University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0206-2023
Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Power consumption prediction of a Household using Machine Learning Models.

Student Name and SRN:

Sam Philip Solo, 22013137

Supervisor: Dr. Alyssa Drake

Date Submitted: May 2 2024

Word Count: 8483

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project **module** or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used ChatGPT, or any other Generative AI tool, to write the report **or code (other than where I have** declared **or referenced it).**

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Sam Philip Solo

Student Name Signature: Sam Philip Solo

Student SRN number: 22013137

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Power consumption has grown due to population growth, lifestyle changes, and technological advancements. This increases the use of renewable resources while simultaneously raising energy prices. Also, as a result of the conflict, there has been a significant increase in power use, which has raised the price of electricity. With technological improvements, it is now feasible to reduce power use using machine learning approaches. Time series is an important aspect in machine learning, and it is employed in a variety of areas, including healthcare, finance, and business. Time series models provide for forecasts and control over power use. As univariate data, a dataset from the UCI Machine Learning Repository was selected, which had a feature called active power in kW. Data preparation was undertaken since the data comprised huge amounts of missing, inconsistent data; also, exploratory data analysis was performed to better comprehend the data. The prediction job was carried out using the Statistical and Classical Model (ARIMA), the Decision Tree Model (XGBoost), and the Deep Learning Model (LSTM). Each model had different findings. Since this investigation was conducted across four years, long-term dependencies were addressed, demonstrating that LSTM outperformed alternative models such as XGBoost and ARIMA. ARIMA, on the other hand, works well with short-term dependencies. This study explores several time series prediction models.

Keywords: Power consumption, Machine Learning, Time Series, ARIMA, XGBoost, LSTM

ACKNOWLEDGEMENT

First of all, I would like to thank God Almighty for his immense blessings upon me to complete this project. Even though various obstacles came into my life, God's guidance was always with me providing with wisdom and knowledge and strengthening me day by day.

I would like to thank Dr. Alyssa Drake, who served as my supervisor since the beginning of this project. Her constant advice advice and support throughout the project made it a success.

I would like to thank Dr Carolyn Devereux, course leader and to all the professors of University of Hertfordshire who helped me gain knowledge and understanding of various subjects throughout the course.

I would like to thank my dear parents and family members, friends, for their encouragement and support throughout this journey.

CONTENTS

1. INTRODUCTION

With the latest inventions, advancements in technology, population, lifestyle, and economic development, the usage of electricity in various parts of the world has increased since 1950s. For a country's growth, from low level to modernizing reaching industrial levels involving quantitative and qualitative improvements, economic development is a big factor which includes agriculture, industry, urbanization, transport, and wealth are some of the major development areas. The rising population makes cause for the increase in demand for electricity as in a household for example, with increase in the number of family members, chances are for high usage. Since the evolution of technology, energy usage is increasing day by day which causes both positives and negatives. The positive side can be replacing existing appliances to energy-efficient ones, vehicles being converted or manufactured to fuel-efficient ones, reducing emissions and so on. While negative sides is, obvious with the increased usage of energy (Bennett, 2023).

As the lifestyle of people and development in various parts escalated, power consumption has also increased. However, the Russia-Ukraine war (Ray, 2024) which began in February 2022, has affected in some countries within Europe and other parts of the world. As the war began during the winter season, Russia targeted power plants and other energy infrastructures, which caused damage and less access for the Ukrainians to electricity and heating systems. As Ukraine gets some sources for its power supply from its defense systems, Russia tries to invade it, making the Ukrainians seek additional resources such as energy, fossil fuels from its neighbouring countries and allies to tackle war. This caused an increase in power consumption and usage, with energy price shot up, making it to turn out for the first energy global energy crisis (Agency, International Energy, 2024). The war not only a crisis on the energy market but also on natural gas, which is a major target for the hike in electricity where gas is a primary resource for energy production. With the consumption of fossil fuels, natural calamities are happening and has affected in various parts of the world, leading to crises in every sector, crop failure and food shortage, raising the price of every meal that's available in the market (Ralston, 2024). In the UK, two-thirds of people say that the cost of living has increased concerning all these factors, making life more and more difficult and the reasons they mentioned were high bills for electricity and gas usage, and electricity is the most demanding compared with other goods and services. This affects people who come under the lower class category to pay more energy bills and also people who require heating during the winter where they can't avoid paying higher bills (Lorusso, 2023).

With this dilemma kept, energy consumption should be reduced. With the advancement of technology and with the existence of the digital world, such as Internet of Things (IoT), concepts such as Artificial Intelligence (AI), Machine Learning (Sarker, 2021), allow to function intelligently with the help of several learning algorithms can predict the usage of power consumption of a household, residential, commercial buildings and so on. With the data available, be it structured or unstructured, we can get some insights, by plotting or generating various visualizations that make it understandable to a customer and then think to go for less power consumption methods. Predicting energy consumption can be done using various statistical and artificial intelligence methods, where the statistical method uses regression, which is useful for predicting energy, correlate energy consumption, and energy-saving predictions (Hai-xiang Zhao, 2012). This prediction with various models can be performed using Time Series (Chirag Deb, 2017), which is an ordered sequence of values recorded over equal intervals of time. In time series, understanding the pattern of the observed data, and model fitting for future predictions plays a significant role. All these rely on how we select the data, past behaviour or trend of the data, and so on. These steps are crucial for power consumption.

1.1Aim

The primary aim of this project is to do time series analysis and predict the power consumption of a household using some machine learning models such as Auto Regressive Integrated Moving Average (ARIMA), Xtreme Gradient Boosting (XGBoost), and Long Short Term Memory (LSTM). The project aims to find which model shows the lowest RMSE value to conclude that the model is considered to be the best for predicting power consumption.

1.2 Objective

The objectives of this project are:

- Examining the dataset:

  For machine learning tasks, the initial objective is to undertake data examination known as data preprocessing. Since large datasets contain errors (or noise), outliers, missing value and inconsistent data, data preprocessing is an important step for machine learning models to work, if not can lead to incorrect values.

- Evaluation of the chosen model

  From the chosen data after the preprocessing steps, a particular model is to be chosen and implemented. Time series predictions can be performed with various model algorithms. For this project, three models such as ARIMA, XGBoost and LSTM are chosen and a comparative analysis is performed for the same.

- Visualization and choosing the right model

  Once model evaluation is done, model performance such as RMSE and MAE is to be performed. The lower values suggest for best model.

2. LITERATURE REVIEW

Data Science is a vast and growing field in every sector, where popularity of forecasting using various machine learning models is still increasing. Forecasting rely on historical data, trends, outcomes and behaviours. Power consumption forecasting has made an escalating demand especially in efficient energy management, prediction of power consumption, price predictions and so on. The literature review on various papers explains various forecasting methodologies used for prediction of energy. These really show the advancements of this field, where varying range of forecasting techniques, from classical statistical, hybrid to deep learning models were implemented using various datasets.

Sachin et al, (2020) explains the use of RNN-LSTM and the traditional ARIMA model in forecasting of electricity consumption. It highlights the positives and negatives of several methods, however establishes the need for more accurate forecasting methods that can perform mid and long-term forecasts. The dataset is taken from UCI Machine Learning Repository, which is used for this project. Sachin et al(2020) perform data preparation by splitting the data into training and testing to perform forecasting models. The results presented good insights on forecasting machine learning algorithms. For short, mid, and long-term dependencies, each houses with house number were chosen. Sachin et al(2020) concluded that for power consumption forecasting, LSTM and RNN beat ARIMA for mid and long-term forecasts. For each type of prediction dependencies (short,mid and long), the RMSE for each values for each day for each block were predicted. The ARIMA results showed the best result with RMSE of 0.01 for short-term and RNN and LSTM showed an RMSE of 0.02 and 0.14 for mid and long-term dependencies respectively.

Alhussein et al, (2020) used the dataset from the project named SGSC initiated by the Australian government. Data contains power consumption data of each household. Several recordings are taken such as the number of persons living in the house, appliances running at particular time, weather conditions, economic lifestyle, daily routines and so on. Alhussein et al. (2020) contributed this work for short-term household forecasting and did this work by collecting customer numbers and forecasting and also pointed out the necessity of precise load prediction for effective power system management and to examine the issues involved in individual household forecasting. The hybrid CNN-LSTM model was used for this project and applied short-term individual household forecasting. Several layers were used for LSTM model to leverage the strengths of CNN and LSTM architectures. To extract valuable features from time-series data, CNN succeeded while LSTM helped capture short- and long-term dependencies. Moreover, this hybrid model was aimed to improve the forecasting accuracy. From the results given by the models, hybrid model showed their effectiveness compared with other models with MAPE(Mean Absolute Percentage Error) of 40.38 %. This analysis concludes the effectiveness in predicting energy consumption. Also, Alhussein et al (2020) performed clustering analysis with hybrid models, with a silhouette score analysis of 15, which suggested a significant level of similarity within clusters. By grouping households having similar consumption patterns, the clustering approach did improve forecasting accuracy, however varies across households where some did not perform with cluster analysis.

Ghanbari et al, (2021) analyses on different models of LSTM (hybrid models) for time series forecasting. The aim of this project was to find out using these models if its possible for predicting power consumpotion for next seven days. Four year measured data of an household is colleceted for this project. The use of LSTM and its architecture for encoder-decoder, where encoder is responsible for collecting input sequences and encoding to its predefined size and decoder is to predict the output sequence value is discussed. The role of encoder- decoder is played by CNN and LSTM in its hybrid state, also bringing the ConvLSTM model which is capable for multistep series forecasting. Various preprocessing steps were conducted along with data preparation such as converting into training and test data. For this project, Ghanbari et al, (2021) used 75 percent as training data for the model and 25 percent for test and evaluation. Evaulation is done using walk-forward, where in this step, model is retrained with acutal data, building the model as historical data and use to predict each steps. In the results, Ghanbari et al, (2021) plots seven days data using the test data, models such as LSTM, Encoder-Decoder LSTM, CNN-LSTM Encoder-Decoder and ConvLSTM were used. These models vary in inputs such as Univariate and Multivariate time series data. Best RMSE scores for each model were chosen.

Chujai et al, (2013) uses the dataset from UCI Machine Learning Repository of electric power consumption of individual household. This project was done using R programming and uses statistical methods. This project aimed to forecast future data based on historical data that was suitable for short-term forecasts. Four components of time series data- seasonality, trend, cyclical and irregularity are identified by understanding the components for accurate forecasting Chujai et al, (2013) mentions various statistical methods such as regression analysis, classical decomposition, Box Jenkins, and smoothing techniques. Classical models such as ARIMA and SARIMAX are included for forecasting. Various data preprocessing steps were conducted and to detect patterns and trends of power consumption in a household data was constructed and decomposed into real-time series periods in daily, weekly, monthly, and quarterly. The best method is chosen using the lowest RMSE (Root Mean Squared Error) and AIC(Akaike Information Criterion) value respectively. For building ARIMA and ARMA models, Box and Jenkins method was used. In ARIMA, order containing variables p,d, and q can be identified from ACF(Autocorrelation function) and PACF(Partial Autocorrelation function). From the results, it was concluded that ARIMA was the best model for monthly and quarterly forecasting, while for daily and weekly forecasting, ARMA showed good result.

Houda et al, (2022) highlights the worldwide demand of electric power consumption, mentions of accurate forecasting. Two methods were mentioned, deterministic and stochastic methods which explains on precdiciton using the past values that is structured in mathematical representations and with non-zero correlation respectively.These points adds to the point that it considers on the patterns and behaviour to enhance forecasting. Dataset was used on electric power consumption of individual household from UCI Machine Learning repository Houda et al() uses a machine learning model known as XGBoost or eXtreme Gradient Boosting, which is based on sequential ensemble learning and uses decision trees and does this work by providing more accurate forecasts based on hourly, daily and monthly data extracted from the dataset.. Data preparation is done for the working of this model. Missing values with 1.25 percent of rows were observed which were replaced. Conversion of observed variables into trend, seasonal, cyclical and irregular and Time Series auto correlation tasks, to calculate the relationship between actual observation and previous one, called lages were perfomed in Exploratory Data Analysis(EDA). The experimental section explained graphically by Houda et al, (2022) plots predictions of hourly, daily and monthly data vs observed (original data or training data). Also, hyper parameter tuning was conducted with comparitice results where the results after after tuning showed on hourly data was 19.5 and 0.026 for RMSE (Root Mean Squared Error) and MAEP (Mean Absolute Percentage Error) respectively was found to show best result. In result XGBoost offers best machine learning model before or after hyper parameter tuning in terms of MAPE under 30 percent error which is explained by the presence of regularities in activites of electrical consumption over time. Moreover, XGBoost is considered to be the best model for prediction of hourly and daily forecasting periods.

