2023

Modelling Effects of Climate on Power Consumption in Tetouan, Morocco

TIME SERIES ANALYSIS AND MODELING (DATS 6313): FINAL TERM PROJECT (FTP)

DANIEL FELBERG

Table of Contents

Table of Figures and Tables

Abstract

Introduction

In 2022, the International Trade Administration reported that Morocco imported “approximately 90 percent of its energy needs”, a considerable amount. Unlike its neighbors, Morocco does not have enough gas or oil reserves to become a major producer, meaning it relies heavily on importing oil products, in order to sustain its electrical grid. That being said, the government has declared that it believes 52% of the country's total installed capacity will come from renewable energy sources by 2030. It is with these considerations that we then understand there is a very real need to be able to model and forecast energy consumption within the country, both in the short-term, and long-term. While creating a model for forecasting the entire country’s power consumption may be extremely ambitious, we may be able to work with relatively smaller data, and create predictions for local areas. A potential candidate that we can utilize for this is the city of Tetouan, situated along the Northern coast of Morocco (*Figure 1*).

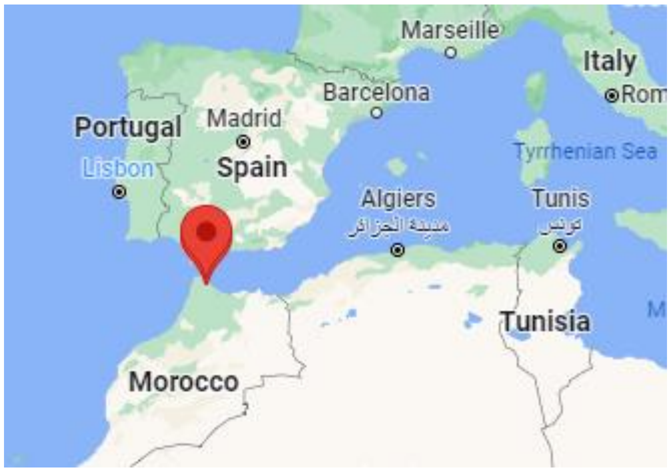


Figure 1: Map Showing Tetouan's Location

Similarly, there is also a need to understand how changes in climate may impact energy consumption, and whether those changes can help predict future energy consumption. As pointed out by the US Environmental Protection Agency (EPA), as temperatures continue to rise across the world, people are expected to consume more electricity for cooling purposes. With rising temperatures, other climate-related changes can also occur: according to US National Oceanic and Atmospheric Administration (NOAA), warmer air can hold more moisture, leading to increased humidity in certain environments. Additionally, with warmer air, the “wind chill index” can also change, meaning that the same windspeed that would normally feel “cool”, may not have that same impact as before. Other geological factors that may also play a role are diffuse flows, which can be briefly summarized as “low-temperature (< 0.2° to ~ 100°C) fluids that slowly discharge through sulfide mounds” (Bemis, Lowell, and Farough, 2012). Having looked at each of these climate-related factors may contribute towards energy consumption, we now turn to the research that has been made towards this effort, specifically in the city of Tetouan.

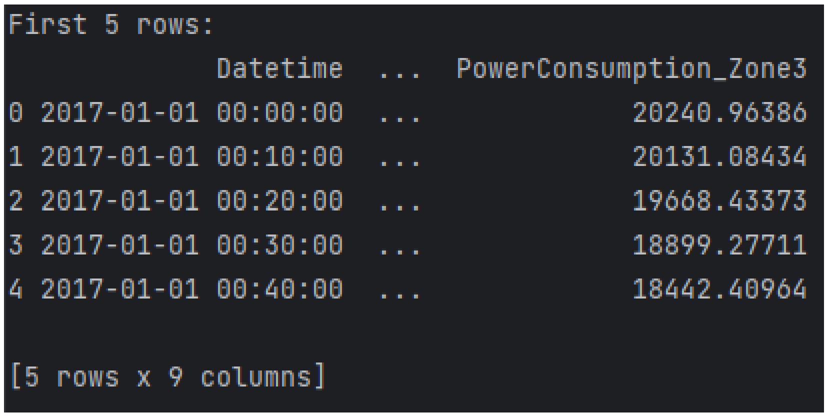
Abdulwahed Salam and Abdelaaziz El Hibaoui (2018), two researchers from Abdelmalek Essaadi University (located in Tetouan) decided to take on the challenge of predicting power consumption using meteorological features. They used weather data that was collected every 5 minutes, for all of 2017, using sensors at the city center of Tetouan, and at the Abdelmalek Essaadi University’s Faculty of Science. The weather data includes temperature, humidity, wind speed, general diffuse flows, and diffuse flows. While the researchers themselves do not explicitly explain the difference between the last 2 variables, for the purposes of our analysis, we can define "general diffuse flows" to encompass a broader category of subsurface fluid movement, whereas "diffuse flows" refer to specific types of fluid emissions, often occurring in geothermal and hydrothermal settings (W.M. Alley, 2009). The importance of establishing this difference will become clear later on, when we look at multi-collinearity between our independent variables.

As for power consumption in Tetouan (measured in Kilowatt-hours, or KWH), the data comes from the Supervisory Control and Data Acquisition System (SCADA) of Amendis, Morocco, a “public service operator in charge of the distribution of drinking water and electricity since 2002” (Salam & Hibaoui, 2018). More specifically, SCADA transforms the delivered power from high voltage (63 kV) down to medium voltage (20 kV), and distributes it across 3 power stations: Quads, Smir and Boussafou (also called zones 1, 2, and 3, but not in that order). As a result, power consumption is measured separately for each zone, and should not be combined (either by mean, or sum total) to represent Tetouan’s total power consumption. While it may be possible to do so with a deeper understanding of the subject at hand, our analysis would not take into account the complexities involved, and other potential *measurable* variables. It should also be noted that because power consumption was collected every 10 minutes, the weather data that was collected was resampled from every 5 minutes, to every 10 minutes as well, using the average of the two sensors used.

Salam’s and Hibaoui’s analysis utilized various machine learning models, and they found that the method which resulted in the least prediction errors was random forest. In this report, I intend to focus on models that are specific to time series analysis. Before we do so, however, it is necessary to perform data pre-processing and Exploratory Data Analysis, to better understand the dataset. From there, we will check the stationarity of the dependent variable, implement differencing if needed, and use time series decomposition to establish Seasonality and Trend. After applying these preliminary procedures we will test the Holt-Winters method, base models (Simple Average, Naïve, Drift, and Simple Exponential Smoothing), Multiple Linear Regression, and lastly, Seasonal Autoregressive Integrated Moving Average (SARIMA), which will also requires us to estimate seasonal and non-seasonal orders and differencing. Finally, once we select our final model, we’ll be able to make h-step ahead predictions, and check the performance of the model. Through all of this, I hope to study the effects of climate on energy, and ultimately determine the best time series model to accurately predict power consumption over time.

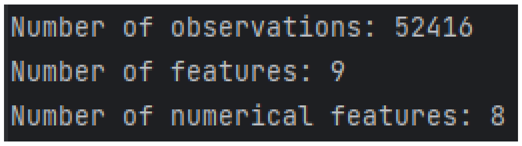
Dataset Description

The first, and most important step in time series analysis is understanding the data at hand. Since we already understand what the data is supposed to measure, as described in the introduction, we now need to look at the dataset itself (after reading it into python using pandas):



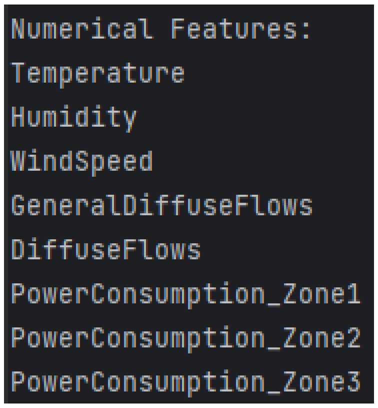
```
First 5 rows:
             Datetime  ...  PowerConsumption_Zone3
0 2017-01-01 00:00:00  ...             20240.96386
1 2017-01-01 00:10:00  ...             20131.08434
2 2017-01-01 00:20:00  ...             19668.43373
3 2017-01-01 00:30:00  ...             18899.27711
4 2017-01-01 00:40:00  ...             18442.40964

[5 rows x 9 columns]
```

Figure 2: First 5 rows of raw data after importing



```
Number of observations: 52416
Number of features: 9
Number of numerical features: 8
```

Figure 3: Number of observations, total features, and numerical features in raw data



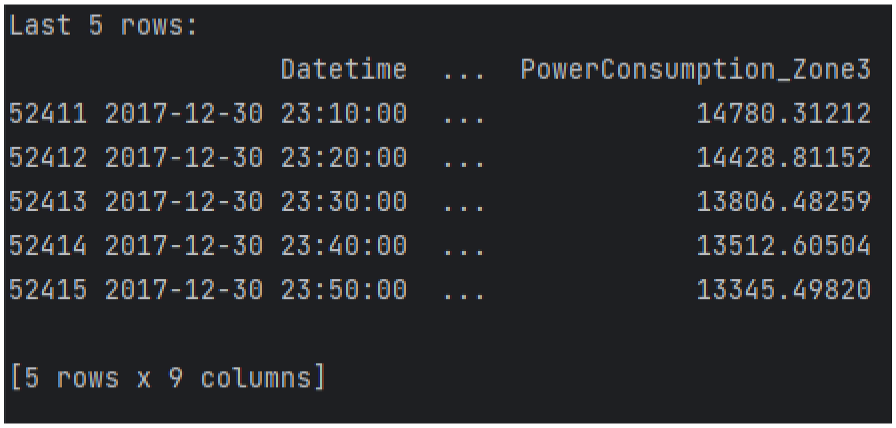
```
Numerical Features:
Temperature
Humidity
WindSpeed
GeneralDiffuseFlows
DiffuseFlows
PowerConsumption_Zone1
PowerConsumption_Zone2
PowerConsumption_Zone3
```

Figure 4: List of numerical features in raw data

In Figure 2, the console outputs show that the raw dataset has and 9 columns, with the left-most column listing the year, month, day, hour, minute, and seconds, in 10 minute intervals. The right-most column shows the power consumption for zone 3. The 'Datetime' value for the first row confirms that our data begins at the start of 2017. Figure 3 shows that we have a total of 52,416 observations, and as shown in Figure 1, we have a total of 9 features in the raw dataset, of which 8 are numerical. Figure 4 then lists all of the numerical features, showing that all columns, except for “Datetime” are numerical. In summary, our features are as follows:

- “Datetime” (shows the year, month, day, hour, minute, and seconds, in 10 minute intervals, that the data was observed)
- “Temperature” (temperature in numerical values)
- “Humidity” (humidity in numerical values)
- “WindSpeed” (wind speed in numerical values)
- “GeneralDiffuseFlows” (general diffuse flows in numerical values)
- “DiffuseFlows” (diffuse flows in numerical values)
- “PowerConsumption_Zone1” (power consumption for zone 1, in KWH)
- “PowerConsumption_Zone2” (power consumption for zone 2, in KWH)
- “PowerConsumption_Zone3” (power consumption for zone 3, in KWH)

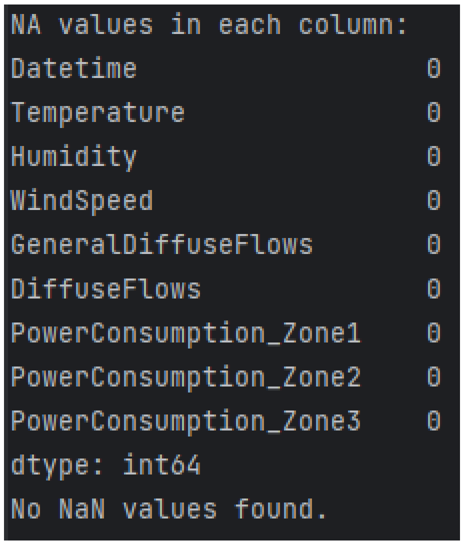
We now have to check if any cleaning might be necessary to further process the raw data:



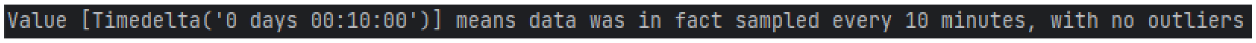
```
Last 5 rows:
                 Datetime  ...  PowerConsumption_Zone3
52411 2017-12-30 23:10:00  ...             14780.31212
52412 2017-12-30 23:20:00  ...             14428.81152
52413 2017-12-30 23:30:00  ...             13806.48259
52414 2017-12-30 23:40:00  ...             13512.60504
52415 2017-12-30 23:50:00  ...             13345.49820

[5 rows x 9 columns]
```

