ASSIGNMENT 3

ANDREW ID: parmenin

18-787: Data Analytics

2/27/23

Niyomwungeri Parmenide ISHIMWE

I, the undersigned, have read the entire contents of the syllabus for course 18-787 (Data Analytics) and agree with the terms and conditions of participating in this course, including adherence to CMU's AIV policy.

Signature: **Niyomwungeri Parmenide ISHIMWE**
Andrew ID: **parmenin**

Full Name: **Niyomwungeri Parmenide ISHIMWE**

LIBRARIES USED

- import pandas as pd
- import numpy as np
- import matplotlib.pyplot as plt
- import statsmodels.api as sm
- from scipy.stats import ttest_ind
- from statsmodels.graphics.tsaplots import plot_acf
- from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error

QUESTION 1:

It was asked to load into the environment (Jupyter notebook), the intraday 15-minute energy demand data for the year 2014, and this was done using the read_csv function from pandas. After loading them, linear interpolation is used to fix day-light saving issues and missing values. Next, the (:00) seconds part is added to the "Time" column to have a valid time format, which helped to make a valid date time by adding the time to the date using pandas' functions i.e., to_datetime[1] and to_timedelta[2] to form a "Datetime" column to the data frame. That process helped to plot the time series of energy demand for every 15 minutes during 2014, and the following graph is generated.

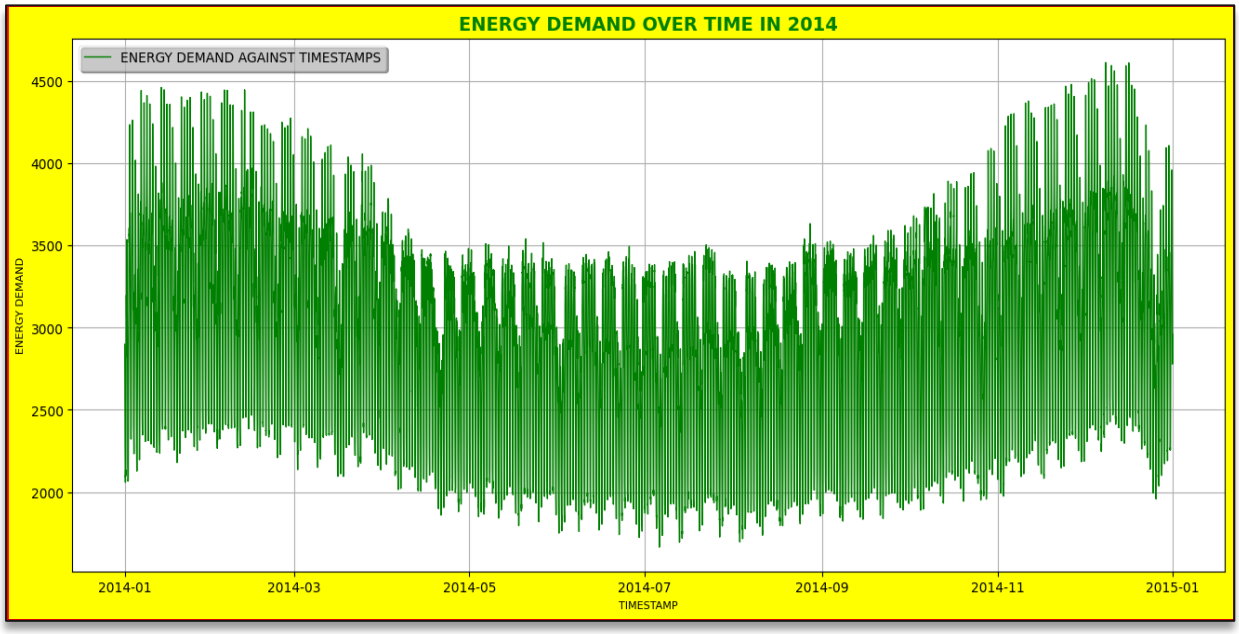

Figure 1: Energy demand against time series for every 15 minutes in 2014.

By examining the produced plot, it can be deduced that energy demand was high in the first three months from January to March, then declined in the following months from April to September, and then climbed in the last three months of 2014 i.e., October to December.

