National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Shivani Nandanikar
Student ID: x22197168

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shivani Shrikant Nandanikar……
Student ID:	x22197168……..……
Programme:	MSc. Data Analytics…… **Year:** 2023-2024…..
Module:	MSc. Research Project…….……
Lecturer:	Dr. Catherine Mulwa…….……
Submission Due Date:	12/08/2024
Project Title:	Impact of Weather Conditions on Renewable Energy Consumption
Word Count:	641…… **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: Shivani Shrikant Nandanikar ……

Date: 12 / 08 / 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

Shivani Nandanikar
Student ID: x22197168

1 Introduction

This configuration manual gives detailed information of the set-up necessary for the environment, software, and hardware used to reproduce the results of the research project. The title of the research is 'Impact of Weather Conditions on Renewable Energy Consumption.' This configuration manual gives detailed descriptions of the software installations, system information, and specific configurations used throughout the research project.

2 Software and hardware requirements

The information of the software used for the implementation of the project are as follows:

Data Collection

Platform from where the dataset is collected – Kaggle website

Link of dataset –

https://www.kaggle.com/code/abhinavdogra002/renewable-energy-and-weather-conditions/input

Environment set-up for implementation

- **Data pre-processing, modelling, evaluation, data visualizations:** Jupyter Notebook
- **Programming language used:** Python
- **Python version used:** Python 3.11.9

Other tools used

- Microsoft Excel
- Lucid chart website
- I Love Pdf website
- Zotero

Hardware requirements of the project are as follows:

- **Operating System:** Microsoft Windows 11 Home Single Language
- **Version:** 10.0.22621 Build 22621
- **System Manufacturer:** LENOVO
- **System Model:** 81W8
- **Processor:** Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1190 Mhz 4 Core(s) 8 Logical Processor(s)
- **RAM:** 8.00 GB
- **System Type:** x64-based PC
- **BIOS Version/Date:** LENOVO DKCN54WW 27-01-2022
- **Disk:** 512 GB SSD