Figure 5: Last 5 rows of raw data

Figure 6: Listing NA values for each column



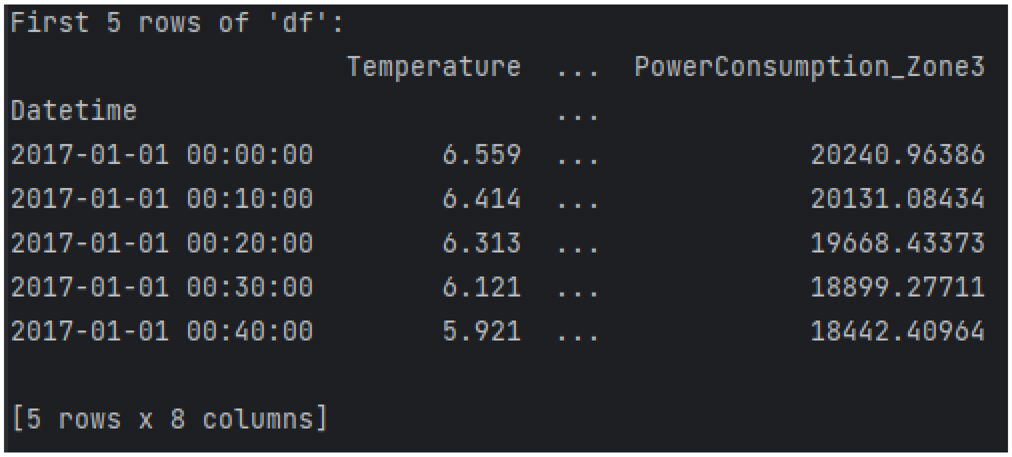
```
NA values in each column:
Datetime                   0
Temperature                0
Humidity                   0
WindSpeed                  0
GeneralDiffuseFlows        0
DiffuseFlows               0
PowerConsumption_Zone1     0
PowerConsumption_Zone2     0
PowerConsumption_Zone3     0
dtype: int64
No NaN values found.
```



```
Value [Timedelta('0 days 00:10:00')] means data was in fact sampled every 10 minutes, with no outliers
```

Figure 7: Checking that data was evenly sampled every 10 mins., with no exceptions

Figures 5-7 confirm that the raw dataset is extremely clean: the data is sampled until the very last 10-minute interval of 2017, with no NA values, nor any unevenly sampled data. We can proceed, but we will create a deep copy of the data that we can use, which we will call “df” so the raw dataset (“df_raw”) is preserved. The “Datetime” column is then indexed to the new dataframe, as shown below:



```
First 5 rows of 'df':
                     Temperature  ...  PowerConsumption_Zone3
Datetime                          ...
2017-01-01 00:00:00        6.559  ...             20240.96386
2017-01-01 00:10:00        6.414  ...             20131.08434
2017-01-01 00:20:00        6.313  ...             19668.43373
2017-01-01 00:30:00        6.121  ...             18899.27711
2017-01-01 00:40:00        5.921  ...             18442.40964

[5 rows x 8 columns]
```

Figure 8: First 5 rows of "df", with "Datetime" as an index

Having indexed the "Datetime" column, we can now easily plot our dependent variable (or variables, since we have a column measuring power consumption for each power zone):

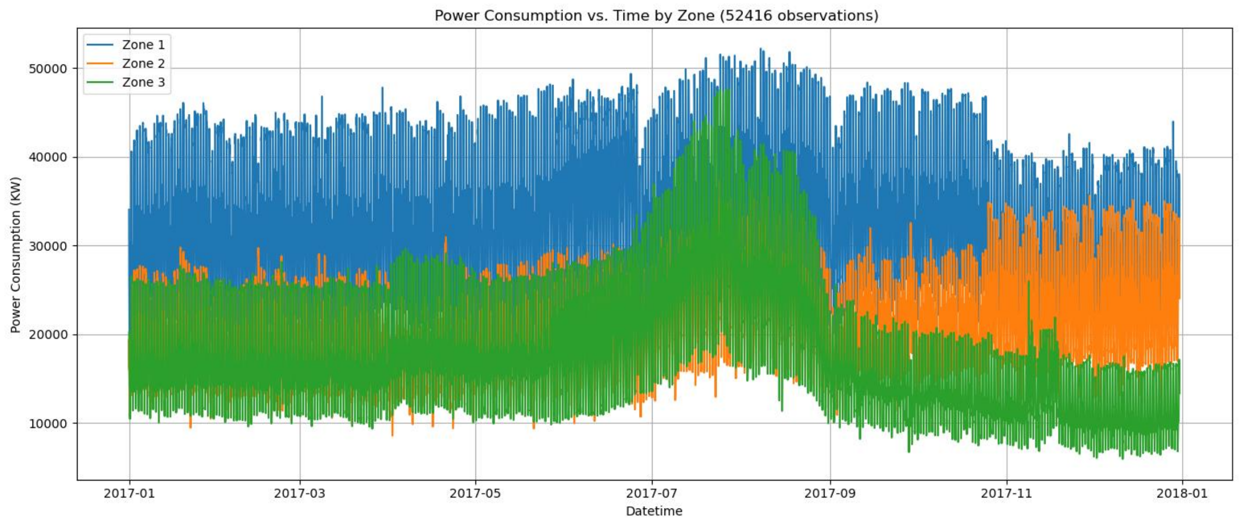
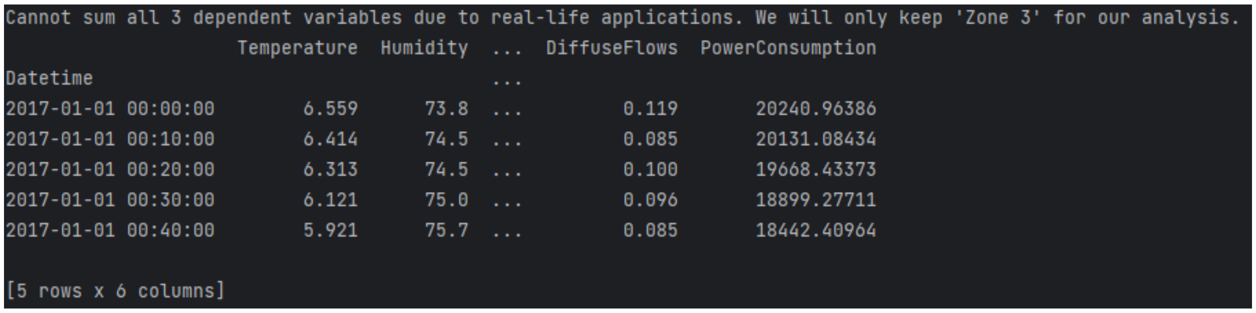


Figure 9: Plotting power consumption vs. time for all 3 zones

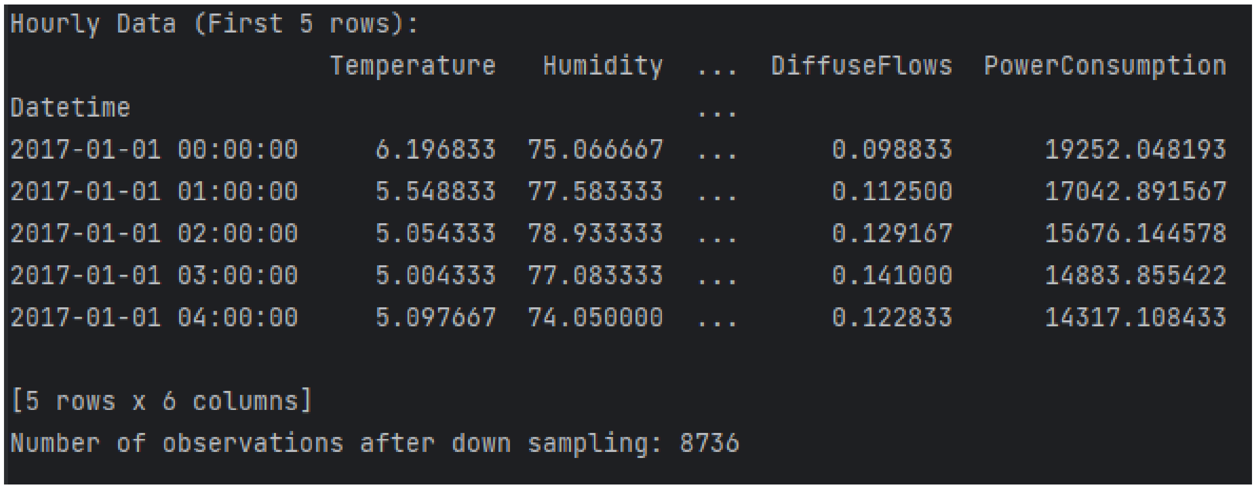
Figure 9 confirms that each zone has an incredibly unique pattern in terms of power consumption, and acts independently of the power consumption in other zones. As discussed in the introduction, we cannot combine all 3 zones into one column. This means our data has 5 features (temperature, humidity, wind speed, general diffuse flows, and diffuse flows) that will serve as our independent variables. To proceed, we now have to choose which column, or zone, we will be using to develop our model. Out of all 3 plots in Figure 9, "Zone 3" seemed the most interesting to use, as it potentially shows a yearly seasonality (if we were analyzing multiple years), or bi-annual trend (if we were only looking at data for the first or last 6 months), with a peak in the middle of the year. The small vertical lines along the plot also potentially indicate some sort of periodic seasonality, possibly daily. Further analysis is needed to confirm these observations. For reference, the zone we have chosen to keep (zone 3) is the station called Smir in the real-world.



```
Cannot sum all 3 dependent variables due to real-life applications. We will only keep 'Zone 3' for our analysis.
                     Temperature  Humidity  ...  DiffuseFlows  PowerConsumption
Datetime                                    ...
2017-01-01 00:00:00        6.559      73.8  ...         0.119       20240.96386
2017-01-01 00:10:00        6.414      74.5  ...         0.085       20131.08434
2017-01-01 00:20:00        6.313      74.5  ...         0.100       19668.43373
2017-01-01 00:30:00        6.121      75.0  ...         0.096       18899.27711
2017-01-01 00:40:00        5.921      75.7  ...         0.085       18442.40964

[5 rows x 6 columns]
```

Figure 10: First 5 rows of 'df', after removing zones 1 and 2, and renaming zone 3 column

Figure 10 confirms we now only have our 5 features (independent variables), and target variable (power consumption in zone 3) in “df”. We should now plot “PowerConsumption” by itself to see if we can observe any other patters. However, before this, as Figures 3 and 9 point out, we currently have 52,416 observations (samples) in our data. This has its advantages and disadvantages: with more observations, we improve our model accuracy, but this comes at the expense of computational power. As a result, I believe down-sampling is a necessary sacrifice we must make, so as to not overcomplicate our models, and more importantly, so that the code can be run with a lower risk of using up too much memory. For this, we will resample “df” as hourly data:



```
Hourly Data (First 5 rows):
                     Temperature   Humidity  ...  DiffuseFlows  PowerConsumption
Datetime                                     ...
2017-01-01 00:00:00     6.196833  75.066667  ...      0.098833      19252.048193
2017-01-01 01:00:00     5.548833  77.583333  ...      0.112500      17042.891567
2017-01-01 02:00:00     5.054333  78.933333  ...      0.129167      15676.144578
2017-01-01 03:00:00     5.004333  77.083333  ...      0.141000      14883.855422
2017-01-01 04:00:00     5.097667  74.050000  ...      0.122833      14317.108433

[5 rows x 6 columns]
Number of observations after down sampling: 8736
```

Figure 11: First 5 rows of "df_hourly" and number of hourly observations

With our down-sampled data, we can now plot “PowerConsumption” (for Smir, or zone 3) against time. To fully observe the patterns that may have been hidden initially, we will plot the full year (8736 hours), the first month (744 hours), the first week (168 hours), and the first day (24 hours).

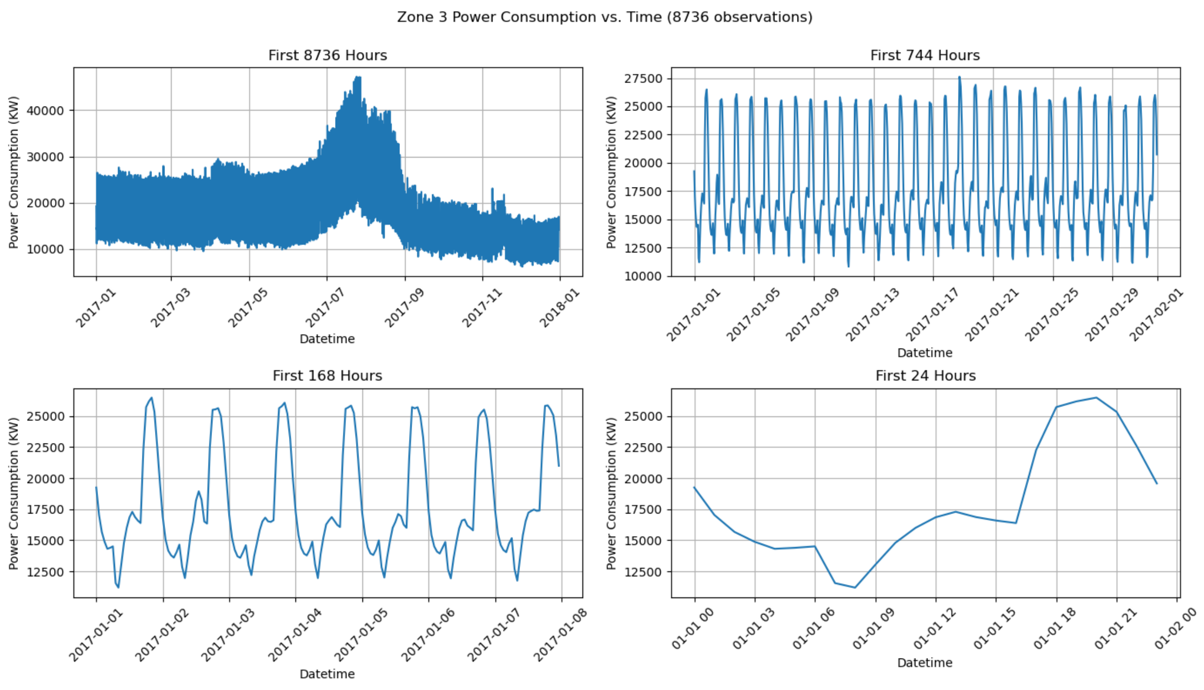


Figure 12: Plotting "PowerConsumption" vs. time (year, month, week, day)

Figure 12 confirms our initial observations: we do see a daily seasonality, repeating approximately every 24 hours, at around 6 PM potentially. Consequently, this means our data may be non-stationary. To confirm this, we can graph the Auto-Correlation Function (ACF) of the dependent variable (“PowerConsumption”):