QUESTION 2:

It was now required to estimate the autocorrelation coefficients for ten days (960 lags) and plot the autocorrelation versus the lag on a ten-day axis. This was done by first calculating the autocorrelation using the statsmodels.api's function (sm.tsa.acf)[3] to visualize how the results look like, then the function plot_acf[4] is used to plot the autocorrelation against the lags with axis labeled in days where the matplotlib's xticks[5] function is used to give ticks for the arrangement of ten 10 days. That has given in the following graph.

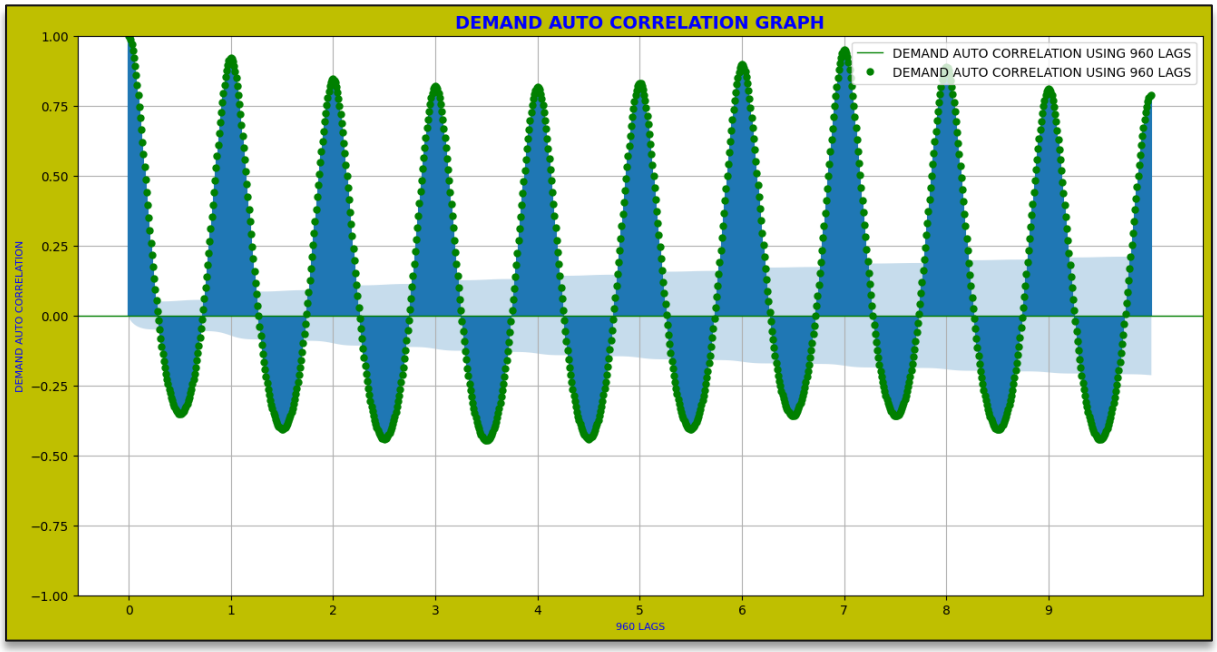

Figure 2: The autocorrelation against 960 lags for 10 days.

We can infer from the graph that strong positive correlations exist between consecutive observations since the positive autocorrelation values are close to 1.

There is also an indication of seasonality in the graph, as the observations see almost the same trend. Most data points in the autocorrelation are significant since they are located outside of the blue region.

QUESTION 3:

It was required to establish a time of year variable that ranged between 0 and 1 and to use a visual to depict how demand fluctuates throughout the year. This was addressed by changing to numeric values, the dates in the created "Datetime" column, and then normalizing to rescale the values into a range of [0,1] using the formula **Xchanged=(X−Xmin)/(Xmax−Xmin)**[6]. With the energy demand against the time of year, the variable is rescaled to the range between 0 and 1, the following plot is produced.

