**FACULTY OF ENGINEERING, COMPUTING AND THE ENVIRONMENT**

**School of Computer Science and Mathematics**

**MSc DEGREE**

**IN**

***Data Science***

Name: Kameswari Naga Bhargavi Kalaparty

ID Number: K2175468

Project Title: Timeseries Energy analysis and forecasting using SMART meter energy data

Date: 18th September 2023

Supervisor: Professor Farzana Rahman

KU London Logo

**WARRANTY STATEMENT**

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data or conclusion of the work

Contents

[List of Figures: 3](#_Toc145948398)

[List of tables: 5](#_Toc145948399)

[Acknowledgement: 6](#_Toc145948400)

[Introduction: 7](#_Toc145948401)

[Abstract: 7](#_Toc145948402)

[Motivation and Background: 7](#_Toc145948403)

[*Research Problem:* 8](#_Toc145948404)

[Aim: 8](#_Toc145948405)

[*Objective:* 8](#_Toc145948406)

[*Summary of thesis contributions:* 9](#_Toc145948407)

[*Thesis Outline:* 10](#_Toc145948408)

[Literature Review: 10](#_Toc145948409)

[Related Theories: 10](#_Toc145948410)

[Review and analysis of related research works: 25](#_Toc145948411)

[Contribution: 28](#_Toc145948412)

[Methodology: 28](#_Toc145948413)

[Model Building Experiments: 42](#_Toc145948414)

[Evaluation and Result Discussion with forecasting: 57](#_Toc145948415)

[Conclusion: 62](#_Toc145948416)

## List of Figures:

* Figure 1: Project Flow
* Figure 2: Components of Time series
* Figure 3: Sample Time Series Decomposition
* Figure 4: Example ACF plot
* Figure 5: Example PACF plot
* Figure 6: RNN Unit
* Figure 7: RNN Architecture
* Figure 8: Forget Gate
* Figure 9: Store gate
* Figure 10: Update gate
* Figure 11: Output gate
* Figure 12: LSTM Model
* Figure 13: BLSTM Model
* Figure 14: MSE Formula
* Figure 15: RMSE Formula
* Figure 16: Methodology of reference paper
* Figure 17: Data Frame of Energy Data
* Figure 18: Quarterly and Monthly visualisation of Power consumption
* Figure 19: Null Values
* Figure 20: Indexed Data Frame
* Figure 21: Seasonality
* Figure 22: Stationarity test result
* Figure 23:Seasonal Decompose using STL
* Figure 24:Seasonal Decompose using Hodirick-Prescott
* Figure 25: Seasonality using seasonal decompose
* Figure 26: ACF and PACF Plots
* Figure 27: EWMA
* Figure 28: DES Additive
* Figure 29: DES Multiplicative
* Figure 30: TES
* Figure 31: Data after Scaling
* Figure 32: Holt-winter’s graphs
* Figure 33: Auto Arima Summary
* Figure 34: Auto Arima Step-wise fit
* Figure 35: Auto Arima Summary with Seasonality
* Figure 36: ARIMA (2,0,2) Summary
* Figure 37: ARIMA (2,0,2) Prédictions on test
* Figure 38: ARIMA (2,0,2) Prédictions Plot
* Figure 39: ARIMA (4,0,1) Prédictions Plot
* Figure 40: Seasonal Component Plot and PACF
* Figure 41: SARIMA plot predictions 1
* Figure 42: SARIMA plot predictions 2
* Figure 43: SARIMA plot predictions 3
* Figure 44: SARIMA plot with X factor
* Figure 45: VARMA plot 1
* Figure 46: VARMA plot 2
* Figure 47: EPOCH Plot
* Figure 48: LSTM Plot
* Figure 49 : LSTM Plot2
* Figure 50 : VARMA Backtesting
* Figure 51 : LSTM Backtesting
* Figure 52 : RSME VARMA
* Figure 53 : ARIMA Forecasts for future
* Figure 54 : SARIMA Forecasts for future
* Figure 55 : VARMA Forecasts for future
* Figure 56 : Linear Regression Forecasts for future

## List of tables:

* Table 1 :Example data set 1
* Table 2 :Example data set 2
* Table 3: performance Table of models in paper
* Table 4: Software Requirements
* Table 5: Hardware Requirements
* Table 6: Energy Data Variables Description
* Table 7: P values associated with features after the Granger Causality test
* Table 8: Holt-winter test predictions
* Table 9: Models RSME
* Table 10: Predictions table for future by best model

## Acknowledgement:

This comprehensive work of energy demand forecasting wouldn’t be possible without constant guidance, support and encouragement of my mentor and supervisor Professor Farzana Rahman. Would also like to take this opportunity to thank my university, my professors and my supervisor to let me have this wonderful opportunity of presenting my take on the energy forecast and demand in UK.

## Introduction:

### Abstract:

Uk’s rising energy demand and consumption is at its most significance currently. This calls for best management that one can give economically and eco-consciously is predicting what is causing the need and what can be expected of the future. This is where data comes in place, when one needs to predict the future they need to understand the past, the past data in the discussion is nothing but the energy consumption that happened in UK till date. SMART meters in homes now capture these details appropriately and effectively not only recording the consumption but also noting down relative factors like temperature and heating day information. Its now in our hands and the energy companies best interest to utilize this data and make meaningful predictions and analysis to help the future of UK’s energy consumption and demand.

### Motivation and Background:

The energy landscape is expected to go into a major transformation with the introduction of smart meters in UK. The are several expectations and changes that are happening in this field. All these swaps and projections are predominantly dependent and revolving around one particular resource which is ‘data’. Data is fuel for the future of any industry and this is not very different when it comes to energy. In the below section we are going to look at the reason behind energy forecasting how and why it is done in detail.

SMART Meters:

SMART meters are advanced devices that are providing reliable and instantaneous data on energy consumption. Data is the major difference between traditional and smart meters. Unlike the former, the latter meter records energy consumption and provides records of energy consumption on a more frequent time basis like hourly consumption. This SMART meter initiative, especially in UK is a free of cost installation service from the energy companies introduced by the government. [1]

What is the significance of the data produced by these SMART meters?

SMART meter data helps in promoting energy awareness by allowing consumers and energy companies to access and monitor their energy consumption which leads to less wastage.

Secondly, this data helps in transparency and budgeting. As the customer is able to track energy he/she can easily identify what appliance is making use of high volumes of energy and when and make financially and ecologically smarted choices.

As discussed the level of energy consumption detection is very high in these meters, hence this can be put to good use to understand anomalies in data thereby detecting issues like malfunctions or energy thefts by uncovering unusual patterns in the energy consumption data.

how does it relate to the objectives of this project?

The data produced by SMART meters is a granular level data. This data is timestamped to every hour at least, this detail equips the user of the data a very detailed understanding on energy which is quite vital for building good energy forecast models like timeseries forecast models. Timeseries forecast models unlike other models help in understanding and forecasting the data based on temporal dependencies, seasonality, stationarity etc.

The SMART meter energy data helps us not only with information on energy usage but also when was this energy used and in what conditions or factors did this consumption happen in. This kind of knowledge helps the user of the data to understand and visualize the relationship between features and their effects on energy consumption. Also helps in forecasting, peak and off peak periods and their fluctuations.

### *Research Problem:*

The foundation of any analysis or study related to energy consumption lies in data availability. In the case of SMART meters, a relatively recent introduction in the UK, the data pool is still in its infancy, with some households yet to adopt SMART meters. As a result, the quantity of data available for essential analysis and forecasting remains limited. Despite the acknowledged advantages of employing high-complexity time series models, including neural networks, for handling time-ordered data, this specific project takes a diverse approach. It spans across various modeling techniques, ranging from straightforward moving average models to sophisticated neural networks. The primary objective is to assess and determine which approach is most effective for the unique SMART meter data landscape, considering its nascent nature.

### Aim:

The first and foremost aim of this project is to improve the accuracy of energy forecasting by using different modelling techniques for our SMART meter dataset.

Secondly, to evaluate the suitability of simple traditional forecasting models and modern/ complex approaches in the context of our dataset.

### *Objective:*

In order to achieve the above mentioned aims of the project there are certain objectives that need to be fulfilled.

A diagram of a flowchart

Description automatically generated

**Figure 1: Project Flow**

The first step includes collection of SMART data from the most recent time period available and then cleaning and pre processing it for appropriate analysis. Visualize the data and further understand the features and their effects on consumption and select the ones most appropriate for this research. Next, checking the stationarity of data. Also, plotting auto correlation and patrial auto correlation plots and gathering the required statistic inferences. Now, selecting a range of forecasting models appropriate for the dataset. Next, develop and finetune these models by adjusting the parameters and hyperparameters accordingly. After that, compare the models in terms of their statistical evaluations and efficiency of forecasting. Next, visualize these results and make appropriate recommendations.

### *Summary of thesis contributions:*

This project provides in depth literature review in terms of energy and timeseries data, machine learning and neural network models associate with respect to timeseries data.

It also provides in depth understanding of SMART meter dataset and its components using different visualizations supporting and answering certain research questions on the dataset.

This study also helps in understanding the entire process involved in time series data analysis and forecasting. Starting from feature understanding and selection, statistical analysis involved, model selection using the statistical inferencing, evaluation of models tested and forecasting results and comparing the error, thereby concluding the right model for the dataset..

### *Thesis Outline:*

The project involving the time series analysis and forecasting of SMART meter data is documented by the following chapters:

Title: Energy Forecasting using SMART Meter Data: Models, Analysis, and Optimization

Chapter 1: Introduction

This chapter provides a quick overview of the entire study involving concepts associated with the project such as motivation, background, research problem and its associated aims and objectives and finally structure of the study.

Chapter 2: Literature Review

This chapter reviews all the concepts necessary for an energy forecasting project in SMART meter time series dataset. Some concepts namely, timeseries energy data, types of time series models, theory of correlation plots used in the modelling, theory of statistical methods used in modelling, evaluation metrics. This section also includes a discussion on similar previous work and existing studies and tries to identify the gap and fill it.

Chapter 3: Contribution

This chapter is exactly where the planning comes to practice. Firstly, visualizations based on research questions are discussed to provide a clear understanding and initial analysis of the SMART energy data. Next, the correlation plots and statistical methods that are discussed are implemented on the dataset. The selected models from above are implemented on the dataset in this section. This part also covers a minor comparison and understanding of the models.

Chapter 4: Conclusion

This chapter summarizes the entire outcome of all the approaches used in chapter 3, further reflects on them to chose the best possible model for relevant dataset. Also justifies if the objectives are achieved and gives a scope and understanding on further research.

## Literature Review:

This section will be looking at different theoretical information regarding the concept of timeseries which will be a support to our study. It will also be discussing similar contributions and works of scholars in the subject of timeseries forecasting.

### Related Theories:

#### Time Series Energy Data:

Timeseries data usually refers to collection of certain data values over a period of time with successive intervals. The intervals could be in any time frame, such as, hours, days, months or years and it could also be continuous. The energy consumption data collected over certain time intervals is nothing but time series energy data. Apart from the measure of energy consumption, this data might also have its related features such as relative temperature on the same day, measurements of house that the data is collected from etc. are few examples. By analyzing such information, one would get useful insights into energy usage patterns and trends, which can result in energy management.

#### Types of Timeseries Data:

Time series data is specifically divided into two types and the major difference between them is of course the time. To explain this in detail, lets look at the two types of time series data we have:

Discrete time series: As the name suggests, the data recorded in accordance to regular intervals in time, such as hours, days, months and years. Hence the measure of time is distinct in this case. Below is an example of discrete data. The time column signifies the distinct split of days in January month, and their associated consumption of energy.

|  |  |
| --- | --- |
| Time | Power in Kw (Kilowatt) |
| 01-January-2022 | 15 |
| 02-January-2022 | 14 |
| 03-January-2002 | 15 |

**Table 1 :Example data set 1**

Continuous time series: It is important to note that, continuous data is also a collection of data based on time points but denser. Below is an example of continuous data, it is also a collection of power in Kw but closer time stamps, of just one day 1st January. Sometimes for analysis, this data is resampled to the table above for better understanding and plotting. [2]

|  |  |
| --- | --- |
| Time | Power in Kw (Kilowatt) |
| 01-January-2022 00:00 | 15 |
| 01-January-2022 01:00 | 15 |

**Table 2 :Example data set 2**

#### Memory of timeseries:

In timeseries memory refers to how the past values in time influence the future data. This understanding helps in analyzing the data in better ways. There are two kind of influences that the past data has on future data in a timeseries setup.

If the past data is quite indicative of future, the data set has a stronger memory calling it long memory in timeseries.

On the other hand, if the past data doesn’t get to impact the future much, then the data is called short memory in timeseries. A point to be noted is that when a dataset associates to short memory, it is difficult to model it make predictions for the unknown future.

How do we measure the memory?

Correlation as a statistical concept helps in understanding how variables relate to each other. Auto correlation plots are the correlation coefficients of different variable in data sets plotted over different timestamps and lags, these are discussed more in further detail in later sections. For now, it needs to be understood that if the graphs drop rapidly at points where we increase the time difference, that means that the variables in dataset are independent and the nature of the data’s memory is short term. However, the rate at which this decay occurs is also very important, if it declines at exponential rate it is exponential decay but some series though they are stationary, they decay at a linear rate which is little slower than exponential one. Which works on the advantage to model accordingly for better forecasts. [3]

#### Components of timeseries:

Causes behind the value of an dependent variable in timeseries are called the components of time series.

A diagram of components of a trend

Description automatically generated**Figure 2: Components of Time series**

There are majorly 4 components in a time series :

1. Trend
2. Seasonality
3. Cyclic
4. Random or irregular movements

Trend:

This usually denotes the overall tendency of data. It is a more smoother of a visualization rather than original. There is no one path that this line follows, it could go upward or downward or fluctuate between them depending on the dataset itself. Most of the timeseries we observe in real world are nonlinear.

Seasonality:

The data in time series is recorded over a span of time, that could be hours, days, months or years. The data which shows same variations over whatever period of time mentioned above is due to the seasonality component in it.

These variations could be anything from natural sources or man-made. Natural can be something like weather, this could effect a sales data set of ‘waterproof apparel’ as people tend to buy these more in rainy season. So may be every rainy season, in every year, the sales of these products will be more compared to other seasons.

Cyclic:

One complete period cycle is called cyclic. The variations observed in data recurring over a year are because of cyclic. These variations are more repetition than periodic.

Random:

As the name suggests, random denotes the random fluctuations in the dataset. Having more domain knowledge about the data set helps in understanding these variations.

