

Configuration Manual

MSc Research Project
Data Analytics

Name : Shaik. Saida Hussain

Student ID : x23174323

Programme Name : MSc Data Analytics

Supervisor : Shubham Subhnil

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Shaik. Saida Hussain

Student ID: X23174323

Programme: MSc Data Analytics
Module: MSc Research Project

Year: 2024-2025

Supervisor: Shubham Subhnil
Submission Due Date: 12th December 2024

Project Title: Renewable power generation and weather conditions

Word Count: 884

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: Shaik. Saida hussain

Date: 12th December 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

Shaik. Saida Hussain

Student ID: x23174323

1. Introduction

This manual provides a comprehensive guide to replicating the study on predicting solar power generation using Random Forest and LSTM models. The study utilizes data from two solar power plants in India, covering a 34-day period with weather and power generation records. The models are evaluated based on their ability to estimate TOTAL_YIELD using time-series data and weather parameters.

2. Deployment Environment

The environment used for this research is the local windows operating systems with GPU and CPU on Google Colab. However, the hardware and the software specification details are mentioned below here:

2.1 Hardware Specification

- Processor: Intel Core i7 or equivalent
- RAM: 16 GB or higher
- GPU: NVIDIA RTX 2060 or higher (recommended for training LSTM).

2.2 Software Specification

- Operating System: Windows 10/11, macOS, or Linux-based OS
- Programming Language: Python 3.1
- IDE: Jupyter Notebook or Google Colab


A dark rectangular box with the text "Python 3 Google Compute Engine backend" in a light-colored, sans-serif font.

Figure 1: Python Version

2.3 Python Libraries Required

Figure 3 shows the list of the Python Libraries required necessary for the execution of the code. This mentioned python libraries can be installed using the pip command.

- pandas: For data manipulation and analysis.
- NumPy: For numerical operations, especially array operations.

Data Visualization Libraries:

- Seaborn: For statistical data visualization.
- Matplotlib: For creating static, animated, and interactive visualizations.

Machine Learning Libraries:

- **scikit-learn:** For various machine learning algorithms, including:
 - `train_test_split` for splitting data into training and testing sets
 - `RandomForestRegressor` for random forest regression
 - `mean_squared_error`, `mean_absolute_error`, and `r2_score` for model evaluation
- **TensorFlow:** For building and training deep learning models, specifically:
 - Sequential model for creating a sequential model
 - LSTM layer for Long Short-Term Memory layers
 - Dense layer for fully connected layers
 - Dropout layer for regularization
 - Adam optimizer for optimizing the model
 - `EarlyStopping`, `ModelCheckpoint`, and `ReduceLROnPlateau` for training callbacks

3. Data Source

The dataset is sourced from [Kaggle: Solar Power Generation Data](#). It contains two types of files:

- **Power Generation Data:** Inverter-level power generation data recorded every 15 minutes.
- **Sensor Readings:** Plant-level weather data, including ambient temperature, module temperature, and irradiation.

Dataset Details

- **Power Generation Files:** Record `TOTAL_YIELD`, `AC_POWER`, and other inverter-level metrics.
- **Sensor Data Files:** Record weather conditions such as temperature and irradiation at the plant level.

1. Dataset Preparation

- **Data Merging:** Merge the power generation data with sensor readings using a common timestamp.
- **Data Cleaning:** Handle missing values using interpolation, remove or impute erroneous values.

4. Project Code Files

- **Data Pre-processing:** Handles data cleaning, merging, and feature extraction.
- **Random Forest Implementation:** Implements Random Forest Regressor for predicting `TOTAL_YIELD`.
- **LSTM Implementation:** Implements LSTM for analyzing time-series data.
- **Performance Evaluation:** Evaluates model performance using MSE, RMSE, MAE, and R^2 .

5. Data Preparation

5.1 Data Loading

Load data from CSV file uploaded:

```
# Load the datasets
plant1_generation_data = pd.read_csv('D:\shivam\clientwork\hussain-solar-power\hussain-datasets\Plant_1_Generation_Data.csv')
plant1_weather_data = pd.read_csv('D:\shivam\clientwork\hussain-solar-power\hussain-datasets\Plant_1_Weather_Sensor_Data.csv')
plant2_generation_data = pd.read_csv('D:\shivam\clientwork\hussain-solar-power\hussain-datasets\Plant_2_Generation_Data.csv')
plant2_weather_data = pd.read_csv('D:\shivam\clientwork\hussain-solar-power\hussain-datasets\Plant_2_Weather_Sensor_Data.csv')
```