3 Implementation

Creation of Virtual Environment

1. Install Jupyter Notebook

2. Open the application

3. Create a new Jupyter notebook

4. Install the required libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline

from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error

from sklearn.model_selection import TimeSeriesSplit
from arch import arch_model

from statsmodels.tsa.vector_ar.var_model import VAR
```

5. Install the required package

   pip install arch

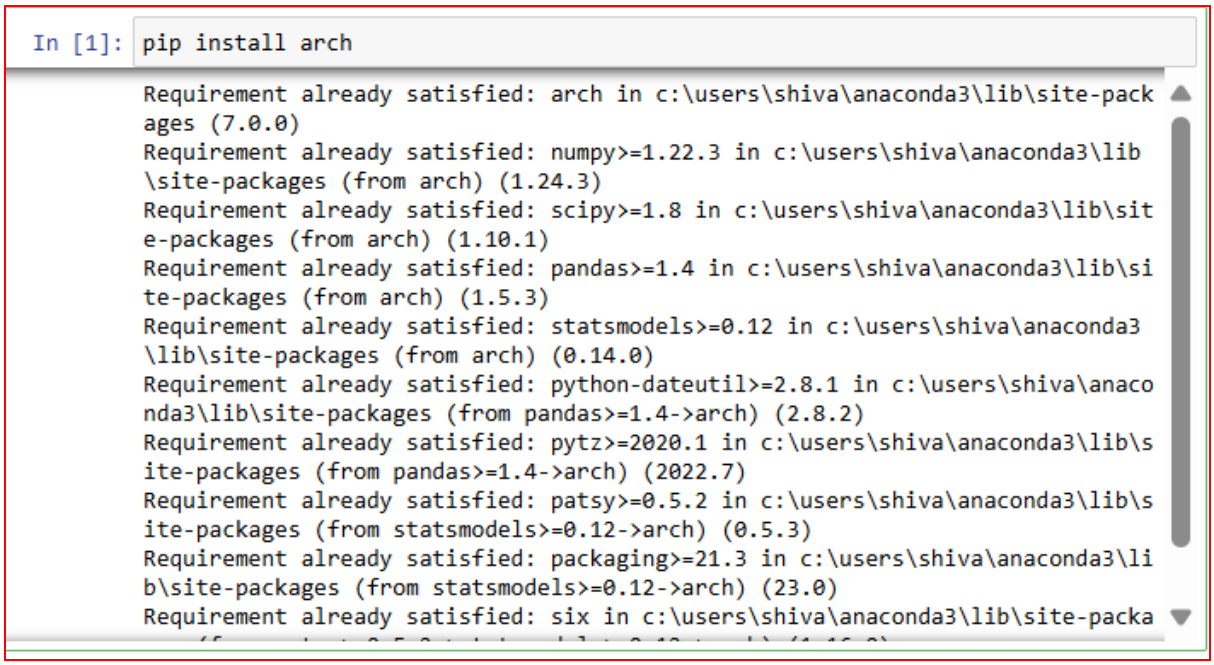

Figure 1: Installation of 'arch' Package

Main implementation and visualizations

I. Data Cleaning process

- Initially recognize and handle the missing values and outliers if present
- Thereafter ensure that they are handled and correctly

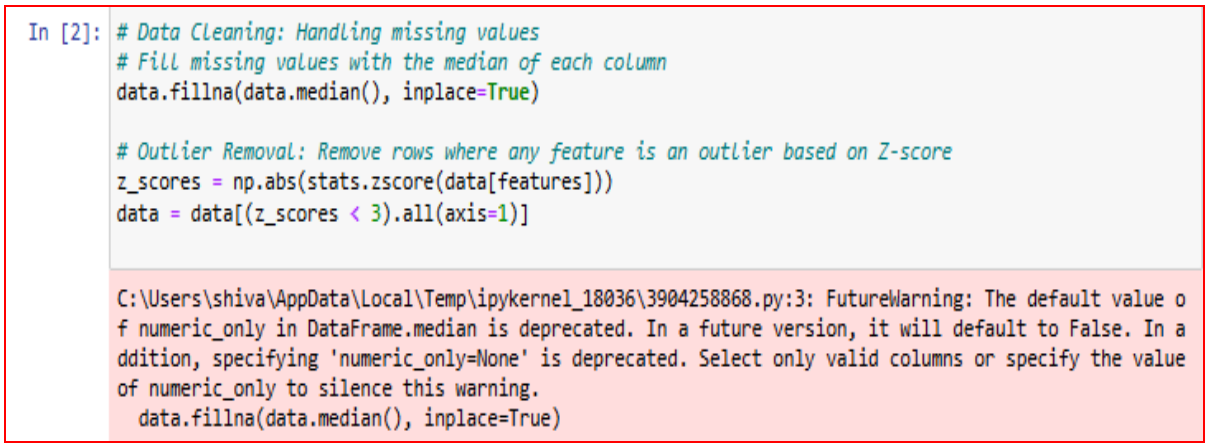

Figure 2: Data Cleaning

II. Standardize the features

- Standardization of the selected features is performed to ensure that all the selected features equally contribute to the analysis.
- It also ensures that the performance of the respective ML model is consistent.

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 3: Standardization of selected features

III. Data Splitting

- Split the collected data into training data and testing data.
- Split the dataset into 80% training and 20% testing data respectively.
- Give a random value of 42 for reproducibility.

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.2, random_state=42)
```

Figure 4: Data Splitting

IV. Modelling

1. Modelling of Polynomial Regression, Random Forest Algorithm, and Gradient Boosting ML Algorithms

- Create the polynomial regression model
- Then train the model
- Perform the predictions
- Calculate evaluation metrics
- Perform cross-validation to avoid model overfitting
- Fit the Random Forest and Gradient Boosting algorithms
- Calculate the evaluation metrics for both the algorithms
- Plot the comparison of the model predictions

```
# Create and train the polynomial regression model
poly_reg = LinearRegression()
poly_reg.fit(X_train_scaled, y_train)

# Predictions
y_train_pred = poly_reg.predict(X_train_scaled)
y_test_pred = poly_reg.predict(X_test_scaled)

# Evaluation
train_mse = mean_squared_error(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print("Polynomial Regression Model")
print("Train MSE:", train_mse)
print("Test MSE:", test_mse)
print("Train R2:", train_r2)
print("Test R2:", test_r2)

# Cross-validation
cross_val_scores = cross_val_score(poly_reg, X_train_scaled, y_train, cv=5, scoring='r2')
print("Cross-validated R2 scores:", cross_val_scores)
print("Average Cross-validated R2 score:", cross_val_scores.mean())

# Advanced models: Random Forest and Gradient Boosting
# Random Forest
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train_scaled, y_train)
y_test_pred_rf = rf_reg.predict(X_test_scaled)
rf_test_r2 = r2_score(y_test, y_test_pred_rf)
print("Random Forest Test R2:", rf_test_r2)

# Gradient Boosting
gb_reg = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_reg.fit(X_train_scaled, y_train)
y_test_pred_gb = gb_reg.predict(X_test_scaled)
gb_test_r2 = r2_score(y_test, y_test_pred_gb)
print("Gradient Boosting Test R2:", gb_test_r2)

# Plot comparison
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_test_pred_rf, c='green', marker='o', edgecolor='white', label='Random Forest')
plt.scatter(y_test, y_test_pred_gb, c='orange', marker='o', edgecolor='white', label='Gradient Boosting')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='black', lw=2, linestyle='--')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Comparison of Model Predictions')
plt.legend()
plt.show()
```