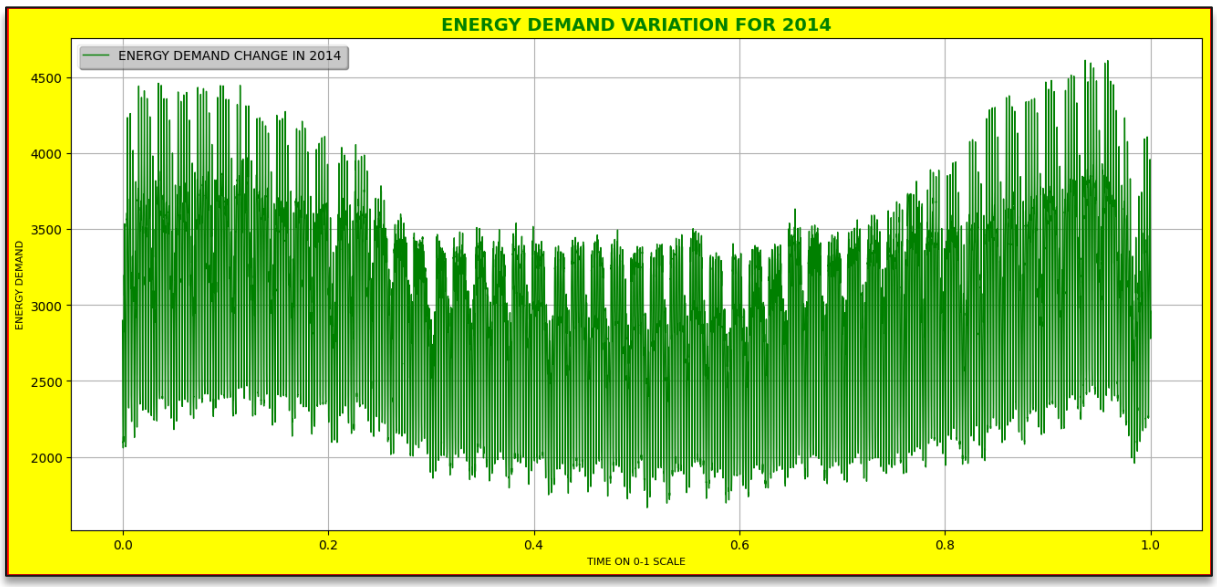

Figure 3: How demand fluctuates throughout the year.

By observing the above plot, it is possible to conclude that energy demand was high in the first three months of 2014, i.e., January to March, then fell in the following months, from April to September, and then increased in the last three months of 2014, from October to December.

QUESTION 4:

It is required to calculate the average demand for each of the 12 months of the year. This was done by firstly getting the names of the months we have in the data using the pandas series function called dt.month_name(), and then using the pandas' groupby function[6] to group demand for each month and then using the mean() function to calculate the monthly average which is depicted in the following table and bar chart.

Month	Demand	Numeric Date	Time Scale
January	3226.265793	1.389873e+18	0.042453
February	3247.410342	1.392422e+18	0.123277
March	3093.453629	1.394971e+18	0.204101
April	2844.273264	1.397606e+18	0.287665
May	2767.462030	1.400241e+18	0.371229
June	2695.365278	1.402876e+18	0.454793
July	2695.177083	1.405512e+18	0.538357
August	2681.552419	1.408190e+18	0.623291
September	2806.761111	1.410825e+18	0.706855
October	2933.151210	1.413460e+18	0.790419
November	3129.710764	1.416096e+18	0.873983
December	3219.443212	1.418731e+18	0.957547