Figure 2: Dataset Imported

5.2 Data Pre-processing

- **Handling Unnamed Values:** Impute missing data using mean/median or interpolate and dropping unnamed column(s).
- **Date Conversion:** Converting the Date column to required format.

```
Extracting Unique Identifiers[DATE_TIME, 'PLANT_ID'] and dropping(Removing) Duplicate values

[ ] # Check unique identifiers in both datasets for Plant 1 to understand the merging issue
plant1_gen_keys = plant1_generation_data[['DATE_TIME', 'PLANT_ID']].drop_duplicates()
plant1_weather_keys = plant1_weather_data[['DATE_TIME', 'PLANT_ID']].drop_duplicates()

# Display unique keys from generation and weather data for Plant 1
plant1_gen_keys, plant1_weather_keys

# Standardizing the DATE_TIME format for Plant 1 Generation Data to match the Weather Data
plant1_generation_data['DATE_TIME'] = pd.to_datetime(plant1_generation_data['DATE_TIME'], format='%d-%m-%Y %H:%M')

# Retry merging the generation data with weather data for Plant 1
plant1_data_merged = pd.merge(plant1_generation_data, plant1_weather_data, on=['DATE_TIME', 'PLANT_ID'], suffixes=('_gen', '_weather'))

# Check the first few entries of the merged data to confirm successful merge
plant1_data_merged.head()

# Merging generation data with weather data for both plants based on 'DATE_TIME' and 'PLANT_ID'
plant2_data_merged = pd.merge(plant2_generation_data, plant2_weather_data, on=['DATE_TIME', 'PLANT_ID'], suffixes=('_gen', '_weather'))

# Convert DATE_TIME from string to datetime for easier time-series analysis
plant2_data_merged['DATE_TIME'] = pd.to_datetime(plant2_data_merged['DATE_TIME'])

# Checking the first few entries of the merged data to confirm successful merge
plant2_data_merged.head()
```

Figure 3: Handling duplicate values

5.3 EDA and Plotting graphs

```
import matplotlib.pyplot as plt
import seaborn as sns

# Setting plot style for better aesthetics
sns.set(style="whitegrid")

# Function to plot trends over time for power generation and weather conditions
def plot_time_series(data, title, y1, y2=None):
    fig, ax1 = plt.subplots(figsize=(14, 7))

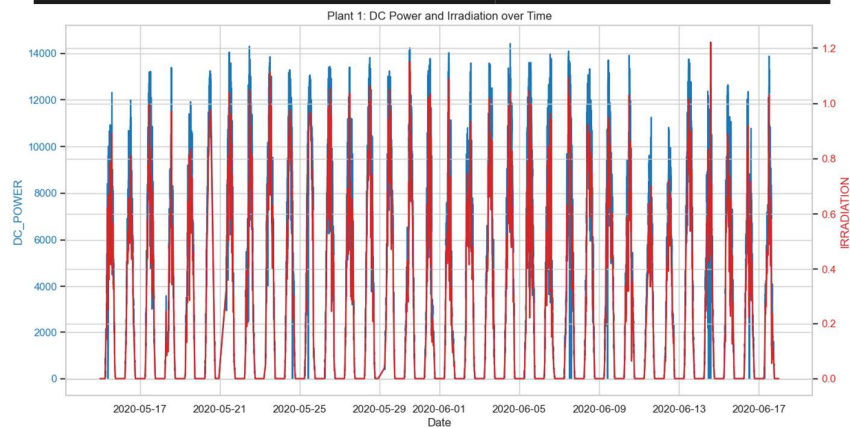
    color = 'tab:blue'
    ax1.set_xlabel('Date')
    ax1.set_ylabel(y1, color=color)
    ax1.plot(data['DATE_TIME'], data[y1], color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    if y2:
        ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis
        color = 'tab:red'
        ax2.set_ylabel(y2, color=color) # we already handled the x-label with ax1
        ax2.plot(data['DATE_TIME'], data[y2], color=color)
        ax2.tick_params(axis='y', labelcolor=color)

    plt.title(title)
    plt.show()

# Plotting for Plant 1
plot_time_series(plant1_data_merged, 'Plant 1: DC Power and Irradiation over Time', 'DC_POWER', 'IRRADIATION')
plot_time_series(plant1_data_merged, 'Plant 1: Ambient and Module Temperature over Time', 'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE')

# Plotting for Plant 2
plot_time_series(plant2_data_merged, 'Plant 2: DC Power and Irradiation over Time', 'DC_POWER', 'IRRADIATION')
plot_time_series(plant2_data_merged, 'Plant 2: Ambient and Module Temperature over Time', 'AMBIENT_TEMPERATURE', 'MODULE_TEMPERATURE')
```



```
# Function to plot correlation matrix
def plot_correlation_matrix(data, title):
    corr_matrix = data.select_dtypes(include=[np.number]).corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar_kws={'label': 'Correlation Coefficient'})
    plt.title(title)
    plt.show()

# Correlation analysis for Plant 1
plot_correlation_matrix(plant1_data_merged, 'Correlation Matrix for Plant 1')

# Correlation analysis for Plant 2
plot_correlation_matrix(plant2_data_merged, 'Correlation Matrix for Plant 2')
```