Figure 5: Polynomian regression, Random Forest and Gradient Boosting modelling

2. Modelling of ARIMA model

- Put ARIMA Model to yearly, monthly, and hourly data
- Plot yearly, monthly, and hourly consumption patterns
- Also check the Stationarity of the respective data
- Perform Decomposition on the respective data
- Perform the forecasting
- Plot the forecast
- Calculate evaluation metrics

```python
# Plot hourly consumption patterns
hourly_data.plot(title='Hourly Energy Consumption', figsize=(12, 6))
plt.xlabel('Hour')
plt.ylabel('Energy delta [Wh]')
plt.show()

# Decomposition for hourly data
decomposition_hourly = seasonal_decompose(hourly_data, model='additive')
fig = decomposition_hourly.plot()
fig.suptitle('Hourly Decomposition')
plt.subplots_adjust(top=0.9)
plt.show()

#Check stationarity
check_stationarity(hourly_data)
hourly_data_diff = hourly_data.diff().dropna()
check_stationarity(hourly_data_diff)

# Clean weather data for hourly (handling missing values)
weather_data_hourly = data[weather_columns].resample('H').mean()''
weather_data_hourly = weather_data_hourly.fillna(method='ffill').fillna(method='bfill')  # Fill NaNs

# Align the lengths of hourly_data and weather_data_hourly
hourly_data, weather_data_hourly = hourly_data.align(weather_data_hourly, join='inner')

model_hourly = SARIMAX(hourly_data, order=(1, 1, 1), exog=weather_data_hourly)
results_hourly = model_hourly.fit()
print(results_hourly.summary())

# Plotting the forecast for hourly energy consumption

forecast_steps_hourly = 24  # Forecast for the next 24 hours
exog_forecast_hourly = weather_data_hourly[-forecast_steps_hourly:]
forecast_hourly = results_hourly.get_forecast(steps=forecast_steps_hourly, exog=exog_forecast_hourly)
forecast_df_hourly = forecast_hourly.summary_frame()

plt.figure(figsize=(12, 6))
plt.plot(hourly_data, label='Actual')
plt.plot(forecast_df_hourly['mean'], label='Forecast', linestyle='--')
plt.fill_between(forecast_df_hourly.index, forecast_df_hourly['mean_ci_lower'], forecast_df_hourly['mean_ci_upper'], color='gray
plt.title('ARIMAX Forecast for Hourly Energy Consumption')
plt.xlabel('Time')
plt.ylabel('Energy delta [Wh]')
plt.legend()
plt.show()

# Model evaluation for hourly data

actual_hourly = hourly_data[-forecast_steps_hourly:]
predicted_hourly = forecast_df_hourly['mean']

# Calculate evaluation metrics
mse_hourly = mean_squared_error(actual_hourly, predicted_hourly)
rmse_hourly = np.sqrt(mse_hourly)
mae_hourly = mean_absolute_error(actual_hourly, predicted_hourly)
mape_hourly = np.mean(np.abs((actual_hourly - predicted_hourly) / actual_hourly)) * 100

# Print the metrics
print(f'Mean Squared Error (MSE) for Hourly Data: {mse_hourly}')
print(f'Root Mean Squared Error (RMSE) for Hourly Data: {rmse_hourly}')
print(f'Mean Absolute Error (MAE) for Hourly Data: {mae_hourly}')
print(f'Mean Absolute Percentage Error (MAPE) for Hourly Data: {mape_hourly:.2f}%')
```

Figure 6: ARIMA model for Hourly data (Similarly for Monthly and Yearly data)