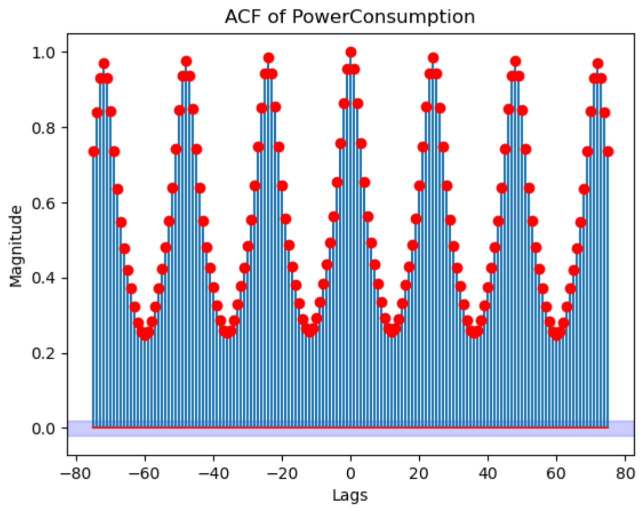


Figure 13: ACF of "PowerConsumption" (75 lags)

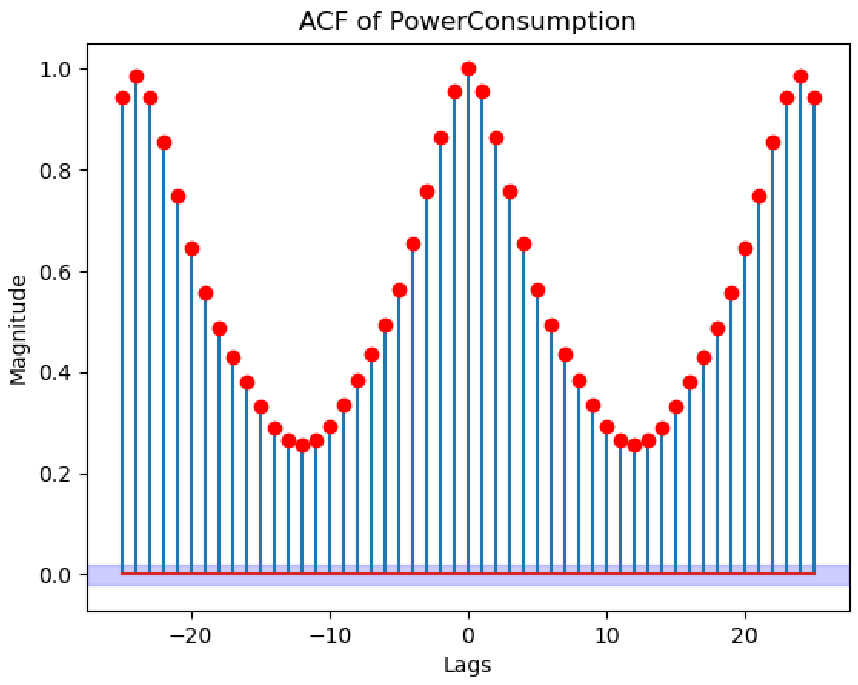


Figure 14: ACF of "PowerConsumption" (25 lags)

Figures 13 and 14 show that our data is in fact, incredibly non-stationary, and the ACF magnitude peaks every 24 hours, confirming what we had hypothesized previously (seasonal period repeats every 24 hours). We will return to this, in order to make our data stationary for some of the models that require it, but for the time being, we will see if we can observe any immediate correlations.

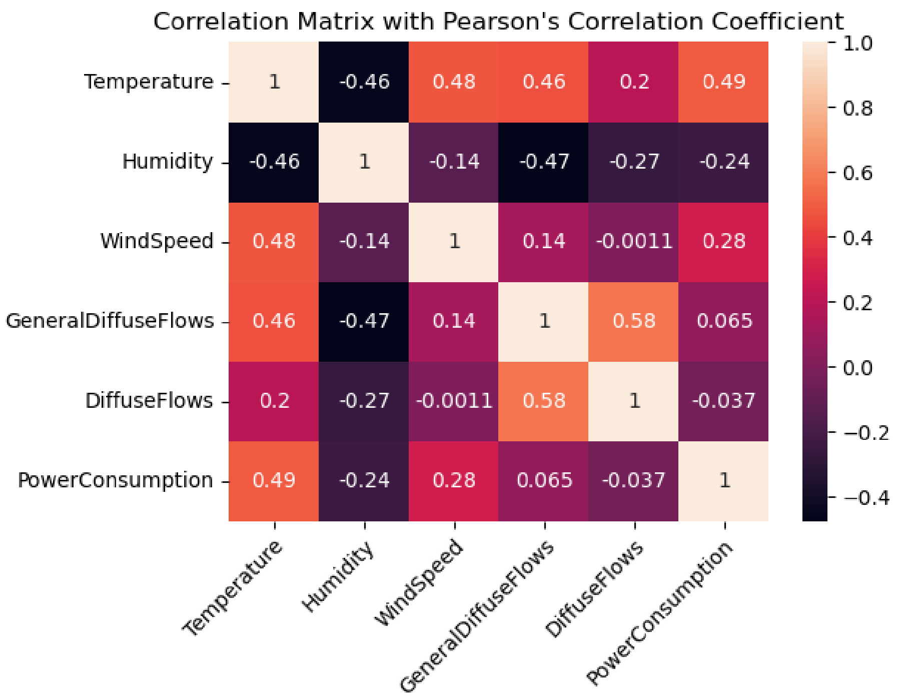


Figure 15: Correlation matrix of hourly data

Figure 15 appears to show that none of the variables are *very* strongly correlated. That being said, we do see some moderate (maybe moderate-to-strong) relationships, such as with “Temperature” and “PowerConsumption”, “Temperature” and “GeneralDiffuseFlows”, and “Temperature” and “WindSpeed”. The strongest correlation, however, appears to be between “GeneralDiffuseFlows” and “DiffuseFlows”, with a calculated Pearson’s Correlation Coefficient of 0.58. This is concerning as it might indicate multi-collinearity in our data. It would make sense, since despite measuring separate types of diffuse flows, they might ultimately be catching the same patterns. This is something that should be looked at going forward, once we get to feature selection.

Before continuing on to create our stationary dataset, we will make an initial train/test split of the hourly data (“df_hourly”), without having made any other modifications beyond what was covered so far:

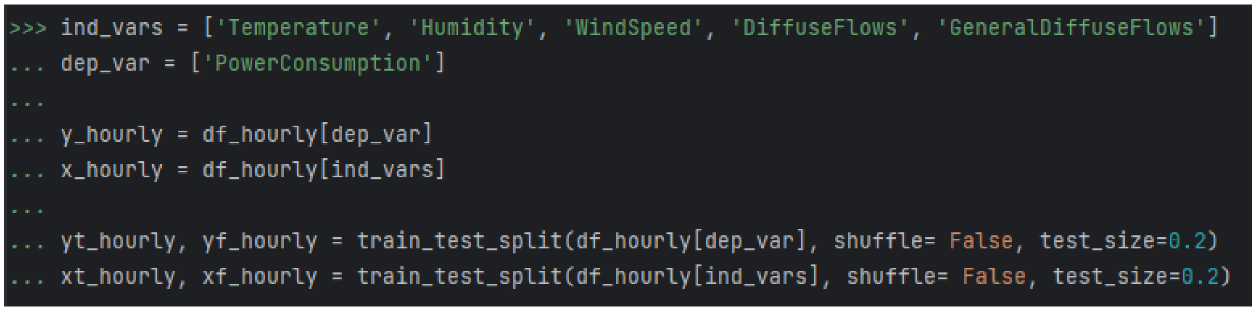

```
>>> ind_vars = ['Temperature', 'Humidity', 'WindSpeed', 'DiffuseFlows', 'GeneralDiffuseFlows']
... dep_var = ['PowerConsumption']
...
... y_hourly = df_hourly[dep_var]
... x_hourly = df_hourly[ind_vars]
...
... yt_hourly, yf_hourly = train_test_split(df_hourly[dep_var], shuffle= False, test_size=0.2)
... xt_hourly, xf_hourly = train_test_split(df_hourly[ind_vars], shuffle= False, test_size=0.2)
```

Figure 16: Train/test split on "df_hourly"

Stationarity

We can now come back to checking for stationarity in our dependent variable ("PowerConsumption"). For this, we will perform both ACF and PACF (Partial Auto-Correlation Function) analysis.

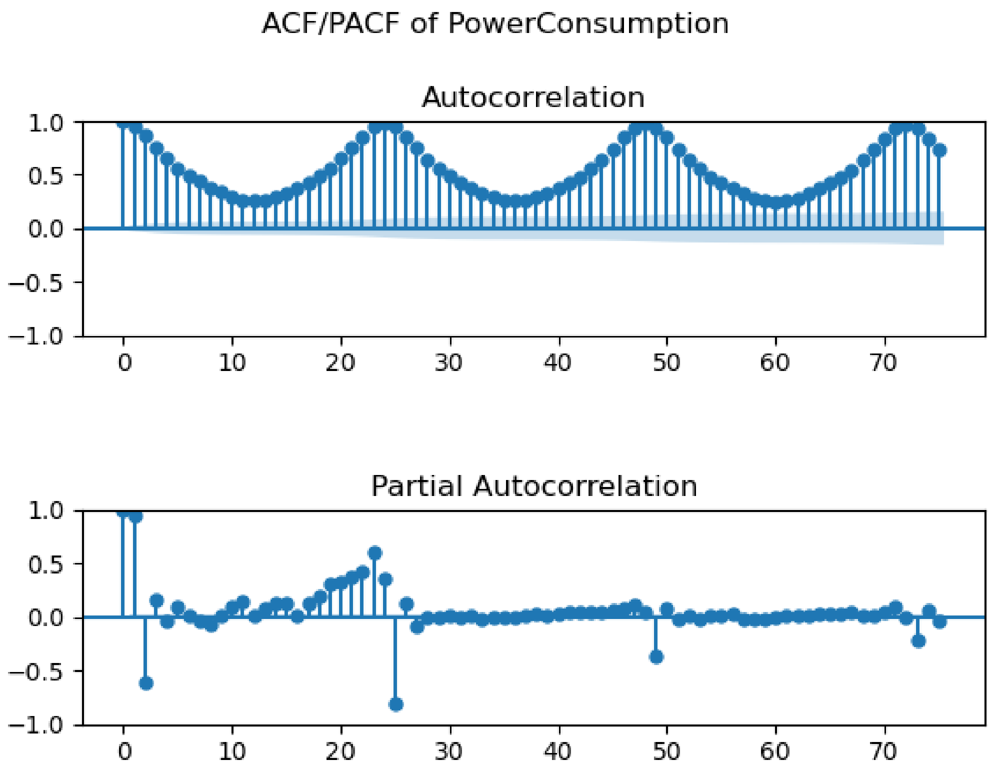
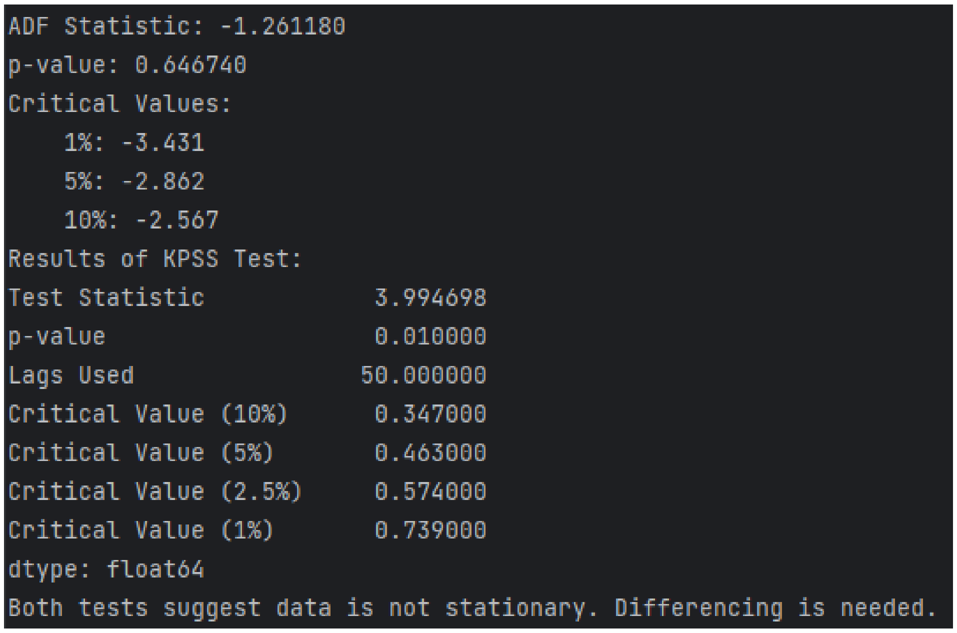


Figure 17: ACF/PACF of "PowerConsumption"

As confirmed with our initial ACF plot, we see a high degree of seasonality in the plots above, with daily peaks every 24 lags. This is significant, as it means that we should perform seasonal differencing in order to make our data stationary. The PACF plot also reflects the spikes we see every 24 lags. To confirm what we see in the ACF/PACF plots, we will now perform ADF and KPSS tests, and plot the rolling mean and rolling variances.



```
ADF Statistic: -1.261180
p-value: 0.646740
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Results of KPSS Test:
Test Statistic             3.994698
p-value                    0.010000
Lags Used                 50.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
Both tests suggest data is not stationary. Differencing is needed.
```