Figure 4: The detailed average demand for each month of the year

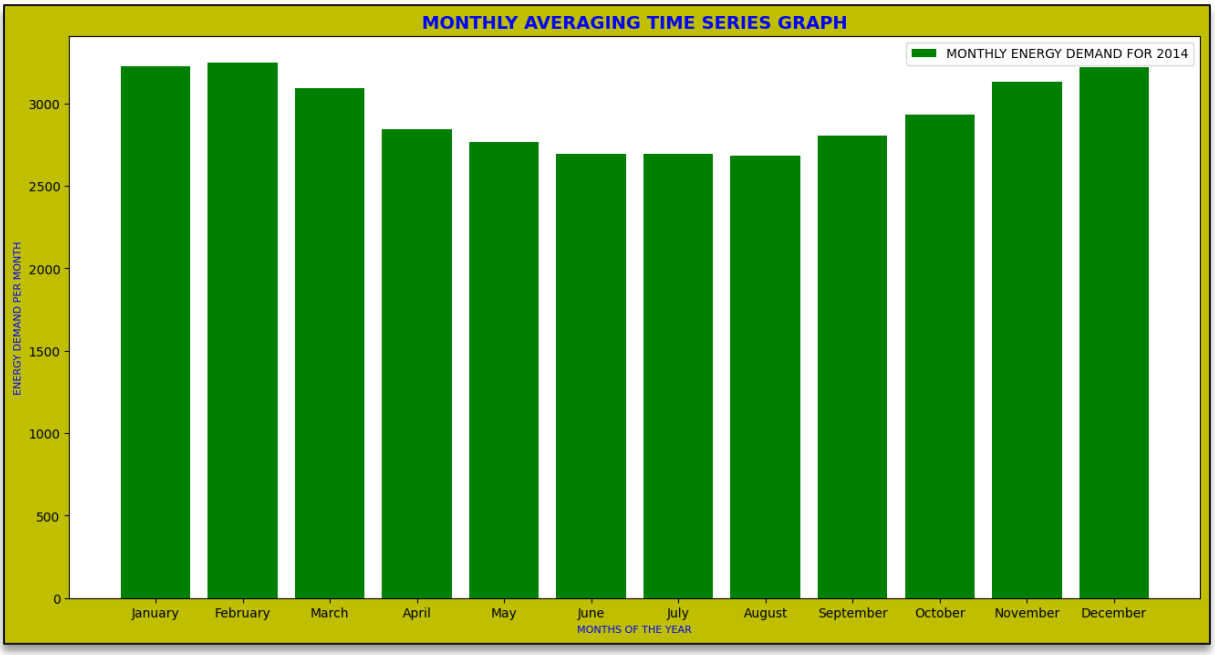

Figure 5: The average demand for each month of the year bar chart

From the above graphics, we can infer that the energy demand was higher in the first three months of the year, from January to March reaching its peak value of 3247 in February. After then, the demand slightly decreased between April to August where it reached its lowest value of 2695 in the summer (August). Furthermore, the demand again started to increase in the last four months of the year i.e., September to December.

QUESTION 5:

For this question, it was required to calculate the average demand for each hour of the day for the whole of the 2014 year. This was done by firstly getting the hour number in the day we have in the data using the pandas series function called dt.hour, and then using the pandas' groupby function to group demand for each hour of the day and then use the mean() function to calculate the hourly average of the day which are depicted in the following bar chart.