Math behind the timeseries:

yt = f (t)

In the above equation, ‘y’ denotes the dependent value recorded in relation to the timeseries ‘t’.

*Additive model:*

yt = Tt + St + Ct + Rt.

Tt, St, Ct, and Rt are the trend value, seasonal, cyclic and random fluctuations at time t respectively. The additive model defines that the trend of the value variable depends on the as the sum of the trend, seasonal, cyclic and random fluctuations together over a time-period t.

*Multiplicative Model:*

yt = Tt × St × Ct × Rt

Similar to additive model, Tt, St, Ct, and Rt are the trend value, seasonal, cyclic and random fluctuations at time t respectively. This model defines that the trend of the value variable depends on the equal proportionality of the trend, seasonal, cyclic and random fluctuations together over a time-period t. [4]

Decomposition of Timeseries:

The statistical method involved in identifying the trend, seasonality and noise, diving them separately and visualising them is called decomposition of timeseries.

This decomposition of timeseries into factors helps one understand the data better and thereby apply better forecasting models.

Every timeseries has two different components: Systematic and non-systematic. Systematic components can be described and modelled including the trend, seasonality and noise.

Now we assume that the components are additive or multiplicative. There are methods in python which automatically perform the equations associated with additive and multiplicative decomposition discussed in the above section. The statsmodels library holds a function called seasonal\_decompose() which does the job, the only thing one needs to do is to mention if the decomposition is additive or multiplicative. [5]

A graph of a graph

Description automatically generated with medium confidence

**Figure 3: Sample Time Series Decomposition**

The above could be a potential output of some random dataset when the seasonal\_decompose() function is used. The data clearly shows a trend which is mostly linear as it increases gradually. However, it can be observed that there is no seasonality to the data or residual.

#### Stationarity in timeseries:

Time series estimates run on an statistical assumption that the data of the timeseries is always constant. In simpler terms, statistical properties like mean and variance remain constant overtime. Now the question is for this to be true the data set in no way should be having any effects of trend and seasonality effecting its values. However, in real world almost all timeseries data have some trend or seasonality effects, when it does it needs to be managed to make valid modelling and predictions.

Stationarity of the data can be related to skewness, if the data when plotted depicts a bell that pretty much means the mean of the data is well distributed around the range and the data is stationary. [6]

Tests for checking the stationarity in data:

First and foremost we look at plots, anything that depicts a bell like curve is stationary.

Also, functions like summary statistics can be called and checked if there is similarity between the statistical values of data such as mean and variance.

Finally we can perform certain tests to understand if the data is stationary, this method is more reliable as we get a quantitative results which can be compared and conclusions can be drawn appropriately rather than visualising and making conclusions over what the eye can see. One such test that can be performed is Augmented Dickey Fuller Test.

Stationarity Tests:

This test basically interprets if the null hypothesis can be rejected or accepted. It calculates the unit roots by using an auto regressive model by applying multiple log values.

H0= The null hypothesis is that if the timeseries has the unit root means that the hypothesis is null and the data is non- stationary.

H1= If it is the other way around the null hypothesis is rejected and data is stationary.

This is interpreted by looking at the p-value, if p-value > 0.05 it is H0 and if p-value <= 0.05 it is H1.

The statsmodel library provides a function called adfuller(X) that can be used to implement the dickey fuller test. X denotes the input series.

There are 2 more tests called KPSS and PP test which similarly check for stationarity by looking for p-value. Will be discussed further in contributions.

#### Literary Analysis of Models for timeseries:

There are many number of timeseries models that can be used to predict data. Also, apart from just time series specific models, some generic machine learning models like k-means give amazing results on certain timeseries data. However, in our case scenario we will be completely focussed on adapting only models specific to timeseries data and will try to see which works better at predictions.

Depending on the variables involved in the analysis of the models, the timeseries of the models are divided into two types: Univariant and Multivariant Models.

As the name suggests, univariant can consider only one variable, its observation with respect to the timestamps. On the other hand, multivariant can consider one dependent variable and the effect of other variables on this variable.

#### Moving Average models:

The stepping stone into timeseries modelling are the moving average models. The way it works is pretty simple, it just averages out the recent data points and repeats the process for each time point to estimate the future values.

Exponentially Weighted Moving Average Model:

EWMA is univariant. This model is a minor expansion of the simple moving average models. This model is predominantly applied not to really make a forecast but just to understand how modelling works and the moving average factor of it, since this factor is going to be one of the components for a higher model that we are going to discuss in later sections. Moving average part of the model calculates the average of the data points with a specified value, denoting the number of lags for the calculation. Depending on the timeseries data this usually varies between couple of months to years. In this model the moving average of the newer observations are given more weight than the older values in the dataset.

EWMAt = Alpha \* rt +(1-Alpha) \* EWMAt-1

Alpha is the weight decided by the user, r is the input.

This is a recursive model, as the predicted value is depending on the historical values of the dataset.

There is function in stats models called x.ewm(span=12,adjust=False), which is just one line function performing the above mathematical formula. X denotes the input value, span denotes the time laps for calculating the moving average.

Holt-Winters Method:

Holt winters is a modelling method (univariant) which is going to take into account the seasonality and trend component of the dataset, unlike the moving average model discussed above. The Holt Winters seasonal method consists of one forecast equation and three smoothing equations. One of the smoothing equations is for the level component denoted by lt, second for the trend component called bt and last for the seasonality denoted by st, and they correspond to three smoothening parameters namely α, β, γ.

This method further has two types namely additive and multiplicative which are very similar to the ones discussed in seasonal decompose section of the report. The former is used when the seasonal component is constant throughout the series, and the latter is used when the seasonality is changing.

Base method: Single Exponential Smoothing : y0=x0, yt= (1- α)yt-1+ αxt

This is going to model, y at sometime t, is equivalent to 1- α term at yt-1(historical value) addition to α at xt.

Holt’s Method: Double Exponential Smoothing: Here a new smoothing factor β is introduced, that is related to the trend component of the data.

lt= (1- α)lt-1+ αxt, lt for level at time t

bt= (1- β)bt-1+ β (lt-lt-1)

fitted model - yt=lt + bt

forecast model – yt+h= lt+hbt (h is periods into future)

Holt-Winters Method: Triple Exponential Smoothing: Here a new smoothing factor γ is introduced, that is related to the seasonality component of the data.

lt= (1- α)lt-1+ αxt

bt= (1- β)bt-1+ β (lt-lt-1)

ct= (1- γ)ct-L+ γ (xt-lt-1-bt-1) {L denotes divisions per cycle}

fitted model - yt=(lt + bt )c

forecast model – yt+m= (lt+mbt) ct-L+1+(m-1) (m is periods into future) [8]

ARIMA:

ARIMA stands for Autoregressive Integrated Moving Average (univariant). These are nothing but 3 components that stand for ‘AR’- Autoregressive Component, ‘I’- Integrated Component and ‘MA’ Moving Average. These models have a linear trend. Point to be noted, ARIMA models are usually used to built a lot more different models. For example, when there is a seasonal component included, we can build SARIMA which is an extension of ARIMA further discussed in later sections.

ARIMA most of the times is applied to the data which is non- stationary. I here denotes the differencing step which is applied one or more times to eliminate the non-stationary nature of the data.

Non-seasonal ARIMA models are denoted by ARIMA (p,d,q) which are non-negative integers.

AR(p) component: Corresponds to the parameter p. Auto regression is nothing but a regression model which utilizes the relationship dependency between current and historical values.

I(d) component: Integrated component. This differences the values (subtracts the current value with a value from history one step at a time), this will in result make the non-stationary data stationary. So the I component is number of times the data is differenced to make it stationary.

MA(q) component: A model that uses the difference between the value and its error from a moving average model applied to lag observations.

The mathematical function behind this forecasting is given by:

Where the values are represented as:

* e is the error term
* c is the constant
* p is the AR lagged value
* q is the MA error value
* d is the differenced value

As discussed in above sections, stationary data has same mean and variance over the time, hence these models predict values whose mean and variance will be same overtime in future as well. To decide the stationarity, the Augmented Dickey Fuller test which was discussed in above sections is used to check the stationarity nature of the data. If the stationarity is positive, the value of q is going to be zero as we perform no differencing.

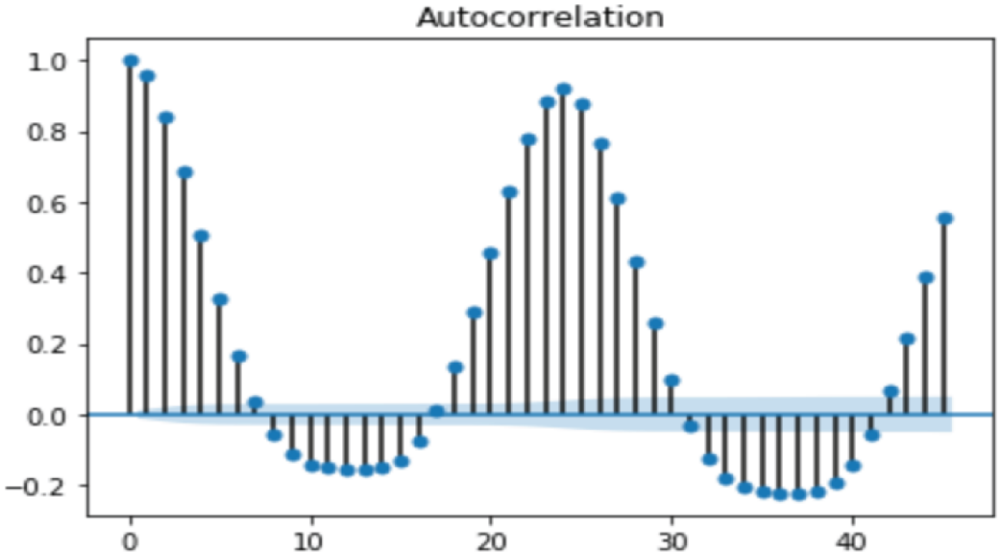
All these differencing techniques and the other values of P and Q are automatically calculated by the statsmodel library by using autoarima() function, when you run this function it reports backs with what values of p,d,q should be and can have better predictability. Apart from this inbuilt functions these values can be predicted from correlation plots.

Auto correlation function and plots:

Auto correlation function is nothing but, in a timeseries the observations that came before are used to compare the ones that came after. In other words, it refers to the degree to which the current time series and the delayed time series are comparable to one another. The greater the degree of correlation that exists between an output and a certain lagging variable, the greater the importance that autoregression places on the variable. If this is the case, the variable is said to have a high degree of predictive power. Finding hidden patterns in the data, such as seasonality and trends, may be made much easier by using autocorrelation, the graphs below can help us figure out the p,d,q values associated with ARIMA and its related models.

Auto Correlation Plot:

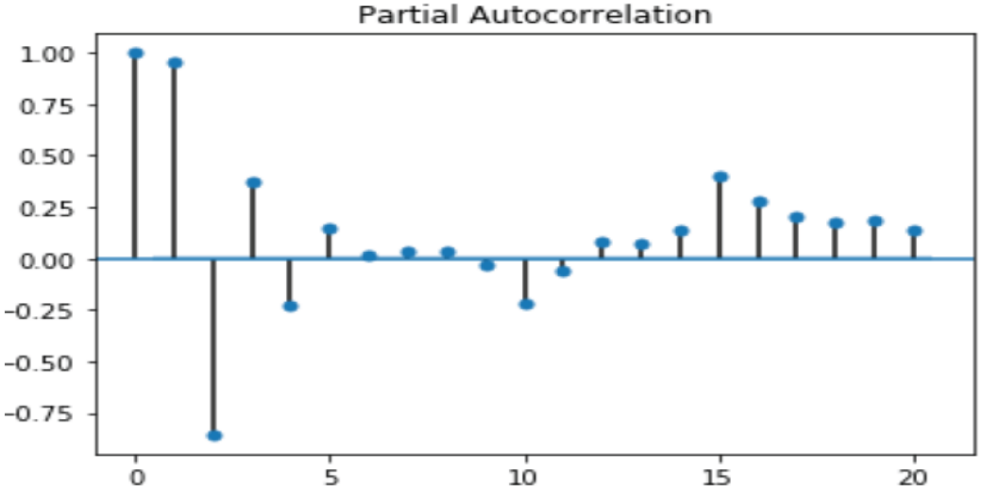
As discussed, the correlation between lagged and original timeseries is shown by ACF plot. The ACF at lag 0 is always 1 because it represents the correlation of the time series with itself at the same time point. The ACF at other lags indicates how much the current value of the time series is correlated with its past values at various lags. If the plot has a sudden drop after few lags, it means that it has non seasonal MA process. If there is a spike at first and a sharp drop it suggests non seasonal AR. The fig below shows an ACF with seasonality.



**Figure 4: Example ACF plot**

Partial Auto Corrélation Plot:

This as well shows the partial auto correlation of the timeseries and its lags however, considers intermediate lags as well. But instead of showing the correlation between current values it shows the correlation between residuals with next lags. Also, it is a function that estimates the additional advantage that comes from adding one more latency. The fig below shows an PACF plot.



**Figure 5: Example PACF plot**

Depending on the spikes and drops of the dataset’s correlation plots, the corresponding p,d,q values are calculated.

SARIMA:

This model is univariant and a upgrade from ARIMA. The only difference is that this model considers the seasonality factor in it, hence the name Seasonal Auto-Regression Intergrade Moving-Average Model.

ARIMA accepts value (p,d,q) and SARIMA along with these accepts an additional set of factors (P,D,Q,m)- these describe the seasonal component of the model. Very similar to p,d,q, the P,D,Q represents the seasonal regression, differencing and moving average coefficients respectively. The m stands for number of datapoints in seasonal cycle. So in case of a monthly data, with early seasonal cycle the value of m will be 12.

Where the following values are represented by:

* p= non-seasonal AR order
* d= non-seasonal differencing
* q= non-seasonal MA order
* P= seasonal AR order
* D= seasonal differencing
* Q= seasonal MA order
* S= seasonal pattern time span

When we implement this using statsmodels library, we import, SARIMAX in which X supports exogenous variable. The exogenous variable is an extra set of variable in the dataset whose values in regards to the future time series are already know and we can utilise these values along with the dependent variables historical values to make predictions on this variable of data.