Figure 18: ADF & KPSS Results of hourly data

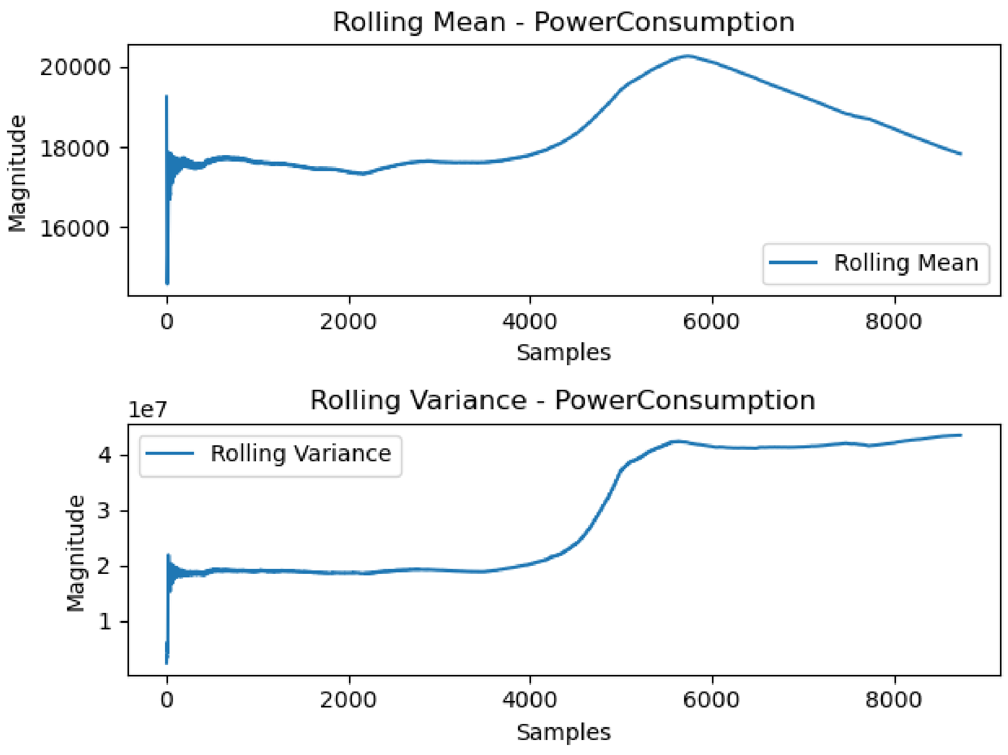


Figure 19: Rolling mean & variance of hourly data ("PowerConsumption")

Figures 18 and 19 confirm what we saw in the ACF/PACF analysis: our dependent variable is in fact, non-stationary. We see this reflected in the ADF and KPSS tests of Figure 18, where our ADF test p-value is approximately 0.6, meaning it is above the significance value of 0.05, whereas the KPSS test p-value is below it. Looking at the rolling mean and variance plots, we see that the mean continues to change drastically all the way towards the end, but variance appears to be slightly more stationary. Nevertheless, we should perform seasonal differencing to make "PowerConsumption" stationary, using 24 as our differencing order (note that we are using a deep copy, so that we can keep our original data).

Figure 20: ACF/PACF After Seasonal Differencing (Periods=24)

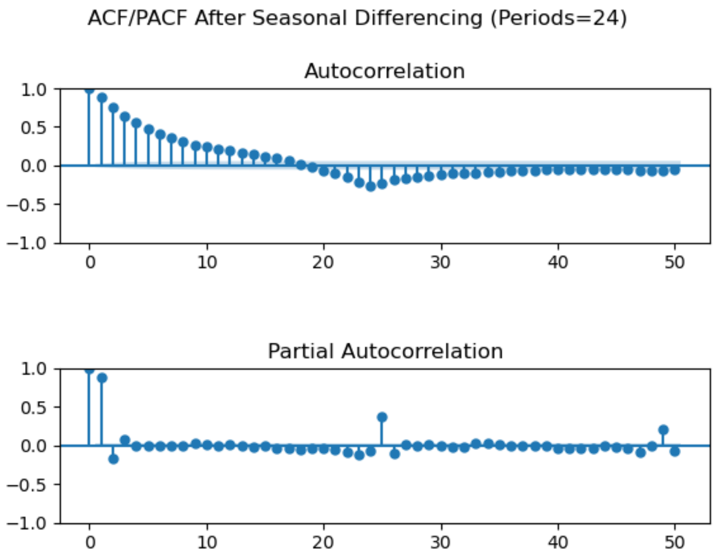
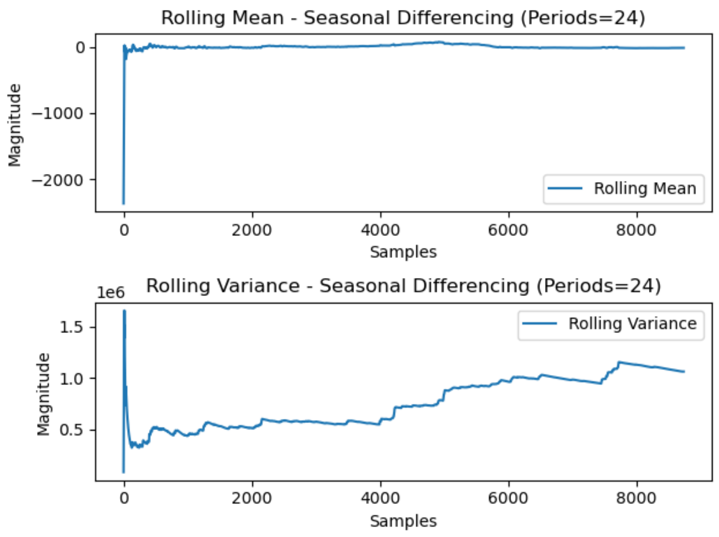


Figure 21: Rolling mean & variance after seasonal differencing



We now see significant improvement in the plots, but the ACF plot and rolling variance suggest we may also have to perform 1st order non-seasonal differencing.

Figure 22: ACF/PACF After Seasonal & 1st Order Diff

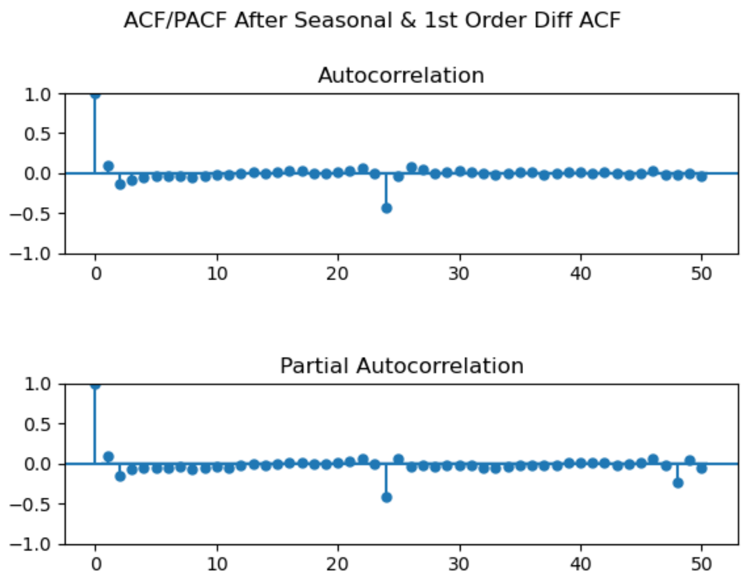
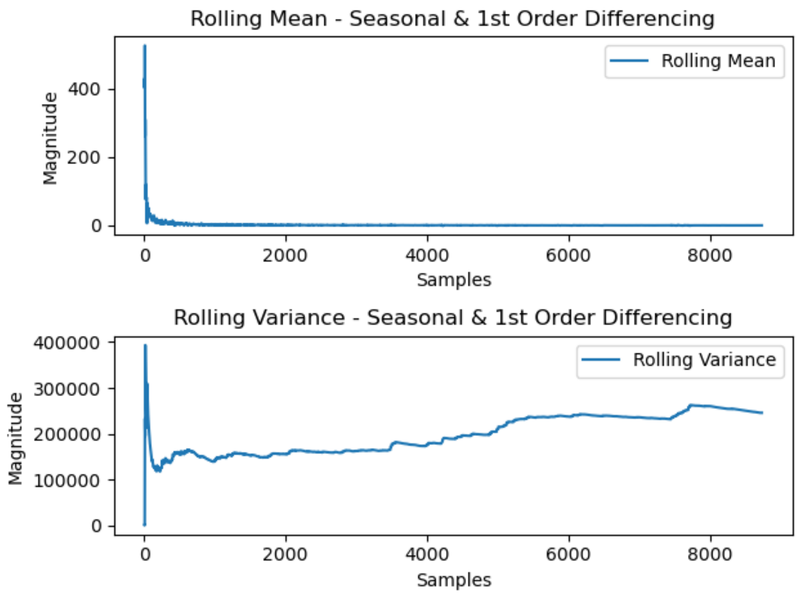
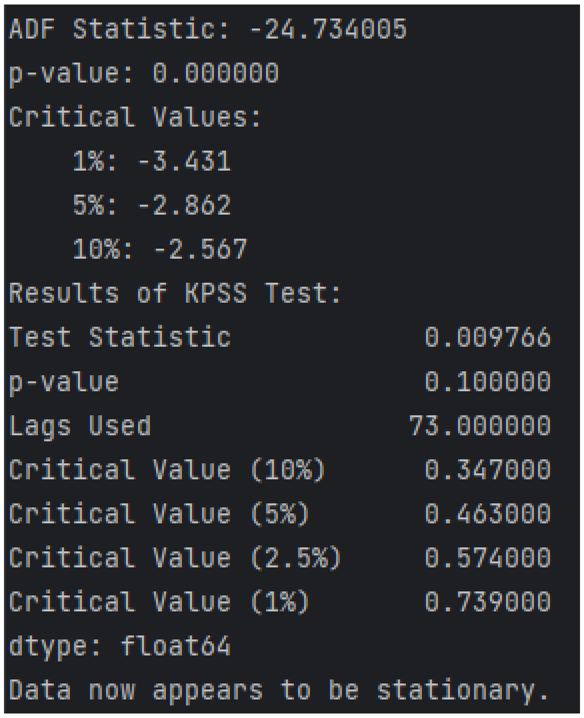


Figure 23: Rolling Mean & Variance after Season & 1st Order Diff



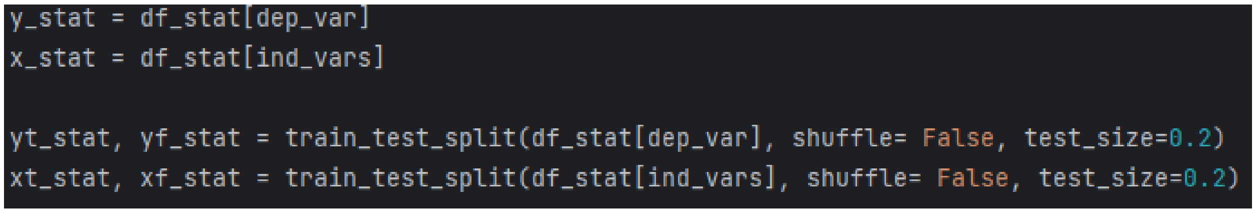
The plots look much better, but there may still be some non-stationarity reflected in the rolling variance plot. To finalize our stationarity analysis, we should check the ADF and KPSS tests again:



```
ADF Statistic: -24.734005
p-value: 0.000000
Critical Values:
    1%: -3.431
    5%: -2.862
    10%: -2.567
Results of KPSS Test:
Test Statistic            0.009766
p-value                   0.100000
Lags Used                73.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
Data now appears to be stationary.
```

Figure 24: ADF & KPSS test of Stationary Data

As we can see in Figure 24, the p-value for the ADF test is significantly below the significance value of 0.05, and KPSS is significantly above it. We can now assume our dependent variable ("PowerConsumption") to be stationary. Now that we have our stationary dataset, we will make an additional train/test split:



```
y_stat = df_stat[dep_var]
x_stat = df_stat[ind_vars]

yt_stat, yf_stat = train_test_split(df_stat[dep_var], shuffle= False, test_size=0.2)
xt_stat, xf_stat = train_test_split(df_stat[ind_vars], shuffle= False, test_size=0.2)
```

Figure 25: Stationary data train/test split

Time Series Decomposition

Now that we have our stationary data, we can more accurately measure the strength of seasonality and trend. For this, we can use the STL (Seasonal and Trend decomposition using Loess) method, which allows us to then calculate the strength of seasonality and trend in our data.