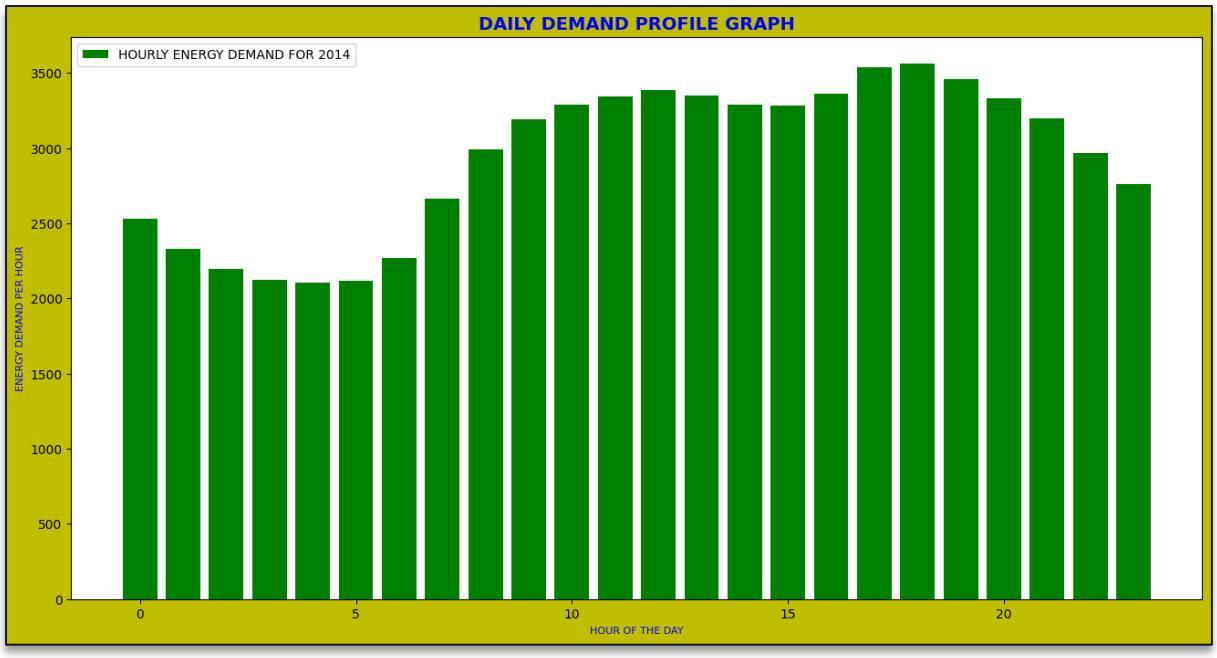

Figure 6: Hourly average or daily demand profile for each hour of the day in 2014.

We can infer from the graph that the energy consumption from late night to the morning (12 am to 5 am) is slightly low reaching its minimum of 2102 at 4 am, as people are asleep and many of the activities aren't operating at full capacity. After 5 am, energy demand starts to increase as people are ready to continue with daily activities at maximum reaching its peak consumption of **3562** at 6 pm before decreasing again.

QUESTION 6:

It was required to calculate and plot a bar graph of the average demand for each day of the week for the whole year. Again, this was done by firstly getting the day name in the weeks we have in the data using the pandas series function called dt.day_name (), and then using the pandas' groupby function to group demand for each day of the week and then use the mean() function to calculate the daily average of the week which are depicted in the following bar chart.

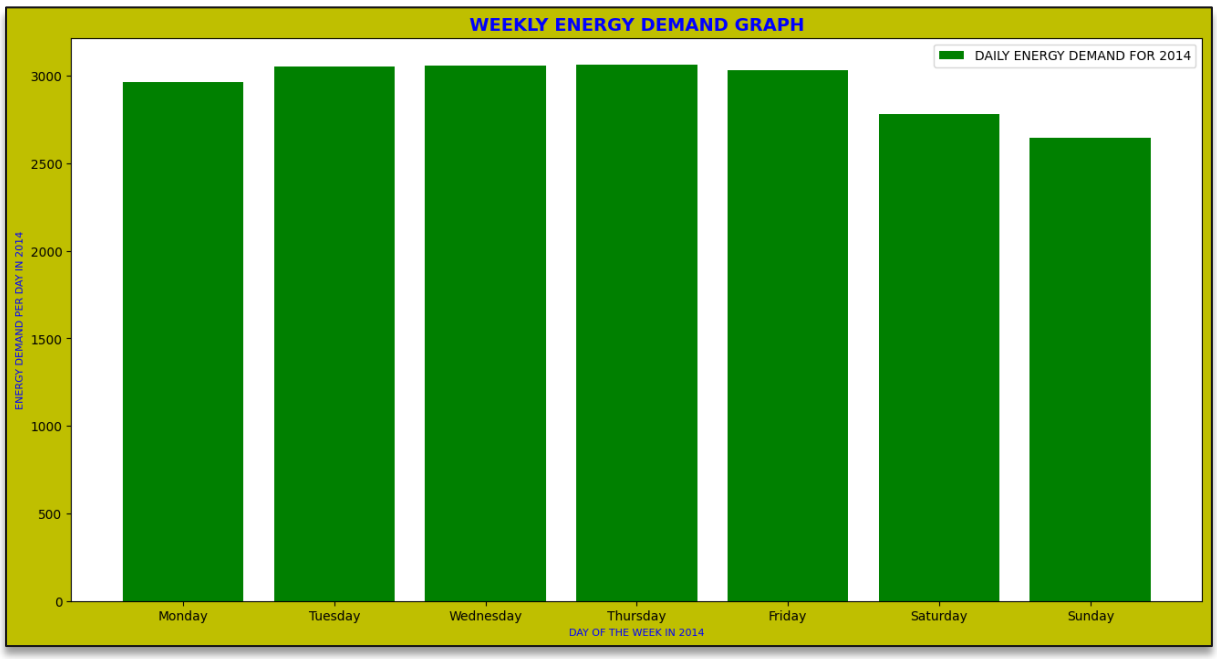

Figure 7: The average demand for each day of the week for the whole year

It can be inferred from the graph that the demand is higher on working days of the week compared to weekends. This can be due to the people working on weekdays and not that much on the weekends when offices and schools are closed which leads to lower consumption.

