1['magType'] = label_encoder.fit_transform(data_cleaned1['magType'])
data_cleaned1['type'] = label_encoder.fit_transform(data_cleaned1['type'])

# Update X and y after outlier removal
X1 = data_cleaned1.drop(columns=['mag', 'time', 'place','date'])
y1 = data_cleaned1['mag']
```

Figure 10: Label Encoding

Split the dataset into train and test subset with 80% of the data for train subset and 20% of the data to test subset.

```
# Train-test split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled1 = scaler.fit_transform(X_train1)
X_test_scaled1 = scaler.transform(X_test1)
```

Figure 11: Split the dataset

Define the Machine learning models into a dictionary for iteration

```
# Define regressors as a dictionary
custom_regressors = {
    "LinearRegression": LinearRegression(),
    "RandomForestRegressor": RandomForestRegressor(random_state=42, n_estimators=100),
    "DecisionTreeRegressor":  DecisionTreeRegressor(random_state=42),
    "KNeighborsRegressor": KNeighborsRegressor(n_neighbors=5),
    "SVR": SVR(kernel='rbf', C=1.0, epsilon=0.1),
    "GradientBoostingRegressor": GradientBoostingRegressor(random_state=42, n_estimators=100),
    "BayesianRidge":  BayesianRidge(),
    "LassoModel": Lasso(alpha=0.1, random_state=42),
    "ElasticNetModel": ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42),
    "AdaBoostRegressor": AdaBoostRegressor()
}
```

Figure 12: machine learning models

```python
# Initialize results storage
results1 = []

# Loop through regressors
for name, model in custom_regressors.items():
    start_time = time.time()

    # Train the model
    model.fit(X_train_scaled1, y_train1)

    # Predict on test set
    predictions = model.predict(X_test_scaled1)

    # Evaluate performance
    r2 = r2_score(y_test1, predictions)
    mse = mean_squared_error(y_test1, predictions)

    elapsed_time = time.time() - start_time

    # Append results
    results1.append({
        "Model": name,
        "mse": mse,
        "R-Squared": r2,
        "Time Taken": elapsed_time
    })

# Convert results to DataFrame
results_df1 = pd.DataFrame(results1)
results_df1.sort_values(by="R-Squared", ascending=False, inplace=True)

# Display results
results_df1
```

Figure 13: Train the models

The results of all the models are compared visually using bar graph.

```python
plt.figure(figsize=(12, 6))
sns.barplot(x='Model', y='R-Squared', data=results_df1, palette='magma')
plt.title('Model Performance')
plt.xticks(rotation=90)
plt.show()
```

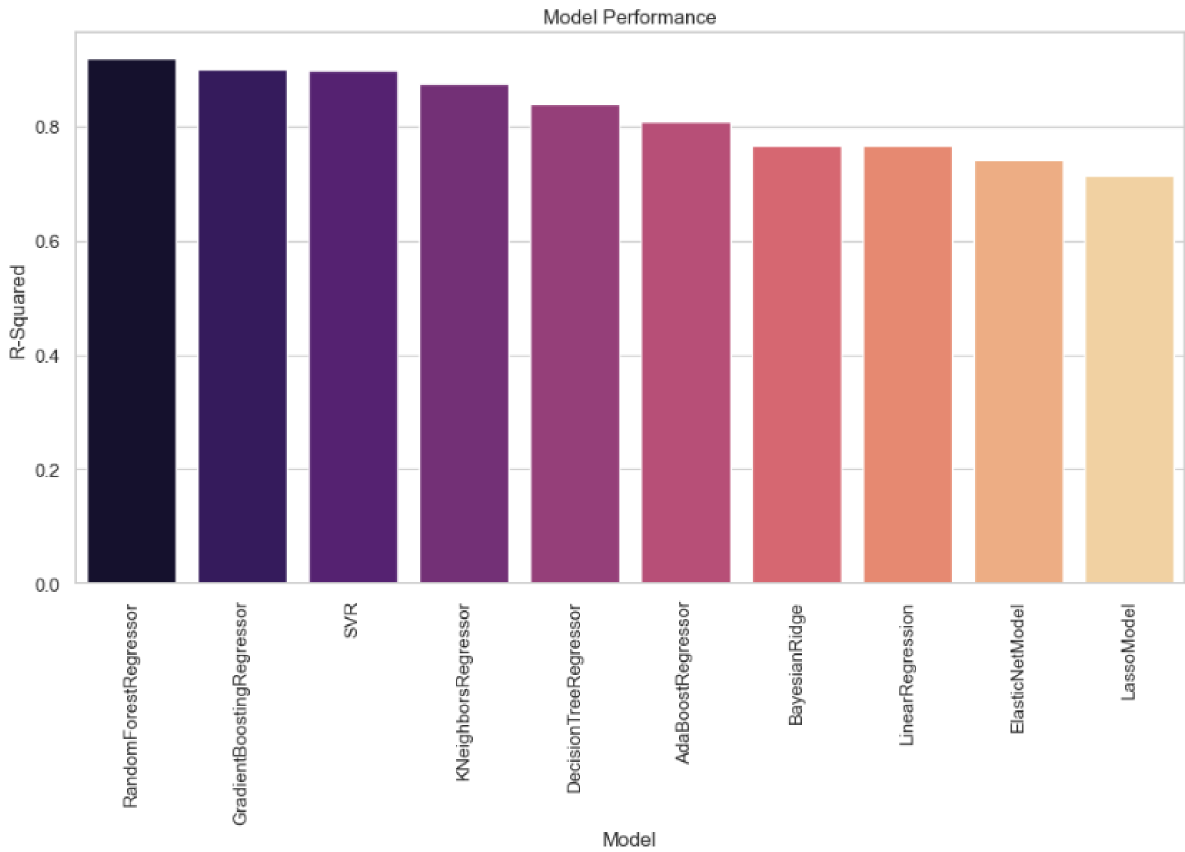

Figure 14: results Comparision