National College of Ireland

# Forecasting of Power plants consumption using Machine Learning Techniques

MSc Research Project
Data Analytics


Mohit Kaushal Jain
Student ID: X21191514

School of Computing
National College of Ireland


Supervisor: Vladimir Milosavljevic

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Mohit Kaushal Jain |
| **Student ID:** | X21191514 |
| **Programme:** | Data Analytics |
| **Year:** | 2023 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Vladimir Milosavljevic |
| **Submission Due Date:** | 14/08/2023 |
| **Project Title:** | Forecasting of Power plants consumption using Machine Learning Techniques |
| **Word Count:** | 995 |
| **Page Count:** | 13 |

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:** | Mohit Kaushal Jain |
| **Date:** | 17th September 2023 |

## 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): | |

# Forecasting of Power plants consumption using Machine Learning Techniques

Mohit Kaushal Jain
X21191514

# 1 Introduction

The configeration manual provides a brief information about the hardware and software requirements for this research. It will also talk about the step by step approach to complete the research project successfully. The following manual is broken into different section for the purpose of information.

# 2 System Requirement

## 2.1 Hardware Requirements

1. **Model Name**: MacBook Air 2020
2. **Operating System**: macOS Ventura 13.4
3. **Processor**: M1 chip
4. **Memory**: 8 GB RAM Storage

## 2.2 Software Requirements

Python language was used for programming. A 3.9.12 Python version was installed from python official website[1]. A jupyter Notebook of 6.4.12 was installed by following instruction from official website of jupyter.[2]. The "jupyter notebook" command is written in terminal to start the notebook as shown in figure 1

# 3 Importing Libraries

There are certain libraries that may be not be installed by default in order to install a library, one needs to use 'pip' command. For example if numpy needs to be installed then the syntax would be "!pip install numpy" in jupyter notebok. The figure 2 shows the commands mentioned for importing libraries. Together, these libraries offer the functionalities needed to effectively handle time series data.