QUESTION 7:

It was required to calculate the daily demand profile for each day of the week by selecting a specific hour for each day and computing the average. This was done by firstly getting the hour number in the day we have in the data using the pandas series function called dt.hour, and then using the pandas' groupby function to group demand for each hour of the day and then use the mean() function to calculate the hourly average of the day and applying the pandas unstack()[7] function to get the variations of the daily demand profiles over 24 hours. With that, the following plot is produced.

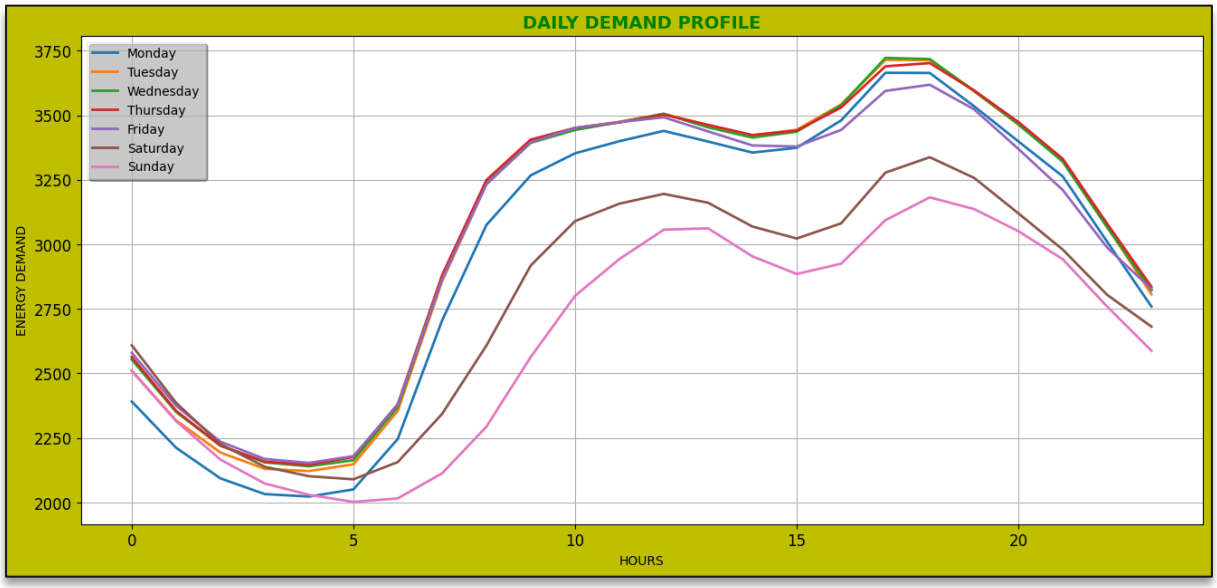

Figure 8: The daily demand profile variation for each day of the week

It can be noticed that the energy consumption between (00:00 pm-05:00 am) is somewhat lower; this is because people are sleeping and companies are operating but not at full capacity, therefore the energy demand is low. People are awake after 5:00 a.m., and everyday activities are underway, therefore demand grows until 8:00 p.m. when people retire for the night. As a result, energy demand reduces after this period.

QUESTION 8:

It was asked to determine whether there is a statistically significant difference between energy demand during the weekend and the weekday. This was done by dividing the data frame into data for weekdays and those weekends and then using the **ttest_ind** function from scipy.stats[8] that calculates the T-test for the means of two independent samples of scores. It helped to yield t-statistics and p-value.

The null hypothesis can be that there is no statistically significant variation in demand throughout the weekend (Saturday and Sunday) and during the working week (Monday through Friday). In addition, the alternative hypothesis can be that there is a statistically significant difference in demand throughout the weekend (Saturday and Sunday) and during the working week (Monday through Friday).