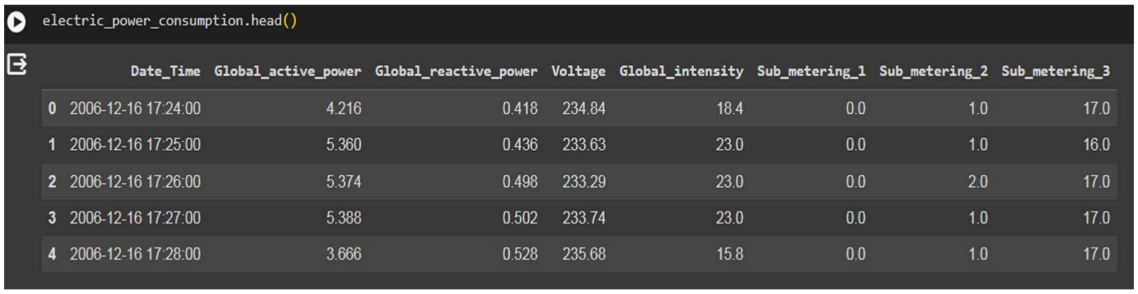
With the context of environmental and economic factors, Deb et al, (2017) explains energy-efficient methods in buildings. The significance of precise energy consumption modelling and forecasting is emphasized in the introduction, especially for existing buildings where occupancy patterns and weather patterns affect energy behavior. The importance of continual monitoring and management of energy use is highlighted, with forecasting playing a critical role in setting boundary conditions and targets for building managers. Several methodologies and applications in building optimisation are reviewed, including simulation-based optimisation, modular-based optimisation systems, and economic model predictive control strategies. Deb et al, (2017) reviews several machine-learning models for energy consumption forecasting. The discussion section examines the benefits and drawbacks of each forecasting technique. For example, artificial neural networks (ANN) are commended for their capacity to map input-output linkages without complex dependencies, but autoregressive integrated moving average (ARIMA) models are known for their universality but complex identification process. Support vector machines (SVMs) are praised for their capacity to handle small samples and nonlinear data, but they are chastised for their lack of outcome clarity. The research focuses on hybrid models, which integrate many strategies to address difficult problems more effectively. Examples of hybrid models include ANN-ARIMA and ARIMA-evolutionary algorithm combinations. These hybrid models are distinguished by their ability to capture both nonlinear and linear components of time series data.

The papers discussed show the importance of machine learning in time series predictions for energy consumption. Several factors depend on the prediction of data such as data preprocessing, feature selection, multistep forecasting, and selection of train and test data for model preparation for power consumption. Deep learning models showed good results compared with classic models. Some papers mention on the implementation of hybrid models was done for prediction tasks to capture some components in time series data.

3. DATASET

The dataset is collected from UC Irvine Machine Learning Repository, which contains collection of datasets, that are used by machine learning community to perform various machine learning algorithms. This repository would not be a success without the assistance of the founders, generators, and people who provided their datasets. Currently, this repository is frequently utilised by students, researchers, and educators to execute machine learning and data mining activities such as model implementation, training, testing, and validation, problem solving, and algorithm learning.

Individual Household Electric Power Consumption dataset includes 2075259 per-minute measurements from a residence in Sceaux, France. The measurements were obtained during a four-year period, from December 2006 to November 2010, and are provided in timestamp format.



```
electric_power_consumption.head()
```

	Date_Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	2006-12-16 17:24:00	4.216	0.418	234.84	18.4	0.0	1.0	17.0
1	2006-12-16 17:25:00	5.360	0.436	233.63	23.0	0.0	1.0	16.0
2	2006-12-16 17:26:00	5.374	0.498	233.29	23.0	0.0	2.0	17.0
3	2006-12-16 17:27:00	5.388	0.502	233.74	23.0	0.0	1.0	17.0
4	2006-12-16 17:28:00	3.666	0.528	235.68	15.8	0.0	1.0	17.0

Figure 1: First five rows of the dataset

3.1 Attribute Information

Attributes, also known as features, are characteristic properties of the data. This describes data instances. These can be used for machine learning algorithms such as training, model evaluation, predictions, and so on. The dataset consists of nine features, each feature explains the following:

- Date: Contains the date in the format yyyy/mm/dd
- Time: Contains the time in the format hh: mm: ss
- Global_active_power: Per minute active power observations consumed by the household (in kilowatt)
- Global_reactive_power: Per-minute reactive power observations consumed by the household (in kilowatt)
- Voltage: Averaged voltage (in Volts)
- Global_intensity: Minute-averaged current intensity (in ampere)
- Sub_metering_1: This is the active energy from Kitchen in that house, containing the Dishwasher, oven, and microwave. It is measured in watt-hour
- Sub_metering_2: This is the active energy from Laundry room in that house, containing washing machine and dryer. It is measured in watt-hour
- Sub_metering_3: This is the active energy for types of climate-controlling equipment such as air conditioner and water heater in that house, measured in watt-hours.

Because this dataset comprises 2075259 metre reading observations, missing values make up around 1.25% of the rows. The missing values are represented by '?'. Missing values can be managed using feature engineering approaches. These stages are explained in the methodology section. Categorical data is included in all of the dataset's characteristics, indicating that the data is organised into categories.

3.2 Exploratory Data Analysis (EDA)

Before implementing a time series model for predictions, it is necessary to understand the dataset. Analyzing a dataset helps to analyze so that it will be easy to summarise the dataset. This is known as Exploratory Data Analysis. Exploratory Data Analysis is important because this helps to identify the patterns, and trends within the data, detect outliers, and find the relationships between the data, which helps to understand more about the dataset. Analysis can be done in various ways, by graphical methods such as barplot, line graph, histogram and so on. EDA can be done using Univariate and Multivariate Time Series Forecasting methods.

For this dataset, the EDA is done using Univariate data, where the column named Global_active_power is chosen and this feature will be used to perform machine learning algorithms. The different EDA performed for this dataset are the following:

A. Outlier Detection of Univariate Data

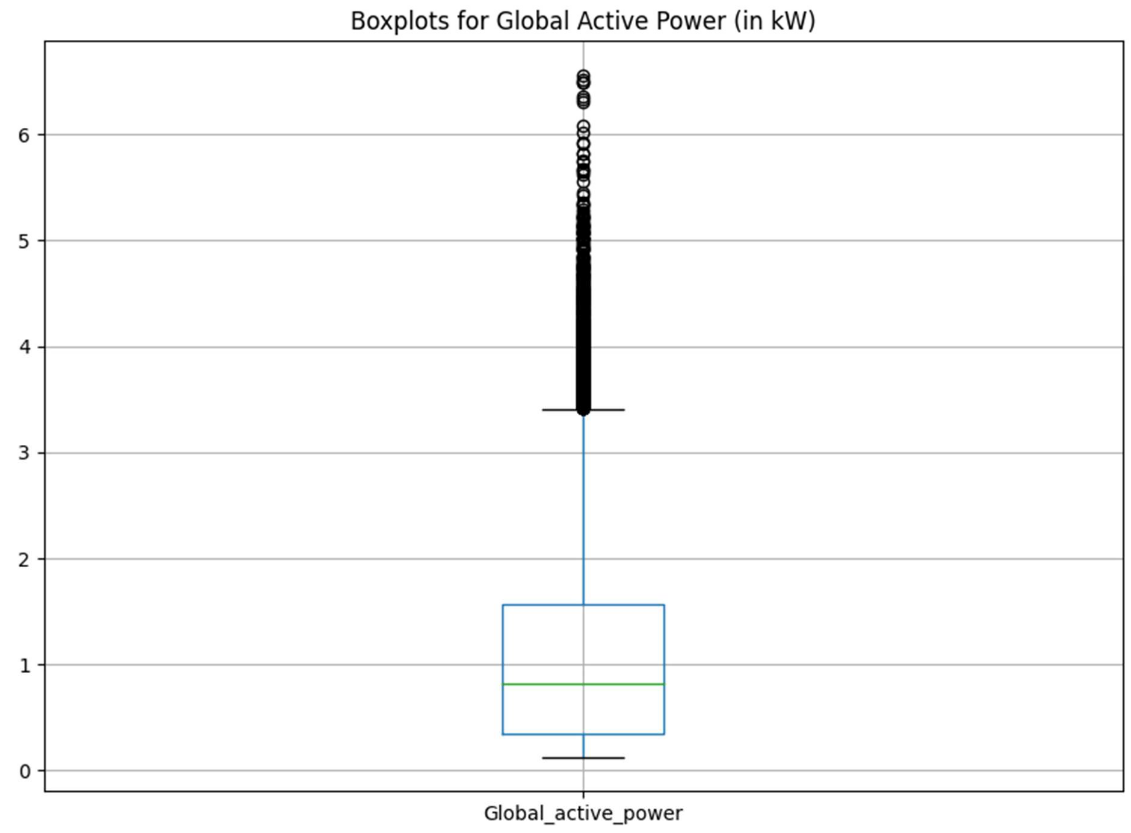


Figure 2: Boxplot for Active Power

Fig 2 shows a box plot of an attribute chosen from the dataset named Global_active_power. Box plots (Third Space Learning, 2024) statistically represent the distribution of the data, where box plot shows pieces of information such as lowest value, lower quartile, median, upper quartile, and highest value. Box plots are best to detect if any outliers exist in the data. Outliers are denoted in black circles which is plotted as individual points. In the figure, the x-axis is Global_active_power and y-axis is numbered from 0 to 6 which describes the piece of information of a boxplot.

B. Particular Time Frame

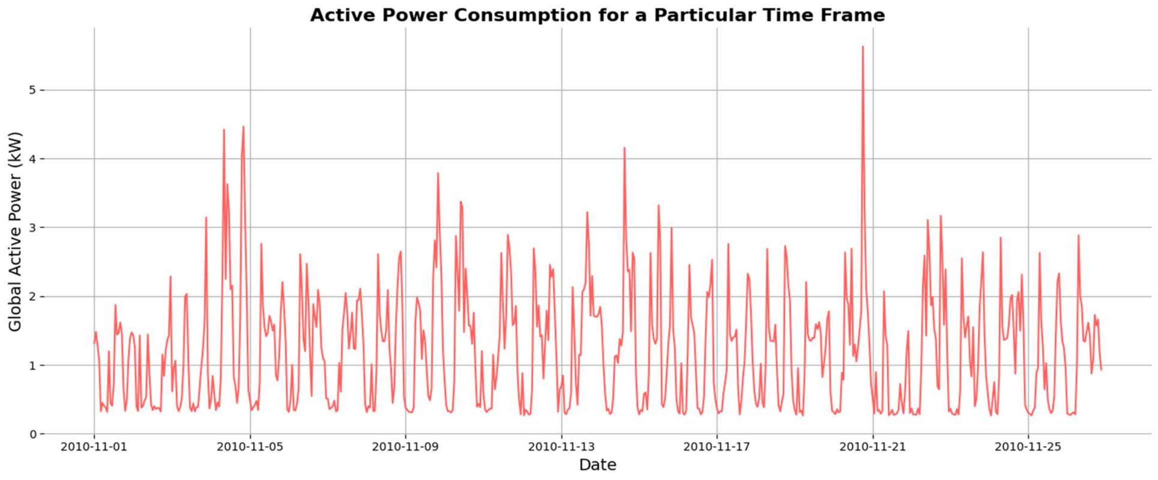


Figure 3: Power Consumption of a Particular Time Frame

Figure 3 depicts the line plot for a certain time window. This task requires univariate data with an attribute named Global_active_power. The line plot was created for November 2010. The graph depicts how data is displayed over time, revealing a daily pattern.

