

Configuration Manual

MSc Research Project
Cloud Computing

Yaseen Ali Khan
Student ID: 22236252

School of Computing
National College of Ireland

Supervisor: Giovani Estrada

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Yaseen Ali Khan.....
Student ID: 22236252.....
Programme: Cloud Computing..... **Year:** 2025.....
Module: MSC Research Project.....
Lecturer: Giovani Estrada
Submission Due Date: 24 April 2025.....
Project Title: Configuration Manual.....
Word Count: 716..... **Page Count:** 8.....

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: Yaseen Ali Khan.....
Date: 24 April 2025.....

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

Yaseen Ali Khan
Student ID: 22236252

1. Introduction

This configuration manual provides a step-by-step instruction on how to set up an EC2 instance using a Python virtual environment to execute an earthquake magnitude prediction application. This system leverages real earthquake data and applies multiple ML models such as Random Forest Regressor, SVR, Neural Networks, and GRU for forecasting purposes.

2. Launch AWS EC2 Instance

- Log in to AWS Management Console.
- Go to EC2 Dashboard → Launch Instance
- Choose: AMI: Ubuntu 20.04
 - Instance Type: t2.medium or higher
 - Storage: 20 GB SSD
- Open ports:
 - 22 (SSH)
 - 8888 (Jupyter Notebook)
- Download and keep the `.pem` key safe.

3. Connect to instance

- **On your terminal:**

```
chmod 400 your-key.pem  
ssh -i "your-key.pem" ubuntu@<EC2_PUBLIC_IP>
```

4. Install core packages

```
sudo apt update && sudo apt upgrade -y  
sudo apt install python3-pip python3-venv git -y
```

5. Setup python Virtual Environment

```
Python3-m venv env  
Source env/bin/activate
```

6. Install python environment

```
pip install --upgrade pip
pip install pandas numpy matplotlib seaborn scikit-learn scipy jupyter tensorflow
keras
```

7. Upload project files

- Create project folder and move files:

```
mkdir disaster_mgmt && cd disaster_mgmt
```

- Upload:

- database.csv
- Disaster_Management.ipynb

8. Start Jupyter Notebook

```
jupyter notebook --ip=0.0.0.0 --port=8888 --no-browser
```

Access via:

http://<EC2_PUBLIC_IP>:8888/?token=...

9. Data Preprocessing Steps

- Load dataset.
- Drop high-missing columns (>70%)
- Impute missing data using median.
- Encode categorical features using `pd.get_dummies`.
- Split dataset into `X_train`, `X_test`, `y_train`, `y_test`.

10. Machine Learning Model Training

- Random Forest Regressor: Trained and tuned using `RandomizedSearchCV`.

```
[ ] 1 # Drop columns with more than 70% missing values
2 missing_threshold = 0.7 * len(df)
3 df.dropna(thresh=missing_threshold, axis=1, inplace=True)

[ ] 1 # Imputing missing values with median
2 df.fillna(df.median(numeric_only=True), inplace=True)

[ ] 1 df.drop(columns=["ID"], errors="ignore", inplace=True)

1 df.isna().sum()

0
Latitude      0
Longitude     0
Type          0
Depth         0
Magnitude     0
Magnitude Type 3
Root Mean Square 0
Source        0
Location Source 0
Magnitude Source 0
Status        0
Datetime      3
dtype: int64
```

- SVR: Uses Pipeline, Box-Cox (with pre-check for positive/finite data).

```
[ ] from sklearn.preprocessing import PowerTransformer

# Positivity check for Box-Cox
if (y <= 0).any():
    y += abs(y.min()) + 1

pt_y = PowerTransformer(method='box-cox')
y_trans = pt_y.fit_transform(y.values.reshape(-1, 1)).flatten()

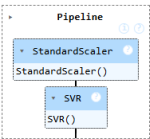
[ ] from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y_trans, test_size=0.2, random_state=42)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svr', SVR(kernel='rbf'))
])

# Train the model
pipeline.fit(X_train, y_train)
```



The diagram shows a Pipeline object containing two steps: StandardScaler and SVR. StandardScaler is the first step, and SVR is the second step. Both steps are represented by blue boxes with their respective class names and constructor signatures.

- Simple Neural Networks (Keras): Standardized features, 3 hidden layers with ReLU.

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.keras import layers
4 from sklearn.preprocessing import StandardScaler
5
6 # Standardization
7 scaler = StandardScaler()
8 X_train_scaled = scaler.fit_transform(X_train)
9 X_test_scaled = scaler.transform(X_test)
10
11 # Simple neural network model
12 nn_model = keras.Sequential([
13     layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
14     layers.Dense(32, activation='relu'),
15     layers.Dense(1)
16 ])
17
```

- Simple Neural Networks (Keras): Standardized features, 3 hidden layers with ReLU.