[1]https://www.python.org/downloads/
[2]https://jupyter.org/install

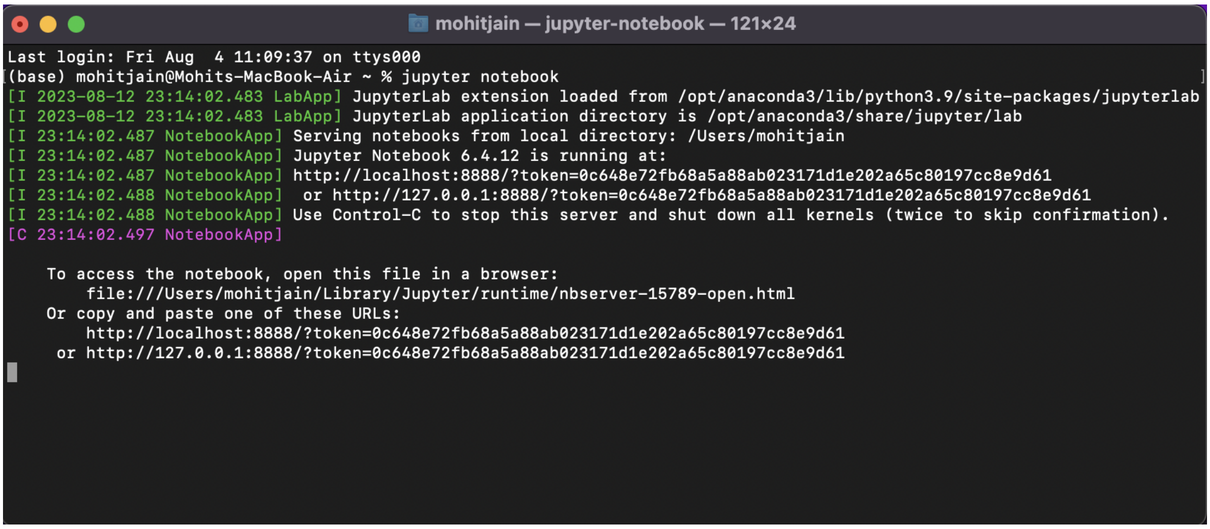


Figure 1: Terminal for Starting Jupyter Notebook

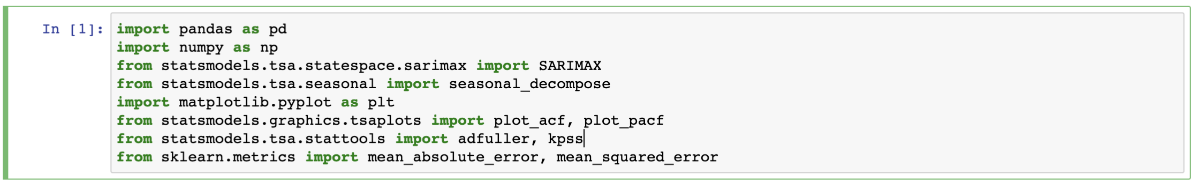


Figure 2: Importing libraries

# 4 Dataset Exploration

The dataset was taken from Kaggle and it contains 52,416 rows and 9 columns [3]. The figure 3 shows the output for first five and last 5 rows of the dataset that was loaded in jupyter notebook.

```
In [2]: df = pd.read_csv('/Users/mohitjain/Desktop/Tetuan City power consumption.csv')

In [3]: df.head()
Out[3]:
```

| | DateTime | Temperature | Humidity | Wind Speed | general diffuse flows | diffuse flows | Zone 1 Power Consumption | Zone 2 Power Consumption | Zone 3 Power Consumption |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1/1/2017 0:00 | 6.559 | 73.8 | 0.083 | 0.051 | 0.119 | 34055.69620 | 16128.87538 | 20240.96386 |
| 1 | 1/1/2017 0:10 | 6.414 | 74.5 | 0.083 | 0.070 | 0.085 | 29814.68354 | 19375.07599 | 20131.08434 |
| 2 | 1/1/2017 0:20 | 6.313 | 74.5 | 0.080 | 0.062 | 0.100 | 29128.10127 | 19006.68693 | 19668.43373 |
| 3 | 1/1/2017 0:30 | 6.121 | 75.0 | 0.083 | 0.091 | 0.096 | 28228.86076 | 18361.09422 | 18899.27711 |
| 4 | 1/1/2017 0:40 | 5.921 | 75.7 | 0.081 | 0.048 | 0.085 | 27335.69620 | 17872.34043 | 18442.40964 |

```
In [4]: df.tail()
Out[4]:
```

| | DateTime | Temperature | Humidity | Wind Speed | general diffuse flows | diffuse flows | Zone 1 Power Consumption | Zone 2 Power Consumption | Zone 3 Power Consumption |
|---|---|---|---|---|---|---|---|---|---|
| 52411 | 12/30/2017 23:10 | 7.010 | 72.4 | 0.080 | 0.040 | 0.096 | 31160.45627 | 26857.31820 | 14780.31212 |
| 52412 | 12/30/2017 23:20 | 6.947 | 72.6 | 0.082 | 0.051 | 0.093 | 30430.41825 | 26124.57809 | 14428.81152 |
| 52413 | 12/30/2017 23:30 | 6.900 | 72.8 | 0.086 | 0.084 | 0.074 | 29590.87452 | 25277.69254 | 13806.48259 |
| 52414 | 12/30/2017 23:40 | 6.758 | 73.0 | 0.080 | 0.066 | 0.089 | 28958.17490 | 24692.23688 | 13512.60504 |
| 52415 | 12/30/2017 23:50 | 6.580 | 74.1 | 0.081 | 0.062 | 0.111 | 28349.80989 | 24055.23167 | 13345.49820 |

Figure 3: Importing Dataset

# 5 Exploratory Data Analysis

To understand the data in a better way Exploratory Data Analysis was carried. With "df.info()" as shown in figure 4. In fig 4 it is observed that the datetime data type is in object so it was converted into datetime object as show in figure 5.

```
In [5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52416 entries, 0 to 52415
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   DateTime                   52416 non-null  object
 1   Temperature                52416 non-null  float64
 2   Humidity                   52416 non-null  float64
 3   Wind Speed                 52416 non-null  float64
 4   general diffuse flows      52416 non-null  float64
 5   diffuse flows              52416 non-null  float64
 6   Zone 1 Power Consumption   52416 non-null  float64
 7   Zone 2  Power Consumption  52416 non-null  float64
 8   Zone 3  Power Consumption  52416 non-null  float64
dtypes: float64(8), object(1)
memory usage: 3.6+ MB
```

Figure 4: Exploratory Data Analysis.