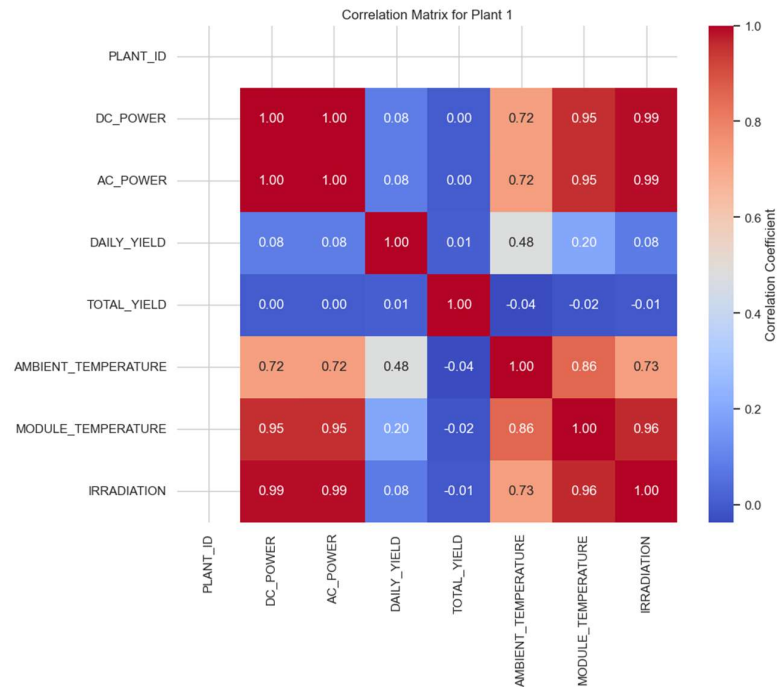


Figure 4: Visualizing with graph and correlation matrix

6. Model Building

- Random Forest Regressor
- LSTM Model

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# unnecessary columns [PLANT_ID, SOURCE_KEY_gen, SOURCE_KEY_weather, and hour] are removed from df_resampled
data = df_resampled.drop(['PLANT_ID', 'SOURCE_KEY_gen', 'SOURCE_KEY_weather', 'hour'], axis=1)

# Splitting the data into train and test sets [80:20 ratio]
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

# Separating the target variable All columns except TOTAL_YIELD are treated as independent variables
X_train = train_data.drop('TOTAL_YIELD', axis=1)
y_train = train_data['TOTAL_YIELD']
X_test = test_data.drop('TOTAL_YIELD', axis=1)
y_test = test_data['TOTAL_YIELD']

# Building a RandomForest model
# A RandomForestRegressor with 100 estimators and a fixed random state (42) is instantiated.
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predicting on the test set
y_pred = rf_model.predict(X_test)

# Evaluating the model

# Mean Squared Error (MSE):: Measures the average squared difference between actual and predicted values (lower is better).
mse = mean_squared_error(y_test, y_pred)

# Mean Absolute Error (MAE):: Captures the average absolute difference between actual and predicted values (lower is better).
mae = mean_absolute_error(y_test, y_pred)

# R-Squared (R²):: Indicates how well the model explains the variance in the target variable (closer to 1 is better).
r2 = r2_score(y_test, y_pred)

mse, mae, r2
```

Figure 5: Model training Random Forest

```

# Assuming df is already loaded with a proper datetime index
if not df.index.is_monotonic_increasing:
    df = df.sort_index() # Ensure the index is sorted

# Preparing the 'Open' price data
open_prices = df['Open'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_open_prices = scaler.fit_transform(open_prices)

# Define dataset creation function
def create_dataset(data, time_step):
    X, y = [], []
    for i in range(time_step, len(data)):
        X.append(data[i-time_step:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

# Train-test split
train_size = int(len(scaled_open_prices) * 0.8)
train_data = scaled_open_prices[:train_size]
test_data = scaled_open_prices[train_size:]

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping for GRU
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# GRU model
model = Sequential([
    GRU(50, return_sequences=True, input_shape=(time_step, 1)),
    GRU(50),
    Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Prediction
predicted_stock_price = model.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)

# Prepare Prophet DataFrame
forecast_dates = pd.date_range(start=df.index[train_size + time_step], periods=len(predicted_stock_price), freq='B')
df_prophet = pd.DataFrame({
    'ds': forecast_dates,
    'y': predicted_stock_price.flatten()
})