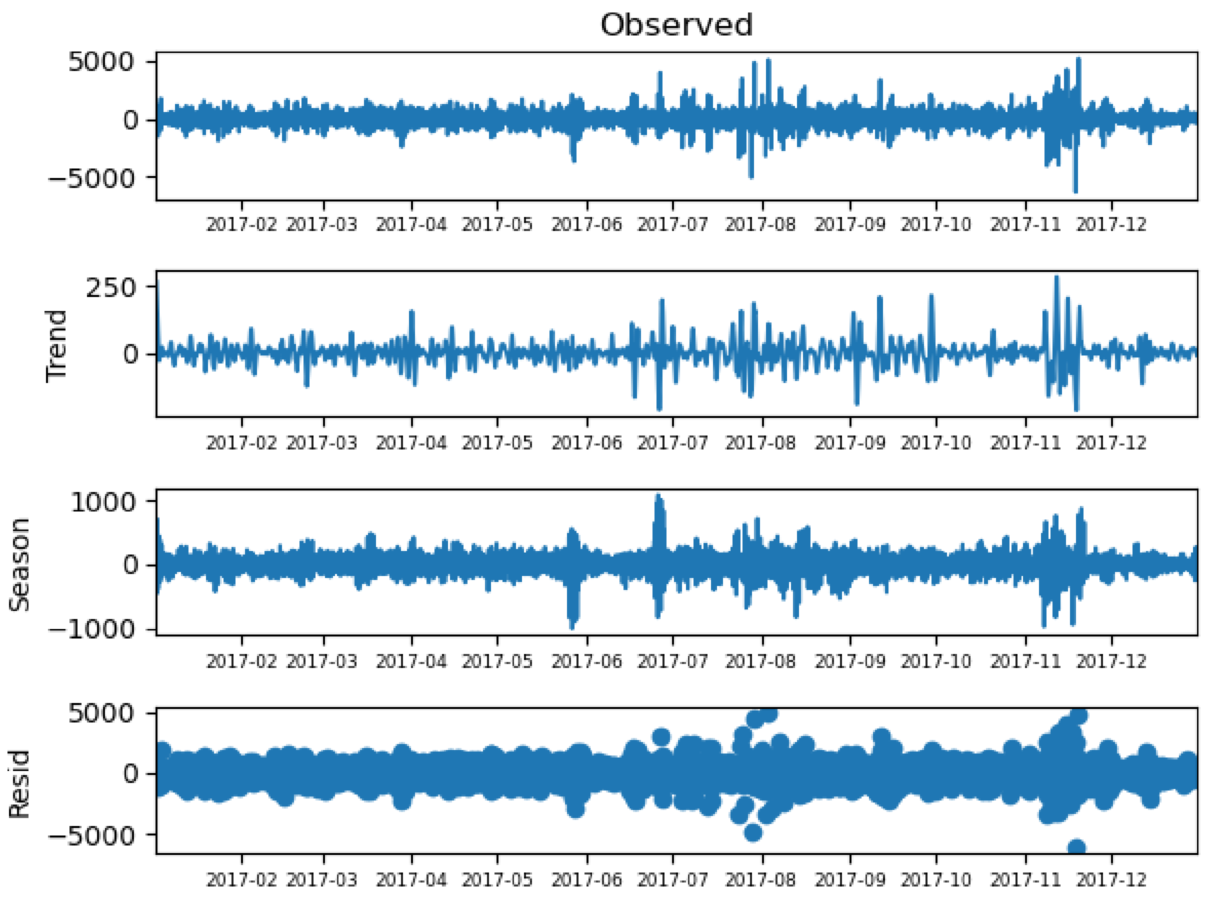
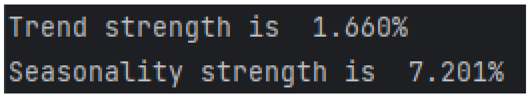


Figure 26: STL Decomposition (stationary data)



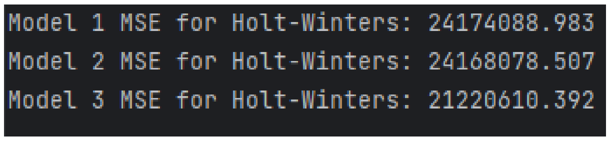
```
Trend strength is  1.660%
Seasonality strength is  7.201%
```

Figure 27: Strength of trend & seasonality

Based on the time series decomposition reflected in figures 26 and 27, we find that our stationary data is very weakly trended, and has very weak seasonality.

Holt-Winters Method

To begin testing some initial models, we have to return to our original non-stationary data. The first model that we should try and fit our data into is the Holt-Winters method, which has as one of its advantages the ability to capture trend and seasonality to capture a linear model. After trying 3 different versions of the model, using MSE as a measure of performance, I found that the one that resulted in the lowest score had a multiplicative parameter for both trend and seasonal. We then tested it on the data to check how well the model performed. Overall, I would not say this model performed well.



```
Model 1 MSE for Holt-Winters: 24174088.983
Model 2 MSE for Holt-Winters: 24168078.507
Model 3 MSE for Holt-Winters: 21220610.392
```

Figure 28: MSE scores for Holt-Winters Models

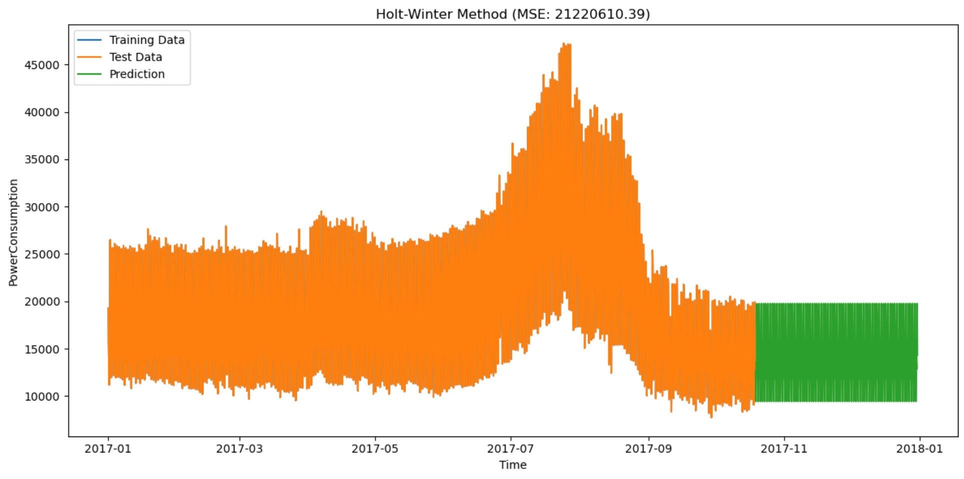


Figure 29: Holt-Winter method forecast plot

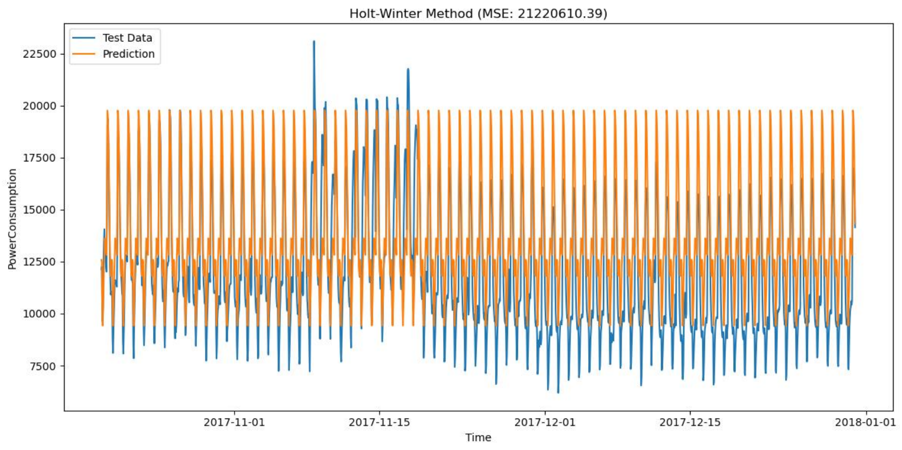
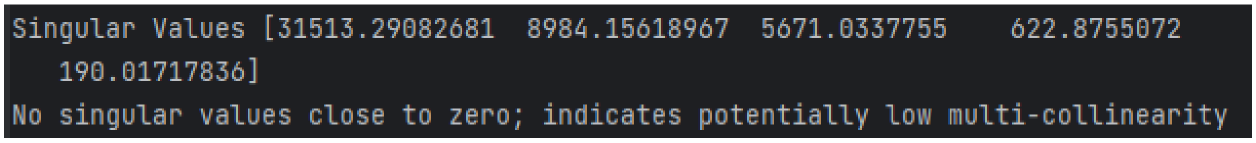


Figure 30: Holt-Winter method forecast (test & prediction)

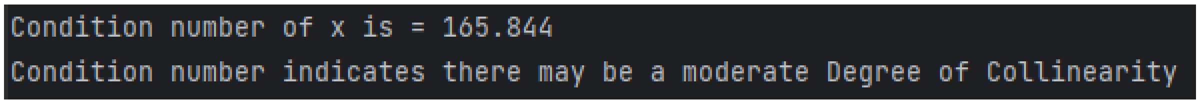
Feature Selection/Dimensionality Reduction

Before we can try out the base models and multiple linear regression, we have to check that our data does not have multi-collinearity, where our independent variables might be correlated with each other. This is a crucial step, as it allows us to improve our models, and remove features if necessary.



```
Singular Values [31513.29082681  8984.15618967  5671.0337755    622.8755072
   190.01717836]
No singular values close to zero; indicates potentially low multi-collinearity
```

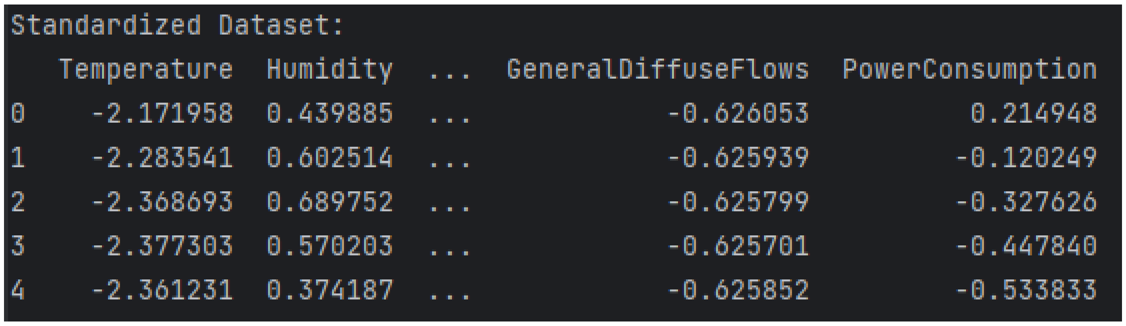
Figure 31: Singular Value Decomposition analysis



```
Condition number of x is = 165.844
Condition number indicates there may be a moderate Degree of Collinearity
```

Figure 32: Condition number

Based on initial analysis, we find that while SVD shows that there is potentially low multi-collinearity (Figure 31), the condition number indicates there may be a moderate Degree of Collinearity, which ranges from 100 to 1,000 (Figure 32). To be sure we create the best models possible, further analysis is needed. To do so, we must standardize our dataset:



```
Standardized Dataset:
   Temperature  Humidity  ...  GeneralDiffuseFlows  PowerConsumption
0    -2.171958  0.439885  ...            -0.626053          0.214948
1    -2.283541  0.602514  ...            -0.625939         -0.120249
2    -2.368693  0.689752  ...            -0.625799         -0.327626
3    -2.377303  0.570203  ...            -0.625701         -0.447840
4    -2.361231  0.374187  ...            -0.625852         -0.533833
```

Figure 33: First 5 rows of standardized dataset

With our standardized dataset, we are now able to perform Principal Component Analysis, Backward Stepwise Regression (using Order of Least Squares), and VIF analysis.

Figure 34: Principal component analysis

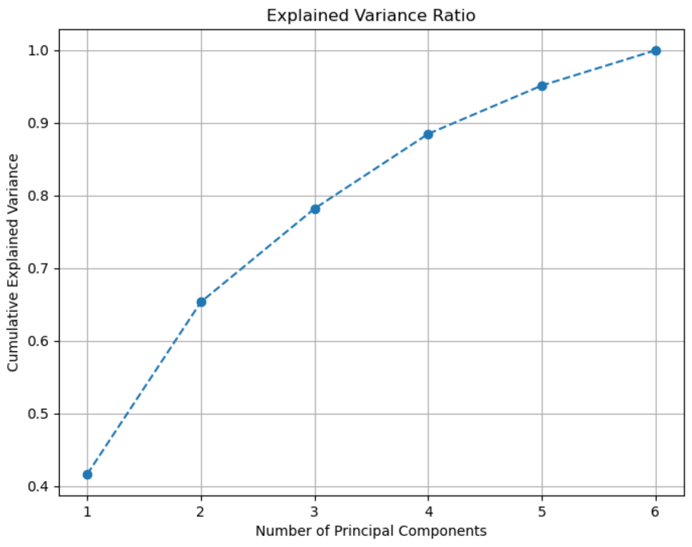
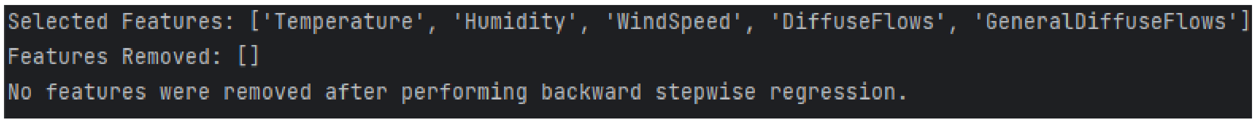
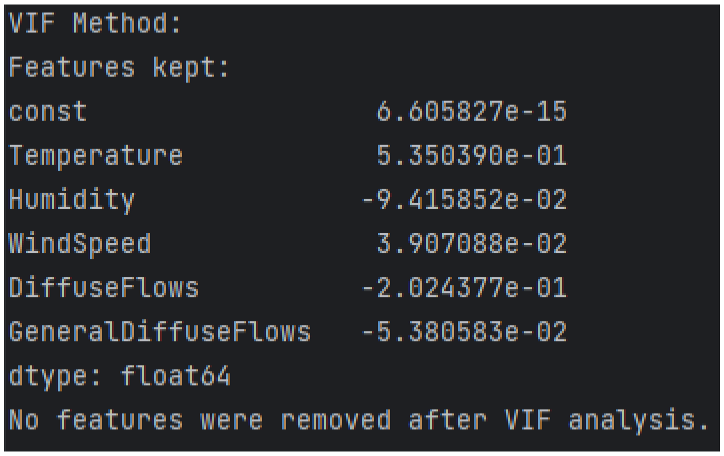


Figure 35: Backward stepwise regression



```
Selected Features: ['Temperature', 'Humidity', 'WindSpeed', 'DiffuseFlows', 'GeneralDiffuseFlows']
Features Removed: []
No features were removed after performing backward stepwise regression.
```

Figure 36: VIF analysis



```
VIF Method:
Features kept:
const                   6.605827e-15
Temperature             5.350390e-01
Humidity               -9.415852e-02
WindSpeed               3.907088e-02
DiffuseFlows           -2.024377e-01
GeneralDiffuseFlows    -5.380583e-02
dtype: float64
No features were removed after VIF analysis.
```

After performing the tests in figures 34-36, we find that no features need to be removed from the model to improve performance. If we look back at the first correlation matrix we plotted, we find that general diffuse flows and diffuse flows are not as strongly correlated as we may have thought, as they are not reducing the model's performance. We will now begin making our models.