[3]https://www.kaggle.com/datasets/ashkanforootan/tetuan-city-power-consumption

```
In [6]: df['DateTime'] = pd.to_datetime(df['DateTime'])
        df.set_index('DateTime', inplace=True)
```

Figure 5: Datetime conversion

## 5.1 Time Series Modelling

The code in figure 6 shows plotting time series model using for loop as the dataset was huge. Here, 'rows_per_subplot' will determine how many data points will be plotted in each subplot. The 'num_subplots' will calculate the total number of subplots needed based on the length of the DataFrame ('df') and the desired number of rows per subplot. After that subplots were created using for loop with start and end index for selecting the data points for plotting.

```
# Define the number of rows you want to plot in each subplot
rows_per_subplot = 1000

# Calculate the number of subplots needed
num_subplots = int(np.ceil(len(df) / rows_per_subplot))

# Create subplots
fig, axs = plt.subplots(num_subplots, 1, figsize=(12, num_subplots * 6))

# Loop through the data and create subplots
for i in range(num_subplots):
    start_idx = i * rows_per_subplot
    end_idx = min((i + 1) * rows_per_subplot, len(df))

    # Plot the selected rows
    axs[i].plot(df.index[start_idx:end_idx], df['Zone 1 Power Consumption'][start_idx:end_idx], label='Zone 1')
    axs[i].plot(df.index[start_idx:end_idx], df['Zone 2  Power Consumption'][start_idx:end_idx], label='Zone 2')
    axs[i].plot(df.index[start_idx:end_idx], df['Zone 3  Power Consumption'][start_idx:end_idx], label='Zone 3')
    axs[i].set_xlabel('DateTime')
    axs[i].set_ylabel('Power Consumption')
    axs[i].set_title(f'Subplot {i+1}')
    axs[i].legend()

# Adjust layout to prevent overlapping labels and titles
plt.tight_layout()

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

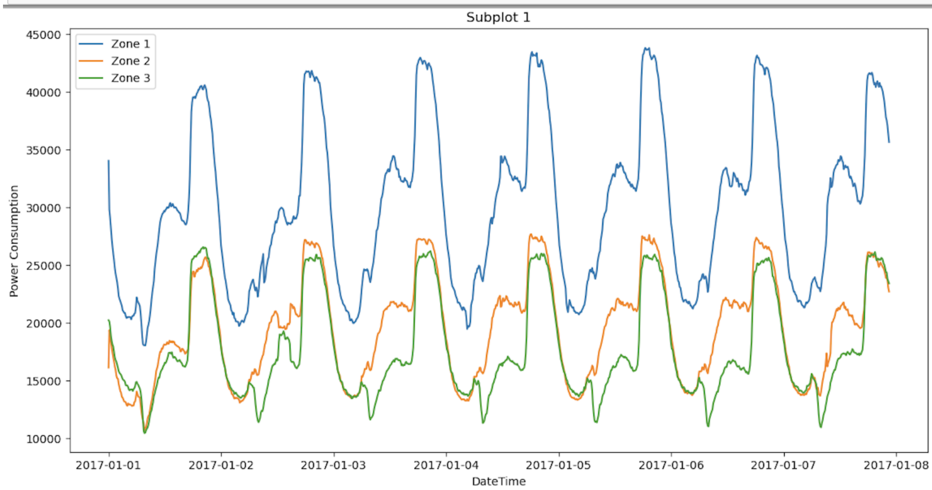


Figure 6: Time Series Modelling

## 5.2 Time Series Decomposition

The code in figure 7 shows the additive decomposition of the time series. The seasonality frequency in the dataset is defined by the period variable. The value was set to 6 for the analysis. The 'seasonal_decompose' function was used to decompose the time series into its components and the model was set to additive.

```
# Create subplots
fig, axs = plt.subplots(num_subplots, 1, figsize=(12, num_subplots * 6))
plt.subplots_adjust(hspace=0.5)
# Specify the frequency of seasonality (assuming 6 timestamps per hour in the dataset)
period = 6
# Loop through the data and create subplots
for i in range(num_subplots):
    start_idx = i * rows_per_subplot
    end_idx = min((i + 1) * rows_per_subplot, len(df))

    # Perform additive decomposition on the current chunk of data
    result = seasonal_decompose(df['Zone 1 Power Consumption'].iloc[start_idx:end_idx],
                                model='additive', period = period)

    # Get the trend, seasonal, and residual components
    trend = result.trend
    seasonal = result.seasonal
    residual = result.resid