```
1 import tensorflow as tf
2 from tensorflow import keras
3 from tensorflow.keras import layers
4 from sklearn.preprocessing import StandardScaler
5
6
7 scaler = StandardScaler()
8 X_train_scaled = scaler.fit_transform(X_train)
9 X_test_scaled = scaler.transform(X_test)
10
11 # feedforward neural network
12 nn_model = keras.Sequential([
13     layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
14     layers.Dense(32, activation='relu'),
15     layers.Dense(1)
16 ])
17
```

 /usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape` super().__init__(activity_regularizer=activity_regularizer, **kwargs)

- GRU Model: Standardized features, 3 hidden layers with tanh activation. Trained the model over 100 epochs.

```

1 # Define a GRU model
2 gru_model = keras.Sequential([
3     layers.GRU(128, activation='tanh', return_sequences=True, input_shape=(X_train.shape[1], 1)),
4     layers.GRU(64, activation='tanh', return_sequences=True),
5     layers.GRU(32, activation='tanh'),
6     layers.Dense(1)
7 ])

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape` to
super().__init__(**kwargs)

[ ] 1 gru_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

```

- Conv1D Model: Standardized features, 3 hidden layers with ReLu activation. Trained the model over 100 epochs.

```

1 conv1d_model = keras.Sequential([
2     layers.Reshape((X_train.shape[1], 1), input_shape=(X_train.shape[1],)),
3     layers.Conv1D(64, kernel_size=5, activation='relu'),
4     layers.MaxPooling1D(pool_size=2),
5     layers.Conv1D(32, kernel_size=3, activation='relu'),
6     layers.GlobalAveragePooling1D(),
7     layers.Dense(32, activation='relu'),
8     layers.Dense(1)
9 ])

/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/reshape.py:3
super().__init__(**kwargs)

[ ] 1 conv1d_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

```

- Bidirectional Model: Standardized features, 3 hidden layers with tanh activation. Trained the model over 100 epochs.

```

[ ] 1 # Define an improved Bidirectional LSTM model
2 bilstm_model = keras.Sequential([
3     layers.Bidirectional(layers.LSTM(128, activation='tanh', return_sequences=True), input_shape=(X_train.shape[1], 1)),
4     layers.Bidirectional(layers.LSTM(64, activation='tanh', return_sequences=True)),
5     layers.Bidirectional(layers.LSTM(32, activation='tanh')),
6     layers.Dense(1)
7 ])

/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/bidirectional.py:107: UserWarning: Do not pass an `input_shape` to
super().__init__(**kwargs)

[ ] 1 bilstm_model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=0.0005), loss='mse', metrics=['mae'])

```

- Bidirectional GRU Conv1D: Standardized features, 3 hidden layers with ReLU activation. Trained the model over 100 epochs.

```

1 # Define a Bidirectional + GRU + Conv1D model
2 bi_gru_conv_model = keras.Sequential([
3     layers.Conv1D(64, kernel_size=5, activation='relu', input_shape=(X_train.shape[1], 1)),
4     layers.MaxPooling1D(pool_size=2),
5     layers.Bidirectional(layers.GRU(128, activation='tanh', return_sequences=True)),
6     layers.Bidirectional(layers.GRU(64, activation='tanh', return_sequences=True)),
7     layers.Bidirectional(layers.GRU(32, activation='tanh')),
8     layers.Dense(1)
9 ])

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning:
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

[ ] 1 bi_gru_conv_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

```

- Extra Trees Regressor: Standardized features like `n_estimators` and `random_state`.

```

1 from sklearn.ensemble import ExtraTreesRegressor, GradientBoostingRegressor
2
3 # Define an Extra Trees Regressor
4 xtr_model = ExtraTreesRegressor(n_estimators=100, random_state=42)
5 xtr_model.fit(X_train_scaled, y_train)
6 y_pred_xtr = xtr_model.predict(X_test_scaled)

```

- Gradient Boosting Regressor: Standardized features like `n_estimators` and `random_state`.

```

[ ] 1 # Define a Gradient Boosting Regressor
2 gbm_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
3 gbm_model.fit(X_train_scaled, y_train)
4 y_pred_gbm = gbm_model.predict(X_test_scaled)

```

11. Monitoring & Debugging Tips

- Handle NaN values before using SVR and neural networks.
- Ensure Box-Cox transformation only on strictly positive, non-zero features.
 - Use StandardScaler before neural models.
 - Monitor training with validation loss to avoid overfitting.

12. Optional Enhancements

- Use TensorBoard for visualizing training.
- Store results in AWS S3 for persistent model/data logs.

- Host inference using Flask + EC2 or SageMaker endpoint.

13.Simulation & Metrics

- Evaluate using MAE, MSE, R^2 .
- Feature importance plotted using `RandomForestRegressor.feature_importances_`.