Base Models

Before we get to MLR, we will first test how reliable base models are on our data. For that, we will use the Simple Average model, Naïve model, and Simple Exponential Smoothing (at different alpha values). They are shown below:

Figure 37: Simple Average model

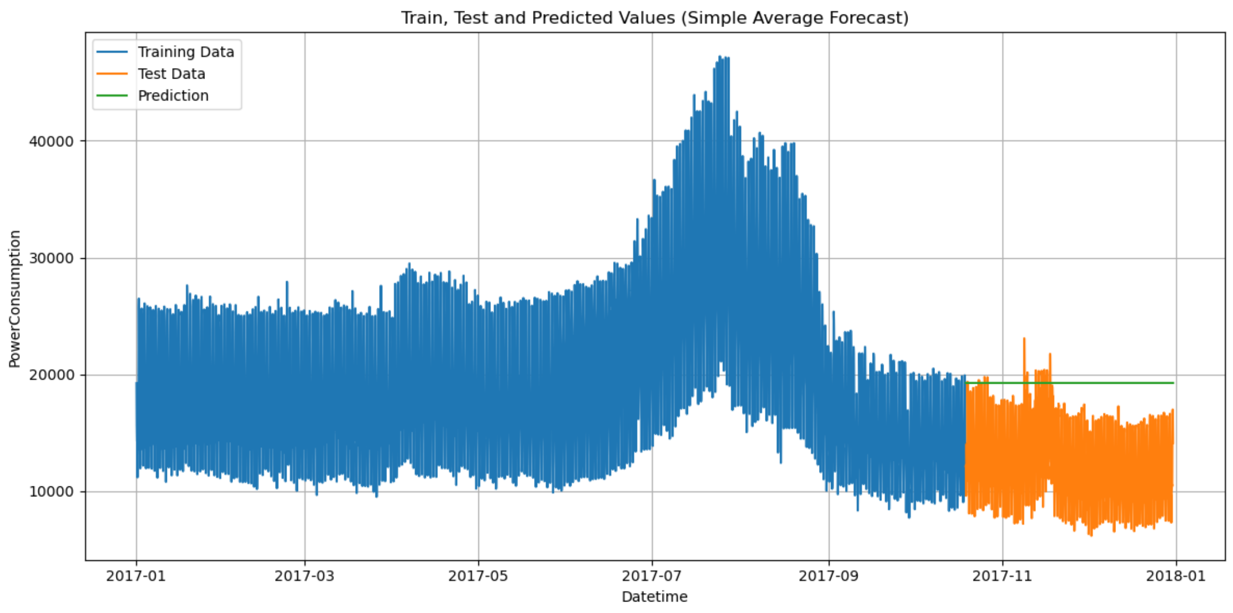


Figure 38: Naive model

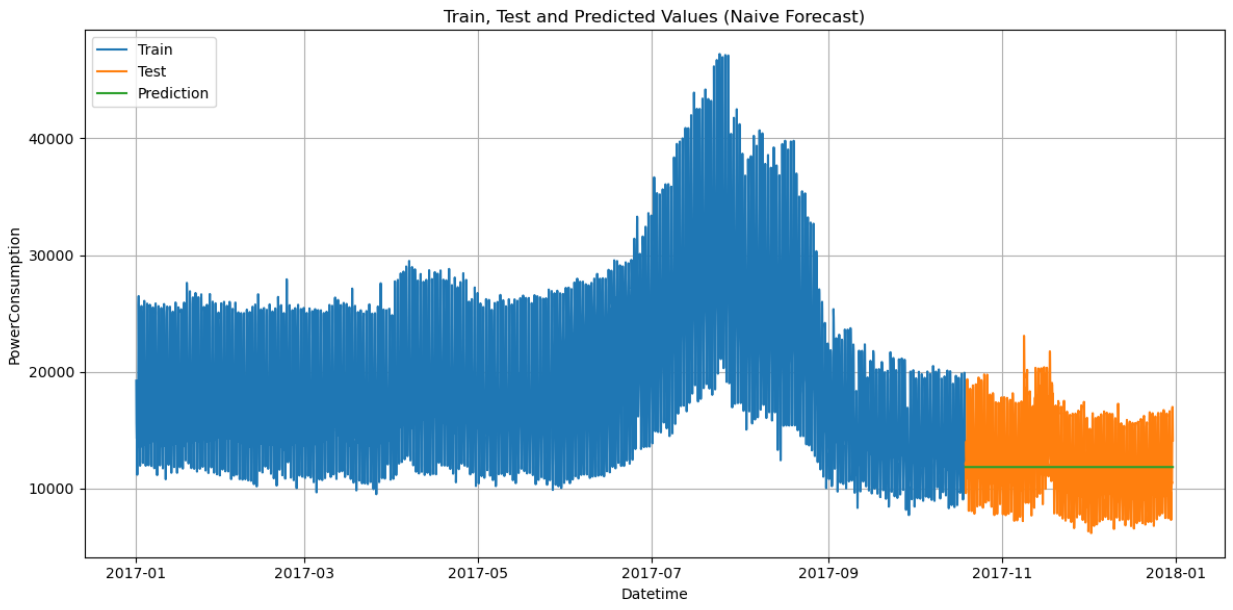


Figure 39: SES model at different alphas

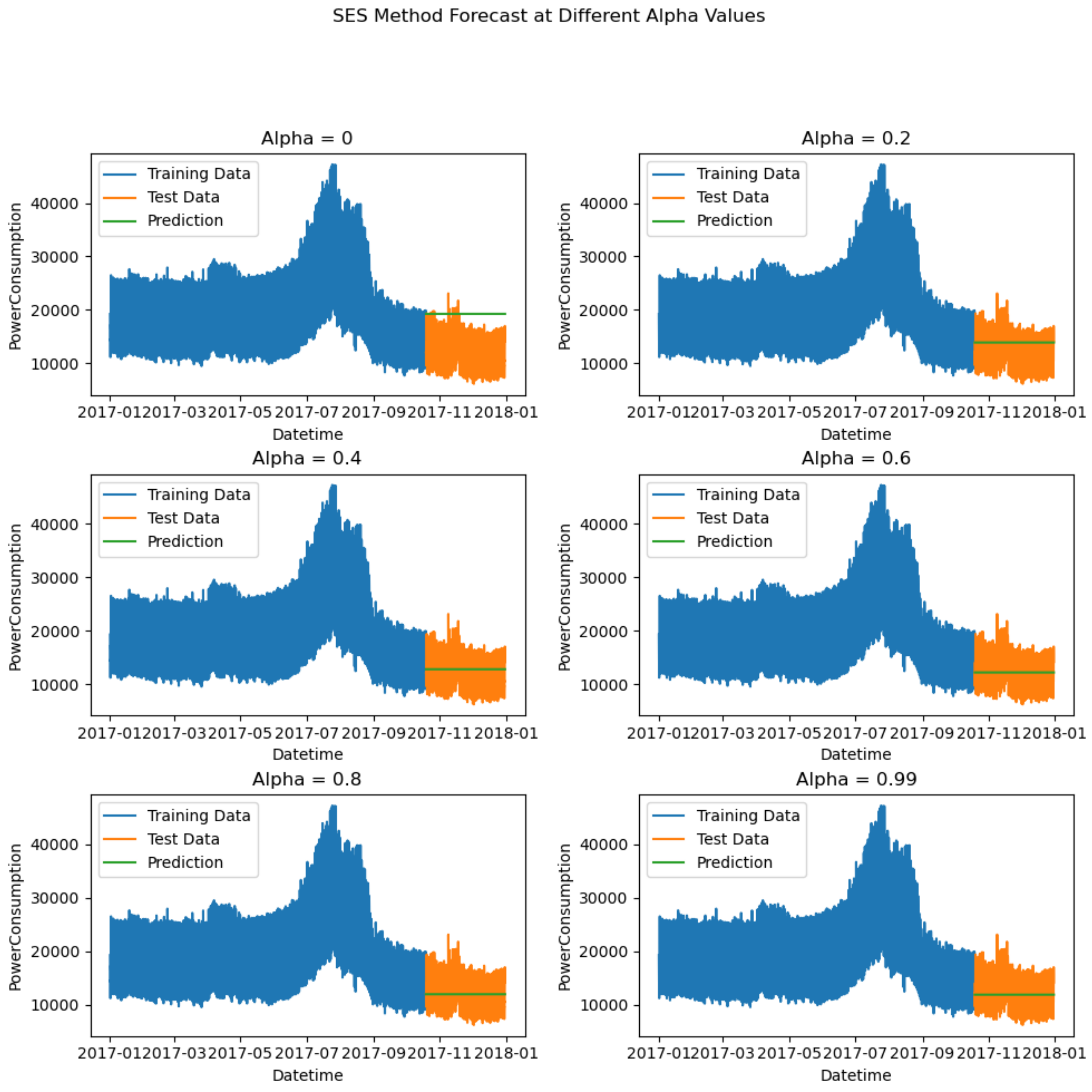
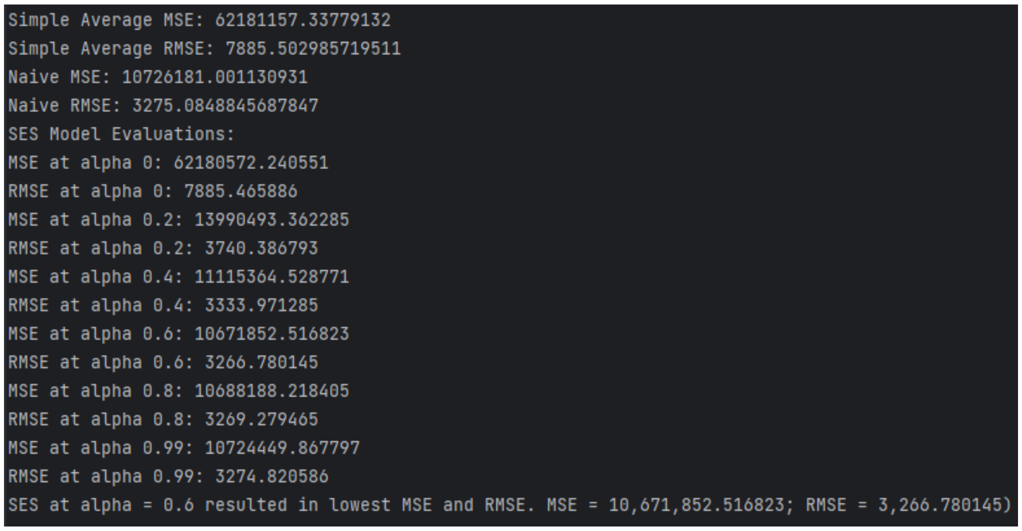


Figure 40: Base models MSE/RMSE scores



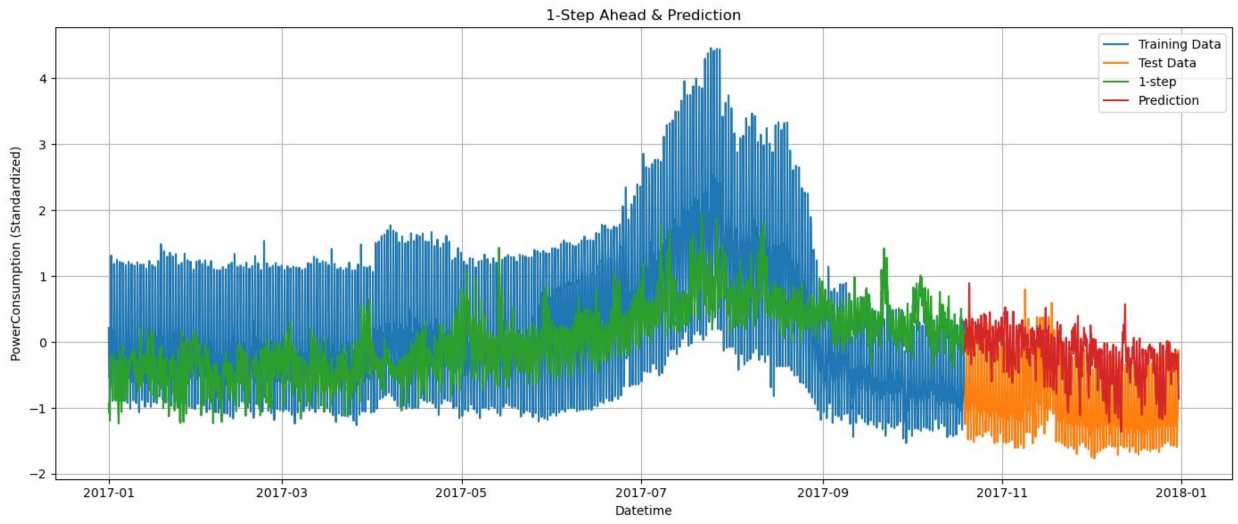
```
Simple Average MSE: 62181157.33779132
Simple Average RMSE: 7885.502985719511
Naive MSE: 10726181.001130931
Naive RMSE: 3275.0848845687847
SES Model Evaluations:
MSE at alpha 0: 62180572.240551
RMSE at alpha 0: 7885.465886
MSE at alpha 0.2: 13990493.362285
RMSE at alpha 0.2: 3740.386793
MSE at alpha 0.4: 11115364.528771
RMSE at alpha 0.4: 3333.971285
MSE at alpha 0.6: 10671852.516823
RMSE at alpha 0.6: 3266.780145
MSE at alpha 0.8: 10688188.218405
RMSE at alpha 0.8: 3269.279465
MSE at alpha 0.99: 10724449.867797
RMSE at alpha 0.99: 3274.820586
SES at alpha = 0.6 resulted in lowest MSE and RMSE. MSE = 10,671,852.516823; RMSE = 3,266.780145)
```

After training and testing the base models on our test data, I find that the model which results in the least amount of errors is SES at alpha 0.6 (MSE = 10,671,852.516823; RMSE = 3,266.780145). So far, this is our best-performing model, but we must check it again multiple linear regression.