From the results of a t-statistics of **51.363399969318884** and p-value of **0.0**. The t-value of **51.363399969318884** demonstrates a significant difference between the means of the two samples, with the weekend demand sample mean considerably different from the working week demand sample mean. we can reject the null hypothesis since the p-value is less than **0.05**, then the alternative hypothesis is preserved. This means that there is a significant variation in demand between the weekend (Saturday and Sunday) and the working week (Monday through Friday) according to the alternative hypothesis.

QUESTION 9:

It was required to divide the data into two halves and use the second half for evaluation purposes while studying the simple benchmark forecasting approach known as persistence. This was done by first extracting the second half of the data from the original data frame and performing simple benchmark forecasting for each forecasting horizon within a day (horizon of 15 minutes, i.e., 96 horizons per day) using a loop, by using the **horizonsForec** list to store the forecasting horizons (i.e., 1 to 96) and **errMAEarr** list to store the mean absolute errors (MAE) for each forecasting horizon. After calculating the MAE, the following graph is produced.

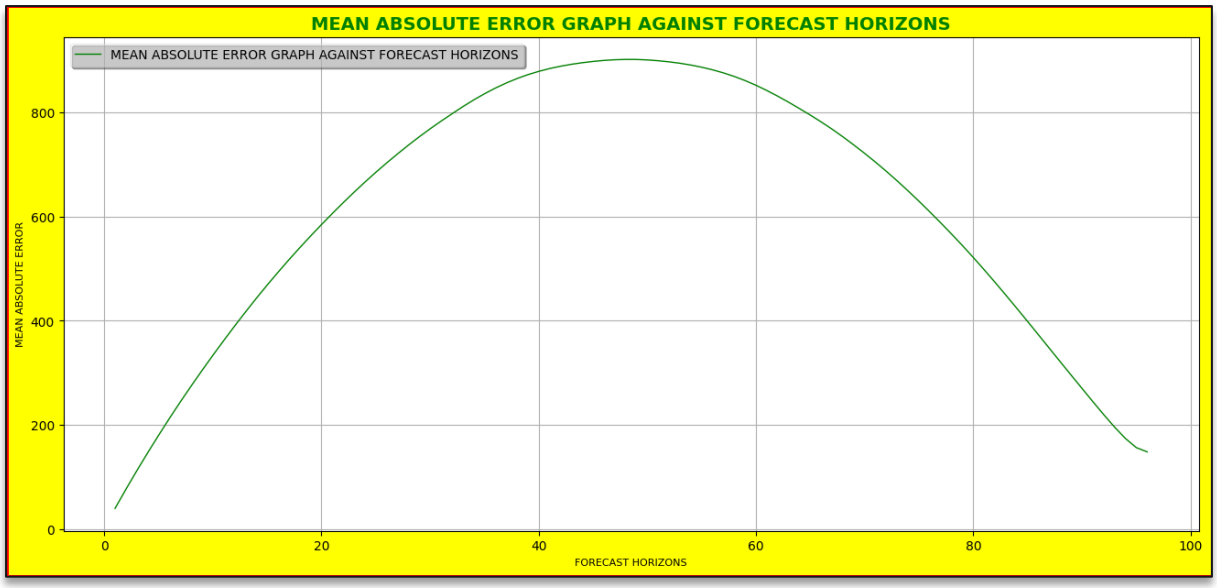

Figure 9: The MAE against forecast horizons for lead times up to one day ahead

By looking at the produced graph, we can infer that as the number of forecasted horizons increases, so does the mean absolute error (MAE) until it reaches the middle of the day at its peak, and then after the middle, MAE begins to decrease as the value of forecasted horizons increases. Therefore, the forecasting horizon with the smallest MAE (the first and the last on the graph above) is the best-performing one, as it produces the most accurate predictions for future demand.

QUESTION 10:

It was required to calculate the mean absolute percentage error for the persistence and plot this against the forecast horizon up to one day ahead. To this, the previous approach is used and instead of calculating the mean absolute error (MAE), the mean absolute percentage error (MAPE) is calculated. After calculating the MAPE, the following graph is produced.

