National College of Ireland

Configuration Manual for Optimizing Renewable Energy Management through Solar Power Forecasting

MSc Research Project
MSCDAD_C

Karthikram Chockalingam Swaminathan
x23127651@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Abid Yaqoob (Abid.Yaqoob@ncirl.ie)

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Karthikram Chockalingam Swaminathan		
Student ID:	X23127651		
Programme:	MSCDAD_C	**Year:**	Sep(2023 – 2024)
Module:	Research Project		
Lecturer:	Abid Yaqoob		
Submission Due Date:	16-09-2024		
Project Title:	Optimizing renewable Energy management - Solar power Forecasting		
Word Count:	1432	**Page Count:**	10

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:	Karthikram CS
Date:	16-09-2024

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 for Optimizing Renewable Energy Management through Solar Power Forecasting

Karthikram Chockalingam Swaminathan
x23127651@student.ncirl.ie

1. Introduction

This configuration manual provides a comprehensive guide to the setup, execution, and management of the research study on "Optimizing Renewable Energy Management through Solar Power Forecasting." The study aims to enhance the accuracy of solar power predictions using advanced machine learning models, thereby contributing to more efficient renewable energy management and grid integration. This manual details the system specifications, software requirements, dataset sources, and step-by-step instructions for executing the code implementation.

2. System Specification

To ensure the smooth execution of the code and to handle the computational demands of training complex machine learning models, the following system specifications are recommended:

- Processor: Intel Core i7 or AMD Ryzen 7 (or equivalent) with at least 4 cores
- RAM: Minimum 16 GB (32 GB recommended for handling large datasets)
- Storage: 500 GB SSD (Solid State Drive) for faster data processing
- GPU: NVIDIA GPU with CUDA support (e.g., NVIDIA GTX 1070 or higher) for accelerating deep learning model training
- Operating System: Windows 10 (64-bit), Ubuntu 18.04 LTS or later, or macOS 10.14 or later
- Python Version: Python 3.7 or later

These specifications will ensure that the system can handle the computational demands of training and deploying the deep learning model, especially when using large datasets and complex neural network architectures.

3. Softwares Used:

The following software and libraries are required for implementing and executing the machine learning models used in this study:

- **Python:** The primary programming language used for coding and executing the models.
- **Jupyter Notebook:** An open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.
- **Pandas:** For data manipulation and analysis, particularly useful in handling large datasets.
- **NumPy:** A fundamental package for scientific computing with Python, used for array operations.

- **Matplotlib & Seaborn:** Libraries for creating static, animated, and interactive visualizations in Python.
- **Scikit-learn:** A machine learning library for Python that provides simple and efficient tools for data mining and data analysis.
- **TensorFlow/Keras:** Deep learning libraries used for implementing and training the Transformer Model.
- **CUDA (Optional):** If using a GPU for accelerating model training, ensure that CUDA and cuDNN are installed and configured correctly.

4. Dataset Source

The dataset used in this study comprises historical solar power data, including attributes such as DC power, AC power, daily and total energy yields, ambient and module temperatures, and irradiation levels. The dataset can be obtained from the following sources:

- Dataset: **Solar Power Plant Dataset**

Ensure that the dataset is downloaded and stored in a directory that is accessible to the Python environment where the code will be executed.

5. Execution of the Code Implementation

To execute the code implementation for this research study, follow the steps below:

1. Setup Python Environment:

- Ensure that Python 3.7 or later is installed on your system.
- Create a virtual environment to manage dependencies and avoid conflicts:

```
python -m venv solar_forecasting_env
```

- Activate the virtual environment:
  - On Windows

```
solar_forecasting_env\Scripts\activate
```

  - On macOS/Linus

```
source solar_forecasting_env/bin/activate
```

- Install the required libraries using pip:

```
pip install pandas numpy matplotlib seaborn scikit-learn tensorflow
```

2. Download and Load the Dataset:

- Download the dataset from the source mentioned in Section 4.
- Place the dataset in the working directory of your Python environment.
- Load the dataset into a pandas Data Frame in the Jupiter Notebook or Python script:

```python
df_solar  = pd.read_csv("mergedsolarmy.csv")
```