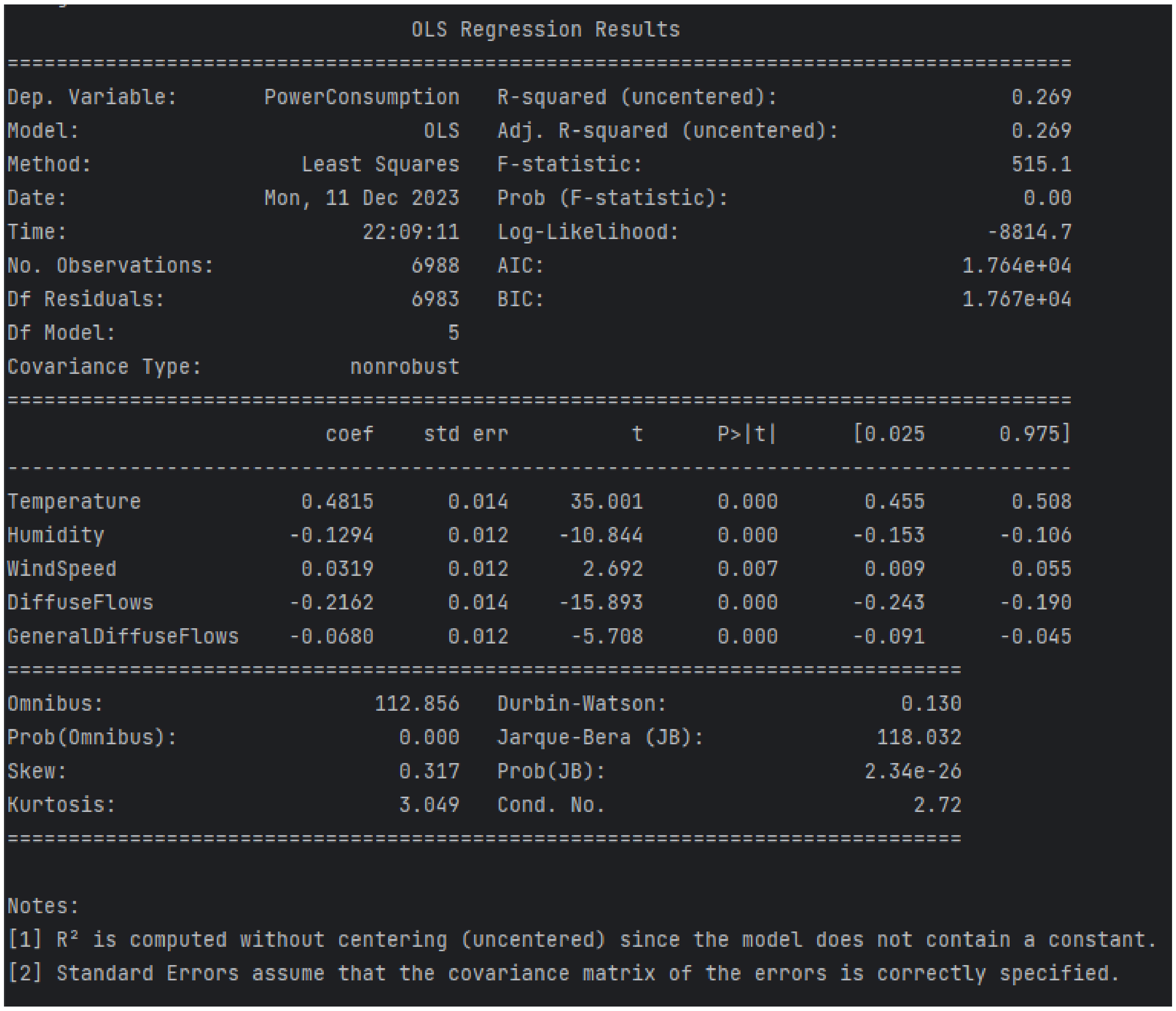
Multiple Linear Regression

For MLR, we will once again be using the standardized data that we used when conducting feature selection. Using Order of Least Squares regression, I created 1-step and h-step predictions, shown below:

Figure 41: MLR (train, test, 1-step, and h-step)

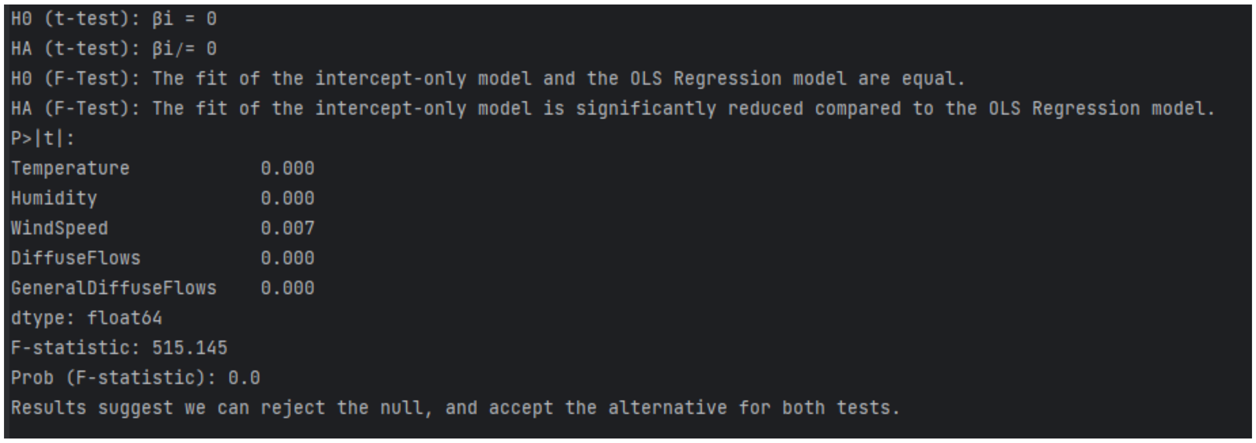


The OLS model summary is also shown below:



```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:      PowerConsumption   R-squared (uncentered):                   0.269
Model:                           OLS   Adj. R-squared (uncentered):              0.269
Method:                Least Squares   F-statistic:                              515.1
Date:               Mon, 11 Dec 2023   Prob (F-statistic):                        0.00
Time:                       22:09:11   Log-Likelihood:                         -8814.7
No. Observations:               6988   AIC:                                 1.764e+04
Df Residuals:                   6983   BIC:                                 1.767e+04
Df Model:                          5
Covariance Type:           nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
Temperature             0.4815      0.014     35.001      0.000       0.455       0.508
Humidity               -0.1294      0.012    -10.844      0.000      -0.153      -0.106
WindSpeed               0.0319      0.012      2.692      0.007       0.009       0.055
DiffuseFlows           -0.2162      0.014    -15.893      0.000      -0.243      -0.190
GeneralDiffuseFlows    -0.0680      0.012     -5.708      0.000      -0.091      -0.045
==============================================================================
Omnibus:                     112.856   Durbin-Watson:                   0.130
Prob(Omnibus):                 0.000   Jarque-Bera (JB):              118.032
Skew:                          0.317   Prob(JB):                     2.34e-26
Kurtosis:                      3.049   Cond. No.                         2.72
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

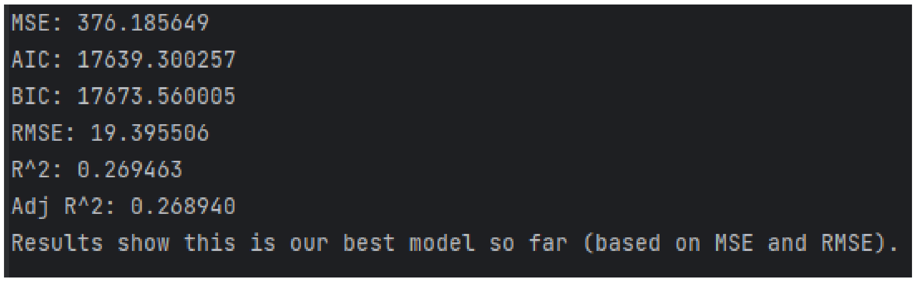
Figure 42: OLS regression results



```
H0 (t-test): βi = 0
HA (t-test): βi/= 0
H0 (F-Test): The fit of the intercept-only model and the OLS Regression model are equal.
HA (F-Test): The fit of the intercept-only model is significantly reduced compared to the OLS Regression model.
P>|t|:
Temperature            0.000
Humidity               0.000
WindSpeed              0.007
DiffuseFlows           0.000
GeneralDiffuseFlows    0.000
dtype: float64
F-statistic: 515.145
Prob (F-statistic): 0.0
Results suggest we can reject the null, and accept the alternative for both tests.
```

Figure 43: t-test & F-Test

Figure 44: Linear Regression Performance