VAR- Vector Auto Regression Models:

These models can be used for multivariant data. As explained in the previous section the forecast variable was effected by exogenous variable when future values are known, but not vice-versa. When variables can effect each other, that’s when these models come into play. Each variable has an equation which explains its evolution based on its own lags, the lags of other variables and error term. This models basically needs a list of variables that effect each other that can be hypothesised. The text book, Forecasting: Principles and Practice has a case where changes in personal consumption expenditures 𝐶𝑡 were forecast based on changes in personal disposable income 𝐼𝑡. [9]

The equation for AR is

𝑦𝑡=𝑐+𝜙1𝑦𝑡−1+𝜙2𝑦𝑡−2+⋯+𝜙𝑝𝑦𝑡−𝑝+𝜀𝑡

c- constant

𝜙 values and their lagged values.

A 𝐾-dimensional VAR model of order 𝑝, **VAR(p)**, has variable 𝑦𝐾.

The equations for a 2-dimensional VAR(1) model is:

    𝑦1,𝑡=𝑐1+𝜙11,1𝑦1,𝑡−1+𝜙12,1𝑦2,𝑡−1+𝜀1,t

    𝑦2,𝑡=𝑐2+𝜙21,1𝑦1,𝑡−1+𝜙22,1𝑦2,𝑡−1+𝜀2,t

Equations for a 2-dimensional VAR(3) model is:

    𝑦1,𝑡=𝑐1+𝜙11,1𝑦1,𝑡−1+𝜙12,1𝑦2,𝑡−1+𝜙11,2𝑦1,𝑡−2+𝜙12,2𝑦2,𝑡−2+𝜙11,3𝑦1,𝑡−3+𝜙12,3𝑦2,𝑡−3+𝜀1,t

    𝑦2,𝑡=𝑐2+𝜙21,1𝑦1,𝑡−1+𝜙22,1𝑦2,𝑡−1+𝜙21,2𝑦1,𝑡−2+𝜙22,2𝑦2,𝑡−2+𝜙21,3𝑦1,𝑡−3+𝜙22,3𝑦2,𝑡−3+𝜀2,t

Same for 3-dimensional VAR(2) model is:

    𝑦1,𝑡=𝑐1+𝜙11,1𝑦1,𝑡−1+𝜙12,1𝑦2,𝑡−1+𝜙13,1𝑦3,𝑡−1+𝜙11,2𝑦1,𝑡−2+𝜙12,2𝑦2,𝑡−2+𝜙13,2𝑦3,𝑡−2+𝜀1,𝑡  
    𝑦2,𝑡=𝑐2+𝜙21,1𝑦1,𝑡−1+𝜙22,1𝑦2,𝑡−1+𝜙23,1𝑦3,𝑡−1+𝜙21,2𝑦1,𝑡−2+𝜙22,2𝑦2,𝑡−2+𝜙23,2𝑦3,𝑡−2+𝜀2,𝑡  
    𝑦3,𝑡=𝑐3+𝜙31,1𝑦1,𝑡−1+𝜙32,1𝑦2,𝑡−1+𝜙33,1𝑦3,𝑡−1+𝜙31,2𝑦1,𝑡−2+𝜙32,2𝑦2,𝑡−2+𝜙33,2𝑦3,𝑡−2+𝜀3,𝑡

VARMA:

The vector autoregressive moving average model, as the name suggests has an extra component called moving average included just like ARIMA.

In a **VARMA(p,q)** the error terms 𝜀t a moving average representation of order 𝑞. Rest everything is just as same as a VAR model.

As an example, the equation for 2-dimensional VARMA(1,1) model is:

    𝑦1,𝑡=𝑐1+𝜙11,1𝑦1,𝑡−1+𝜙12,1𝑦2,𝑡−1+𝜃11,1𝜀1,𝑡−1+𝜃12,1𝜀2,𝑡−1+𝜀1,𝑡  
    𝑦2,𝑡=𝑐2+𝜙21,1𝑦1,𝑡−1+𝜙22,1𝑦2,𝑡−1+𝜃21,1𝜀1,𝑡−1+𝜃22,1𝜀2,𝑡−1+𝜀2,𝑡

Granger Causality Test:

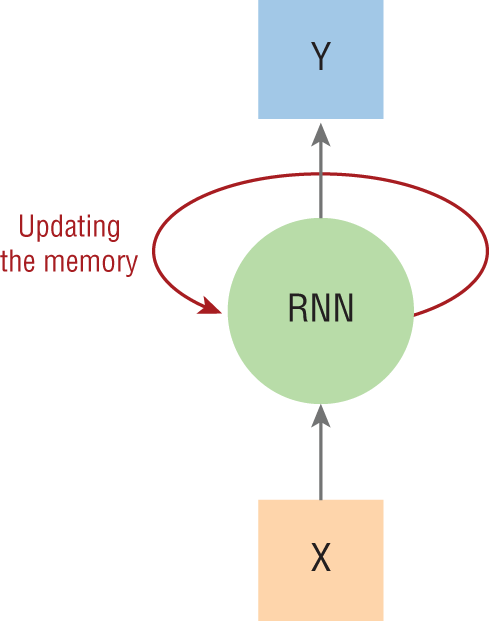
Now, the question is when there are a lot of variables involved in the dataset, how do we decide which will effect the value field the most rather than trail and erroring all variables in the dataset. This is when a statistical test named Granger Causality Test comes in place. This test basically understands and decides weather if the one variable is effecting the forecast of other in given timeseries. Thereby, it is helping in defining the potential relationships between variables, helping in trending. This test is present in statsmodels library and calculates the p values at different lags. If the p values are less than 0.05 at least for one of the lags, that’s when a relationship can be established and null hypothesis can be rejected. Grangercausalitytests(df[x,y],maxlags=4) is the function in statsmodels. [x,y] denote the variables in comparison, maxlags defines the number of lags to be performed on them.

Deep Learning:

Deep Learning models, specifically Recurrent Neural Networks (RNNs) and their variants, Convolutional Neural Networks (CNNs), and Transformers, have been successfully applied to various time series forecasting and analysis tasks. These models are powerful tools for forecasting, however they are quite data hungry. The model selection, hyperparameter tuning and evaluation help make a suitable choice.

Recurrent Neural Networks:

One of the important types of neural networks for timeseries forecasting are the RNN’s. In the 1980s, recurrent neural networks were developed that are capable of successfully extracting patterns from input data, they are a type of artificial neural network, RNNs have recently acquired prominence because to the growing processing power offered by GPUs. Each and every unit in RNN serves as an internal memory to have the information about it. Also, it has loops inside that are responsible for carrying information across the neurons when reading the input. RNNs are especially useful for sequential data, as shown in thefig below

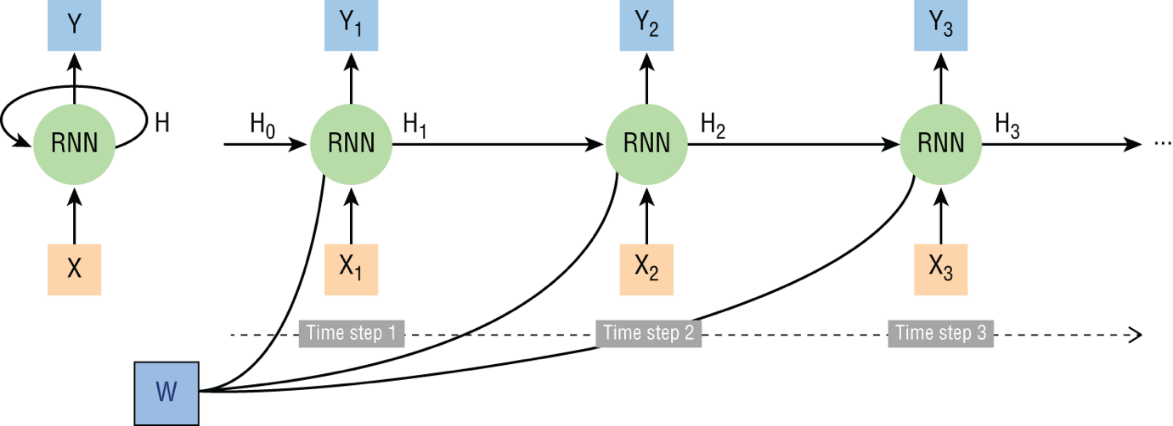


**Figure 6: RNN Unit**

If X is the variable that is being entered, then Y is the variable that is being produced.

The below figure shows the characteristics of RNN network, they share the parameters across each layer of the network. Whereas the feedforward network has different nodes across the network while RNN have same weight parameter within each layer of the record.

* X as the input
* Y as the output
* H is the hidden layer



**Figure 7: RNN Architecture**

LSTM:

A very common issue that an RNN faces is that, especially after a while, when you train the model on large sequences, the network will begin to forget the nature of data that came at the beginning by overwriting the weights. The information is lost at each step in RNN. The solution to this problem is that we need to have some sort of long and short term memory in place for our networks. That’s how the LSTM – Long Short Term Memory cell is created to address this issue of RNN.

There are four main operations in an LSTM cell,

* **Forget gate:**

By transferring the prior state to one of the sigmoid functions, the irrelevant portions of the previous state are forgotten are shown in below Figure.

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 8: Forget Gate**

* Store gate:

Determines whatever information from the past is valuable, then stores it in the cell state as shown in below Figure.

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 9: Store gate**

* Update gate:

1. In addition to the hidden state, LSTM additionally keeps track of a distinct cell state (t).
2. These gate actions update c(t) as shown in below Figure. In a favourable scenario, these gate operations can permit whole cell state to pass to the following cell.

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 10: Update gate**

* Output gate:

The information that must be transmitted into the following cell is controlled by the output gate shown in below Figure

A screenshot of a computer

Description automatically generated with medium confidence

**Figure 11: Output gate**

Diagram

Description automatically generated

**Figure 12: LSTM Model**

The above figure shows the entire model of LSTM. The inputs are normal from a recurrent neuron, ht-1 and xt, there is also a third input here called as cell state and denoted by ct-1. As per the output, we output ht as usual but also a cell state called ct. The very step involved is called, forget gate layer, this is where it is going to be decided what information does it get to throw away from the cell state. So we pass in the ht-1 and xt after performing a linear transformation, with weights and biased terms into a sigmoid (σ) function. Since it’s a σ layer, the output is always going to be between 0 and 1. 1 represents to keep it and 0 to get rid of it. The next step gets to decide what new information needs to be stored into the cell state (ct). In here, the first part is σ and the second part is hyperbolic tangent layer (tanh). Now, σ is the input gate later represented by it. The process is same again with inputs as ht-1 and xt with wi and bi pass that into the σ function and now we have an output values between 0 to 1 again. The tanh has same inputs through it creates candidate values. These could be added to the state. Now the old cell state ct-1 needs to be updated to ct and pass to ct+1. This is achieved by by adding it to new candidate values. Now the new output is again passed to the sigmoid layer and we put the cell state to the tanh, that will push the values to be between -1 and 1 and multiply the output by sigmoid gate that finally gives us ht.

Bi-directional LSTM

A Bidirectional LSTM is a sequence processing model that consists of two LSTMs, one taking input in a forward direction and the other in a backward direction. The below figure shows how it works.

Diagram

Description automatically generated

**Figure 13: BLSTM Model**

Keras library has really nice API that gets nearly all these processes of LSTM and BLSTM done with just a call of a function.

LSTMS that we will particularly focus on are listed below:

Vanilla LSTM: The term "vanilla" is often used to distinguish the basic LSTM architecture from its more complex variants. Here only layer of LSTM is added.

Bidirectional LSTM: Bidirectional LSTM layer is added instead of simple LSTM as discussed in above section. Reset everything remains the same.

Multilayer LSTM: In the above 2 architecture, only single layers are added, here multiple LSTM layers are added to increase the complexity of the model. The number of layers however depends on what works best with the dataset.

*Linear Regression:*

This model is majorly applied on timeseries data when one wants to know the relationship and effect of one feature on other for accurate model building. In short, it deals with multivariate data.

The process involves in preparation of lag features and combining the dependent and independent lags creating a linear model, fitting using OLS (ordinary least squared).

Formula Involved: Y = β0 + β1 \* X + ε

Y: Dependent variable

X: Independent variable

β0: Intercept

β1: Coefficient of the independent variable

ε: Error term

GARCH model in conjunction with liner regression is predominantly used in financial datasets however, this study would like to experiment and see how it works on energy data. GARCH stands for Generalized Autoregressive Conditional Heteroskedasticity. This particularly models the volatility or variance of the variable’s residual.

#### Comparison of performance of Models

The models that are tried to forecast can be compared to each other by considering metrics like below:

MSE  (Mean squared error): calculated the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals.

A mathematical equation with numbers and symbols

Description automatically generated

**Figure 14: MSE Formula**

Where,

* yi is the ith observed value.
* ŷi is the corresponding predicted value.
* n = the number of observations.

RMSE (Root Mean Squared error):  is the square root of Mean Squared error. It measures the standard deviation of residuals.

A math equations with numbers and symbols

Description automatically generated with medium confidence

**Figure 15: RMSE Formula**

RMSE, which stands for Root Mean Squared Error, serves as a metric that quantifies the accuracy of a regression model's predictions. It does so by measuring the average magnitude of the differences between the predicted values generated by the model and the actual observed values in the dataset. In other words, RMSE tells us how close or far off our model's predictions are from the real values. The lower the RMSE, the better the model's predictions coincide with the actual data points.

On the other hand, R-squared, also known as the Coefficient of Determination, provides information into how effectively the predictor variables included in the model can account for and explain the variability observed in the response variable. R-squared is a value between 0 and 1, where 0 indicates that the predictor variables do not explain any of the variance in the response variable, and 1 suggests that the predictor variables perfectly explain all the variance. Essentially, R-squared serves as a measure of goodness-of-fit, helping us assess the proportion of variation in the response variable that our model can elucidate.

So, to summarize, while RMSE concentrates on the precision of the model's predictions by evaluating the accuracy of these predictions in comparison to the actual data points, R-squared delves into the model's capability to elucidate the variance observed in the response variable, by quantifying how well the predictor variables contribute to explaining the underlying patterns in the data. Both of these metrics are vital tools in assessing the performance and validity of a model, with RMSE addressing prediction accuracy and R-squared focusing on explanatory power. [10]

### Review and analysis of related research works:

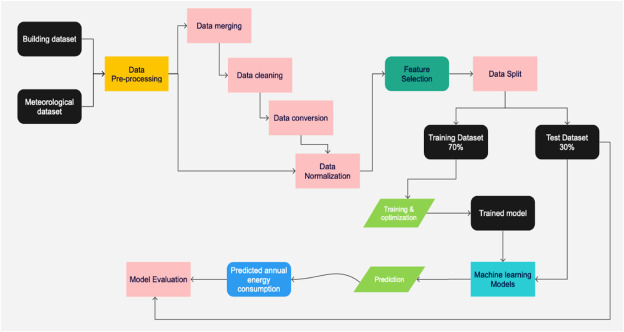
This section will be discussing some of the important and key papers that influenced this study of timeseries of smart meter data. The two papers that are being discussed are taken as a representation of couple of other papers/ researchers reviewed. Almost all of them fall under similar analysis.