Python

```python
df_solar.columns
```

Python

```
Index(['SNO', 'SOURCE_KEY', 'DC_POWER', 'AC_POWER', 'DAILY_YIELD',
       'TOTAL_YIELD', 'DATE_TIME', 'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE',
       'IRRADIATION', 'DATE', 'TIME', 'DAY', 'MONTH', 'WEEK', 'HOURS',
       'MINUTES', 'TOTAL MINUTES PASS', 'DATE_STRING', 'SOURCE_KEY_NUMBER'],
      dtype='object')
```

3. Explore the Dataset:

- Explore the dataset to understand the data, its structure, and relationships between the features.

```python
# View the first 5 rows of the dataset
df_solar.head()
```

```python
#shape of the dataset
df_solar.shape
```

```python
# View the dataset information
df_solar.info()
```

```python
# View the dataset statistics
df_solar.describe()
```

4. Data Cleaning:

- Perform Cleaning by handling missing values, removing duplicates, and converting categorical variables into numerical ones.

```python
# Check for missing values
df_solar.isnull().sum()
```

```python
#check the datatypes we have on our dataset
df_solar.dtypes
```

5. Data Preprocessing:

- Perform feature engineering, and normalization as outlined in the methodology of the research study with Solar power plant inventory efficiency calculations.

- It calculates the Average and Standard Deviation in the Data Frame.

```python
avg = df_solar.AC_POWER.mean()
std_dev = df_solar.AC_POWER.std()
std_dev
```

- It calculates the z-score for each value in the AC_POWER column and stores these z-scores in a new column named score within the df_solar Data Frame.

```
df_solar['z_score'] = df_solar.AC_POWER.apply(lambda x:(x-avg)/std_dev)
```

- It returns the number of unique dates present in the DATE column of the df_solar Data Frame.

```
df_solar['DATE'].nunique()
```

- The code calculates the first and third quartiles of the AC_POWER values in the df_solar Data Frame

```
q1 = df_solar.AC_POWER.quantile(q=0.25)
q2 = df_solar.AC_POWER.quantile(q=0.75)
```

```
iqr = q2-q1
iqr
```

```
solar_dc_power = df_solar[df_solar['DC_POWER'] > 0]['DC_POWER'].values
solar_ac_power = df_solar[df_solar['AC_POWER'] > 0]['AC_POWER'].values
```

- Solar power Inverter Efficiency calculation.

```
solar_dc_power.max()
```

```
solar_ac_power.max()
```

```
solar_plant_eff = (np.max(solar_ac_power)/np.max(solar_dc_power ))*100
print(f"Power conversion Efficiency ratio AC/DC of Solar Power Plant:  {solar_plant_eff:0.3f} %")
```

```
AC_list=[]
for i in df_solar['AC_POWER']:
    if i>0:
        AC_list.append(i)
AC_list
AC_list.sort()
AC_list.reverse()
len(AC_list)
```

```
#Here we take all nonzero DC values and plot them on histogram
DC_list=[]
for i in df_solar['DC_POWER']:
    if i>0:
        DC_list.append(i)
DC_list
DC_list.sort()
DC_list.reverse()
len(DC_list)
```

6. Vizualize:

Data visualization helps us to understand the data distribution, and correlations between the features. We can use various seaborn plots such as scatter, histogram, and heatmap to visualize the dataset.

- Calculate the Ambient Temperature over the Months.

```
# AMBIENT_TEMPERATURE over months
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_solar, x='MONTH', y='AMBIENT_TEMPERATURE')
plt.title('AMBIENT_TEMPERATURE over Months')
plt.xlabel('Month')
plt.ylabel('AMBIENT_TEMPERATURE')
plt.show()
```

- Daily Yield over time calculation

```
# DAILY_YIELD over time
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_solar, x='DATE', y='DAILY_YIELD')
plt.title('DAILY_YIELD over Date')
plt.xlabel('Date')
plt.ylabel('DAILY_YIELD')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

- Output Power Vs efficiency

```
#Output power Vs Efficiency
plt.figure(figsize=(16,8))
AC_list.sort()
DC_list.sort()
eff = [i/j for i,j in zip(AC_list,DC_list)]

plt.plot(AC_list,eff,color='darkgreen')
plt.xlabel('Output power in kW')
plt.ylabel('Efficiency AC/DC')
plt.title('Output Power vs Efficiency')
plt.show()
```

- Calculation of Z score

```
# Histogram of z-scores
plt.figure(figsize=(10, 6))
sns.histplot(df_solar['z_score'], bins=20)
plt.title('Distribution of z-scores')
plt.xlabel('z-score')
plt.ylabel('Frequency')
plt.show()
```

7. Split Data into Train and Test Sets:

- Now we will split the data into training and testing sets. We will use 80% of the data for training and 20% for testing.