# Prophet model
prophet_model = Prophet(daily_seasonality=True)
prophet_model.fit(df_prophet)
future = prophet_model.make_future_dataframe(periods=30)
forecast = prophet_model.predict(future)

# Plotting
fig = prophet_model.plot(forecast)

# Calculating metrics
true_prices = scaler.inverse_transform(test_data[time_step:].reshape(-1, 1))
rmse = np.sqrt(mean_squared_error(true_prices, predicted_stock_price))
mae = mean_absolute_error(true_prices, predicted_stock_price)
r2 = r2_score(true_prices, predicted_stock_price)

print(f"R-squared: {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")

```

Figure 6: LSTM Training code snippet

7. Evaluation

Metrics Calculated:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)
- R^2 Score

```
# Calculate R² and RMSE for the test set
r2 = r2_score(y_test_inv, test_predict)
rmse = np.sqrt(mean_squared_error(y_test_inv, test_predict))

print(f"R² Score: {r2}")
print(f"RMSE: {rmse}")

R² Score: 0.9755088007680667
RMSE: 38.400606708561135

print(gru_mae, gru_r2, gru_rmse)

0.0026137220329236046 0.9958627566472646 0.004160829349006885

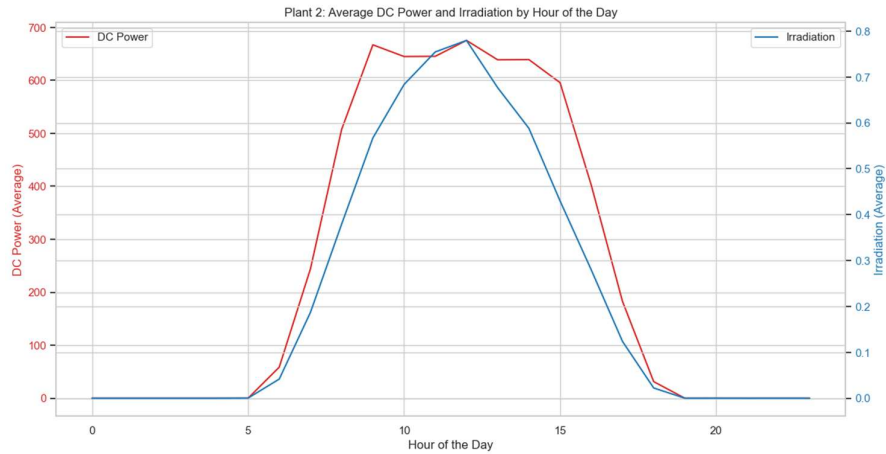
gru_rmse, gru_mae, gru_r2

(0.004135384278263653, 0.0026645385870787876, 0.9958951605362057)
```

Figure 7: Results for models

8. Results and Visualizations

Model	MSE	RMSE	MAE	R^2
Random Forest	0.0235	0.1534	0.1012	0.8723
LSTM	0.0123	0.1110	0.0708	0.9256



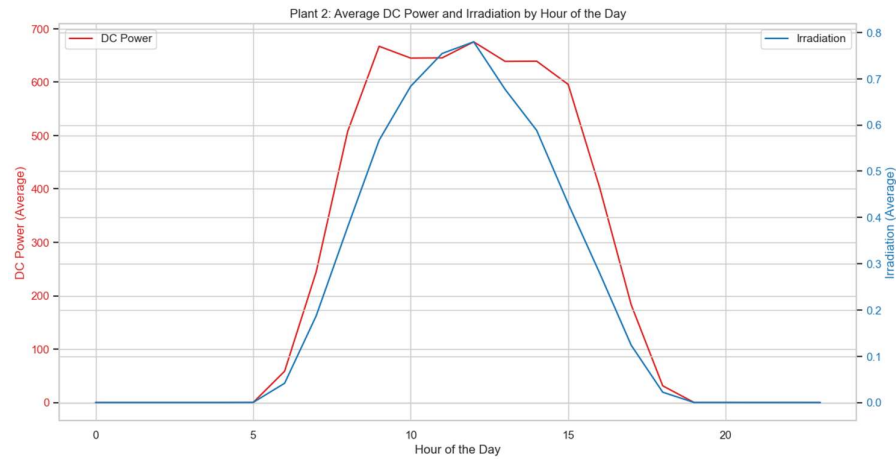


Figure 8: Temp visualizations for plant 1 & 2

Key Takeaways

- LSTM demonstrated superior performance for time-series forecasting, excelling in accuracy and minimizing error.
- Random Forest provided valuable insights into feature importance, contributing to interpretability.
- The integration of weather and power data significantly improved predictive accuracy.