    # Plot the decomposed components in the current subplot
    axs[i].plot(df.index[start_idx:end_idx], df['Zone 1 Power Consumption'].iloc[start_idx:end_idx]
            , label='Original Data')
    axs[i].plot(trend.index, trend, label='Trend', color='orange')
    axs[i].plot(seasonal.index, seasonal, label='Seasonality', color='green')
    axs[i].plot(residual.index, residual, label='Residuals', color='red')
    axs[i].set_ylabel('Power Consumption')
    axs[i].set_title(f'Subplot {i+1}')
    axs[i].legend()

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

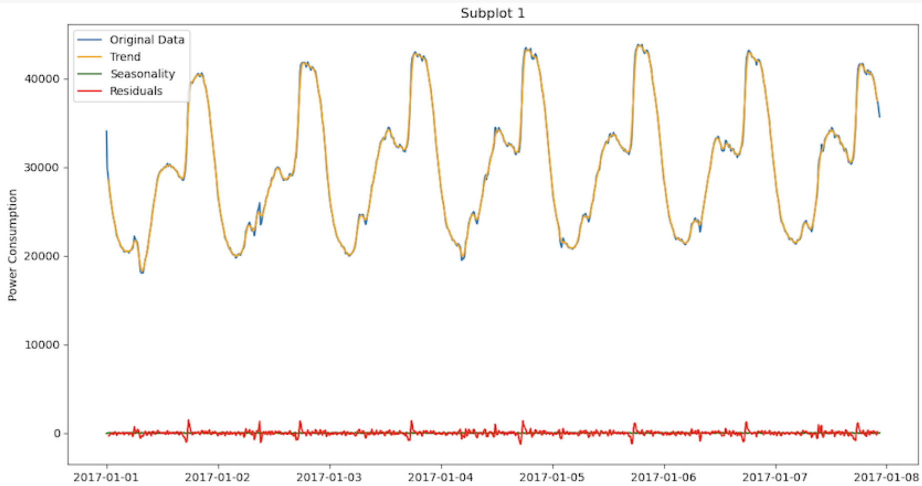


Figure 7: Additive time series decomposition for zone 1

## 5.3 Plot for temperature, Humidity and zone wise energy consumption

The code in figure 8 is about grouping time series data by month and then plotting the average temperature over each month. The "resample('M')" function is used to group the data by months ('M' represents monthly frequency). Finally the mean value is taken for resampled months. The figure 8 shows a line plot for temperature. The figure 9 creates a line graph to visualize the average humidity over each month. The figure 10 creates a line graph for zone wise energy consumption in different colours.

```
# Group the data by month and compute the mean for each month
monthly_data = df.resample('M').mean()
```

```
#Plot Temperature
#plt.plot(df['Temperature'], label='Temperature', color='tab:red')
plt.plot(monthly_data['Temperature'], label='Temperature', color='tab:red')
plt.title("Monthly plot for Temperature")
```

```
Text(0.5, 1.0, 'Monthly plot for Temperature')
```

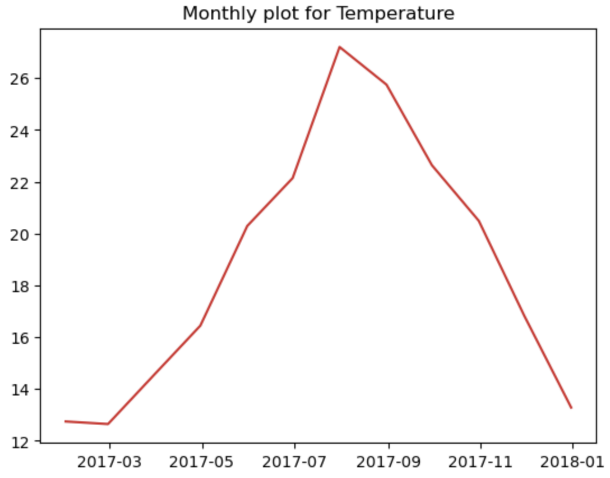


Figure 8: Plotting of Temperature

```
# Plot Humidity
#plt.plot(df['Humidity'], label='Humidity', color='tab:blue')
plt.plot(monthly_data['Humidity'], label='Humidity', color='tab:blue')
plt.title("Monthly plot for Humidity")
```

```
Text(0.5, 1.0, 'Monthly plot for Humidity')
```