C. Bar plots

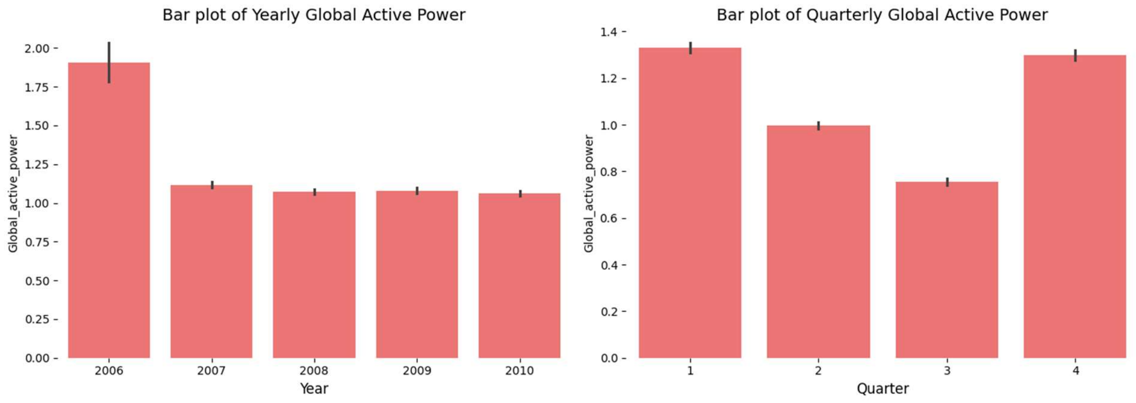


Figure 4: Seasonal Trend in Yearly and Quaterly Global Active Power Consumption

The bar plots in Figure 4 depict the shifting time zones of the database's Global Active Power column throughout an annual and quarterly time frame. According to the first bar plot of yearly worldwide active power, 2006 has the largest utilisation, with an active power range of 1.75 kW. This is because the data for the month of December 2006 was available, and December was thought to have a strong demand for heating appliances owing to the cold. While active power in the remaining bar plots remained consistent between 2007 and 2010, at 1.10 kW. The second figure, which depicts four quarters, demonstrates that the first and second quarters of each year of the dataset had the highest utilisation, with active power exceeding 1.2kW. This rise might be attributed to an increase in power use, since heaters and other mechanical equipment will be in high demand throughout the winter season.

D. Subplots

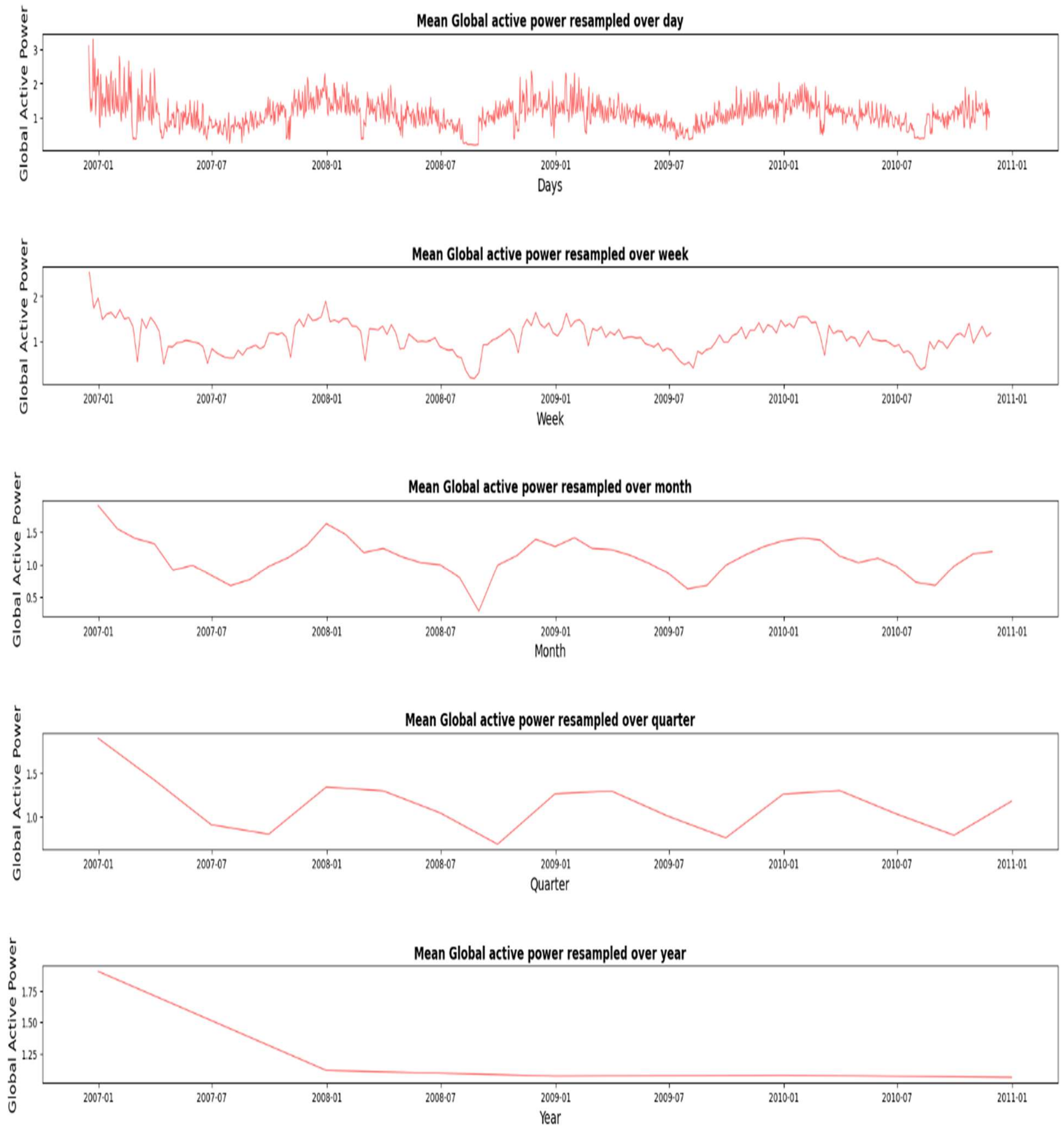


Figure 5: Subplots of Global Active Power Consumption

Fig 5 shows subplots of mean global active power consumption measured over a particular time, over day, over week, over month, over quarter and over year respectively. X-axis shows time period and the y-axis shows mean active power. The analysis of this graph shows varying fluctuations over time, which covers the period from January 2007 (December 2006, which is the starting data is also taken as mean) to November 2010. These line plots are plotted by taking the mean of Global active power and resampling with the day, week, month, quarter and year respectively.

E. Average over Time

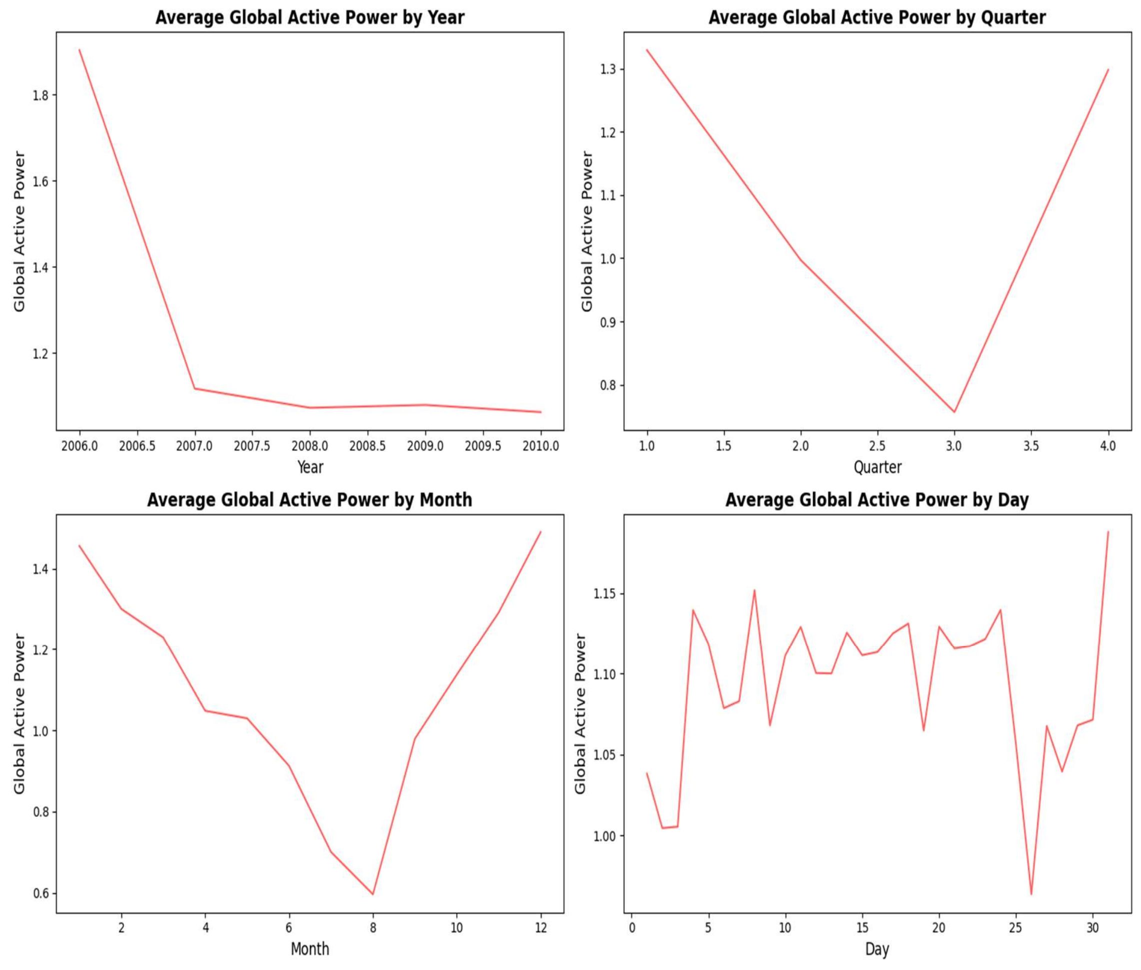


Figure 6: Average of Global Active Power Consumption over time

The four subplots showing line graphs in Fig 6 show the average global active power consumption varying over Year, Quarter, Month, and Day respectively where the mean of each time was taken. From the analysis of each line plot, it is clear that the active power varies over time. The end of the year 2006 shows a decline in the active power from 1.8 kW where in the first month of the year 2007, active power reaches 1.10 kW. The line graphs for the quarter and monthly shows a similar trend since the usage of energy was more needed with the start of the winter season where the reading for this house taken from this dataset requires the usage of heating appliances.

3.3 Ethical Requirements

After all the dataset contains numerical data, and readings taken from an electric meter from a house over four years by the creators, the dataset does not contain any personal information. So it meets Ethical requirements and is licensed under a public license Creative Commons 4.0 International.

4. METHODOLOGY

4.1 Overview

This section discusses on how the chosen dataset was used for initial stages before implementing machine learning models for predictions. Several stages done in the methodology are shown in Fig 7. As data contains a set of measurements, the first step, data preprocessing is necessary before the implementation of machine learning models because as data becomes huge, so it has missing values, outliers and so on. Once this stage is cleared, analysis of the data is important. This stage of analyzing the dataset is known as Exploratory Data Analysis. This stage is necessary to understand the behaviour, trends, patterns and so on of the data. Machine learning model requires two types of data, training and testing data. For this, based on how the data is to be predicted using machine learning, data splitting is done. Once the data is splitted, a model is implemented, where, from the training data which was split, is trained and then uses test data for prediction. A particular model chosen is evaluated and then visualized accordingly.

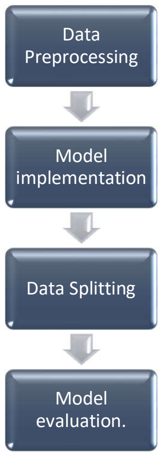


Figure 7: Flow diagram of various stages in time series analysis

This project was built on Google Colabaratory using Python programming language. Several import libraries were used for this project which include the following:

4.1.1 Python header files

i. Numpy: Numpy stands for numerical python, is a fundamental package that provides arrays, and mathematical functions for numerical operations and data manipulations.

ii. Pandas: For data manipulation and analysis, this package is used. With this package, data structures like DataFrame creation, with functionalities for reading, writing and processing data are possible

iii. Matplotlib: This package is useful for visualization purposes. This provides various types of plots such as line plots, histograms, bar charts, and many more.

iv. Seaborn: Seaborn is also used for visualization purposes, however useful for the creation of statistical graphs such as matrices (correlation, confusion), scatter plots, boxplots, and many more.