The first paper that will be discussed is published in Science Direct in 2022.

***Scope:*** The scope of the paper was to design and develop efficient models for predicting energy consumption of a building. Now this paper is unique in a way as the building in consideration of the study wasn’t an operational one, this is at an early design phase. The overall aim of the study was to help in construction of energy efficient buildings and also aid in lessening the overall energy consumption of newly constructed or buildings.

***Background:*** According to this study, building energy prediction models are divided into two types, one being forward models and the other data driven models. Forward or physics based models require a lot of inputs about the building as well the information on environmental factors such as heating, ventilation, air conditioning, insulation, thermal properties etc. these are quite difficult to access. The data driven models on the other hand, are based on mathematics. These models are trained on timestamped data of energy consumption only. There has also been a discussion about the models suitable for this kind of analysis ANN has been seen as the best when there is a large data to start with. SVM’s for good results and Decision trees are also considered for their less complex applications.

***Methodology and Conclusions:*** This research basically used a large dataset. The study applied almost 9 machine learning algorithms namely: ANN, Gradient Boosting (GB), K Nearest Neighbour (KNN), Deep Neural Network (DNN), Random Forest, Decision tree Stacking, Support Vector Machine (SVM) and Linear Regression (LR). The methodology is flowcharted as below.



**Figure 16: Methodology of reference paper**

First the raw data is pre-processed, later EDA is done, after that feature selection, train and test split, optimisation, training the model, prediction and evaluation are done. The data is a combination of 2 sets. The first is the energy consumption data of 5000 residential buildings in UK. There were couple of variable which were considered independent like temperature, wind speed etc. and the dependent variable is the consumption. The other dataset is the meteorological dataset collected from Meteostat repository, this data was a collection of 10 area postcodes of residential buildings and was averaged monthly.

After all the models are applied below was a table which showed the performance of each.

| Model | Training Time | R-squared | MAE | RMSE | MSE |
| --- | --- | --- | --- | --- | --- |
| Deep Neural Network | 5.2s | 0.95 | 0.92 | 1.16 | 1.34 |
| Artificial Neural Network | 3.7s | 0.94 | 0.95 | 1.20 | 1.45 |
| Gradient Boosting | 1.8s | 0.92 | 1.10 | 1.40 | 1.95 |
| Support Vector Machines | 2.0s | 0.90 | 1.22 | 1.61 | 2.61 |
| Random Forest | 2.5s | 0.89 | 1.32 | 1.69 | 2.85 |
| K Nearest Neighbours | 1.4s | 0.77 | 1.90 | 2.40 | 5.78 |
| Decision Tree | 1.2s | 0.74 | 1.99 | 2.55 | 6.48 |
| Linear Regression | 1.4s | 0.73 | 2.02 | 2.59 | 6.72 |
| Stacking | 1.3s | 0.73 | 2.04 | 2.60 | 6.76 |

**Table 3: Performance Table of models in paper**

This is further visualised using boxplot in the research. Overall the conclusion was that the Decision Tree algorithm had the least training time hence was the quickest model to run. However, the Deep Neural network model performed the best in terms of all the error terms R- squared, MAE, RMSE and MSE.

The next paper that will be discussed in this study will be a smart meter data which is very close to this project. [11] [12]

*Scope:* The scope of this study was to forecasting residential energy demand using smart meter data and cluster based aggregation.

*Background:* As the paper was specifically on smart meter data, it discussed the importance of energy usage and how smart meters help in understanding the energy consumption behaviour. It also discussed on how energy companies now have so much abundance of data and responsibility to study and analyse this data for bigger benefits. It has also been discussed in the paper that the models like, Linear Regression, ARMA and Generalized Additive Models to Neural Networks and Support Vector Regression have already been applied in this field but was more of larger spaces such predicting electric loads of market segments, this might have a large data as it considers a lot of customers or sometimes an entire country. However, this research mainly focussed on electric consumption on residential customers.

*Methodology and Conclusions:*The paper first discussed the dataset, which basically used a research data for 5000 Irish homes which has smart meters installed in them. This study majorly focussed on tariffs and their dependencies on time of usage. Next, it discussed about feature selection. The data set had very restricted information and mostly time like week, day and hour and weather information along with consumption. Correlation was used and automatic feature selection was applied. As far as features were concerned the algorithms which used the features were SVM, LR and MLP. There was also SARIMA applied for the energy consumption variable. However, the SV, LR and MLP outperformed SARIMA.

Apart from these two papers there were certain generalised projects on timeseries analysis which particularly didn’t research on smart meter data, these were also observed to experiment only on univariant data. Two such papers were forecasting analysis on superstore sales data and Jetrail Tariff dataBoth of these weren’t energy related but they were good examples of timeseries data forecasting. The Jetrail Tariff project had good number of features as well but again only worked on models such as Moving average, Simple Exponential smoothing and Holt winters. The Super store project worked more on SARIMA and ARIMA. It did work on Prophet model too which checked the effect of holidays on the sales data, thereby making it Bivariant. [13][14]

#### Research Gap:

In the first paper discussed, though the data was mentioned as hourly/ sub hourly, there were no specific timeseries algorithms applied for the study. Although the dataset had many variables to play with all the nine algorithms they weren’t specific to timeseries data. Also, there was deep learning applied using ANN however, it was also quite popular that timeseries data best performs with RNN models which was missing. Now coming to the second paper, the data was more closer to this study as the data involved was smart meter data. This paper also focussed more on ML models such as SVM and LR rather than timeseries specific models however, there was one application of Seasonal ARIMA which was outperformed by the others.

After considerable research and study in smart meter data, it is observed that the UK is still in the process of adapting to this and making all the households installed with smart meters. Hence the data of these smart meters at the moment are guarded by energy companies and the available data for open research is quite limited. Data was encountered to be a prevailing issue in energy data especially involving smart meters.

In the first paper although there were a lot of dependent variables they were only applied on general ML models and in second paper there wasn’t much information on other dependent variables apart from just weather. This was also applied through grid search on other algorithms but not on SARIMA as SARIMA always handles univariant data.

In both the papers feature selection is done through grid search and this paper aims at trying other ways of selecting causality. Also this study aims at applying timeseries specific algorithms such as Holt Winters, ARIMA, SARIMA, VARIMA, LSTM and Moving Average to a limited dataset. Apart from that, this particular study will try to explore a multivariant model and see if other dependent variables actually effect the energy consumption in timeseries models.

## Contribution:

### Methodology:

Till now, this paper discusses the concepts that might be applicable in case of a timeseries problem. Now, the application plan, execution of experiments and the results will be discussed and reported in this section. Before going ahead with the plan, lets look at the dataset and analyse it.

#### Tools And Techniques:

*Software Requirements:*

|  |  |  |
| --- | --- | --- |
| S.no | Tools | Uses |
| 1 | Python | Programming Language |
| 2 | Google Colab | IDE |
| 3 | Operating system | Windows 11 |
| 4 | Dataset | UK Data Services |
| 5 | Python libraries | NumPy, Pandas, Matplotlib, Sklearn |
| 6 | Deep learning libraries | Keras, TensorFlow |
| 7 | Time series libraries | ARIMA, SARIMAX, VARMAX, HOLT Winters, ACF, PACF, Augmented Dickey Fuller test, KPSS, PP, Arch |
| 8 | GPU (not mandatory) | 8 GB |
| 9 | Office 365 | Documentation |

**Table 4: Software Requirements**

*Hardware Requirements:*

|  |  |  |
| --- | --- | --- |
| S.no | Type | Hardware |
| 1 | Operating System | Windows 11 |
| 2 | RAM | 8 GB |
| 3 | Processor | 12th Gen i5 |
| 4 | GPU | 4 GB |

**Table 5: Hardware Requirements**

#### Dataset and its source:

The work on this research paper is focused on the ‘Smart Energy Research Lab: Energy Use in GB

domestic buildings 2021’ dataset, which is under Safeguarded Access at UK data services website. This is a publicly available dataset. The data collected from Smart energy research lab is 13000 random homes which is a broad representation of homes in GB.

Data Collection:

This data is collected through a secured access from the website. This is an aggregated dataset varies over, years, months and times of the day. The data when acquired was in CSV format.

Data Description:

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Class | Description | Example |
| fuel | String | Fuel that the value represents | Net  electricity |
| unit | String | Unit of the value | Wh |
| summary\_stat | String | What statistical summary the value  represents | Median |
| subsample | String | What subsample of SERL Observatory homes  were used | All |
| summary\_time | String | Time period that the value relates to, either a  specific half hour or a daily value. Note that  for half hourly statistics, the reported time is  local time, and the value is for the energy  used in the preceding 30 minutes. | 00:30 |
| time\_period | String | Time period that the data is summarized over | 2021 |
| segmentation\_variable\_  1 | String | Variable used to segment the Observatory  sample | num\_occu  pants |
| segment\_1\_value | String | Segment of the above segmentation variable  that this value relates to (any value that the  segmentation variable can take, or a merge  of 2+ values to ensure at least 10 cases are  included) | >=5 |
| value | Float | Value of the specified summary statistic,  summary period and segment | 8.168 |
| n\_sample | Float | Number of cases in the distribution this value  was drawn from | 10,764 |
| n\_statistic | Float | Number of cases used to calculate the value.  For mean and standard deviation n\_sample =  n\_statistic. For centiles n\_statistic = 10 as we  take the 10 closest values to the centile and  report the mean of these in the value field for  SDC reasons. | 10 |
| mean\_temp | Float | Mean external temperature in ⁰C over the  summary time during the time period | 10.378 |
| mean\_hdd | Float | Mean heating degree days over the summary  time during the time period. Note that this is  NaN for half hourly summary times as heating  degree days are defined for daily periods. | 5.74 |
| decimal\_places | Float | Number of decimal places the value has been  rounded to | 3 |
| weekday\_weekend | String | Whether the value was calculated using only  data from weekdays, weekends or both | Weekday |
| mean\_floor\_area | Float | Mean floor area in m2 for dwellings in the  SERL observatory with segment\_1\_value and  a floor area value from an EPC | 100.026 |
| n\_mean\_floor\_area | Float | Number of cases used to calculate  mean\_floor\_area | 1437 |
| mean\_bedrooms | Float | Mean number of bedrooms for dwellings in  the SERL observatory with segment\_1\_value  and with an answer to the number of  bedrooms question (B6) in the SERL survey | 2.918 |
| n\_mean\_bedrooms | Float | Number of cases used to calculate  mean\_bedrooms | 2586 |
| mean\_occupants | Float | Mean number of occupants for dwellings in  the SERL observatory with segment\_1\_value  and with a valid answer to the number of  occupants question (C1\_new) in the SERL  survey | 2.297 |
| n\_mean\_occupants | Float | Number of cases used to calculate  n\_mean\_occupants | 2581 |

**Table 6: Energy Data Variables Description**

Data Ethics and Privacy:

1. Data used in these project is accessed from Smart Energy Research Lab: Statistical Data, 2019-2021 which is under Safeguarded Access at UK data services website. UK data service comes under UK data protection act of 2018. GDPR has strict rules and regulations against how personal data should be handled and protected. Hence, privacy of the data is always in place.
2. The data collected from Smart energy research lab is 13000 random homes which is a broad representation of homes in GB. Hence the data has no bias towards any ethic group and is completely reliable and transparent.
3. The reliability of data can also be verified using confidence intervals.

Data Preprocessing:

The data pre-processing is a process where we make the data ready and available for analysis and forecasting. The energy data set that is dealt with currently in this particular project is a timeseries data set. However, in a time series dataset to make relevant analysis we need the each particular timestamp to have only one particular value to analyse. Now to make this possible we first read the dataset of choice using pandas.

Since the main aim behind this study is to forecast using timeseries, for any data to be considered as timeseries, one of its column containing the time stamps should be in a datetime format. The time stamps are converted to datetime format by calling datetime function. Now when we visualise the data there are a lot of time stamps for one particular day, this could cause a lot of noise in graph, hence in the study we only consider the data with month and date information. This is done by creating a time stamp and putting it in a for loop and extracting all 24 days (2 years data). Now, under summary\_stat column, there is a data for each percentile and mean and median. For our study, mean value can be considered as it covers 50% of data and ignoring the rest. The filtered data now looks as below.

A screenshot of a computer

Description automatically generated

**Figure 17: Data Frame of Energy Data**

Visualisations to understand the dataset:

**A diagram of a graph

Description automatically generatedA graph of a graph with red and black lines

Description automatically generated with medium confidence**

**Figure 18: Quarterly and Monthly visualisation of Power consumption**

The main associated column to our dataset is the value column. This denotes the energy consumption that happened over the time line. The above two graphs show the exact points in the energy stands over the timeline on y-axis. From the graphs its easily understood that the q3 i.e. months June, July and August had the least consumption. Quarter 1 i.e. January, February and March had most consumption.

#### EDA- Exploratory Data Analysis:

This is a crucial part of data analysis, where we deal with aspects such as data cleaning, data transformation to fit the suitability of a model.

*Null Values:*

Firstly, we started off by looking at null values in the dataset. Isnull()sum() gave total number of null values. The variables, mean\_bedrooms and mean\_occupants have 48 missing values. By further analysing these columns, it is understandable that these columns can have decimal values as number of bedrooms can only be a finite number lie 2 or 3. Hence, the decimals have been rounded using df.round() function, where df is the dataframe. Now, considering the rows with missing values, instead of deleting these missing valued rows, they are replaced by the values 2 for mean\_bedrooms and 1 for mean\_occupants. This is done by looking at the other variables in the same row, from the domain knowledge, it can be estimated that the similar floor area houses might have similar number of bedrooms and occupants, this variable with similar value has been observed and compared with missing value fields, it then has been inferred that the houses with similar floor area had 2 bedrooms and 1 occupant and hence its been filled out accordingly. Now the dataset has no null values as shown below.

***A screenshot of a computer program

Description automatically generated***

**Figure 19: Null Values**

*Negative Values:*

Datasets depending on domain can have negative values, but logically thinking, power consumption either happens or doesn’t hence the values can only be positive or zero. Similarly, other numeric variables in the dataset like mean\_temp, mean\_hdd can have positive values or zero hence this check.

The dataframe’s variable is given and all the values <=0 are displayed. There were two values seen. Using .loc function, the location is printed. It is observed that both the negative values have the same mean floor area, occupants and bedrooms. However for same values the value field had varied values hence cannot infer a particular value so considered the data frame without these 2 rows.