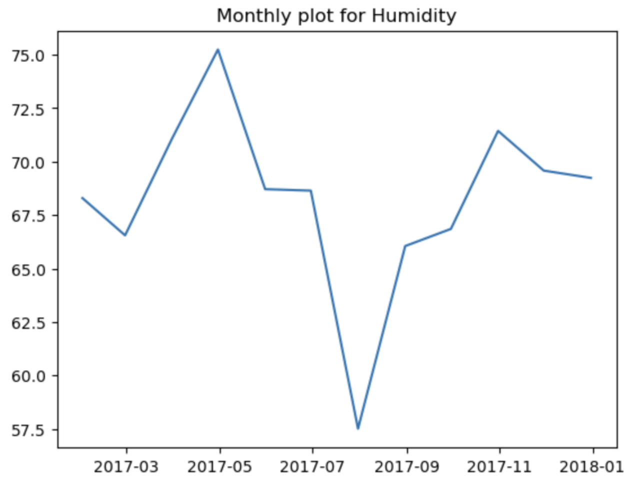


Figure 9: Plotting of Humidity

```
# Plot Power Consumption for each zone
#plt.plot(df['Zone 1 Power Consumption'], label='Zone 1 Power Consumption', color='tab:orange')
plt.plot(monthly_data['Zone 1 Power Consumption'], label='Zone 1 Power Consumption', color='tab:orange')
#plt.plot(df['Zone 2  Power Consumption'], label='Zone 2 Power Consumption', color='tab:green')
plt.plot(monthly_data['Zone 2  Power Consumption'], label='Zone 2 Power Consumption', color='tab:green')
#plt.plot(df['Zone 3  Power Consumption'], label='Zone 3 Power Consumption', color='tab:purple')
plt.plot(monthly_data['Zone 3  Power Consumption'], label='Zone 3 Power Consumption', color='tab:purple')
plt.title("Zone wise monthly power consumption")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f946846bd90>
```

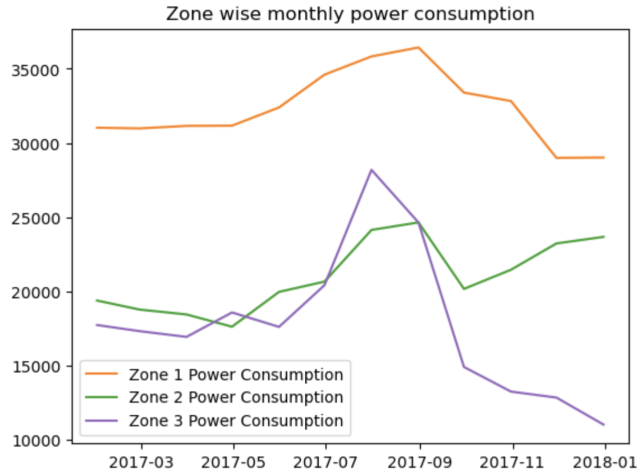


Figure 10: Plotting of Zone wise energy consumption

# 6 Assumption tests

The assumption tests that were performed were related to autocorrelations. The names of various time series variables (such as power consumption for each zone, humidity, and temperature) that are significant for the analysis are listed in the variables list. The variables list includes the names of various time series variables that are relevant for the analysis, such as power consumption for each zone, humidity, and temperature.

After that Augmented Dickey Fuller(ADF) test is performed. The figure 11 shows the code and results of ADF test. The 'result[0]' represents the calculated ADF statistic, 'result[1]' provides the p-value associated with the test and 'result[4]' gives critical values at different significance levels.

The KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test is used in the code segment to evaluate the stationarity of various time series variables. The figure 12 shows the results of KPSS test. The 'kpss_result[0]' represents the calculated KPSS statistic, kpss_result[1] provides the p-value associated with the test, and kpss_result[3] gives critical values at different significance levels.