4.1.2 Header files for various models

i. statsmodels: The Statsmodels package is a statistical model for estimation purposes, and conducting statistical tests. For this project, a statistical model named ARIMA is used for prediction tasks. With this package, we can plot various graphs such as autocorrelation function (ACF) and partial autocorrelation (PACF) plots are possible to find the order of ARIMA models. A statistical test called Augmented Dickey-Fuller (ADF) test is conducted for testing the stationary of the time series.

ii. Xgboost: This package is used to import xgboost library to perform gradient boosting and to perform parallel tree boosting.

iii. Keras: This library imports Keras package, which is a high-level neural networks, enabling an interface for building and training deep learning models such as LSTM. With keras, we can import various libraries such as keras.models, keras.layers, keras.callbacks and so on. For this project, keras.models is used for creating Sequential neural network model. Keras.layers import is to add different types of layers to neural network model such as Dense, LSTM and many more.

iv. Sklearn: This is a library for performing Machine Learning Tasks. For this project, sklearn is imported to find mean absolute error(MAE) and mean squared error (MSE) for RMSE value.

4.2 Explanation of Various Stages in Methodology

4.2.1 Data Preprocessing

Data preprocessing (Tobias Geisler Mesevage, 2021) is an essential phase in data mining and analysis that involves transforming data into a machine-understandable language and turning it into an organized format. It is critical to create error-free, well-organized data for any machine learning model to perform its many responsibilities correctly. If raw data were taught, the outputs would be useless and incomprehensible. Raw data requires data cleaning since it is always noisy or contains mistakes, inconsistencies, and missing values.

Machine learning extracts features, also known as attributes, variables, or fields, from a dataset to characterize the data's qualities. These may or may not include structured data, therefore managing them is a critical effort. Various phases in data preparation include:

- Cleaning: Removes errors, inconsistencies and irrelevant information
- Integration: Data combined from multiple sources

- Reduction: Data dimensionality reduction, making important data in use.
- Transformation: Data conversion into a suitable format for further steps such as analysis, splitting, model implementation, evaluation and visualization.

4.2.2 Chosen models

For this project, a comparative analysis of three models is performed and for time series prediction, these models have found to show good results from various papers. The chosen models for this project are the following

a) ARIMA (Autoregressive Integrated Moving Average)

ARIMA (Hayes, 2024) is a famous statistical model for predicting trends in machine learning. This model employs regression analysis, a powerful statistical method for determining the strength of one variable (dependent variable) in relation to a set of other variables (independent variable), whereas linear regression is well-known for determining the line of best fit by establishing a relationship between two variables. To make ARIMA stationary, it is necessary to comprehend the data, pick appropriate orders, and validate the model using multiple methodologies. This model employs three components for time series, which are AR, I, and MA.

- AR or Autoregression refers to a model that shows a changing variable and captures linear relationship between current and past values. This regresses on lagged, or prior values.
- I or Integrated refers to differencing of raw time series observations to make it stationary. This ensures statistical properties and computes the differences between consecutive observations. Differencing involves subtracting the current value from the previous value to remove seasonality.
- MA or Moving average is regressed on the past forecast error. This component accounts for short-term dependencies between current and past values.

ARIMA model uses orders as parameters, they are p, d, and q. Each parameter have various functions and contain integral values:

  - p: This represents the number of lag observations in the model. This parameter also represents the order of AR components.
  - d: This represents the number of times the time series or the raw data is referenced, also known as order of differencing.
  - q: This order explains on the number of past forecast errors included in the MA model, also known as the order of MA component.

For implementing ARIMA model, a package called statsmodels is used. This package is used for conducting statistical tests, models, and statistical data exploration.

For this project, the ARIMA model is used for prediction purposes. Firstly, for all machine learning models, preprocessed data is chosen. We choose the Global_active_power column from this dataset, and for time series, since one variable or column is used, this is called as Univariate Time Series. We are required to check if the data is free from seasonality and is stationary, so graphs such as Auto Correlation (ACF) and Partial Auto Correlation (PACF).

Auto Correlation (ACF)

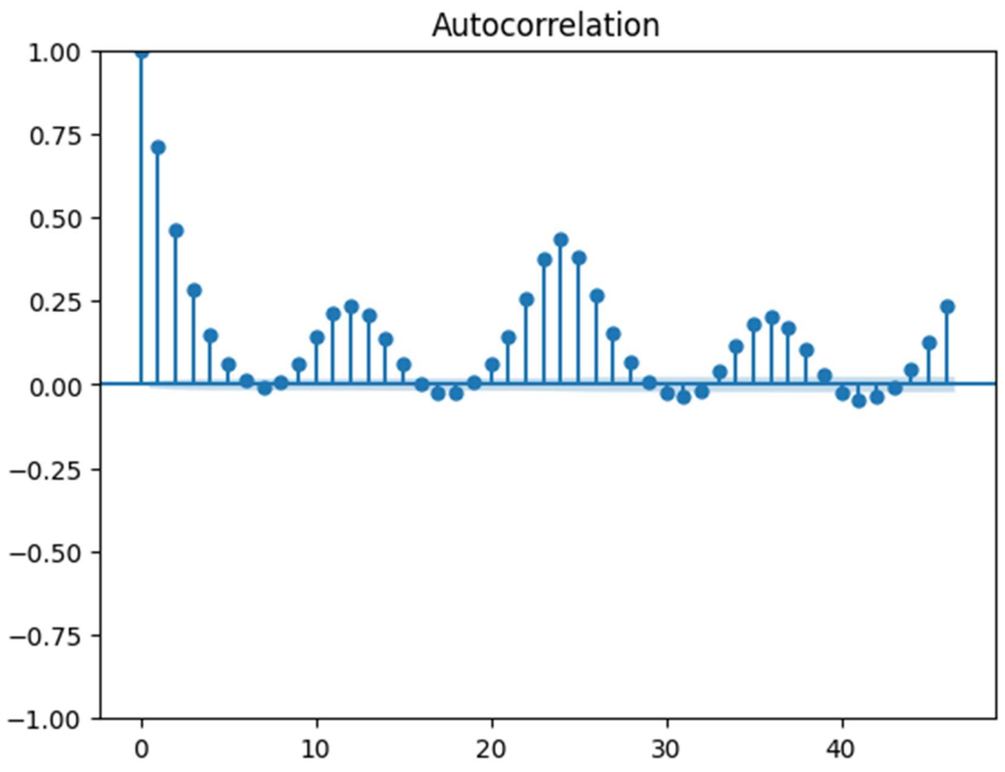


Figure 8: Auto Correlation Graph

Auto Correlation (ACF) is a method for calculating and identifying the correlation between two data at different locations in a time series. This aids in correlation analysis by determining lags, the number of intervals between two observations, and the ACF function, which assesses correlations for various lags. ACF aids in understanding how previous values impact present values in time series, as well as assessing whether a time series is random or stable by detecting seasonality or trend.

Fig plots an autocorrelation graph having x-axis, which represents the lags indicating the number of intervals between two observations, and y- y-axis shows autocorrelation coefficient ranging from 1.0 to -1.0 respectively. From this graph, there is a strong positive autocorrelation at lag 0 and becomes weaker from lag 1. This shows a positive correlation in time series, meaning that the values tend to be followed by higher values. From each ten lags, the ACF plot shows a fluctuation becoming stronger and weaker respectively.

Partial Auto Correlation (PACF)

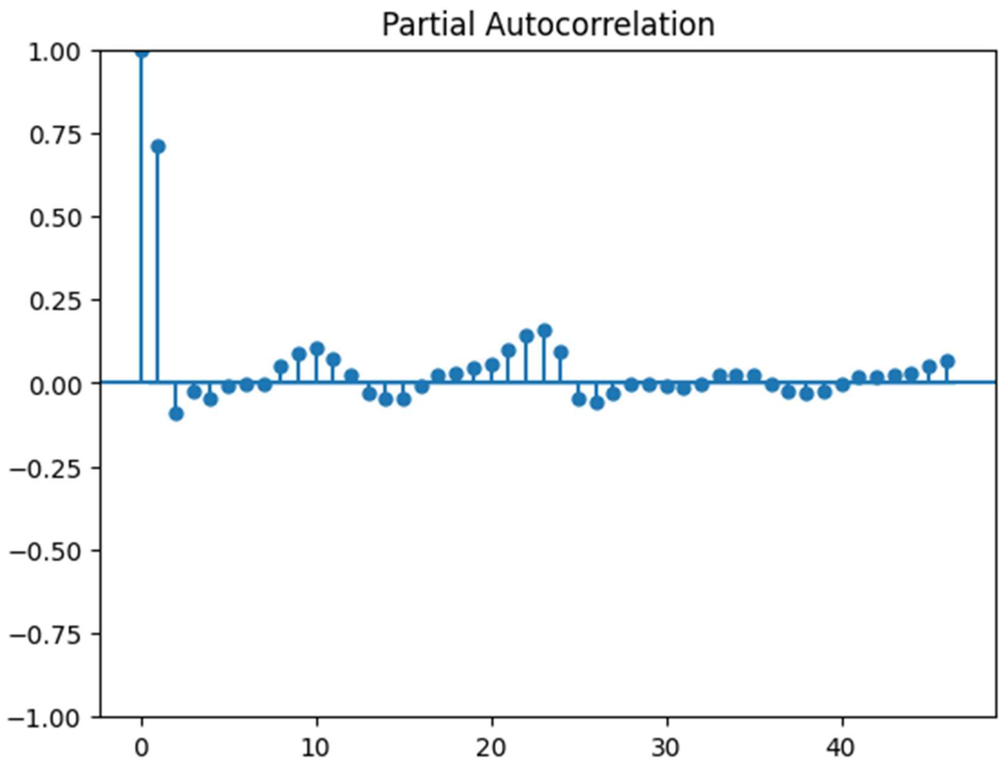


Figure 9: Partial Auto Correlation Graph

Partial AutoCorrelation (PACF) is similar to ACF but displays only a correlation between two observations by taking its immediate previous observations. PACF focuses on direct impact of past values on the current value in a time series.

Similar to ACF plot, x-axis shows lags and y-axis shows correlation between the original series and lagged series, after removing the effect of correlations at shorter lags. The correlation at lag 0 correlates to 1.00, which shows a perfect correlation at lag 0, however lags 1 and 2 shows short-term dependencies in the time series. But going longer, points appear close to zero, which shows there is no long-term dependence.

It is concluded from the ACF and PACF plots that time series has short-term dependence with fewer long-term dependence. The next step of this model is to perform a statistical test known as Augmented Dickey fuller test (G, 2023) to determine if the time series is stationary.

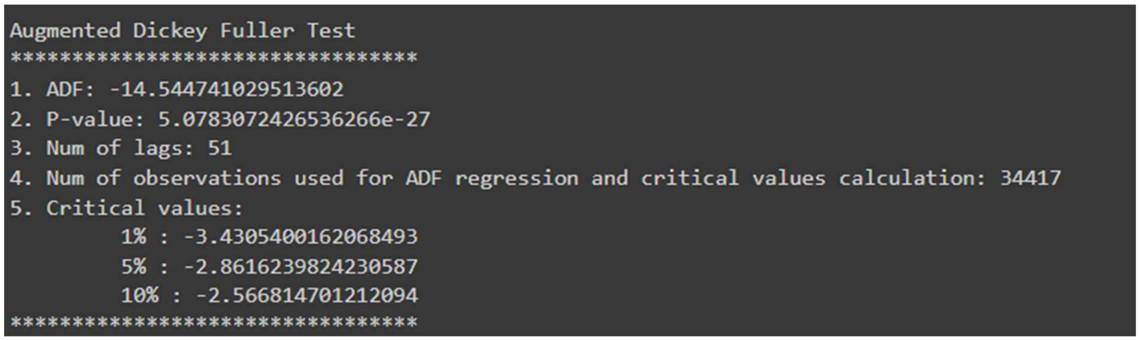
Augmented Dickey Fuller (ADF) Test.

This is a statistical test that is used in time series analysis to determine if the time series is stationary or not. For ARIMA models to work, it is essential to make the time series stationary. Stationary does not vary with statistical properties such as mean, variance, covariance and standard deviation. Stationary of time series is necessary as it makes easier for statistical models for prediction, this also helps to understand the data better and forecasting models. The ADF test is commonly used to determine nonseasonal differencing order in ARIMA modeling. The test statistic and p-values help infer whether a series is stationary or non-stationary. The ADF test involves hypothesis testing with null and alternate hypotheses. Null Hypothesis (H_0) executes the series is non-stationary, or series has a unit root and Alternate Hypothesis(Ha or H_1), which executes that the series is stationary. If Null Hypothesis (H_0) is less than 0.05, i.e, Critical value and p-value < 0.05, we reject Null Hypothesis, hence time series is stationary.