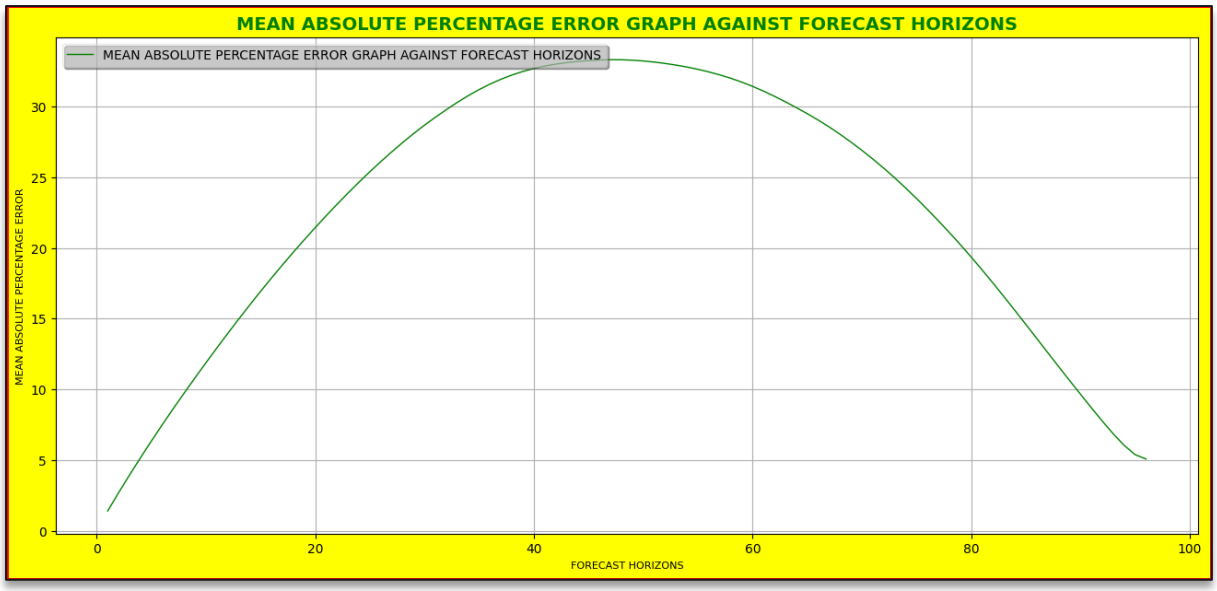

Figure 10: The MAPE for the persistence against the forecast horizon up to one day ahead

After plotting the above graph, we see that as the value of forecasted horizons grows, so does the mean absolute error (MAPE) until it reaches the middle of the day at its peak, and then after the middle, MAPE begins to decrease as the value of forecasted horizons increases. Generally, a lower MAPE indicates a more accurate model, which means that for the above plot, the first and the last forecasting horizons can produce more accurate results than others.

REFERENCES

[1] ‘pandas.to_datetime — pandas 1.5.3 documentation’. https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html (accessed Feb. 21, 2023).

[2] ‘pandas.to_timedelta — pandas 1.5.3 documentation’. https://pandas.pydata.org/docs/reference/api/pandas.to_timedelta.html (accessed Feb. 21, 2023).

[3] ‘statsmodels.api.tsa.acf Example’. https://programtalk.com/python-more-examples/statsmodels.api.tsa.acf/ (accessed Feb. 12, 2023).

[4] K. Drelczuk, ‘ACF (autocorrelation function) — simple explanation with Python example’, *Medium*, May 15, 2020. https://medium.com/@krzysztofdrelczuk/acf-autocorrelation-function-simple-explanation-with-python-example-492484c32711 (accessed Feb. 12, 2023).

[5] ‘matplotlib.pyplot.xticks — Matplotlib 3.7.0 documentation’. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html (accessed Feb. 21, 2023).

[6] ‘Python | Pandas dataframe.groupby()’, *GeeksforGeeks*, Nov. 19, 2018. https://www.geeksforgeeks.org/python-pandas-dataframe-groupby/ (accessed Feb. 21, 2023).

[7] ‘Pandas DataFrame: unstack() function’, *w3resource*, Aug. 19, 2022. https://www.w3resource.com/pandas/dataframe/dataframe-unstack.php (accessed Feb. 21, 2023).

[8] ‘scipy.stats.ttest_ind — SciPy v1.10.1 Manual’. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html (accessed Feb. 25, 2023).