*Indexing:*

Indexing is a crucial step involved in timeseries analysis and forecasting. To start with certain decomposition techniques and timeseries models do not work without timeseries having the timestamped column as an index column with datetime format in place, this is like a prerequisite as this allows the techniques and models to understand trends, patterns and relationships over the time period. Along with this indexing provides, good order and sequence for the observations making analysis and modelling easy. Also, in some cases dataset is a combination of different datasets and they need to aligned on some common point and the indexed column with date information can serve as the best alignment.

This is achieved quite easily in python by using set\_index function, but before performing this it is necessary to see to it that the data is in date time format readable by Pandas. The indexed dataset looks as below.

A screenshot of a computer

Description automatically generated

**Figure 20: Indexed Data Frame**

#### Statistical Inferences:

*Time Resampling using Down sampling:*

Time Resampling is a process in which the time frequencies are adjusted from a denser samples to smaller. There are certain inbuilt resampling methods available in Pandas which help in segregating the sample size based on the option selected. Foe example, if there is a dataset which has time stamps and observations for every hour and when it is resampled using the function resample(M).mean() with attribute value as ‘M’, the function aggregates all the hourly time stamps and calculates monthly value. However, in the dataset associated with this project, we use a technique called down sampling as the data is already appropriately timestamped. The issue with the current dataset is that it has multiple values associated to a single month data as it has readings of gas and electric as well. So instead of taking higher frequency data like days and reducing it to months, groupby() is used to group all the values with similar dates together according to the index i.e. according to the dates, and calculate a mean. A point to be considered here, the energy overall is aggregated here as the units of measurement for both gas and electricity is generalised to KWh already. To confirm the correlation between these two, a graph is plotted using the aggregated values and it looks smooth with a proper trend component, as shown below.

A blue line on a white background

Description automatically generated

**Figure 21: Seasonality**

*Stationarity:*

The importance of Stationarity is vast not only in timeseries but generally in machine learning. The reason behind this is quite straightforward, models such as ARIMA assume that the data used in modelling is stationary. The models best perform on stationarity for predictions.

There are couple of tests inbuilt in form of functions in statsmodels that one can call to understand if data is stationary or not. These tests prove or reject null hypothesis. In case if tests prove that the data is non stationary, a method called differencing can be used to make the data stationary before using it to model.

Now, the aggregated dataset called as ‘sum\_aggregated\_df’ is put through three different tests called ADF (Augmented Dicky-fuller test), KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test and the PP (Phillips-Perron).

ADF’s null hypothesis is that, the dataset has unit root indicating non stationary. In other words, non-stationary.

If the p-value of ADF is less than 0.05, null-hypothesis is rejected and dataset is stationary.

KPSS’s null hypothesis is that the time series is stationary around a deterministic trend. Also, If the p-value of KPSS is less than 0.05, null-hypothesis is rejected and dataset is non-stationary.

PP’s null-hypothesis also depends on unit root, and the p-value less than 0.05 rejects null-hypothesis and interprets that data is stationary.

All these tests are run in stats models by calling, adfuller(), kpss() and adfuller(regression='ct') respectively. For the dataset in case, two tests prove that the data is stationary out of the three, which can be considered and assumed that data is stationary. The result is shown in the picture below. S denotes Stationarity and NS non stationarity.

A screenshot of a computer

Description automatically generated

**Figure 22: Stationarity test result**

*Seasonal Decompose:*

Seasonal Decomposition is a technique used in statmodels, to separate the data into its components covering the trend, seasonality, residual and cyclic. There are many techniques that can be used to achieve this and many new techniques being researched to work on these components. For the dataset in study, there will be 3 types of techniques applied.

Starting with a very simple decomposition technique called STL(Seasonal and Trend Decompose using Loess), trend implies overall behaviour of data, seasonal captures patterns in the data and residual as the name suggests handles the noise. This is a simple additive model where the original value is a sum of all three components.

This technique is called by a function: STL() and the output is as the picture below.

A graph of blue lines and dots

Description automatically generated with medium confidence

**Figure 23:Seasonal Decompose using STL**

The plot clearly shows an upward trend, there is definite seasonality and it is cyclic in nature. To look further on the cyclic part, Hodrick-Prescott Filter Decomposition technique is applied. This technique focuses on two components, trend and cyclical component. The result of the plot is as follows and there is a definite cyclical trend to the data.

A graph with a line and a blue line

Description automatically generated

**Figure 24:Seasonal Decompose using Hodirick-Prescott**

In the above plot, the green line is the cyclical component, it clearly states that the energy usage is at its highest during the months of January in both years and gradually decreases to its least in July and then thereafter has an upward trend.

Finally, to sum up the discussion on seasonal decomposition, the most notable technique is the good old seasonaldecomposition() function, here the data is tested on multiplicative as the above models tested additive component. The plot is shown below. The trend is better depicted here while using multiplicative rather than additive, seasonality is pretty much the same and the residual is almost 1 all over the timeseries for energy consumption variable.

A graph on a screen

Description automatically generated

**Figure 25: Seasonality using seasonal decompose**

*Correlation Analysis:*

It is quite important to understand the concept of correlation in timeseries analysis and forecasting. It explains about dependencies, relationships and co-movements of the data.

Correlation plots are used to visualise and understand the dependencies. The two plots are namely:

1. ACF (Auto Correlation Function) plot
2. Partial Autocorrelation Function (PACF) Plot

ACF plots some vertical lines and these lines are plotted at various lag values denoting the horizontal axis(x axis). These lags represented by number associated to shifting the timeseries by itself. This is to see the correlation between the value to itself at different time lags.

Similar to ACF, PACF plots values across vertical lines on different time lags. The former corresponds to correlation coefficients and the latter to partial coefficient. Partial means nothing but the value which is not explained by the mutual correlation. These values help majorly in defining the parameters of models like ARIMA and its variations.

The graphs below show the correlation plots for the data set in study. The technique of understanding the variables for ARIMA through these graphs will be discussed in further sections in detail. [15]

A graph with blue dots and lines

Description automatically generatedA graph with blue lines and dots

Description automatically generated

**Figure 26: ACF and PACF Plots**

*Causality in Timeseries:*

As the suggests, causality is nothing but the dependency of one variable on the other in a timeseries. Most of the timeseries models are univariant, meaning they deal with single variable. However, as discussed in the earlier sections this project aims at exploring models on multivariate data, meaning more variables in timeseries data. To make different variables work together in forecasting, one must have a knowledge on how the dependent variable is getting effected by the independent ones. In order to check this condition, there is a predefined function in statsmodels called *grangercausalitytests.*

This test works on hypothesis just like other statistical tests, null hypothesis proves that there is no causality and alternative proves vice versa. The only decision that one should make in the grangercausalitytests(X, maxlags) function is to decide the lag order i.e. number of past values to be considered. This value needs to be experimented and can start with a constant value 5. When the experiment is one on all the valid variables of data set and compared with the energy consumption value, below are the results.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | value and mean\_temp | value and mean\_hdd | value and mean\_floor\_area | value and mean\_bedrooms | value and mean\_occupants |
| P-value at lag 1 | 0.05 | 0.05 | 0.5 | 0.5 | 0.5 |
| P-value at lag 2 | 0.5 | 0.2 | 0.3 | 0.3 | 0.3 |
| P-value at lag 3 | 0.8 | 0.7 | 0.5 | 0.5 | 0.5 |
| P-value at lag 4 | 0.00 | 0.10 | 0.8 | 0.8 | 0.8 |
| P-value at lag 5 | 0.02 | 0.3 | 0.8 | 0.8 | 0.8 |

**Table 7: P values associated with features after the Granger Causality test**

Amongst the given lags, if at least one of them has the p-value <= 0.05, there is a good chance of causality between those two variables and can be used for forecasting. Otherwise it is safe to assume that they do not have required causality for forecasting. According to the above table the variable *mean\_temp* and *mean\_hdd* have p-values less than 0.05 in certain lags and hence have good causality with the *value* variable which is the energy consumption value.

*Smoothening Techniques:*

Smoothening techniques are applied by using some sort of statistical components like mean to make the data much smoother, meaning reduce noise. For the data set in study, moving averages and exponential smoothening techniques have been applied.

*Exponential Moving Average****:*** The way that it works is, it calculates the weighted average of past data points with recent ones giving importance to these by giving higher weights whereas the older ones get lesser weights. *ewm(span=12,adjust=False).mean()* function is used to achieve this. Span is the period that needs to be considered for higher weights. Note: before performing the function, the index frequency should be given according to the data, in this case it is ‘MS’ as it is monthly data. When the smoothened data is plotted it looks like the orange line and blue depicts the original trend of the data.

A graph of a graph

Description automatically generated with medium confidence

**Figure 27: EWMA**

*Double Exponential Smoothening:*This works very similar to EWMA, however, it considers two components, level and trend component. *ExponentialSmoothing(sum\_aggregated\_df['value'], trend='add').fit().fittedvalues.shift(-1),* function can becalled to perform this. Shift is nothing but it shifts the values by -1. The trend can be multiplicative or additive, both are performed in this case. The plot is shown below. It can be observed that these values are more closely related to the original.

A graph of a mountain

Description automatically generated with medium confidence

**Figure 28: DES Additive**

A graph of colored lines

Description automatically generated

**Figure 29: DES Multiplicative**

*Triple Exponential Smoothening:*As the name suggests there are three components included in this technique and rest everything is similar to the double exponential smoothening. The seasonality component is also incorporated in this technique. The *ExponentialSmoothing(sum\_aggregated\_df['value'],trend='add',seasonal='add',seasonal\_periods=12)* is the function in place. Just like before seasonality can be multiplicative or additive. Both are used and the best plot is given below. This is the closest to the original compared to other smoothening techniques.

A line graph on a white background

Description automatically generated

**Figure 30: TES**

This smoothened data can be further used for modelling and evaluate the model performance according to the error and choose which works effectively.

*Feature Scaling:*

When a model is built on data with features having different ranges it will jeopardize the model. So to overcome this we use feature scaling techniques like standardization and normalization and the one we have used in the model is standardization using standardscaler from sklearn library.

So when the standardscaler performed on our data will change the input features into the same range with mean of zero and standard deviation of one which in turn allow our model to learn more and be robust to any discrepancies.

A table of numbers with black text

Description automatically generated

**Figure 31: Data after Scaling**

### Model Building Experiments:

This can be defined as the process where we create and train algorithms. The dataset in study has been processed through the below discussed algorithms.

*Train test split:*

The first step of any model building for that matter is splitting the entire data into training and testing sets called test train split. Model is fit on training data, and forecast of this data to the same length of time that the test data is and then compare the forecasted results to the test data. Typically, the size of the test data is 20% of the sample. This also depends on how long the data is and how much one wants to forecast. A better way to interpret this the test size should ideally be at least as large as the maximum forecast.

*Holt Winters Method:*

The first model that the study starts with is Holt winters, it is a relatively easy model for timeseries forecasting and more like a stepping stone. This method handles data with three components level, trend and seasonality.

*fitted\_model=ExponentialSmoothing(train\_data['TESmul12'],initialization\_method='estimated',trend='add',seasonal=12).fit()*

The model is fit to the training data using the above line of code. Here the training data variable is the exponentially smoothened *value* field of the dataset. The trend is given as multiplicative as the additive is tried and multiplicative fits the best. The season value is 12 as it denotes the time period and this is a monthly data. Now by making the model learn using training data, the values are forecasted using .*forecast(6*) and then compared to the test data, the 6 denotes the time period of the number of predictions expected, better to be the length of test data. Below table is the test predictions according to the model and the graphs denote a train test and prediction plots.

|  |  |
| --- | --- |
| 2022-01-01 | 41.957293 |
| 2022-02-01 | 39.376187 |
| 2022-03-01 | 32.978864 |
| 2022-04-01 | 23.309518 |
| 2022-05-01 | 17.166641 |

**Table 8: Holt-winter test predictions**

A graph with blue lines

Description automatically generatedA screenshot of a graph

Description automatically generated

**Figure 32: Holt-winter’s graphs**

The graphs show a similar trend visually, the method predicted good for starters. To further check the statistical validation, there are errors calculated and discussed in the results section.

*Arima:*

As discussed in the literature review, ARIMA is a combination of three models. Auto Regressive, Integrated and moving average. These three values are represented in model by using (p,d,q) and the main goal to achieve results is to know and find these orders. Component ‘d’ is nothing but differencing. For the next tow, there are two methods involved in doing this. First and the traditional one is to read the plots through ACF and PACF plots. In correlation section of this study we have the plots available for us. If the autocorrelation plot shows positive auto correlation at first lag (lag-1) then it suggests to use the AR component. If it shows negative then it suggests to use MI component.

p: Number of lagged observations included in the model.

d: Number of times raw observations are differenced.

q: The size of moving average window.

Now to analyse what these values are, one can look at the PCAF plot. A sharp drop at lag ‘k’ suggests an AR-k model can be used. And after if it has a gradual decline it suggests an MA model too. In our graph, there is a sharp drop at 2 hence the first term can be 2. The data in the study is stationary so there will be no differencing applied, hence the value of d is 0. Finally, the q component is always less than or equal to p component. Hence it can be between 2 and 1, whatever works the best.

The second way to figure out these values will be discussed now, as observed it’s a daunting task to read these plots and assume values. It also more often effective to run a grid search. This is done by using pmdarima (pyramid arima) a separate library designed to perform these grid search functions across the combinations of p,d,q. It uses AIC (Akaike information criterion) as a metric to make comparisons between the performances of different models.

*auto\_arima(sum\_aggregated\_df['value'],seasonal=False).summary()*

The output of the above code is :

A screenshot of a data sheet

Description automatically generated

**Figure 33: Auto Arima Summary**

Indicating the best model to be with model SARIMA and order (4,0,1)

Above mentioned is the line of code to achieve this. Initially seasonality is taken as false. The AIC for this model is 126. What this practically does is understood stepwise by running the below code:

*stepwise\_fit= auto\_arima(sum\_aggregated\_df['value'],start\_p=0,start\_q=0,max\_p=6,max\_q=3,seasonal=False,trace=True)*

It basically, suggest to start from p and q values at 0 and max is 3.

A screenshot of a computer program

Description automatically generated

**Figure 34: Auto Arima Step-wise fit**

Both these functions above suggest the same without seasonality but or data is seasonal hence in the code *seasonal* is now given as true. Below is now the new suggested model.