ADF test uses adfuller() function in statmodels.tsa.stattools package. The following outputs are generated:

- p-value: This represents the probability of observing a test statistic. With the p-value we can identify if time series is stationary or not.
- Value of test statistic: This tells how much data was used for the test. Having the sufficient number of observations is important for reliability of ADF test
- Number of lags considered: This indicates the number of lags included in ADF. Also accounts for autocorrelation, leading to more accurate test result
- Critical value cut-offs: These represent the thresholds for rejecting null hypothesis at different significance levels such as 1%, 5% and 10 %.

For this project, we have done this test and the results are the following:



```
Augmented Dickey Fuller Test
********************************
1. ADF: -14.544741029513602
2. P-value: 5.0783072426536266e-27
3. Num of lags: 51
4. Num of observations used for ADF regression and critical values calculation: 34417
5. Critical values:
        1% : -3.4305400162068493
        5% : -2.8616239824230587
        10% : -2.566814701212094
********************************
```

Figure 10: Augmented Dickey-Fuller Test

Explaination are the following:

- ADF value shows -14.5447, which is a key value. Since it is a negative value, this indicates stronger evidence against non-stationary time series.
- P-value: This represents the probability of observing a test statistic. P value which we received is a small value that is less than 0.05. So this proves alternate hypothesis, or the time series is stationary.
- No of lags: This indicates number of lagged terms. We got 51, this leads to accurate result.
- No of observations used for ADF regression: This shows how much data was used for the test. For this test, a total of 34417 values are used.

- Critical values: These represent thresholds for rejecting null hypothesis for various levels such as 1%, 5%, and 10%. The values were negative for this proves a strong reject for null hypothesis.

Once we confirm that the data is stationary, the preprocessed data chosen is then split into two, training and test data, 90 percent of the data is used for training and 10 percent is used for test. With this, ARIMA model training is initialized, model is performed fitting. Once fitting is done, predictions are done for the test data.

b) XGBoost

XGBoost or eXtreme Gradient Boosting (Simplilearn, 2023) is a gradient boosting tool that is meant to be very efficient, portable, and versatile. This is a supervised machine learning method that performs numerous tasks such as classification and regression, and it falls under the category of ensemble learning. XGBoost is an implementation of gradient-boosting decision trees. This provides parallel tree boosting that solves issues accurately and rapidly, and is intended to be fast, easy to use, and perform well on huge datasets. In addition, this model trains numerous decision trees and then aggregates the findings, giving it an edge over other algorithms in terms of learning speed. This model is used to train and evaluate data on huge datasets for prediction purposes.

Features of XGBoost

- Offers regularization, that allows to control overfitting.
- Handle sparse datasets such as missing values and data processing steps.
- XGBoost has block structre for parallel learning which scales up to multicore machines or clusters. This helps reduce memory usage when training models.
- Highly efficient and scalable interpretation of boosting algorithm.
- XGBoost requires non-continuous memory access, hence, optimal usage of hardware is key for this model, so does the term Cache awareness. This allocates internal buffers in each thread.
- XGBoost employs out-of-core computation, which optimises available disc space and maximises use when manipulating large datasets.

XGBoost is also known as ensemble learning since it integrates various models to improve overall performance, predictability, and accuracy in machine learning. Ensemble learning, tagging, boosting, and stacking are some of the essential insights for improving performance and model quality.

Bagging employs a voting method for classification and averaging for regression, reducing total variance by taking the average performance of several estimates. Bagging helps to minimize variance by generating many decision trees in parallel. Furthermore, this addresses the issue of overfitting.

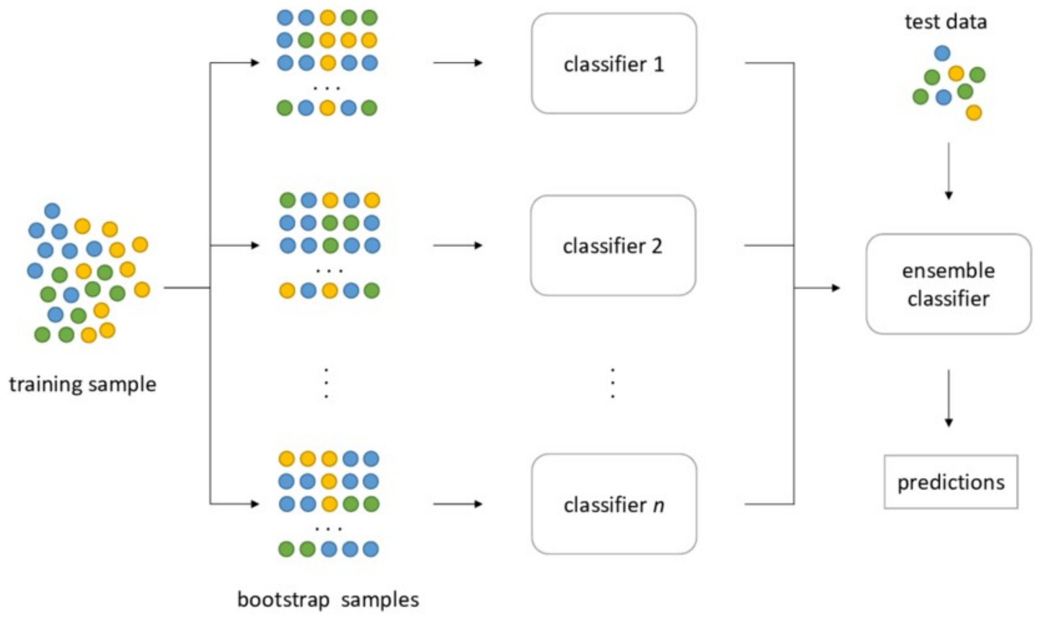


Figure 11:Bagging using Sequential Bootstrap Techniques
Source(https://hudsonthames.org/bagging-in-financial-machine-learning-sequential-bootstrapping-python/)

Boosting approach matches poor learners and converts them into strong learners. The trees are created successively, with each consecutive tree aiming to minimize the mistakes of the preceding tree. Each tree learns from each stem (or predecessor) and corrects it with residual mistakes. As previously stated, weak learners provide critical information for prediction, allowing boosting strategies to deliver robust outcomes.

For this project, we have used XGBoost model for prediction of power consumption of a household. Firstly, preprocessed data was chosen, later on cyclical features such as hour, month, year, day of week, month, year and so on are implemented as features using sine and cosine transformations. Lagging features for certain variables were created for temporal patterns. Later the data was split into train and test. We have chosen 90 percent as train and 10 percent as test, and creates feature matrices X_train, X_test, Y_train, and Y_test for train and test sets respectively. With the chosen sets, XGBoost model fitting is done. For fitting to happen, train data set is chosen. Later, test set is chosen for prediction.

c) <u>LSTM</u>

LSTM (GEEKSFORGEEKS, 2024) stands for Long Short-Term Memory. LSTM is an improved version of recurrent neural network (RNN), which is a deep learning model that is trained to process and convert a sequential data input into specific sequential data output. This deep learning model is useful for prediction tasks and is useful for long-term dependancies in sequential data, which makes it well-suited for various tasks such as time series forecasting, speech recognition, and so on.

LSTM address the problem of traditional RNN, which has a single hidden state, by introducing memory cell that can hold the information for an extended period. These cells allow LSTM to collect or discard the information, also known as vanishing gradient problem. Memory cell consists of three gates, which decide what information to add, remove and generate output. These gates are input gate, forget gate and output gate. Input gate controls what informations are added to the memory cell, forget gate controls what information is removed from the memory cell and, output gate is to control the information from the memory cell.

Architecture and Working of LSTM

The figure below shows the architecture of LSTM model, this contains four neural networks and different memory blocks called cells. Following cells where the information is retained is explained below:

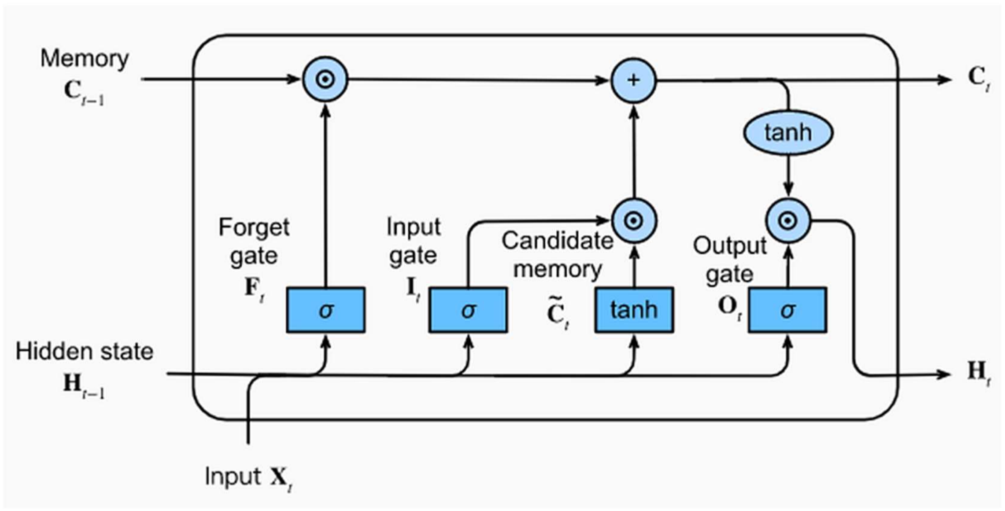


Figure 12: LSTM Architecture (Source: https://d2l.ai/chapter_recurrent-modern/lstm.html)

Forget Gate

This type of memory cell removes any information that is no longer useful in the cell state. Two inputs are fed to the gate and multiplied with weight matrices followed by addition of bias. For getting a binary output, this result is passed through an activation function. The output becomes 0 if the piece of information is forgotten and 1, if information is retained for future use respectively.

Input Gate and Candidate Memory

These gates has the function of inserting information that is received from the cell state. The information is regulated using sigmoid function, then it filter the values as how it was done in foreget gate with inputs. Candidate memory is responsible for generating candidate vector and uses tangent function. This ensures that all values of candidate vector will be from -1 to 1, where it normalizes the information which will be added to the cell state. The input gate is responsible for the generation of a selector vector which will be multiplied with candidate vector element by element. This result is then added to the cell state vector.

Output Gate

This gate determines the value of hidden state from the current cell state by extracting useful information. This information is generated in output gate. This gate works with the function of multiplication between selector vector and candidate vector. With the hyperbolic tangent function, where after multiplication, vector values of the cell state are normalized from -1 to 1 getting a hidden state. This helps to control stability. This multiplied vector from the output gate is then sent as output for next cell, also said as an input to the next cell.

Compared with traditional RNN, the LSTM overcomes the problem of vanishing gradient problem by dealing with recall or forgetting from each cell under the gating mechanism explained above. The problem of vanishing gradient occurs for RNNs after several long sequences of model training.

For this project, we have chosen a univariate feature from the dataset named Global_active_power. Firstly, this column is converted to float datatype and is then normalized to -1 to 1. This data is then transformed into MinMax scaler by transforming the value. For the working of LSTM, this step is necessary as mentioned about the gates or memory cells. Once this step is done, this is converted into train and test data sets, such as X_train, Y_train, X_test, and Y_test respectively.

4.2.3 Data Splitting

For any machine learning models, data splitting is important. From data splitting, we ensure that chosen model is trained, and then tested on a final, independent variable. Since we chose the dataset of electric power consumption, we can split the data into two, 90 percent of the data to be training set and the remaining 10 percent as test set. Based on the model, the train and test data are converted into X_train, X_test, Y_train and Y_test respectively. These train and test sets are used for XGBoost and LSTM model, ARIMA does the work with train and test variables.

From the training set, we can learn by understanding patterns and relationships within data, training data is representative and unbiased. The test set, however, checks for final model's accuracy on new data and provides an unbiased estimate of model performance.