The program is aimed at generating a pair of visualizations, namely the AutoCorrelation Function (ACF) visualization and the Partial AutoCorrelation Function (PACF) visualization, with the purpose of evaluating the autocorrelation present in a temporal sequence. The figure 13 shows the results of ACF and PACF graphs.

```python
# Select the time series data for each variable
variables = ['Zone 1 Power Consumption'
             ,'Zone 2  Power Consumption'
             , 'Zone 3  Power Consumption',
             'Humidity', 'Temperature']

for variable in variables:
    time_series_data = df[variable]

    # Perform the Augmented Dickey-Fuller test
    result = adfuller(time_series_data)

    # Print the ADF test results for each variable
    print(f'ADF Statistic for {variable}:', result[0])
    print(f'p-value for {variable}:', result[1])
    print(f'Critical Values for {variable}:', result[4])
    print('\n')
```

```
ADF Statistic for Zone 1 Power Consumption: -32.12127853462614
p-value for Zone 1 Power Consumption: 0.0
Critical Values for Zone 1 Power Consumption: {'1%': -3.4304749044184266, '5%': -2.8615952052
42518, '10%': -2.566799383915253}


ADF Statistic for Zone 2  Power Consumption: -25.22216377100368
p-value for Zone 2  Power Consumption: 0.0
Critical Values for Zone 2  Power Consumption: {'1%': -3.4304749044184266, '5%': -2.861595205
242518, '10%': -2.566799383915253}


ADF Statistic for Zone 3  Power Consumption: -16.36686797515679
p-value for Zone 3  Power Consumption: 2.835133086903964e-29
Critical Values for Zone 3  Power Consumption: {'1%': -3.4304749044184266, '5%': -2.861595205
242518, '10%': -2.566799383915253}


ADF Statistic for Humidity: -17.184247931293186
p-value for Humidity: 6.616075836617727e-30
Critical Values for Humidity: {'1%': -3.4304749044184266, '5%': -2.861595205242518, '10%': -
2.566799383915253}


ADF Statistic for Temperature: -9.459827585705547
p-value for Temperature: 4.384185727809652e-16
Critical Values for Temperature: {'1%': -3.4304749044184266, '5%': -2.861595205242518, '10%':
-2.566799383915253}
```

Figure 11: ADF test

```python
for variable in variables:
    time_series_data = df[variable]

    # Perform the KPSS test
    kpss_result = kpss(time_series_data)

    # Print the KPSS test results for each variable
    print(f'KPSS Statistic for {variable}:', kpss_result[0])
    print(f'p-value for {variable}:', kpss_result[1])
    print(f'Critical Values for {variable}:', kpss_result[3])
    print('\n')
```

```
KPSS Statistic for Zone 1 Power Consumption: 6.018599702888408
p-value for Zone 1 Power Consumption: 0.01
Critical Values for Zone 1 Power Consumption: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574,
'1%': 0.739}


KPSS Statistic for Zone 2  Power Consumption: 16.860093845930926
p-value for Zone 2  Power Consumption: 0.01
Critical Values for Zone 2  Power Consumption: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574,
'1%': 0.739}


KPSS Statistic for Zone 3  Power Consumption: 8.92788914353952
p-value for Zone 3  Power Consumption: 0.01
Critical Values for Zone 3  Power Consumption: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574,
'1%': 0.739}


KPSS Statistic for Humidity: 0.5777992157502823
p-value for Humidity: 0.024654616749974337
Critical Values for Humidity: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}


KPSS Statistic for Temperature: 11.643586215500964
p-value for Temperature: 0.01
Critical Values for Temperature: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
```

Figure 12: ADF test

```
# Plot ACF and PACF plots to assess autocorrelation
fig, axes = plt.subplots(1, 2, figsize=(10, 4))
plot_acf(time_series_data, ax=axes[0], lags=30)
plot_pacf(time_series_data, ax=axes[1], lags=30)
plt.tight_layout()
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'y
w' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walk
er ('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```

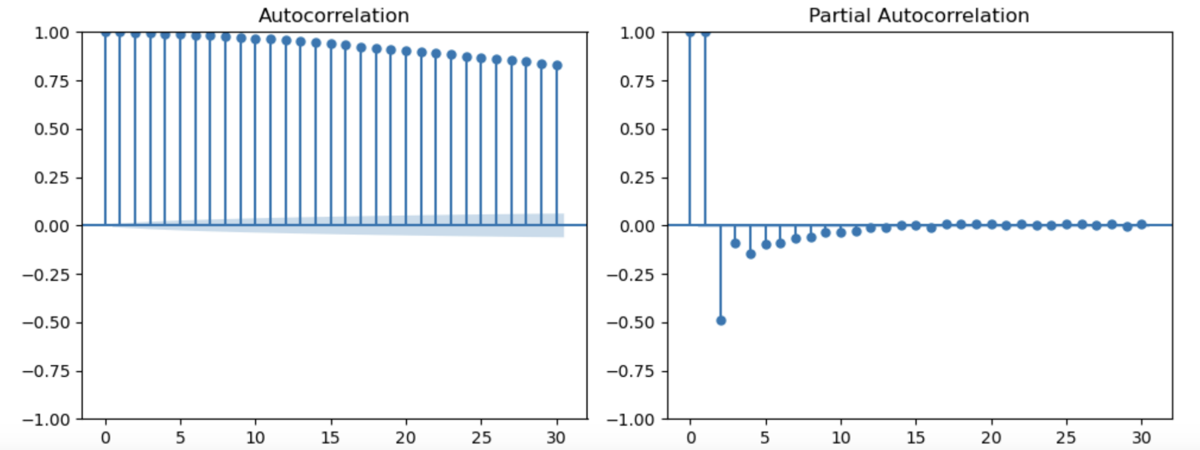


Figure 13: ACF and PACF graphs

# 7 Initialising variables for SARIMAX

The code in figure 15 shows order and seasonal order components required for time series analysis for SARIMAX. SARIMAX is a time series forecasting model that includes seasonality to the basic ARIMA model.