A screenshot of a data sheet

Description automatically generated

**Figure 35: Auto Arima Summary with Seasonality**

Now even though the autoarima function suggests SARIMA as the model. SARIMA typically has an extra set of components along with (p,d,q) called (P,D,Q). These values however are not suggested by atomarima. Hence, in this study we are going to apply ARIMA excluding the seasonal component and also later SARIMA by guessing (P,D,Q) values through PACF.

ARIMA (2,0,2)

This is the model applied with seasonality. The process to run a model is as follows:

1. Once autoarima is competed, the dataset is divided into train and test. Instead of using percentages here the cell values are used and the testing set had six months of length in it.

*train = sum\_aggregated\_df.loc[:'2021-06-01']*

*test = sum\_aggregated\_df.loc['2021-06-01':]*

1. Next step is considering the training values of the dataset and fitting them to the model.

*model = ARIMA(train['value'],order=(2,0,2))*

*results = model.fit()*

*results.summary()*

The last line, results.summary gives the results of the model ran. Information like AIC.

A screenshot of a data sheet

Description automatically generated

**Figure 36: ARIMA (2,0,2) Summary**

1. Now according to the model, the predictions are made using the below code:

*start=len(train)*

*end=len(train)+len(test)-1*

*predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('ARIMA(2,0,2) Predictions')*

len(train) is computing the length of the training data and assigning it to the start. And same with end, these define the length of the predictions. The predictions according to the model are as follows and the graph denotes the predictions (orange line) and original test value (blue line) to see how the model performed. The values are coherent as the graph suggests, hence a good model.

A screenshot of a computer

Description automatically generated

**Figure 37: ARIMA (2,0,2) Prédictions on test**

A graph with blue and orange lines

Description automatically generated

**Figure 38: ARIMA (2,0,2) Prédictions Plot**

Now when the same process id repeated to the order (4,0,1). The results are as follows:

A graph with blue and orange lines

Description automatically generated

**Figure 39: ARIMA (4,0,1) Prédictions Plot**

This graph looks close enough too, hence we use statistical error variables to identify the best model, discussed in later sections.

*SARIMA:*

As auto arima suggested the best model for this particular data is SARIMA with an order (2,0,2) however, it did not specify the seasonal conditions of P,D,Q here, hence going with understanding the PCAF plot. And the seasonality.

A blue line on a white background

Description automatically generated

A graph with blue lines and dots

Description automatically generated

**Figure 40: Seasonal Component Plot and PACF**

From first figure above it can be observed that there is a definite seasonality every year to this plot and the second graph denotes the partial correlation, there is a drop at 2 and the differencing is zero as the series is stationary. As discussed above, P=2, D=0 and Q can be 2 and 1.

All the modelling steps discussed in ARIMA are applied.

1. The data is split into train test

*train = sum\_aggregated\_df.iloc[:12]*

*test = sum\_aggregated\_df.iloc[12:]*

Here we are diving data into 12 and 12, one year for training and one for testing. As testing out the hyperparameters is also crucial in model building.

1. Model is fit using the below code and forecasted:

*model = SARIMAX(train['value'],order=(2,0,2),seasonal\_order=(2, 0, 2, 12))*

*results = model.fit()*

*fcast = results.predict(len(sum\_aggregated\_df),len(sum\_aggregated\_df)+5,typ='levels').rename('SARIMA(2,0,2)(2,0,2,12) Forecast')*

The results when plotted are as below:

A graph of a graph

Description automatically generated with medium confidence

**Figure 41: SARIMA plot predictions 1**

The forecasted values are mimicking the original trend but not exactly. Further statistical analysis can prove the accuracy of the model.

A graph on a white background

Description automatically generated

**Figure 42: SARIMA plot predictions 2**

The graph above shows, predictions and original test value data.

The same process is repeated however seasonality q value is changed to 1. The values are plotted for predicted and test values for the model (2,0,2)(2,0,1).

A graph on a white background

Description automatically generated

**Figure 43: SARIMA plot predictions 3**

The graph is quite similar to the other seasonal values, further statistical analysis can prove the accuracy of the model.

*SARIMA with X factor:*

As the name suggests there is an extra factor of consideration for this model called exogenous factor. This is nothing but another field from the dataset that are influencing the target variable.

All the modelling steps are performed accordingly.

The model is fit using the below code:

*model = SARIMAX(train['value'],exog=train['mean\_temp'],order=(2,0,2),enforce\_invertibility=False)*

*results = model.fit()*

*results.summary()*

Here the exog defines the variable that effects the target. This variable is selected using causality factor from Granger Causality test.

Values are forecasted using similar code from SARIMA and the values are plotted against test values.

A graph with blue and orange lines

Description automatically generated

**Figure 44: SARIMA plot with X factor**

The values look to be good but not as great as SARIMA. Also, this is done to understand the model and its accuracy for the dataset. However, this isn’t a valid model to apply for forecasting as it works best when the exogenous variable’s value of the future is known and we are trying to predict the current value of target. Then this model does best to predict the future values of target by using the effect of the exogenous variable.

*VARMA (Vector Auto Regression and Moving Average):*

This a very critical model for this study as it deals with multivariate data. To understand the relationships between the variables of data it uses vectors. Rest is very similar to ARIMA without the integrated component which is nothing but the differencing done to eliminate stationarity. Since we have the data which is stationary this part might not make huge difference.

All the steps involved are pretty much similar to ARIMA.

Auto arima is run all variables involved and best case scenario for all variables is considered. The variables to be involved in this are again selected using causality from the Granger test.

The first model considered only one variable with best causality i.e., mean\_temp. It is fit using the below code:

*model = VARMAX(train[['value','mean\_temp']], order=(2, 2), trend='c')*

*results = model.fit(maxiter=1000, disp=False)*

*results.summary()*

Under the training data used to fit for model, this model takes in as many variables as possible. Hence including mean\_temp variable here. The values are forecasted and plotted as below compared to test values.

A graph of a graph

Description automatically generated with medium confidence

**Figure 45: VARMA plot 1**

This looks like a good correlation.

To understand the best model, another variable mean\_hdd is added and the graph is plotted.

**Figure 45: VARMA plot 1**

A graph of a person

Description automatically generated with medium confidence

**Figure 46: VARMA plot 2**

By far this looks like the best model as the lines are not extremely overlapping which might indicate overfitting.

*Recurring Neural Networks:*

Recurring Neural networks are specifically designed to deal with sequence data. In this case, it’s a time series, they also deal with Sentences, Audios, Music and a lot more.

LSTMs:

An prevailing issue with RNN is that it tends to forget the initial inputs and in LSTM we can define what needs to be forgotten and what we need to remember. LSTM stands for long short term memory cell. Using libraries like Keras and Tensorflow help implementing and building LSTMS networks. For the study dataset we try to apply 3 types of LSTMS and see which works better.

The process is as follows:

1. The index frequency is set to ‘MS’ as this is a monthly data.
2. The next set is to scale the data, the standardscaler is used in this case and called upon the function standardscaler().
3. The data is split to train and test using the below code. It pretty much divides the data to 75% and 25%, the former named to be train and the latter test.

# Calculate the split point

split\_point = int(0.75 \* len(selected\_df))

# Split the data into training and testing sets using iloc

train = selected\_df.iloc[:split\_point]

test = selected\_df.iloc[split\_point:]

1. Time Series Generator: Now these series should be loaded to label as batches, traditionally numpy is used transform the regular batch and tag it with a label appropriately, luckily, now keras has a timeseries generator object that does this part automatically. The code is given below to run this object. The test data is our input is hence the n\_input is 17 and label is 1. Then we pass in the sequence to the generator. The generator gives a tuple as output. X is the input and y is the prediction.

*n\_input = 17*

*n\_features = 1*

*generator = TimeseriesGenerator(scaled\_train, scaled\_train, length=n\_input, batch\_size=1)*

1. Now we define the model, by using the below code for vanilla LSTM:

*model = Sequential ()*

*model.add (LSTM (50, activation ='relu', input\_shape = (17, 1)))*

*model.add (Dense (1))*

One layer of LSTM is added with 50 neurons. The activation feature is given as ‘relu’- rectified linear unit, next the input is (17,1), 17 is the observations sent as test and 1 is since the its is univariant. After this a singular dense neuron is added, to aggregate all the 50 neurons to a single prediction.

1. Now, we compile everything before fitting giving the optimiser as ‘adam’ and the loss is measured in ‘mse‘ which is mean squared error.

*model.compile(optimizer='adam', loss='mse')*

1. The summary is looked at by using model.summary(). Now we fit this to the training generator using

*model.fit\_generator(generator,epochs=100)*

The more epoch we use will take more time to train, epoch is nothing but a single run through to all of the training data.

1. The loss is plotted and looks as below:

A graph of a number of people

Description automatically generated with medium confidence

After 45 epoch’s the data is no more variant so the epoch value can be 45.

**Figure 47: EPOCH Plot**

1. Just like other models, based on the models train data, predictions are made and plotted against the test to understand how well it predicted the test data. To achieve this the scaled data is reversed:

*first\_eval\_batch = first\_eval\_batch.reshape((1, 17, 1)).*

1. The test precitions are generated using the below code:

*test\_predictions = []*

*first\_eval\_batch = scaled\_train[-n\_input:]*

*current\_batch = first\_eval\_batch.reshape((1, n\_input, n\_features))*

*for i in range(len(test)):*

*current\_pred = model.predict(current\_batch)[0]*

*# store prediction*

*test\_predictions.append(current\_pred)*

*# update batch to now include prediction and drop first value*

*current\_batch = np.append(current\_batch[:,1:,:],[[current\_pred]],axis=1)*

1. The scaler is inversed:

*true\_predictions = scaler.inverse\_transform(current\_batch.reshape(1,-1))*

1. Graph is plotted between the test predictions and original test values.

A graph with a line

Description automatically generated

**Figure 48: LSTM Plot**

This model in entirety performed the least the there is no similar trend between the test value and predictions. Preductions run a flat line almost.

Two more models called stacked LSTM and Bidirectional LSTM are run on the same data to understand if they would give better results.

The Stacked LSTM is nothing but we add more layers of LSTM than the vanilla one. The model definition code is as below:

*stacked LSTM*

*model = Sequential()*

*model.add (LSTM (100, activation ='relu', return\_sequences =True, input\_shape = (17, 1)))*

*model.add (LSTM (100, activation ='relu'))*

*model.add (Dense (1))*

remaining 11 steps remain the same as vanilla lstm and the output of the plot is as below.

A graph with a line

Description automatically generated

**Figure 49 : LSTM Plot2**

The bidirectional LSTM is nothing but we process input sequence from both instead of one like traditional LSTM. The model definition code is as below:

#*Bidirectional LSTM Modelling in Keras:*

*model = Sequential ()*

*model.add (Bidirectional (LSTM (100, activation ='relu'), input\_shape = (17, 1)))*

*model.add (Dense (1))*

*Linear regression using Garch:*

The final model that is built is the linear regression from scratch using backwards stepwise feeding and ordinary least squared to estimate.

The first step is to perform a backward stepwise feature selection, it starts with all the features of the dataset and keeps removing least significant ones depending on the p-value. The code is as below:

*def backward\_stepwise(X, Y, threshold\_out):*

*included=list(X.columns)*

*while True:*

*changed=False*

*for feature in included:*

*model= sm.OLS(Y, sm.add\_constant(pd.DataFrame(X[included]))).fit()*

*p\_values=model.pvalues*

*worst\_pval=p\_values[feature]*

*if worst\_pval > threshold\_out:*

*included.remove (feature)*

*changed= True*

*print (f'Removing {feature} with p-value {worst\_pval}')*

*if not changed:*

*break*

*return included*

Here, X and Y are the inputs, X- features and Y- Dependent Variable. Threshold\_out removes the p-values greater than the specified, include will have all the possible features. While is given to loop the iteration until all values are satisfied.

The next process is according to the correlation p and q are defined. Code is as below.

*#this is with p=2 and q=2*

*check=backward\_stepwise(X,Y,threshold\_out=0.1)*

*final=sm.OLS(Y, sm.add\_constant(pd.DataFrame(X[check]))).fit()*

*print(final.summary())*

*final.fittedvalues*

*final.predict()*

*# Load or generate your regression residuals data (squared residuals)*

*squared\_residuals = final.resid\*\*2*

*# Fit a GARCH model to the squared residuals*

*model = arch\_model(squared\_residuals, vol='Garch', p=1, q=1) # Adjust p and q as needed*

*results = model.fit()*

*# Forecast the volatility for the next 6 data points*

*forecast\_horizon = 6*

*forecasted\_volatility = results.forecast(horizon=forecast\_horizon)*

*# Generate forecasts for the upcoming data points using OLS and GARCH-forecasted volatility*

*upcoming\_X = np.column\_stack((np.ones(forecast\_horizon), np.random.rand(forecast\_horizon, 3))) # Adjust as needed*

*# Ensure the number of columns in upcoming\_X matches the number of coefficients in ols\_results.params*

*if upcoming\_X.shape[1] != len(results.params):*

*raise ValueError("Number of columns in upcoming\_X does not match the number of coefficients in the OLS model.")*

*# Calculate the forecasted values using OLS coefficients and GARCH-forecasted volatility*

*forecasted\_values = np.dot(upcoming\_X, results.params) + np.sqrt(forecasted\_volatility.variance.values[-1])*

*# Print the forecasted values*

*print("Forecasted Values:")*

*print(forecasted\_values)*

Once the features are selected, the next line fits the linear regression. Next the predicted values are generated, squared residuals are calculated, squaring them helps modelling the volatility. Next this we are fitting the model to GARCH which is predominantly used in finance but tried to do it to understand how it could work on energy data. At last, the values are forecasted.

### Evaluation and Result Discussion with forecasting:

When evaluating timeseries models, it is important to consider its temporal nature. Common evaluation techniques used in this study varied according to the models.

1. Backtesting: A technique that involves fitting a model to the training data and predictions are made according to the test size of the data. Once that is done the test values and predictions are compared by plotting to learn weather if the model performed according to the train data. If there is a entire detrend in these values that means the model doesn’t work.

Backtesting is done for all the eight models and the best and the bad fitted models are shown in below graphs.

A graph of a person

Description automatically generated with medium confidence

**Figure 50 : VARMA Backtesting**

The above graph shows the test values in blue and predictions in orange. Though the lines started a little far from each other the they has really good match from September. This model is VARMA using multi-regression of three features.

A graph with a line

Description automatically generated

**Figure 51 : LSTM Backtesting**

The above graph is a classic example of a badly performing model on the data, here the blue line suggests the test data and the orange suggests the predictions. The predictions are far off from the test though they started well, the model failed to read the trends in data. It is a flat line with no seasonality. This model is an RNN model using LSTM layers.