4.2.4 Model implementation & evaluation

For power consumption prediction, we have implemented three models. The results of the three models are the following:

a) ARIMA model

After fitting using the train data, a model summary is generated. The summary shows that the ARIMA model chooses the order (4,1,4) with the parameters p, d, and q having the values 4,1,4 accordingly. The findings indicate various values for AIC, BIC, and HQIC, which assure model complexity, fit, and so forth. Using these results, the RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) of the model are determined with the skleran.metrices package.

b) XGBoost

Once the stage of creating lagging features, data splitting to train and test are done, it is then prepared for model fitting. For model fitting, XGBoost Regressor function is used with train data and prediction is done with the test data. Model performance is evaluated using RMSE and MAE.

c) LSTM

This model uses a single layer for performing the prediction of power consumption. For this model, the model is trained with train data sets. For this task, 50 iterations (or epochs) were initialized. Each epoch involves model's weights based on the training data. As 50 iterations were conducted, the loss (or error) is computed during this training, this describes how well the model is performing or fitting the training data. If the loss value is lower, it is desirable. Just as loss, validation loss is performed on validation dataset, this assess how well the model is trained on an unseen data. This helps to track overfitting, which happens when model performs well on training data but poorly on unseen data.

5. RESULTS

The table below shows the comparative outcome of each model by performing model performance tests.

Model	**RMSE**	**MAE**
ARIMA	0.9566022139578871	0.7538292914506157
XGBoost	0.5819740456286109	0.4372154366987499
LSTM	0.4765418555643306	0.31940782820265057

Table 1: Model Performance Values

Mean Absolute Error (ScienceDirect, 2022) is used for assess errors between paired observations such as actual value vs predicted value. MAE is calculated as the average of the absolute error values.

Formula for calculating MAE is:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} | x_i - x |$$

Root Mean Squared Error (SAP, 2023) is used to calculate the mean difference between values predicted by a model and actual values. This returns a result indicating how well a certain model can perform and forecast the target value.

Formula on calculating RMSE is:

$$RMSE = \sqrt{\frac{SSE_W}{W}} = \sqrt{\frac{1}{W}\sum_{i=1}^{N} w_i u_i^2}$$

Visualization on Each Model

a) ARIMA

From the line plot of this model, the blue line shows the actual values, and the red line shows the predicted values. The date, which runs from July 2010 to December 2010, is shown by the x-axis. Global active power is represented by the y-axis.
The general trend in the data appears to have been caught by the forecasting ARIMA model. The actual and expected figures may not always match up, especially between September and November of 2010. Although the accuracy of the model as a whole cannot be determined from this graph, the visual comparison indicates that the ARIMA model forecasts global active power reasonably well.

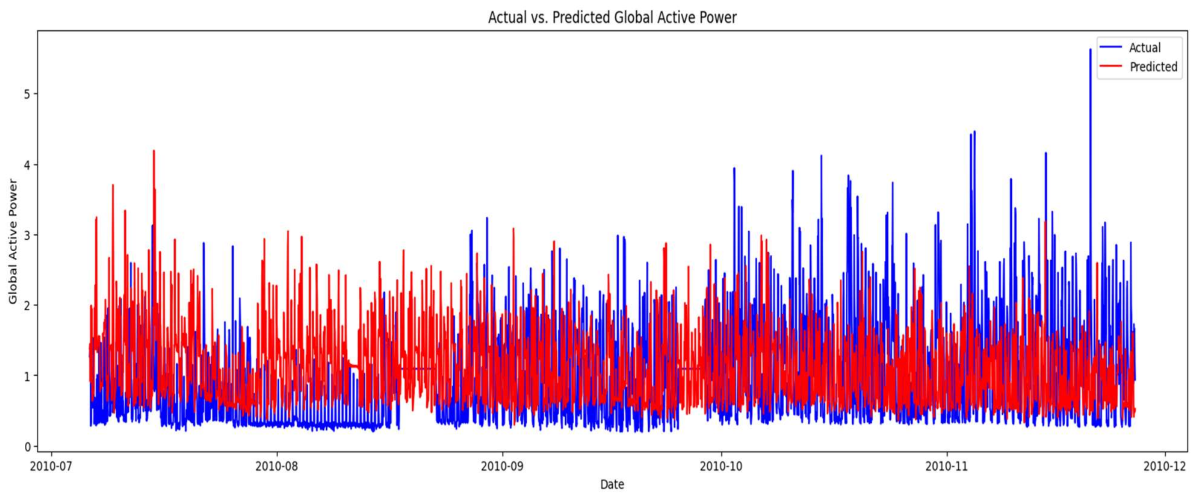


Figure 13: Actual vs Predicted plot for ARIMA model

b) <u>XGBoost</u>

This line plot shows the actual and predicted line plot of this model using the test data. The blue line and red line shows the actual and predicted values of the data plotted respectively. The data chosen and plotted is from July 2010 to November 2010, which is taken as x-axis. However the axis shows upto December 2010. Global active power is plotted on y-axis, denoting the active power in kW (kilowatt). This model is able to plot a good forecast, however, there is a lag on capturing the overall trend between September and November 2010.

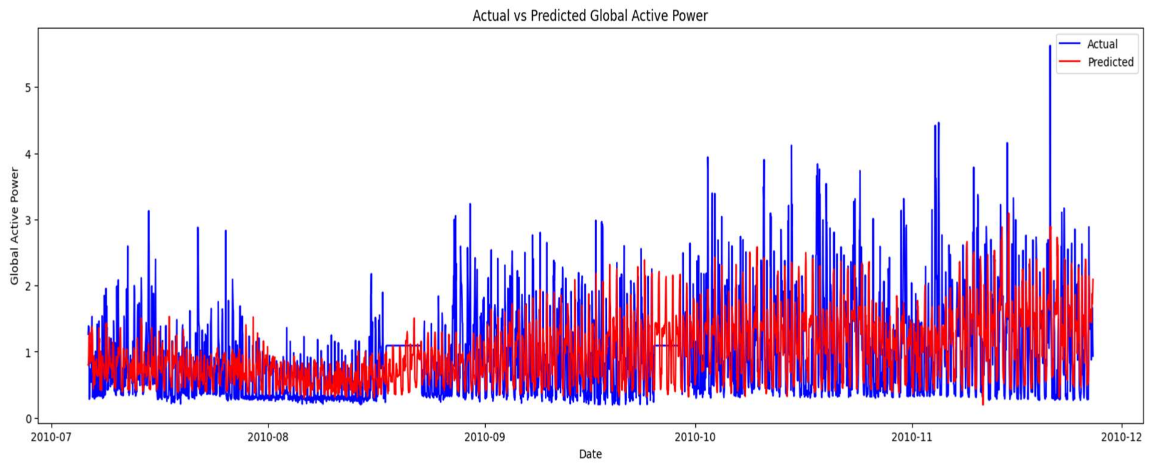


Figure 14: Actual vs Predicted plot for XGBoost model

c) <u>LSTM</u>

The plot below plots a model loss graph on training data and test data. Fifty iterations were conducted and is clear from the plot that test loss has lower loss value, which indicates that the model is performing well on test set. After 50 epochs, the loss value on the train data comes to 0.0085 and validation loss to 0.0055 respectively.

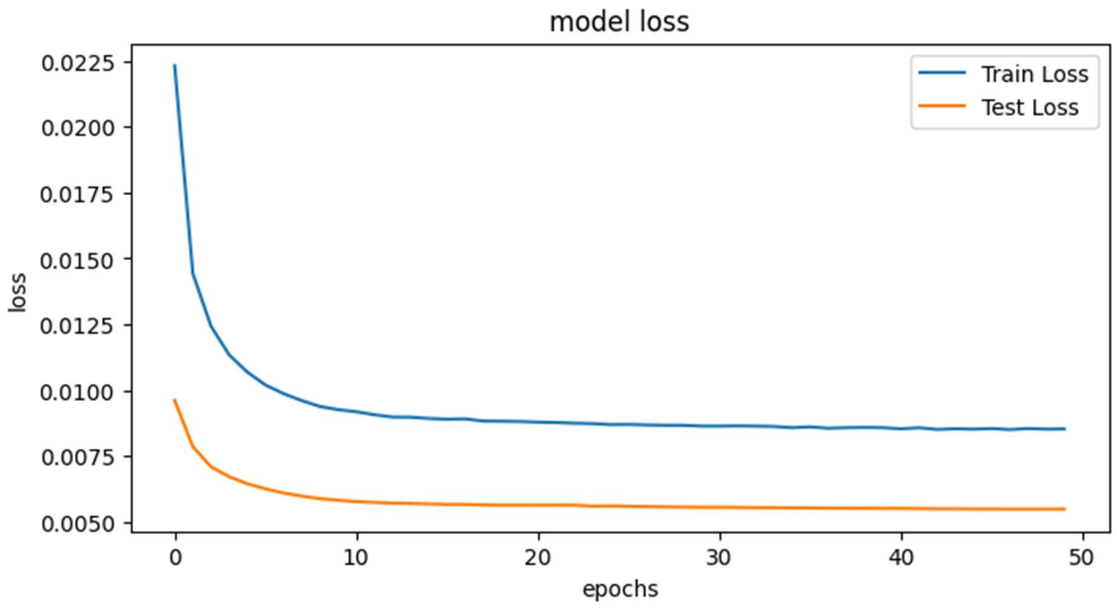


Figure 15: Model loss plot for LSTM Model

The graph plots the actual vs predicted global active power of an household using the test data that was predicted using the LSTM model. The line plot represents actual vs predicted as blue and red respectively. Similar to the other models, this model uses the same data from July to November 2010, but is represented as time steps in x-axis. Global_active_power is represented in y-axis. The prediction is able to capture along with the overall trend and concludes that LSTM provides a reasonable forecast.

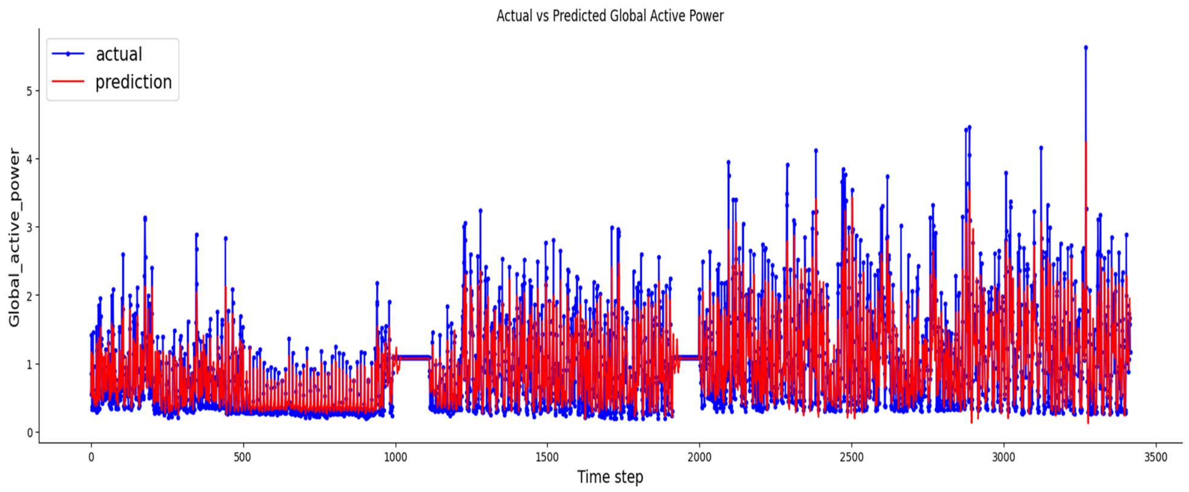


Figure 16: Acutal vs Predicted plot for LSTM Model

6. ANALYSIS AND DISCUSSION

Comparison of the model

From the results, after performing the model tests, it is concluded that LSTM model has the lowest RMSE and MAE value, which means the model exhibed the best performance and is the best for prediction and provides more accurate results for this project. RMSE value was around 0.48 and MAE of 0.32 respectively. LSTM is best for sequential data, enabling to capture long-term dependencies. However, the other models, namely XGBoost and ARIMA lacked in performance for long-term, but XGBoost was able to outperform ARIMA with its feature of ensemble learning method. ARIMA model might not have captured the sequential nature of the data, which led to higher errors.

Reason for Performance

The capacity of the LSTM model to identify long-term dependencies in the data is probably its strongest suit. In contrast to seasonality and historical values, LSTM is able to capture more intricate linkages that impact changes in active power. Although XGBoost outperformed ARIMA, it may not have fully grasped the sequential character of the data, which resulted in greater error rates.