```
#splitted the dataset into 80:20 ratio where 80% dataset is splitted to training
#and 20 % dataset is splitted to test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

```
# Standardize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

8. Built and Evaluate the Model:

In this step, we will build the various models and evaluate their performance.

- Gradient boosting:

```
# Build the Gradient Boosting Model
gb_model = GradientBoostingRegressor(n_estimators = 35,random_state=42)
# Fit the model on the training data
gb_model.fit(X_train, y_train)
```

Then find the R2 score ,Mean Square Error , and mean absolute error for the Model.

```
y_pred_gb = gb_model.predict(X_test)
R2_Score_gb = round(r2_score(y_pred_gb,y_test) * 100, 2)
print("R2 Score for Gradient Boosting Model: ",R2_Score_gb,"%")
```

```
gb_mean_sq = mean_squared_error(y_pred_gb,y_test)
print("Mean Square Error of Gradient Boosting Model: ",gb_mean_sq)
```

```
gb_mean_ab = mean_absolute_error(y_pred_gb,y_test)
print("Mean Absolute Error of Gradient Boosting Model: ",gb_mean_ab)
```

Print the results of the Gradient Boosting.

```
print("Results For Gradient Boosting: ")
cross_check_gb = pd.DataFrame({'Actual' : y_test , 'Predicted' : y_pred_gb})
cross_check_gb['Error'] = cross_check_gb['Actual'] - cross_check_gb['Predicted']
cross_check_gb.head()
```

```
cross_check_gb['percentage'] = (((cross_check_gb['Error']).abs())/cross_check_gb['Actual'])*100
cross_check_gb.head()
```

- KNN Regressor:

```
# Build the K Nearest Neighbours Model with 4 number of neighbors
knn_model = KNeighborsRegressor(n_neighbors=4)
# Fit the model on the training data
knn_model.fit(X_train, y_train)
```

Then find the R2 score ,Mean Square Error , and mean absolute error for the Model.

```
R2_Score_knn = round(r2_score(y_pred_knn,y_test) * 100, 2)
print("R2 Score for K Nearest Neighbours: ",R2_Score_knn,"%")
```

```
print("Mean Square Error of K Nearest Neighbours Model: ",knn_mean_sq)
```

```
knn_mean_ab = mean_squared_error(y_pred_knn,y_test)
print("Mean Absolute Error of K Nearest Neighbours Model: ",knn_mean_ab)
```

Print the results of the KNN.

```
cross_check_knn['percentage'] = (((cross_check_knn['Error']).abs())/cross_check_knn['Actual'])*100
cross_check_knn.head()
```

- Decision Tree

```
# Standardize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Then find the R2 score ,Mean Square Error , and mean absolute error for the Model.

```
y_pred_dr = dt_model.predict(X_test)
R2_Score_dt = round(r2_score(y_pred_dr,y_test) * 100, 2)
print("R2 Score for Decision Tree Model: ",R2_Score_dt,"%")
```

```
dt_mean_sq = mean_squared_error(y_pred_dr,y_test)
print("Mean Square Error of Decision Tree Model: ",dt_mean_sq)
```

```
dt_mean_ab = mean_absolute_error(y_pred_dr,y_test)
print("Mean Absolute Error of Decision Tree Model: ",dt_mean_ab)
```

Results for Decision Tree .

```
print("Results For Decision Tree: ")
cross_check_dr = pd.DataFrame({'Actual' : y_test , 'Predicted' : y_pred_dr})
cross_check_dr['Error'] = cross_check_dr['Actual'] - cross_check_dr['Predicted']
cross_check_dr.head()
```