```
# Specify the order and seasonal_order parameters for SARIMAX
order = (1, 1, 1)  # (p, d, q)
seasonal_order = (1, 0, 1, 10)  # (P, D, Q, S)
```

Figure 14: Order and Seasonal orders

The code in figure 16 focuses on dividing a dataset into training and testing sets.

```
# Specify the order and seasonal_order parameters for SARIMAX
order = (1, 1, 1)  # (p, d, q)
seasonal_order = (1, 0, 1, 10)  # (P, D, Q, S)
```

Figure 15: Order and Seasonal orders

# 8 SARIMAX Modelling and predictions

The code in figure 17 talks about the model building of SARIMAX time series.
The code in figure 18 involves forecasting using the fitted SARIMA model for Zone 1 power consumption, evaluating the forecasted values, and calculating Mean Squared

```
# Split the data into 80% training and 20% testing
train_data, test_data = train_test_split(df, test_size=0.2, shuffle=False)
```

Figure 16: Training and testing sets

Error (MSE) and Root Mean Squared Error (RMSE) metrics. 'forecast1' contains forecasted results of time series, 'predicted_zone1' stores the predicted mean values for Zone 1 power consumption from the forecast, forecasted_values holds the forecasted values obtained from predicted_zone1, actual_values contains the actual power consumption values from the last 12 observations in the test data. The 'mean_squared_error(actual_values, forecasted_values)' calculates the MSE between the actual and forecasted values and the square root of MSE gives RMSE.

The similar code is also there for Zone 2 power consumption, Zone 3 power consumption, Temperature and Humidity.

```python
# Create and fit the SARIMA model for zone 1 power consumption
#model = SARIMAX(df['Zone 1 Power Consumption'], order=order, seasonal_order=seasonal_order)
model1 = SARIMAX(train_data['Zone 1 Power Consumption'], order=order, seasonal_order=seasonal_order)
results1 = model1.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.57382D+00    |proj g|=  9.20021D-02

At iterate    5    f=  7.56881D+00    |proj g|=  3.52700D-03

At iterate   10    f=  7.56875D+00    |proj g|=  8.19087D-04

At iterate   15    f=  7.56865D+00    |proj g|=  5.46129D-03

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     19     22      1     0     0   6.768D-05   7.569D+00
  F =   7.5686168200734656

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

Figure 17: SARIMAX Model Creation.

```python
forecast1 = results1.get_forecast(steps=12)
predicted_zone1 = forecast1.predicted_mean
print(predicted_zone1)

forecasted_values = np.array(predicted_zone1)

# Extract the last 12 actual test values
actual_values = np.array(test_data['Zone 1 Power Consumption'][-12:])

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(actual_values, forecasted_values)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
2017-10-19 04:40:00    25666.045587
2017-10-19 04:50:00    25753.540162
2017-10-19 05:00:00    25823.748687
2017-10-19 05:10:00    25880.851264
2017-10-19 05:20:00    25926.696875
2017-10-19 05:30:00    25958.508600
2017-10-19 05:40:00    25995.375771
2017-10-19 05:50:00    26027.272135
2017-10-19 06:00:00    26045.157748
2017-10-19 06:10:00    26066.080167
2017-10-19 06:20:00    26081.482192
2017-10-19 06:30:00    26093.950397
Freq: 10T, Name: predicted_mean, dtype: float64
Mean Squared Error (MSE): 42747195.128234446
Root Mean Squared Error (RMSE): 6538.133917887767
```

Figure 18: SARIMAX Predictions