```
MSE: 376.185649
AIC: 17639.300257
BIC: 17673.560005
RMSE: 19.395506
R^2: 0.269463
Adj R^2: 0.268940
Results show this is our best model so far (based on MSE and RMSE).
```

Figure 45: ACF of MLR residuals

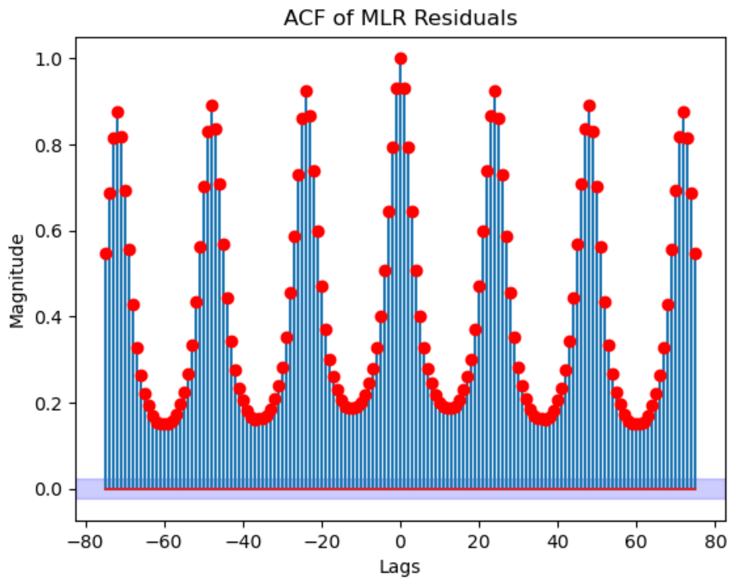


Figure 46: Variance and mean of MLR residuals

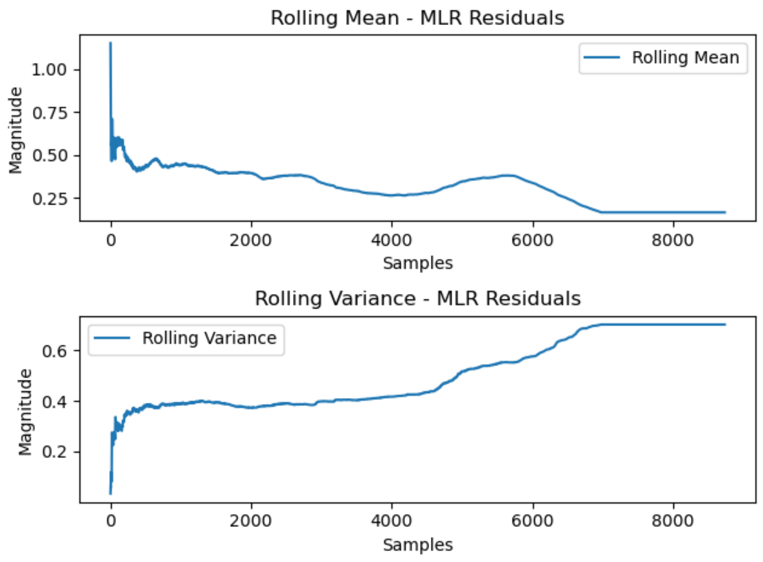
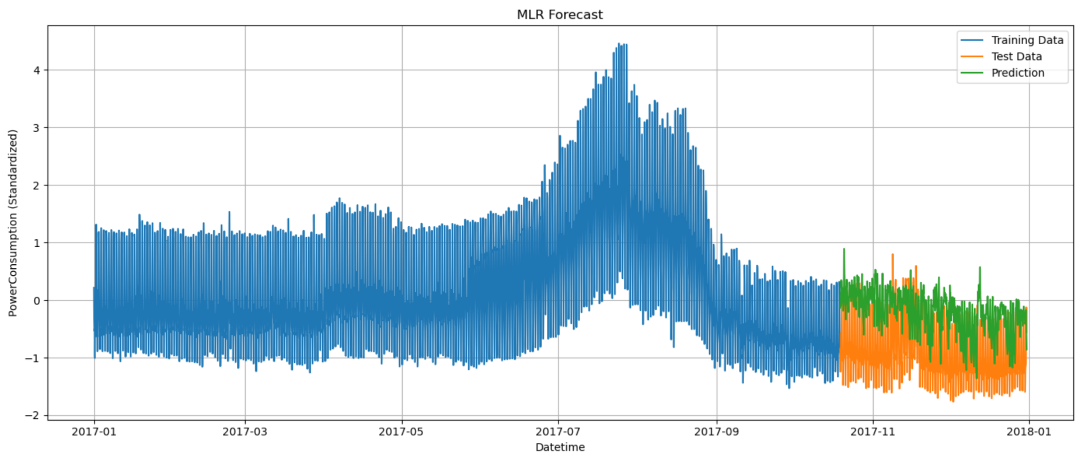


Figure 47: MLR train, test, and prediction plot

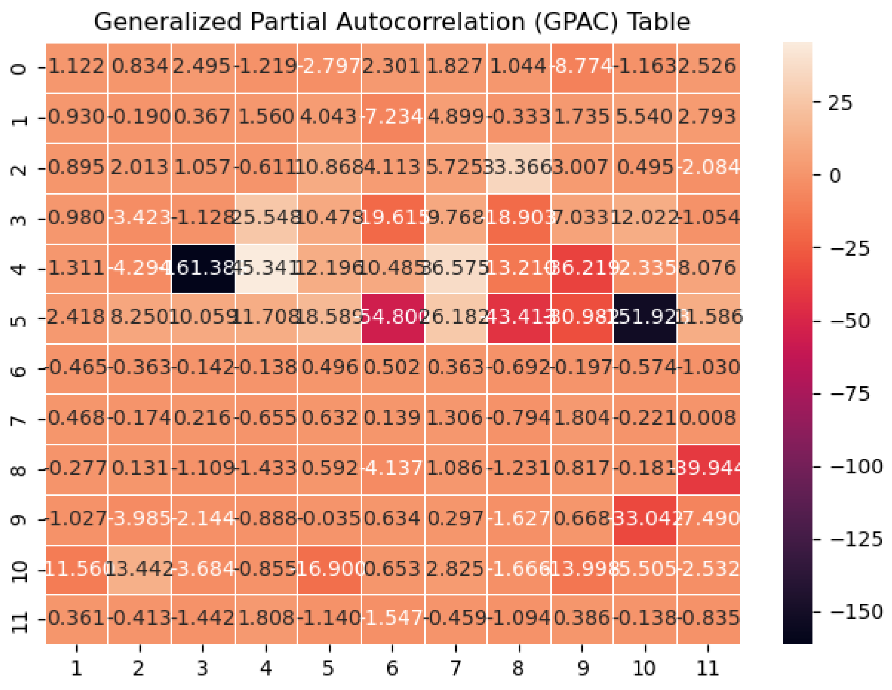


Overall, results show that while linear regression resulted in our lowest MSE so far, the residuals are non-stationary, meaning that the model can be improved upon. We now turn to SARIMA to try and create our new best model.

SARIMA Model

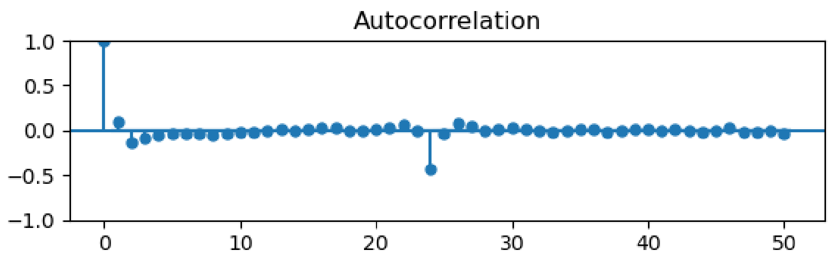
Since our data required seasonal differencing to become stationary, we should use the SARIMA model. To determine the best parameters for our model we can use the GPAC table below to find AR and MA orders.

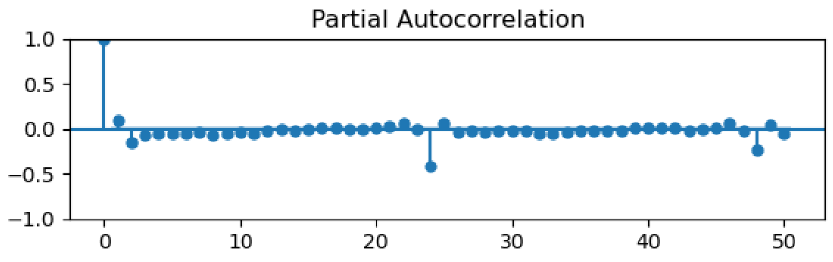
Figure 48: GPAC Table


Generalized Partial Autocorrelation (GPAC) Table
25
0
−25
−50
−75
−100
−125
−150

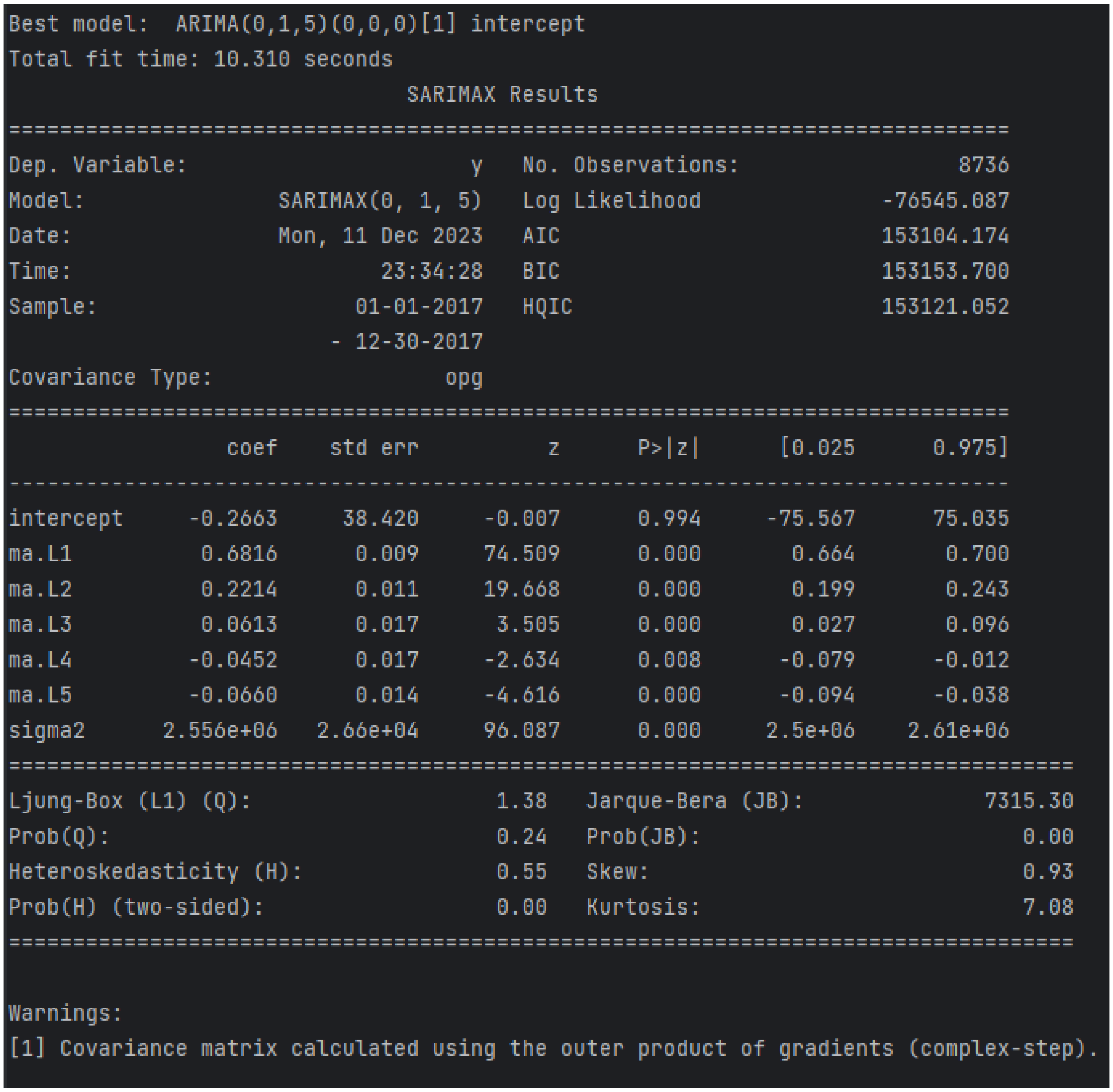
	1	2	3	4	5	6	7	8	9	10	11
0	1.122	0.834	2.495	-1.219	-2.797	2.301	1.827	1.044	-8.774	-1.163	2.526
1	0.930	-0.190	0.367	1.560	4.043	-7.234	4.899	-0.333	1.735	5.540	2.793
2	0.895	2.013	1.057	-0.611	10.868	4.113	5.725	33.366	3.007	0.495	-2.084
3	0.980	-3.423	-1.128	25.548	10.478	19.615	9.768	18.903	7.033	12.022	-1.054
4	1.311	-4.294	161.38	45.341	12.196	10.485	36.575	13.210	36.219	2.335	8.076
5	2.418	8.250	10.059	11.708	18.589	54.800	26.182	43.413	30.982	251.921	11.586
6	0.465	-0.363	-0.142	-0.138	0.496	0.502	0.363	-0.692	-0.197	-0.574	-1.030
7	0.468	-0.174	0.216	-0.655	0.632	0.139	1.306	-0.794	1.804	-0.221	0.008
8	0.277	0.131	-1.109	-1.433	0.592	-4.137	1.086	-1.231	0.817	-0.181	39.944
9	1.027	-3.985	-2.144	-0.888	-0.035	0.634	0.297	-1.627	0.668	33.042	7.490
10	11.560	13.442	-3.684	-0.855	16.900	0.653	2.825	-1.666	13.998	5.505	-2.532
11	0.361	-0.413	-1.442	1.808	-1.140	-1.547	-0.459	-1.094	0.386	-0.138	-0.835

Stationary ACF





SARIMA Forecast Function



```
Best model:  ARIMA(0,1,5)(0,0,0)[1] intercept
Total fit time: 10.310 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 8736
Model:               SARIMAX(0, 1, 5)   Log Likelihood              -76545.087
Date:                Mon, 11 Dec 2023   AIC                         153104.174
Time:                        23:34:28   BIC                         153153.700
Sample:                    01-01-2017   HQIC                        153121.052
                         - 12-30-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.2663     38.420     -0.007      0.994     -75.567      75.035
ma.L1          0.6816      0.009     74.509      0.000       0.664       0.700
ma.L2          0.2214      0.011     19.668      0.000       0.199       0.243
ma.L3          0.0613      0.017      3.505      0.000       0.027       0.096
ma.L4         -0.0452      0.017     -2.634      0.008      -0.079      -0.012
ma.L5         -0.0660      0.014     -4.616      0.000      -0.094      -0.038
sigma2      2.556e+06   2.66e+04     96.087      0.000     2.5e+06    2.61e+06
===================================================================================
Ljung-Box (L1) (Q):                   1.38   Jarque-Bera (JB):              7315.30
Prob(Q):                              0.24   Prob(JB):                         0.00
Heteroskedasticity (H):               0.55   Skew:                             0.93
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.08
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Summary and Conclusions

We found that the MLR resulted in the best performance of our data. However, this is not supposed to be the case, and given more time, and effort, a SARIMA model should be able to replace the MLR.

Appendix: Python Codes Developed for the FTP

```python
#%% ------------- Importing Packages -------------

print("------------- Importing Packages -------------")

#%% Importing Packages

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf
import statsmodels.tsa.holtwinters as ets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.tsa.api import SimpleExpSmoothing
from statsmodels.tsa.arima.model import ARIMA
import numpy.linalg as linalg
import random
import levenberg_marquardt as LM
from sktime.forecasting.arima import AutoARIMA
import pmdarima as pm
from pmdarima.arima.utils import ndiffs, nsdiffs
from sktime.forecasting.sarimax import SARIMAX

#%% ------------- Data Preprocessing -------------

print("------------- Data Preprocessing -------------")

#%% Importing Dataset, Printing head, # of obs&feats, list of num feats, and
NaNs

df_raw = pd.read_csv("powerconsumption.csv", parse_dates=["Datetime"])
print("First 5 rows:")
print(df_raw.head())

print(f'Number of observations: {len(df_raw)}')

all_features = df_raw.describe(include='all').columns.tolist()
print(f'Number of features: {len(all_features)}')

numerical_features = df_raw.describe(include=[np.number]).columns.tolist()
print(f'Number of numerical features: {len(numerical_features)}')
print("Numerical Features:")
for i in numerical_features:
    print(i)

print("Last 5 rows:")
print(df_raw.tail())

print("NA values in each column:")
```

References

- https://www.trade.gov/country-commercial-guides/morocco-energy
- https://sci-hub.se/10.1109/IRSEC.2018.8703007
- https://sps.columbia.edu/news/four-reasons-why-morocco-becoming-renewable-energy-powerhouse#:~:text=Unlike%20many%20other%20countries%20in,oil%2C%20gas%2C%20and%20coal.
- https://www.epa.gov/climateimpacts/climate-change-impacts-energy
- https://www.weather.gov/lmk/humidity
- https://www.jstor.org/stable/24861156
- https://doi.org/10.1016/B978-012370626-3.00015-6.