Comparison with the Literature Review

Some publications reviewed in the literature review refer to short-term and long-term dependency. For short-term dependencies, the ARIMA model is regarded the best model, whereas LSTM and XGBoost are considered equally preferable for long-term relationships. However, these differ from data to data.

Limitations to the Work

There may be some limits to the work, such as an LSTM model being susceptible to overfitting if trained on a limited dataset and if data was not preprocessed well. When compared to the ARIMA model, interpreting LSTM work might be complicated.

Applications based on the best model

The findings directly address the project's goal of establishing an accurate model for estimating global active power. The LSTM's exceptional performance makes it an excellent choice for practical applications. Its capacity to recognise complicated patterns can help utilities optimise power generation and grid management.

7. CONCLUSION

Machine learning plays a crucial role in prediction tasks for time series. It allows to capture complexities such as relationships within time series data, allowing for more accurate forecasts and handling large datasets effectively, with insights. As several challenges such as overfitting, wrong predictions and data requirements, machine learning is a powerful tool for time series. From this project, power consumption of an household is predicted with three models and it was concluded that LSTM is considered as the best model, since the data we choose was longer, it proved that it is best for long term predictions than other models. However, XGBoost came closer to the mark, ARIMA showed poor results, concluding that ARIMA is best for short term predictions. The results facilitate better decision-making and underscore the importance of advanced modelling techniques and extends to various industries such as healthcare, finance, energy, and domains showing the importance of time series.

For the future work, other factors such as weather data, economic indicators can be linked with this data to show on real-world scenario and use it to predict. Fine-tuning can be implemented so that the model's performance can be improved. Also, forecasting for several days, months or years can be done to understand how the power consumption will be in the future with the past values.

8. REFERENCES

Agency, International Energy, 2024. *Russia's War on Ukraine.* [Online]
Available at: https://www.iea.org/topics/russias-war-on-ukraine
[Accessed 13 April 2024].

Bennett, A., 2023. *Why is energy consumption increasing?.* [Online]
Available at: https://www.internetgeography.net/topics/why-is-energy-consumption-increasing/
[Accessed 13 April 2024].

Chirag Deb, F. Z. J. Y. S. E. L. K. W. S., 2017. Renewable and Sustainable Energy Reviews. *A review on time series forecasting techniques for building energy,* Volume 74, p. 902–924.

El Houda, B. N. a. L. L. a. A. M., 2022. International Conference on Pattern Analysis and Intelligent Systems. *Time Series Analysis of Household Electric Consumption with XGBoost Model,* pp. 1-6.

GEEKSFORGEEKS, 2024. *Deep Learning | Introduction to Long Short Term Memory.* [Online]
Available at: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/amp/
[Accessed 25 April 2024].

G, V. K., 2023. *Statistical Tests to Check Stationarity in Time Series.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2021/06/statistical-tests-to-check-stationarity-in-time-series-part-1/
[Accessed 24 April 2024].

Hai-xiang Zhao, F. M., 2012. A review on the prediction of building energy consumption. *Renewable and Sustainable Energy Reviews,* 16(6), pp. 3586-3592.

Hayes, A., 2024. *Autoregressive Integrated Moving Average (ARIMA) Prediction Model.* [Online]
Available at: https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp#:~:text=An%20autoregressive%20integrated%20moving%20average%2C%20or%20ARIMA%2C%20is%20a%20statistical,values%20based%20on%20past%20values.
[Accessed 23 April 2024].

Lorusso, D. M., 2023. *The Effects of High Energy Prices on UK Consumers.* [Online]
Available at: https://from.ncl.ac.uk/the-effects-of-high-energy-prices-on-uk-consumers#:~:text=In%20this%20regard%2C%20according%20to,electricity%20bills%20as%20a%20cause.
[Accessed 13 April 2024].

M M Sachin, M. P. B. a. A. S. P., 2020. *Analysis of energy consumption using RNN-LSTM and ARIMA Model.* [Online]
Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1716/1/012048
[Accessed 19 April 2024].

MUSAED ALHUSSEIN, K. A. ,. (. M. I. S. I. H., 2020. *Hybrid CNN-LSTM Model for Short-Term Individual Household Forecasting,* Volume 8, pp. 180544-180557.

Pasapitch Chujai, N. K. a. K. K., 2013. Proceedings of the International MultiConference of Engineers and Computer Scientists 2013. *Time Series Analysis of Household Electric Consumption with ARIMA and ARMA Models,* Volume 1, pp. 295-300.

Ralston, J., 2024. *Two years of Russia's war on Ukraine: the gas crisis, price rises and energy security.* [Online]
Available at: https://eciu.net/insights/2024/two-years-of-russias-war-on-ukraine-the-gas-crisis-price-rises-and-energy-security
[Accessed 13 April 2024].

Ray, M., 2024. *Russia-Ukraine War.* [Online]
Available at: https://www.britannica.com/event/2022-Russian-invasion-of-Ukraine
[Accessed 13 April 2024].

Reza Ghanbari, K. B., 2021. 26th International Computer Conference. *Multivariate Time-Series Prediction Using LSTM,* pp. 1-5.

SAP, 2023. *Root Mean Squared Error (RMSE).* [Online]
Available at:
https://help.sap.com/docs/SAP_PREDICTIVE_ANALYTICS/41d1a6d4e7574e32b815f1cc87c00f42/5e5198fd4afe4ae5b48fefe0d3161810.html
[Accessed 30 April 2024].

Sarker, I. H., 2021. *Machine Learning: Algorithms, Real-World Applications and Research Direction.* [Online]
Available at: https://doi.org/10.1007/s42979-021-00592-x
[Accessed 13 April 2024].

ScienceDirect, 2022. *Mean Absolute Error.* [Online]
Available at: https://www.sciencedirect.com/topics/engineering/mean-absolute-error
[Accessed 30 April 2024]

Simplilearn, 2023. *What is XGBoost? An Introduction to XGBoost Algorithm in Machine Learning.* [Online]
Available at: https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#what_is_xgboost_algorithm
[Accessed 20 April 2024].

Third Space Learning, 2024. *Box Plot.* [Online]
Available at: https://thirdspacelearning.com/gcse-maths/statistics/box-plot/
[Accessed 15 February 2024].

Tobias Geisler Mesevage, 2021. *What Is Data Preprocessing & What Are The Steps Involved?.* [Online]
Available at: https://monkeylearn.com/blog/data-preprocessing/
[Accessed 20 April 2024].

UCI Machine Learning Repository, 2012. *Individual Household Electric Power Consumption.* [Online]
Available at:
https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption
[Accessed 24 January 2024].

UCI Machine Learning Repository, 2023. *UCI Machine Learning Repository About.* [Online]
Available at: https://archive.ics.uci.edu/about
[Accessed 24 January 2024].

APPENDIX

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

pip install pmdarima

from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller

import xgboost as xgb
from xgboost import plot_importance

# Import necessary functions from keras
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *

# Import MinMaxScaler from sklearn
from sklearn.preprocessing import MinMaxScaler

# Import mean squared error and mean absolute error from sklearn
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# Import early stopping from keras callbacks
from keras.callbacks import EarlyStopping
from keras.utils import set_random_seed, plot_model


from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt


def data_preprocessing(dataset):


    print("------------ DATA PREPROCESSING--------------")


    electric_power_consumption = pd.read_csv(dataset, sep=';',
                                             parse_dates={'Date_Time':                ['Date',                 'Time']},
infer_datetime_format=True,
                                             low_memory=False, na_values=['nan', '?'])
    # Print info about the dataframe
    print(electric_power_consumption.info())


    print("............................................")


    print("-------- First 4 rows of the data -----------")


    # Display the first few rows of the dataframe
    print(electric_power_consumption.head())


    print("............................................")


    print("-------------- Missing values  -------------")


    # Check for missing values
    print(electric_power_consumption.isnull().sum())


    # Fill missing values with column means
    electric_power_consumption.fillna(
        electric_power_consumption.mean(), inplace=True)
    # Set the index to the datetime column
    electric_power_consumption.set_index('Date_Time', inplace=True)

    # Resample the data into hourly intervals and aggregate using mean
    newdf = electric_power_consumption.resample('H').mean()

    print("...........................................")

    print("----------- No missing values  ------------")

    # Check for missing values after resampling
    print(newdf.isnull().sum())

    print("...........................................")

    newdf.describe()

    return newdf

def ed_analysis(dataset):

  newdf = dataset
  # Create new columns for year, quarter, month, and day
  newdf['year'] = newdf.index.year
  newdf['quarter'] = newdf.index.quarter
  newdf['month'] = newdf.index.month
  newdf['day'] = newdf.index.day
  newdf['weekday'] = (newdf.index.weekday < 5).astype(int)

  print("Boxplot for the feature 'Global Active Power")

  print("*********************************")
```

```
# Plot boxplots for each feature
data1 = newdf.loc[:, ['Global_active_power']]
plt.figure(figsize=(10, 7))
data1.boxplot()
plt.title('Boxplots for Global Active Power (in kW)')
#plt.xticks(rotation=45)
plt.show()

print("Active Power Consumption for a Particular Time Frame")

dataset_one_month = newdf[(newdf.index.year == 2010) &
                          (newdf.index.month == 11)]

plt.figure(figsize=(14, 6))
plt.plot(dataset_one_month.index,
         dataset_one_month['Global_active_power'], color='#ff6361')
plt.ylabel('Global Active Power (kW)', fontsize=14)
plt.xlabel('Date', fontsize=14)
plt.title('Active Power Consumption for a Particular Time Frame',
          fontsize=16, fontweight='bold')
plt.tight_layout()
plt.grid(True)
sns.despine(bottom=True, left=True)
plt.show()
print("**********************************")

print("Bar plots for Yearly and Quaterly Global Active Power")

# Create a figure with 2 subplots
plt.figure(figsize=(14, 5))

# Plot the first subplot showing the bar graph of yearly global active power
```

```
plt.subplot(1, 2, 1)
# Adjust the subplot's width
plt.subplots_adjust(wspace=0.2)
# Create the bar graph using Seaborn's barplot function
sns.barplot(x="year", y="Global_active_power", data=newdf, color='#ff6361')
# Label the x-axis
plt.xlabel('Year', fontsize=12)
# Add a title to the plot
plt.title('Bar plot of Yearly Global Active Power', fontsize=14)
# Remove the top and right spines of the plot
sns.despine(left=True, bottom=True)
# Add a tight layout to the plot
plt.tight_layout()

# Plot the second subplot showing the bar graph of quarterly global active power
plt.subplot(1, 2, 2)
# Create the bar graph using Seaborn's barplot function
sns.barplot(x="quarter", y="Global_active_power", data=newdf, color='#ff6361')
# Label the x-axis
plt.xlabel('Quarter', fontsize=12)
# Add a title to the plot
plt.title('Bar plot of Quarterly Global Active Power', fontsize=14)
# Remove the top and right spines of the plot
sns.despine(left=True, bottom=True)
# Add a tight layout to the plot
plt.tight_layout()

print("*********************************")

print("Subplots of Global Active Power Consumption (in kW) over Time")

# Create a figure with specified size
fig = plt.figure(figsize=(25, 15))
```

```
# Adjust the subplot spacing
fig.subplots_adjust(hspace=1)

# Create first subplot
ax1 = fig.add_subplot(5, 1, 1)
# Plot the resampled mean of Global_active_power over day with different color
ax1.plot(newdf['Global_active_power'].resample(
   'D').mean(), linewidth=1, color='#ff6361')
# Set the title for the subplot
ax1.set_title('Mean Global active power resampled over day',
              fontsize=14, fontweight='bold')
# Set major tick parameters for the subplot
ax1.tick_params(axis='both', which='major')
# Set x-axis label
ax1.set_xlabel('Days', fontsize=14)
# Set y-axis label
ax1.set_ylabel('Global Active Power', fontsize=14)

# Create second subplot
ax2 = fig.add_subplot(5, 1, 2, sharex=ax1)
# Plot the resampled mean of Global_active_power over week with different color
ax2.plot(newdf['Global_active_power'].resample(
   'W').mean(), linewidth=1, color='#ff6361')
# Set the title for the subplot
ax2.set_title('Mean Global active power resampled over week',
              fontsize=14, fontweight='bold')
# Set major tick parameters for the subplot
ax2.tick_params(axis='both', which='major')
# Set x-axis label
ax2.set_xlabel('Week', fontsize=14)
# Set y-axis label
ax2.set_ylabel('Global Active Power', fontsize=14)
```