- Random Forest

```
# Build the Random Forest model
rdf_model = RandomForestRegressor(random_state=42,max_depth = 6)
# Fit the model on the training data
rdf_model.fit(X_train,y_train)
```

Then find the R2 score ,Mean Square Error , and mean absolute error for the Model.

```
R2_Score_rdf = round(r2_score(y_pred_rdf,y_test) * 100, 2)
print("R2 Score for Random Forest Model: ",R2_Score_rdf,"%")
```

```
rdf_mean_sq = mean_squared_error(y_pred_rdf,y_test)
print("Mean Square Error of Random Forest Model: ",rdf_mean_sq)
```

```
print("Random Forest Regressor (MSE) for Differnt Depths:")
print("\t Mean Square Error For Depth 1: ",rf_mse1)
print("\t Mean Square Error For Depth 3: ",rf_mse2)
print("\t Mean Square Error For Depth 5: ",rf_mse3)
print("\t Mean Square Error For Depth 6: ",rdf_mean_sq)
```

```
rdf_mean_ab = mean_absolute_error(y_pred_rdf,y_test)
print("Mean Absolute Error of Random Forest Model: ",rdf_mean_ab)
```

Results for Random Forest.

```
print("Results For Random Forest: ")
cross_check_rdf = pd.DataFrame({'Actual' : y_test , 'Predicted' : y_pred_rdf})
cross_check_rdf['Error'] = cross_check_rdf['Actual'] - cross_check_rdf['Predicted']
cross_check_rdf.head()
```

- Transformer Model

```
# Build the transformer model
transformer_encoder = build_transformer_model(input_shape, head_size, num_heads, ff_dim, num_transformer_blocks, mlp_units, dropo

transformer_model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer
])

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

Then find the R2 score ,Mean Square Error , and mean absolute error for the Model.

```
R2_Score_transformer = round(r2_score(y_pred_transformer,y_test) * 100, 2)
print("R2 Score for Transformer Model: ",R2_Score_transformer,"%")
```

```
transformer_mean_sq = mean_squared_error(y_pred_transformer,y_test)
print("Mean Square Error of Transformer Model: ",transformer_mean_sq)
```

```
transformer_mean_ab = mean_absolute_error(y_pred_transformer,y_test)
print("Mean Absolute Error of Transformer Model: ",transformer_mean_ab)
```

Results of the Transformer Model.

```
# Standardize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

9. Model Training Results:

- Train the machine learning models using the preprocessed data. The code will include the training and evaluation of various models such as Gradient Boosting, KNN, Decision Tree, Random Forest, and Transformer Model.

```
# Build the Gradient Boosting Model
gb_model = GradientBoostingRegressor(n_estimators = 35,random_state=42)
# Fit the model on the training data
gb_model.fit(X_train, y_train)
```

Python

```
GradientBoostingRegressor
GradientBoostingRegressor(n_estimators=35, random_state=42)
```

```
# Build the K Nearest Neighbours Model with 1 number of neighbors
knn_model1 = KNeighborsRegressor(n_neighbors=1)
# Fit the model on the training data
knn_model1.fit(X_train, y_train)
```

Python

```
KNeighborsRegressor
KNeighborsRegressor(n_neighbors=1)
```

```
# Build the Decision Tree model
dt_model = DecisionTreeRegressor(max_depth=6)
# Fit the model on the training data
dt_model.fit(X_train,y_train)
```

Python

```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=6)
```

```
# Build the Random Forest model for depth 1
rf_model1 = RandomForestRegressor(random_state=42,max_depth = 1)
# Fit the model on the training data
rf_model1.fit(X_train,y_train)
```

Python

```
RandomForestRegressor
RandomForestRegressor(max_depth=1, random_state=42)
```

```
# Train the model
history = transformer_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

Python

```
Epoch 1/50
481/481 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 391259.4688 - val_loss: 29349.0254
Epoch 2/50
481/481 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 22143.2754 - val_loss: 9279.3457
Epoch 3/50
481/481 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 7639.1860 - val_loss: 4186.4888
Epoch 4/50
```

10. Prediction and Evaluation:

- **Comparison of Machine language models.**
- Once the models are trained, use them to make predictions on the test dataset. Evaluate the predictions using the specified metrics (MSE, MAE, R^2)