1. Statistical measures- Mean Squared Error (MSE) and Root Mean Squared Error (RMSE):

These are calculated for all models by just calling a function called mean-squared\_error() and rmse(). The best RMSE again is the VARMA model with 3 features just like backtesting, the results are shown in the picture. The RMSE value is the error value and it should be compared to the mean of the original value. This pictures shows good rmse value as its just 2 and the average is somewhere around 18. So for every 18 there is 2 error. 11% error can be understood.

A screenshot of a computer code

Description automatically generated

**Figure 52 : RSME VARMA**

The below table shows the RMSE values for all the models:

|  |  |
| --- | --- |
| **Model** | **RMSE** |
| ARIMA (2,0,2) | 6.17 |
| ARIMA (4,0,1) | 15.6 |
| SARIMA (2,0,2,)(2,0,2,12) | 16.0 |
| SARIMA (2,0,2,)(2,0,1,12) | 3.2 |
| VARMA 2 Features | 3 |
| VARMA 3 Features | 2.8 |
| LSTMS(all three) | 9.3 |
|  |  |

**Table 9: Models RSME**

From the above results it is very well understood that the least error is for the VARMA 3 feature model, next best performing was SARIMA with seasonality order (2,0,1,12). Amongst both ARIMA’s the ARIMA (2,0,2) with seasonality is performing better than the one without considering seasonality.

*Refitting and forecasting:*

Refitting the model defined model and forecasting the future with model predictions is the final step involved in the process of modelling. The reason we do this is because at the end any model should be able to predict future, the best performing models according to RMSE scores are given here and made to predict future.

The code is pretty common for all the models, the train data and train predicted data is first concatenated and labelled as combined\_data. The model is then fit to the comnined\_ data a new\_forecast variable is created and the forecasted values according to the model are put in this. At the end, the combined\_ data and the new \_forecast are plotted and observed the pattern.

The below pictures show the predictions of all the well performing models, in theseARIMA (4,0,1), SARIMA (2,0,2)(2,0,2,12) and LSTM aren’t displayed as the error is too high and the back testing wasn’t great too.

A screenshot of a graph

Description automatically generatedA number of numbers on a white background

Description automatically generated

**Figure 53 : ARIMA Forecasts for future**

The model associated is ARIMA. The pictures above on the left shows the plots of test and predictions which are pretty close to each other. Predicted values are shown towards the right.

A graph with red and blue lines

Description automatically generatedA screenshot of a computer

Description automatically generated

**Figure 54 : SARIMA Forecasts for future**

The model associated is SARIMA, the left shows the plot of predictions and test and the right figure shows the energy predicted in those months.

A screenshot of a computer

Description automatically generatedA graph with blue lines

Description automatically generated

**Figure 55 : VARMA Forecasts for future**

The above model is VARMA with 3 features respectively. This by far the best performing models. The predicted values for future months are in the first figure and the associated plot in the next figure.

A blue and green line graph

Description automatically generated

**Figure 56 : Linear Regression Forecasts for future**

The above figure shows the linear regression from scratch and the green line depicts the future forecast. This model compared to others discussed above predicted differently.

## Conclusion:

From the above analysis, of different models their predictions and error values, the best performed model amongst all was the VARMAX with 3 components. This model had the least error when compared to the rest of the models and also handled the multivariate data. Hence for the Energy dataset in study the predicted values for energy consumption in future are listed below, these values are not just predicted by the previous years energy consumption but also the effect of the features: mean temperature and mean hdd played a vital role in forecasting these values for future reference.

|  |  |  |  |
| --- | --- | --- | --- |
| **Month and Year of forecast** | **Energy consumption in KWh** | **Mean Temperature** | **Mean HDD** |
| 01-01-2023 | 35 | 6 | 8 |
| 01-02-2023 | 31 | 7 | 6 |
| 01-03-2023 | 25 | 9 | 5 |
| 01-04-2023 | 19 | 11 | 4 |
| 01-05-2023 | 14 | 13 | 3 |
| 01-06-2023 | 11 | 14 | 2 |

**Table 10: Predictions table for future by best model**

The future of this study can be that, there are many new models coming into the existence every day, not only models but also techniques involving seasonal decomposition and other statistical measures which are helping one to understand the trends and seasonality better. These models can be explored on newly available datasets and analysis can be done accordingly. Especially, energy companies can have abundance of this data exponentially, that can be put to a great use by following these techniques.

## References:

1. *Smart meters: A guide for Households* (no date) *GOV.UK*. Available at: https://www.gov.uk/guidance/smart-meters-how-they-work (Accessed: 18 September 2023).
2. Anish, A. (2020) *Time Series analysis*, *Medium*. Available at: https://medium.com/swlh/time-series-analysis-7006ea1c3326 (Accessed: 18 September 2023).
3. Joshi, ~ Prateek (2022) *What is long memory in time series analysis*, *Perpetual Enigma*. Available at: https://prateekvjoshi.com/2016/08/27/what-is-long-memory-in-time-series-analysis/ (Accessed: 18 September 2023).
4. *Components of time series analysis: Trends and mathematical models - TOPPR*. Available at: https://www.toppr.com/guides/business-mathematics-and-statistics/time-series-analysis/components-of-time-series/ (Accessed: 18 September 2023).
5. Brownlee, J. (2020) *How to decompose time series data into trend and seasonality*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/ (Accessed: 18 September 2023).
6. Nielsen, A. (2019) *Practical time series analysis*. O’Reilly Media.
7. Brownlee, J. (2020a) *How to check if time series data is stationary with python*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/time-series-data-stationary-python/ (Accessed: 18 September 2023).
8. *NIST/SEMATECH e-Handbook of Statistical Methods*. Available at: https://www.itl.nist.gov/div898/handbook/ (Accessed: 18 September 2023).
9. *Forecasting: Principles and practice (2nd ed)* (no date) *11.2 Vector autoregressions*. Available at: https://otexts.com/fpp2/VAR.html (Accessed: 18 September 2023).
10. *Medium*. Available at: https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better- (Accessed: 18 September 2023).
11. *Explore scientific, technical, and medical research on ScienceDirect* (no date) *ScienceDirect.com | Science, health and medical journals, full text articles and books.* Available at: https://www.sciencedirect.com/ (Accessed: 18 September 2023).
12. *Effective transfer learning algorithm in spiking neural networks | IEEE ...* Available at: https://ieeexplore.ieee.org/abstract/document/9489375 (Accessed: 18 September 2023).
13. Li, S. (2018) *An end-to-end project on time series analysis and forecasting with python*, *Medium*. Available at: https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b (Accessed: 18 September 2023).
14. Kumari, K. (2023) *Time series forecasting using Python*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2022/06/time-series-forecasting-using-python/ (Accessed: 18 September 2023).
15. Business, F.S. of (no date) *Identifying the orders of Ar and MA terms in an Arima model*. Available at: https://people.duke.edu/~rnau/411arim3.htm (Accessed: 18 September 2023).

## Appendices

!pip install scipy

!pip install arch

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from datetime import datetime, timedelta

from dateutil.relativedelta import relativedelta

from scipy import stats

from statsmodels.tsa.stattools import adfuller, kpss

from datetime import datetime

import matplotlib.dates as mdates

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.tsa.x13 import x13\_arima\_select\_order

# from statsmodels.tsa import x12

from statsmodels.tsa.seasonal import STL

import statsmodels.api as sm

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf

from statsmodels.tsa.holtwinters import SimpleExpSmoothing

from arch import arch\_model

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from statsmodels.tsa.statespace.exponential\_smoothing import ExponentialSmoothing

import statsmodels.api as sm

# Import the models we'll be using in this section

from statsmodels.tsa.stattools import acovf,acf,pacf,pacf\_yw,pacf\_ols

from pandas.plotting import lag\_plot

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf

from statsmodels.tsa.stattools import grangercausalitytests

from sklearn.metrics import mean\_squared\_error

from statsmodels.tools.eval\_measures import rmse

data = pd.read\_csv("Dissertation.csv")

datastamp='mean'

# Define the start date as 'Jan-20'

start\_date = datetime.strptime('Jan-20', '%b-%y')

# Generate the list of month-year values from 'Jan-20' to 'Dec-21'

date\_values = [(start\_date + relativedelta(months=i)).strftime('%b-%y') for i in range(24)]

# Extract rows with the generated month-year values

filtered\_df = data[data['summary\_time'].isin(date\_values)]

# Extract rows with the generated month-year values

filtered\_df = filtered\_df[filtered\_df['summary\_stat']==datastamp]

# filtered\_df = filtered\_df[filtered\_df['segmentation\_variable\_1']=='None']

# Convert the 'Date' column to datetime format

filtered\_df['summary\_time'] = pd.to\_datetime(filtered\_df['summary\_time'], format='%b-%y')

# Convert the 'Date' column to the desired format 'dd-mm-yyyy'

filtered\_df['summary\_time'] = filtered\_df['summary\_time'].dt.strftime('%m-%d-%Y')

df=filtered\_df

df

### EDA:

#Counting number of null values in the dataset

null\_values=df.isnull().sum()

print(null\_values)

column\_values = df['mean\_bedrooms']

print(column\_values)

column\_to\_round = 'mean\_bedrooms'

df[column\_to\_round] = df[column\_to\_round].round(decimals=0)

bedroomvalue = df['mean\_bedrooms']

print(bedroomvalue)

column\_values = df['mean\_occupants']

print(column\_values)

column\_to\_round = 'mean\_occupants'

df[column\_to\_round] = df[column\_to\_round].round(decimals=0)

bedroomvalue = df['mean\_occupants']

print(bedroomvalue)

column\_to\_fill = 'mean\_bedrooms'

replacement\_value = 2

df[column\_to\_fill] = df[column\_to\_fill].fillna(replacement\_value)

column\_to\_fill = 'mean\_occupants'

replacement\_value = 1

df[column\_to\_fill] = df[column\_to\_fill].fillna(replacement\_value)

null\_values=df.isnull().sum()

print(null\_values)

# Filter non-positive values

non\_positive\_values = df[df['value'] <= 0]

# Count the number of non-positive values

count\_non\_positive = len(non\_positive\_values)

# Print the count

print("Count of non-positive values:", count\_non\_positive)

# Filter non-positive values and display only the column of interest

non\_positive\_values = df.loc[df['value'] <= 0, 'value']

# Display the non-positive values

print(non\_positive\_values)

# Filter out non-positive values

df = df[df['value'] > 0]

plt.hist(df['mean\_temp'], bins='auto')

# Filter out non-positive values

df = df[df['value'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# Plot the transformed column

plt.hist(df['value'], bins='auto')

plt.xlabel('Transformed Values')

plt.ylabel('Frequency')

plt.title('Histogram of Transformed Column')

plt.show()

# Filter out non-positive values

df = df[df['mean\_temp'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# Plot the transformed column

plt.hist(df['mean\_temp'], bins='auto')

plt.xlabel('Transformed Values')

plt.ylabel('Frequency')

plt.title('Histogram of Transformed Column')

plt.show()

# Filter out non-positive values

df = df[df['mean\_hdd'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# Plot the transformed column

plt.hist(df['mean\_hdd'], bins='auto')

plt.xlabel('Transformed Values')

plt.ylabel('Frequency')

plt.title('Histogram of Transformed Column')

plt.show()

# Filter out non-positive values

df = df[df['mean\_floor\_area'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# Filter out non-positive values

df = df[df['mean\_bedrooms'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# Filter out non-positive values

df = df[df['mean\_occupants'] > 0]

# Apply Box-Cox transformation to the remaining values

transformed\_data, lambda\_value = stats.boxcox(df['value'])

# Print the estimated lambda value

print("Estimated lambda value:", lambda\_value)

# # Convert the 'time\_column' to datetime format

df['summary\_time'] = pd.to\_datetime(df['summary\_time'])

df = df.sort\_values(by='summary\_time', ascending=True)

# Set the 'time\_column' as the new index

df.set\_index('summary\_time', inplace=True)

# Verify the updated DataFrame

Df

sum\_aggregated\_df = df.groupby(df.index)['value','mean\_temp','mean\_hdd','mean\_floor\_area','n\_mean\_floor\_area','mean\_bedrooms','n\_mean\_bedrooms','mean\_occupants','n\_mean\_occupants'].mean()

print(sum\_aggregated\_df)

plt.plot(sum\_aggregated\_df['value'])

print("Observations if stationary tests")

X=sum\_aggregated\_df['value']

stats\_adf=[]

stats\_kpss=[]

stats\_pp=[]

#ADF TEST

adf= adfuller(X.dropna())

adf\_test\_stat=adf[0]

adf\_P\_value=adf[1]

if adf\_P\_value <=0.05:

stats\_adf="ADF Stationary"

else:

stats\_adf="ADF NS"

#KPSS TEST

kpss\_result=kpss(X.dropna())

kpss\_test\_stat=kpss\_result[0]

kpss\_p\_value=kpss\_result[1]

if kpss\_p\_value >=0.05:

stats\_kpss= "KPSS S"

else:

stats\_kpss="KPSS NS"

#PP TEST

pp\_result=adfuller(X.dropna(),regression='ct')

pp\_test\_stat=pp\_result[0]

PP\_p\_value=pp\_result [1]

if PP\_p\_value <=0.05:

stats\_pp="PP S"

else:

stats\_pp= "PP NS"

print(stats\_adf+stats\_kpss+stats\_pp)

# print("Series is stationary if:")

#additive seasonal decompose

stl = STL(X)

result = stl.fit()

seasonal, trend, residual = result.seasonal, result.trend, result.resid

result.plot();

# Hodrick-Prescott Filter Decomposition

cycle, trend = sm.tsa.filters.hpfilter(X)

# Plot the trend component

plt.figure(figsize=(12, 6))

plt.plot(sum\_aggregated\_df.index, trend, label='Trend', color='blue')

plt.plot(sum\_aggregated\_df.index, cycle, label='Cycle', color='green')

plt.xlabel('Date')

plt.ylabel('Trend')

plt.title('Hodrick-Prescott Filter Decomposition - Trend Component')

plt.legend()

plt.grid(True)

plt.show()

# Seasonal decomposition of the aggregated values

result = seasonal\_decompose(sum\_aggregated\_df['value'], model='multiplicative')

# Plot the original data, trend, seasonal, and residual components

plt.figure(figsize=(10, 6))

result.plot();

w=result.trend\*result.resid\*result.seasonal/result.seasonal

w

#ACF

title = 'Autocorrelation'

lags = 23

plot\_acf(sum\_aggregated\_df['value'],title=title,lags=lags);