3. Modelling of SARIMA + GARCH model

- Initially install the ‘arch’ package
- Check for stationarity of the data
- Perform differencing if necessary
- Fit the SARIMA model to the data
- Fit GARCH model to residuals of SARIMA model
- Final SARIMA model fitting to entire data
- Plot the forecast
- Calculate evaluation metrics

```
pip install arch

# CHECKING FOR STATIONARITY AND PERFORMING THE DIFFERENCING IF REQUIRED

# Check stationarity
def check_stationarity(timeseries):
    result = adfuller(timeseries)
    print('ADF Statistic: %f' % result[0])
    print('p-value: %f' % result[1])
    for key, value in result[4].items():
        print(f'Critical Values:\n   {key}, {value}')

check_stationarity(hourly_data)

# Differencing to make data stationary
hourly_data_diff = hourly_data.diff().dropna()
check_stationarity(hourly_data_diff)

# FIT SARIMA MODEL TO HOURLY DATA

# Initial SARIMA model fitting
model = SARIMAX(hourly_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 24), exog=weather_data_hourly)
results = model.fit()
print(results.summary())

# Fit GARCH model to the residuals of the SARIMA model
residuals = results.resid
garch_model = arch_model(residuals, vol='Garch', p=1, q=1)
garch_results = garch_model.fit(disp='off')
print(garch_results.summary())

# Check if the GARCH model has improved the residuals
garch_residuals = garch_results.resid
```

```
# FINAL MODEL FITTING

# Fit final SARIMA model on entire dataset
final_model = SARIMAX(hourly_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 24), exog=weather_data_hourly)
final_results = final_model.fit()
print(final_results.summary())


# Forecasting
forecast_steps = 24  # Forecast for the next 24 hours
forecast = final_results.get_forecast(steps=forecast_steps, exog=weather_data_hourly[-forecast_steps:])
forecast_df = forecast.summary_frame()

# Plotting the forecast
plt.figure(figsize=(12, 6))
plt.plot(hourly_data, label='Actual')
plt.plot(forecast_df['mean'], label='Forecast', linestyle='--')
plt.fill_between(forecast_df.index, forecast_df['mean_ci_lower'], forecast_df['mean_ci_upper'], color='gray', alpha=0.3)
plt.title('SARIMA Forecast for Hourly Energy Consumption')
plt.xlabel('Time')
plt.ylabel('Energy delta [Wh]')
plt.legend()
plt.show()

# Model evaluation
actual = hourly_data[-forecast_steps:]
predicted = forecast_df['mean']

# Calculate evaluation metrics
mse = mean_squared_error(actual, predicted)
rmse = np.sqrt(mse)
mae = mean_absolute_error(actual, predicted)
mape = np.mean(np.abs((actual - predicted) / actual)) * 100

# Print the metrics
print(f'Mean Squared Error (MSE) for Hourly Data: {mse}')
print(f'Root Mean Squared Error (RMSE) for Hourly Data: {rmse}')
print(f'Mean Absolute Error (MAE) for Hourly Data: {mae}')
print(f'Mean Absolute Percentage Error (MAPE) for Hourly Data: {mape:.2f}%')
```

Figure 7: Modelling of SARIMA + GARCH model

4. Modelling of VAR Model:

- Load the data
- Select relevant columns for VAR model
- Check for stationarity
- Fit VAR model to the data and on the training data
- Plot the forecast
- Calculate the evaluation metrics

```
# LOAD AND PREPROCESSING OF DATA

# Load data
data = pd.read_csv('solar_weather.csv')

# Convert 'Time' column to datetime
data['Time'] = pd.to_datetime(data['Time'], format='%Y-%m-%d %H:%M:%S')
# Set 'Time' as index
data.set_index('Time', inplace=True)

# Select relevant columns for VAR model (e.g., solar, wind energy consumption, and weather conditions)
data_var = data[['isSun', 'wind_speed', 'temp', 'humidity', 'pressure', 'rain_1h']]

# Check for missing values and handle them
data_var = data_var.fillna(method='ffill').fillna(method='bfill')

# Check stationarity
def check_stationarity(timeseries, name):
    result = adfuller(timeseries)
    print(f'ADF Statistic for {name}: {result[0]}')
    print(f'p-value for {name}: {result[1]}')
    for key, value in result[4].items():
        print(f'Critical Values for {name}:\n   {key}, {value}')

for column in data_var.columns:
    check_stationarity(data_var[column], column)

# Differencing to make data stationary if necessary
data_var_diff = data_var.diff().dropna()

for column in data_var_diff.columns:
    check_stationarity(data_var_diff[column], column)
```

```
 # Fit the VAR model
# Split the data into training and testing sets
n_train = int(len(data_var_diff) * 0.8)
train, test = data_var_diff[:n_train], data_var_diff[n_train:]

# Fit VAR model on the training data
model = VAR(train)
lag_order = model.select_order(maxlags=15)
print(lag_order.summary())

# Forecasting
forecast_steps = len(test)
forecast = var_model.forecast(train.values[-var_model.k_ar:], steps=forecast_steps)
forecast_df = pd.DataFrame(forecast, index=test.index, columns=test.columns)

# Plotting the forecast
variables = ['isSun', 'wind_speed']
for var in variables:
    plt.figure(figsize=(12, 6))
    plt.plot(data_var_diff.index, data_var_diff[var], label=f'Actual {var}')
    plt.plot(forecast_df.index, forecast_df[var], label=f'Forecast {var}', linestyle='--')
    plt.title(f'VAR Forecast for {var}')
    plt.xlabel('Time')
    plt.ylabel(f'{var} delta [Wh]')
    plt.legend()
    plt.show()

# Model evaluation
mse_results = {}
for var in variables:
    mse = mean_squared_error(test[var], forecast_df[var])
    mse_results[var] = mse
    print(f'Mean Squared Error for {var}: {mse}')
```

Figure 8: Modelling of VAR model