```
algorithms = [gb_model,knn_model,dt_model,rdf_model,transformer_model]
algorithm_names = ['Gradient Boosting Regressor', 'K-Nearest Neighbors Regressor', 'Decision Tree Regressor','Random Forest Regre
scores = []

for model, name in zip(algorithms, algorithm_names):
    y_pred = model.predict(X_test)

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

    scores.append((name, mse, mae, r2))
```

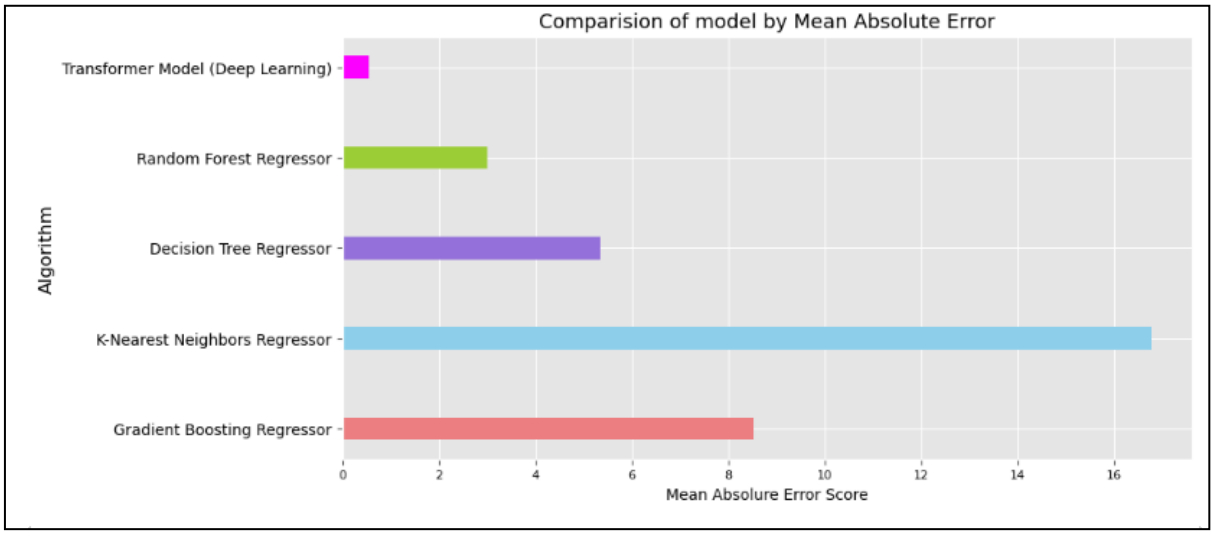
```
print('Comparision of Results btw Machine Learning Models:\n')
score_df = pd.DataFrame(scores, columns=["Model", "MSE", "MAE", "R2"])
score_df
```

	Model	MSE	MAE	R2
0	Gradient Boosting Regressor	114.620528	8.538632	0.999153
1	K-Nearest Neighbors Regressor	594.540303	16.777431	0.995607
2	Decision Tree Regressor	41.174763	5.345243	0.999696
3	Random Forest Regressor	14.604895	3.011707	0.999892
4	Transformer Model (Deep Learning)	1.098629	0.588929	0.999992

According to the findings, the Transformer Model is the best-performing model for predicting solar power, closely followed by the Random Forest and Gradient Boosting Regressors models.

6. Visualization:

- Visualize the results using Matplotlib and Seaborn to create plots that compare actual vs. predicted values, error distributions, and model performance.



7. Run the Jupyter Notebook:

- If using Jupyter Notebook, run each cell sequentially, ensuring that each step executes without errors. Monitor the output of each cell to verify the correctness of the code execution.

8. Save and Document Results:

- Save the results, including model performance metrics and visualizations. Document the findings as per the research study requirements.

This configuration manual provides all the necessary steps and guidelines to set up, execute, and manage the code implementation for optimizing renewable energy management through solar power forecasting. Follow each section carefully to ensure that the research study's objectives are achieved successfully.

References

Python: https://www.python.org
Dataset Source: https://www.kaggle.com/datasets/pythonafroz/solar-power