#PACF

title='Partial Autocorrelation'

lags=11

plot\_pacf(result.seasonal,title=title,lags=lags);

sum\_aggregated\_df['value'].expanding().mean().plot(figsize=(12,5))

ax = sum\_aggregated\_df['value'].plot(figsize=(20,6),xlim=['2020-01-01','2021-12-01'],ylim=[0,50])

ax.yaxis.grid(True)

ax.xaxis.grid(True)

sum\_aggregated\_df['EWMA12'] = sum\_aggregated\_df['value'].ewm(span=12,adjust=False).mean()

sum\_aggregated\_df[['value','EWMA12']].plot();

sum\_aggregated\_df.index.freq = 'MS'

sum\_aggregated\_df.index

span = 12

alpha = 2/(span+1)

sum\_aggregated\_df['EWMA12'] = sum\_aggregated\_df['value'].ewm(alpha=alpha,adjust=False).mean()

sum\_aggregated\_df['SES12']=SimpleExpSmoothing(sum\_aggregated\_df['value']).fit(smoothing\_level=alpha,optimized=False).fittedvalues.shift(-1)

sum\_aggregated\_df.head()

from statsmodels.tsa.holtwinters import ExponentialSmoothing

sum\_aggregated\_df['DESadd12'] = ExponentialSmoothing(sum\_aggregated\_df['value'], trend='add').fit().fittedvalues.shift(-1)

sum\_aggregated\_df.head()

subset\_df = sum\_aggregated\_df[['value', 'EWMA12', 'DESadd12']].iloc[:24]

# Plot the selected columns

subset\_df.plot(figsize=(12, 6))

# Configure x-axis limits

plt.autoscale(axis='x', tight=True)

# Show the plot

plt.show()

sum\_aggregated\_df['DESmul12'] = ExponentialSmoothing(sum\_aggregated\_df['value'], trend='mul').fit().fittedvalues.shift(-1)

sum\_aggregated\_df.head()

subset\_df = sum\_aggregated\_df[['value', 'EWMA12', 'DESadd12', 'DESmul12']].iloc[:24]

# Plot the selected columns

subset\_df.plot(figsize=(12, 6))

# Configure x-axis limits

plt.autoscale(axis='x', tight=True)

# Show the plot

plt.show()

sum\_aggregated\_df['TESadd12'] = ExponentialSmoothing(sum\_aggregated\_df['value'],trend='add',seasonal='add',seasonal\_periods=12).fit().fittedvalues

sum\_aggregated\_df[['value','TESadd12','DESadd12']].plot(figsize=(12,6)).autoscale(axis='x',tight=True);

sum\_aggregated\_df[['value','DESadd12']].plot(figsize=(12,6)).autoscale(axis='x',tight=True);

sum\_aggregated\_df['TESmul12'] = ExponentialSmoothing(sum\_aggregated\_df['value'],trend='mul',seasonal='mul',seasonal\_periods=12).fit().fittedvalues

sum\_aggregated\_df[['value','TESadd12','TESmul12']].plot(figsize=(12,6)).autoscale(axis='x',tight=True);

sum\_aggregated\_df.head()

sum\_aggregated\_df.tail()

train\_data = sum\_aggregated\_df.loc[:'2021-12-01']

test\_data = sum\_aggregated\_df.loc['2021-08-01':]

fitted\_model = ExponentialSmoothing(train\_data['TESmul12'],initialization\_method='estimated',trend='add',seasonal=12).fit()

test\_predictions = fitted\_model.forecast(6).rename('HW Forecast')

test\_predictions

train\_data['TESmul12'].plot(legend=True,label='TRAIN')

test\_data['TESmul12'].plot(legend=True,label='TEST',figsize=(12,8));

train\_data['TESmul12'].plot(legend=True,label='TRAIN')

test\_data['TESmul12'].plot(legend=True,label='TEST',figsize=(12,8))

test\_predictions.plot(legend=True,label='PREDICTION');

lag\_plot(sum\_aggregated\_df['value']);

# ACF array

acf(sum\_aggregated\_df['value'])

sum\_aggregated\_df.info

sum\_aggregated\_df.index.freq = 'MS'

sum\_aggregated\_df[['value','mean\_hdd']].plot(figsize=(16,5))

sum\_aggregated\_df['value'].iloc[2:].plot(figsize=(16,5),legend=True);

sum\_aggregated\_df['mean\_temp'].shift(2).plot(legend=True);

grangercausalitytests(sum\_aggregated\_df[['value','mean\_temp']],maxlag=5);

grangercausalitytests(sum\_aggregated\_df[['value','mean\_hdd']],maxlag=5);

#has most p values greater than 0.5

grangercausalitytests(sum\_aggregated\_df[['value','mean\_floor\_area']],maxlag=5);

#has most p values greater than 0.5

grangercausalitytests(sum\_aggregated\_df[['value','mean\_bedrooms']],maxlag=5);

grangercausalitytests(sum\_aggregated\_df[['value','mean\_occupants']],maxlag=5);

from statsmodels.graphics.tsaplots import month\_plot,quarter\_plot

month\_plot(sum\_aggregated\_df['value']);

dfq = sum\_aggregated\_df['value'].resample(rule='Q').mean()

quarter\_plot(dfq);

import pandas as pd

import numpy as np

%matplotlib inline

# Load specific forecasting tools

from statsmodels.tsa.arima\_model import ARMA,ARMAResults,ARIMA,ARIMAResults

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf # for determining (p,q) orders

!pip install pmdarima

from pmdarima import auto\_arima # for determining ARIMA orders

auto\_arima(sum\_aggregated\_df['value'],seasonal=False).summary()

stepwise\_fit= auto\_arima(sum\_aggregated\_df['value'],start\_p=0,start\_q=0,max\_p=6,max\_q=3,seasonal=False,trace=True)

auto\_arima(sum\_aggregated\_df['value'],seasonal=True).summary()

auto\_arima(sum\_aggregated\_df['value']).summary()

auto\_arima(sum\_aggregated\_df['value']).summary()

from statsmodels.tsa.arima\_model import ARMA,ARMAResults,ARIMA,ARIMAResults

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf # for determining (p,q) orders

from pmdarima import auto\_arima # for determining ARIMA orders

!pip install statsmodels

import statsmodels.api as sm

from statsmodels.tsa.arima.model import ARIMA

train = sum\_aggregated\_df.loc[:'2021-06-01']

test = sum\_aggregated\_df.loc['2021-06-01':]

model = ARIMA(train['value'],order=(2,0,2))

results = model.fit()

results.summary()

# Obtain predicted values

start=len(train)

end=len(train)+len(test)-1

predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('ARIMA(2,0,2) Predictions')

test['value'].plot(legend=True, figsize=(12,8))

predictions.plot(legend=True)

rooterror = rmse(test['value'], predictions)

rooterror

error = mean\_squared\_error(test['value'], predictions)

error

test['value'].mean()

# Combine the training and test data, including the previously predicted values

combined\_data = pd.concat([train['value'], predictions])

# Retrain the ARIMA(2,0,2) model with the updated dataset

model = ARIMA(combined\_data, order=(2, 0, 2))

results = model.fit()

# Generate a new months forecast

df\_forecast\_new = results.forecast(steps=16)

# Display the new forecasted values

print(df\_forecast\_new)

combined\_data.plot(legend=True,color='orange')

df\_forecast\_new.plot(legend=True,color='red')df\_forecast\_new.plot(legend=True,color='red')

train = sum\_aggregated\_df.iloc[:12]

test = sum\_aggregated\_df.iloc[12:]

model = ARIMA(train['value'],order=(4,0,1))

results= model.fit()

results.summary()

# Obtain predicted values

start=len(train)

end=len(train)+len(test)-1

predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('ARIMA(4,0,1) Predictions')

test['value'].plot(legend=True, figsize=(12,8))

predictions.plot(legend=True)

error = mean\_squared\_error(test['value'], predictions)

error

rooterror = rmse(test['value'], predictions)

rooterror

test['value'].mean()

# Combine the training and test data, including the previously predicted values

combined\_data = pd.concat([train['value'], predictions])

# Retrain the ARIMA(2,0,2) model with the updated dataset

model = ARIMA(combined\_data, order=(4, 0, 1))

results = model.fit()

# Generate a new months forecast

df\_forecast\_new = results.forecast(steps=20)

# Display the new forecasted values

print(df\_forecast\_new)

combined\_data.plot(legend=True,color='orange')

df\_forecast\_new.plot(legend=True,color='red')

# Set one year for testing

train = sum\_aggregated\_df.iloc[:12]

test = sum\_aggregated\_df.iloc[12:]

model = SARIMAX(train['value'],order=(2,0,2),seasonal\_order=(2, 0, 2, 12))

results = model.fit()

results.summary()

start=len(train)

end=len(train)+len(test)-1

predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('SARIMA(2,0,2) Predictions')

test['value'].plot(legend=True,figsize=(12,6))

predictions.plot(legend=True)

error = mean\_squared\_error(test['value'], predictions)

error

rooterror = rmse(test['value'], predictions)

rooterror

test['value'].mean()

model = SARIMAX(train['value'],order=(2,0,2),seasonal\_order=(2, 0, 2, 12))

results = model.fit()

fcast = results.predict(len(sum\_aggregated\_df),len(sum\_aggregated\_df)+5,typ='levels').rename('SARIMA(2,0,2)(2,0,2,12) Forecast')

sum\_aggregated\_df['value'].plot(legend=True,figsize=(12,6))

fcast.plot(legend=True)

# Combine the training and test data, including the previously predicted values

combined\_data = pd.concat([train['value'], predictions])

# Retrain the SARIMA(2,0,2) model with the updated dataset

model = SARIMAX(combined\_data,order=(2, 0, 2),seasonal\_order=(2, 0, 2, 12))

results = model.fit()

# Generate a new months forecast

df\_forecast\_new = results.forecast(steps=16)

# Display the new forecasted values

print(df\_forecast\_new)

combined\_data.plot(legend=True,figsize=(12,12),color='blue')

df\_forecast\_new.plot(legend=True,color='red')

model = SARIMAX(train['value'],order=(2,0,2),seasonal\_order=(2, 0, 1, 12))

results = model.fit()

results.summary()

start=len(train)

end=len(train)+len(test)-1

predictions = results.predict(start=start, end=end, dynamic=False, typ='levels').rename('SARIMA(2,0,2) Predictions')

test['value'].plot(legend=True,figsize=(12,6))

predictions.plot(legend=True)

error = mean\_squared\_error(test['value'], predictions)

error

rooterror = rmse(test['value'], predictions)

rooterror

test['value'].mean()

# Combine the training and test data, including the previously predicted values

combined\_data = pd.concat([train['value'], predictions])

# Retrain the SARIMA(2,0,2) model with the updated dataset

model = SARIMAX(combined\_data,order=(2, 0, 2),seasonal\_order=(2, 0, 2, 12))

results = model.fit()

# Generate a new months forecast

df\_forecast\_new = results.forecast(steps=16)

# Display the new forecasted values

print(df\_forecast\_new)

combined\_data.plot(legend=True,figsize=(12,12),color='blue')

df\_forecast\_new.plot(legend=True,color='red')

cols = ['value','mean\_temp','mean\_hdd','mean\_floor\_area','n\_mean\_floor\_area','mean\_bedrooms','n\_mean\_bedrooms','mean\_occupants','n\_mean\_occupants']

for col in cols:

sum\_aggregated\_df[col] = sum\_aggregated\_df[col].astype(int)

sum\_aggregated\_df.head()

auto\_arima(sum\_aggregated\_df['value'],exogenous=sum\_aggregated\_df['mean\_temp'],seasonal=True,m=1).summary()

model = SARIMAX(train['value'],exog=train['mean\_temp'],order=(2,0,2),enforce\_invertibility=False)

results = model.fit()

results.summary()

start=len(train)

end=len(train)+len(test)-1

exog\_forecast = test[['mean\_temp']] # requires two brackets to yield a shape of (35,1)

predictions = results.predict(start=start, end=end, exog=exog\_forecast).rename('SARIMAX Predictions')

test['value'].plot(legend=True,figsize=(12,6),title=title)

predictions.plot(legend=True)

error = rmse(test['value'], predictions)

error

auto\_arima(sum\_aggregated\_df['value'])

auto\_arima(sum\_aggregated\_df['mean\_temp'])

# Calculate the split point

split\_point = int(0.75 \* len(sum\_aggregated\_df))

# Split the data into training and testing sets using iloc

train = sum\_aggregated\_df.iloc[:split\_point]

test = sum\_aggregated\_df.iloc[split\_point:]

model = VARMAX(train[['value','mean\_temp']], order=(2, 2), trend='c')

results = model.fit(maxiter=1000, disp=False)

results.summary()

df\_forecast = results.forecast(6)

df\_forecast

sum\_aggregated\_df['value'].iloc[18:].plot(figsize=(12,5),legend=True).autoscale(axis='x',tight=True)

df\_forecast['value'].plot(legend=True);

sum\_aggregated\_df['mean\_temp'].iloc[18:].plot(figsize=(12,5),legend=True).autoscale(axis='x',tight=True)

df\_forecast['mean\_temp'].plot(legend=True);

error = rmse(sum\_aggregated\_df['value'].iloc[18:], df\_forecast['value'])

error

# Extend your dataset with the forecasted values

sum\_aggregated\_df = pd.concat([train, df\_forecast])

# Check the updated DataFrame

print(sum\_aggregated\_df.tail(10)) # Just to verify that the new data is appended

order = (2, 2)

# Build and fit the VARMAX model with the extended dataset

model = VARMAX(sum\_aggregated\_df[['value', 'mean\_temp']], order=order, trend='c')

results = model.fit(maxiter=1000, disp=False)

# Generate a new 6-month forecast

df\_forecast\_new = results.forecast(steps=12)

# Display the new forecasted values

print(df\_forecast\_new)

sum\_aggregated\_df['value'].plot(legend=True,figsize=(12,12),color='blue')

df\_forecast\_new['value'].plot(legend=True,color='red')

columns\_to\_normalize = ['value', 'mean\_temp','mean\_hdd']

scaler = StandardScaler()

sum\_aggregated\_df[columns\_to\_normalize] = scaler.fit\_transform(sum\_aggregated\_df[columns\_to\_normalize])

print(df)

train = sum\_aggregated\_df.iloc[:18]

test = sum\_aggregated\_df.iloc[18:]

model = VARMAX(train[['value','mean\_temp','mean\_hdd']], order=(2, 2), trend='c')

results = model.fit(maxiter=50, disp=False)

results.summary()