```
# Create third subplot
ax3 = fig.add_subplot(5, 1, 3, sharex=ax1)
# Plots the resampled mean of Global_active_power over month with different color
ax3.plot(newdf['Global_active_power'].resample(
    'M').mean(), linewidth=1, color='#ff6361')
# Set the title for the subplot
ax3.set_title('Mean Global active power resampled over month',
              fontsize=14, fontweight='bold')
# Set major tick parameters for the subplot
ax3.tick_params(axis='both', which='major')
# Set x-axis label
ax3.set_xlabel('Month', fontsize=14)
# Set y-axis label
ax3.set_ylabel('Global Active Power', fontsize=14)

# Create fourth subplot
ax4 = fig.add_subplot(5, 1, 4, sharex=ax1)
# Plots the resampled mean of Global_active_power over quarter with different color
ax4.plot(newdf['Global_active_power'].resample(
    'Q').mean(), linewidth=1, color='#ff6361')
# Set the title for the subplot
ax4.set_title('Mean Global active power resampled over quarter',
              fontsize=14, fontweight='bold')
# Set major tick parameters for the subplot
ax4.tick_params(axis='both', which='major')
# Set x-axis label
ax4.set_xlabel('Quarter', fontsize=14)
# Set y-axis label
ax4.set_ylabel('Global Active Power', fontsize=14)

# Create a fifth subplot
ax5 = fig.add_subplot(5, 1, 5, sharex=ax1)
# Plots the resampled mean of Global_active_power over year with different color
```

```
ax5.plot(newdf['Global_active_power'].resample(
    'A').mean(), linewidth=1, color='#ff6361')
# Set the title for the subplot
ax5.set_title('Mean Global active power resampled over year',
              fontsize=14, fontweight='bold')
# Set major tick parameters for the subplot
ax5.tick_params(axis='both', which='major')
# Set x-axis label
ax5.set_xlabel('Year', fontsize=14)
# Set y-axis label
ax5.set_ylabel('Global Active Power', fontsize=14)

plt.show()

'''
'''


print("********************************")
print("Subplots of Average Global Active Power Consumption (in kW) over Time")

# Create a figure with 2 rows and 2 columns and set its size to 14x8
plt.figure(figsize=(15, 10))

# First subplot in the first row, first column
plt.subplot(2, 2, 1)
# Group data by year and take the mean of the 'Global_active_power' column
grouped_by_year = newdf.groupby(
    newdf.index.year).Global_active_power.agg('mean')
# Plot the mean of 'Global_active_power' by year with purple color
grouped_by_year.plot(color='#ff6361')
# Set the x label to be empty
plt.xlabel('Year', fontsize = 12)
plt.ylabel('Global Active Power', fontsize = 12)
```

```
  # Set the title to 'Average Global Active Power by Year' with font size 12 and font weight
'bold'
  plt.title('Average Global Active Power by Year',
           fontsize=14, fontweight='bold')


  # Second subplot in the first row, second column
  plt.subplot(2, 2, 2)
  # Group data by quarter and take the mean of the 'Global_active_power' column
  grouped_by_quarter = newdf.groupby(
     newdf.index.quarter).Global_active_power.agg('mean')
  # Plot the mean of 'Global_active_power' by quarter with purple color
  grouped_by_quarter.plot(color='#ff6361')
  # Set the x label to be empty
  plt.xlabel('Quarter', fontsize = 12)
  plt.ylabel('Global Active Power', fontsize = 12)
  # Set the title to 'Average Global Active Power by Quarter' with font size 12 and font weight
'bold'
  plt.title('Average Global Active Power by Quarter',
           fontsize=14, fontweight='bold')


  # Third subplot in the second row, first column
  plt.subplot(2, 2, 3)
  # Group data by month and take the mean of the 'Global_active_power' column
  grouped_by_month = newdf.groupby(
     newdf.index.month).Global_active_power.agg('mean')
  # Plot the mean of 'Global_active_power' by month with purple color
  grouped_by_month.plot(color='#ff6361')
  # Set the x label to be empty
  plt.xlabel('Month', fontsize = 12)
  plt.ylabel('Global Active Power', fontsize = 12)
  # Set the title to 'Average Global Active Power by Month' with font size 12 and font weight
'bold'
  plt.title('Average Global Active Power by Month',
           fontsize=14, fontweight='bold')
  # Fourth subplot in the second row, second column
  plt.subplot(2, 2, 4)
  # Group data by day and take the mean of the 'Global_active_power' column
  grouped_by_day = newdf.groupby(
    newdf.index.day).Global_active_power.agg('mean')
  # Plot the mean of 'Global_active_power' by day with purple color
  grouped_by_day.plot(color='#ff6361')
  # Set the x label to be empty
  plt.xlabel('Day', fontsize = 12)
  plt.ylabel('Global Active Power', fontsize = 12)
  # Set the title to 'Average Global Active Power by Day' with font size 12 and font weight 'bold'
  plt.title('Average Global Active Power by Day',
          fontsize=14, fontweight='bold')

  # Use tight_layout to adjust the subplots so that they fit into the figure area
  plt.tight_layout()

  # Show the plot
  plt.show()

def arima_initialize(data):


  newdf = data
  data1 = newdf.loc[:, ['Global_active_power']]
  #variable plot is set so that column Global active power will be executed.

  '''

  "Plots for Auto Correlation and Partial Auto Correlation"

  '''
```

```
print("*********************************")

print("Auto Regressive Integreted Moving Average (ARIMA) Model")

plot = data1['Global_active_power']

acf_original = plot_acf(plot)
pacf_original = plot_pacf(plot)

print("*********************************")

print("Augmented Dickey Fuller Test ")

print("*********************************")

def ad_test(dataset):
  dftest = adfuller(dataset, autolag='AIC')
  print("1. ADF:", dftest[0])
  print("2. P-value:", dftest[1])
  print("3. Num of lags:", dftest[2])
  print(
      "4. Num of observations used for ADF regression and critical values calculation:",
dftest[3])
  print("5. Critical values: ")
  for key, val in dftest[4].items():
    print("\t", key, ":", val)

ad_test(data1['Global_active_power'])

'''
Splitting data into Training and test data, where training data contains 90
% while test data contains the remaining data.
'''
```

```
  print("********************************")

  print("Shapes of Train and Test Data ")

  print("********************************")

  train_size = int(len(data1) * 0.9)
  train, test = data1[:train_size], data1[train_size:]
  print(train.shape, test.shape)

  print("********************************")

  return train, test, data1
def arima_fit(train, test):



   print("..........................................")
   p_values=range(0,5)
   d_values=range(0,2)
   q_values=range(0,5)
   for p in p_values:
     for d in d_values:
       for q in q_values:
         order = (p, d, q)
         predictions = []
         try:
           model = ARIMA(train, order=order)
           model_fit = model.fit()
           y_pred = model_fit.forecast(steps=len(test))[0]
           predictions.extend(y_pred)
           error = mean_squared_error(test, predictions)
           print("ARIMA%s RMSE = %.2f" % (order, error))
           if error < best_rmse:
             best_rmse = error
             best_params = order
         except:
           continue
   arima_model_order = model_fit.summary()
   print("********************************")
   print(arima_model_order)
   n_periods = int(len(test))
   predictions = model_fit.predict(n_periods=n_periods)

   # Slice the predictions array to only include the last n_periods
   sliced_predictions = predictions[-n_periods:]

   # Calculate RMSE



   rmse = sqrt(mean_squared_error(test, sliced_predictions))
   # Calculate MAE
   mae = mean_absolute_error(test, sliced_predictions)

   print("********************************")

   print("Root Mean Squared Error and Mean Absolute Error of the data. ")

   print("********************************")

   print(f'RMSE: {rmse}')
   print(f'MAE: {mae}')

   return test, sliced_predictions, arima_model_order

def xgboost_initialize(data):
  """
```

```
Returns:

"""
print("********************************")

print("XGBoost Model")

print("********************************")

def encode(data, col, max_val):
  data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
  data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
  data.drop(col, axis=1, inplace=True)
  return data

# Setting lagging time of 15 mins, 30 mins and 1 hr respectively.

def get_lag(data, col, lagtime):
  for i in range(1, lagtime + 1):
    if len(pd.Series(col)) == 1:
      data[col+"_lag"+str(i)] = data[col].shift(i*15)
    else:
      for col_j in col:
        data[col_j+"_lag"+str(i)] = data[col_j].shift(i*15)
  return data

# Create Time Series Features
data['date'] = data.index
data['hour'] = data['date'].dt.hour
data = encode(data, 'hour', 24)
data['dayofweek'] = data['date'].dt.dayofweek
data = encode(data, 'dayofweek', 7)
```

```
data['month'] = data['date'].dt.month
data = encode(data, 'month', 12)
# newdf['year'] = newdf['date'].dt.year
data['dayofyear'] = data['date'].dt.dayofyear
data = encode(data, 'dayofyear', 365)
data['dayofmonth'] = data['date'].dt.day
data = encode(data, 'dayofmonth', 31)
#newdf['weekofyear'] = newdf['date'].dt.weekofyear
data.drop('date', axis=1, inplace=True)

# adding lagging feature
# adding time lags of 15min,30min as features
lagtime = 2
lag_feature = ['Global_reactive_power', 'Voltage', 'Global_intensity',
            'Sub_metering_1', 'Sub_metering_2', 'Sub_metering_3']
newdf = get_lag(data, lag_feature, lagtime)

# adding time lags of 2 hours as features
data = get_lag(data, 'Global_active_power', 8)
data.drop(lag_feature, axis=1, inplace=True)
data.dropna(inplace=True)

split_date = pd.to_datetime('2010-07-05')
df_train = data.loc[data.index <= split_date].copy()
df_train.shape

train_size = int(len(data) * 0.9)
train, test = newdf[:train_size], newdf[train_size:]

# The target is forecasting Global_active_power.
X_train, y_train = train.iloc[:, 1:], train.iloc[:, 0]
X_test, y_test = test.iloc[:, 1:], test.iloc[:, 0]
```

```
  print("********************************")

  return X_train, y_train, X_test, y_test, train, test

def xgboost_fit(data):

  X_train, y_train, X_test, y_test, train, test = xgboost_initialize(
    power_consumption)

  print("********************************")

  reg = xgb.XGBRegressor(n_estimators=1000)
  reg.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_test, y_test)],
        early_stopping_rounds=50, verbose=False)

  print("********************************")

  test['Prediction'] = reg.predict(X_test)
  df_all = pd.concat([test, train], sort=False)

  print("********************************")

  # Calculate RMSE
  rmse = np.sqrt(mean_squared_error(
    test['Global_active_power'], test['Prediction']))

  # Calculate MAE
  mae = mean_absolute_error(test['Global_active_power'], test['Prediction'])

  print("Root Mean Squared Error (RMSE):", rmse)
  print("Mean Absolute Error (MAE):", mae)

  actual_plot = test['Global_active_power']
  prediction_plot = test['Prediction']

  return reg, actual_plot, prediction_plot

  print("********************************")


def lstm_initialize(newdf):
  print("********************************")

  print("Long Short Term Memory (LSTM).")

  print("********************************")

  data1 = newdf.loc[:, ['Global_active_power']]

  #Transform the Global_active_power column of the data DataFrame into a numpy array of
float values

  dataset = data1.Global_active_power.values.astype('float32')
  #Reshape the numpy array into a 2D array with 1 column

  dataset = np.reshape(dataset, (-1, 1))
  #Create an instance of the MinMaxScaler class to scale the values between 0 and 1
  scaler = MinMaxScaler(feature_range=(0, 1))

  #Fit the MinMaxScaler to the transformed data and transform the values

  dataset = scaler.fit_transform(dataset)

  #Split the transformed data into a training set (90%) and a test set (10%)
  train_size = int(len(dataset) * 0.90)
  test_size = len(dataset) - train_size
  train, test = dataset[0:train_size, :], dataset[train_size:len(dataset), :]
  def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

  # reshape into X=t and Y=t+1
  look_back = 30
  X_train, Y_train = create_dataset(train, look_back)
  X_test, Y_test = create_dataset(test, look_back)

  # reshape input to be [samples, time steps, features]
  X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
  X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

  return scaler, X_train, Y_train, X_test, Y_test

def lstm_fit(data):
  scaler, X_train, Y_train, X_test, Y_test = lstm_initialize(data)
  # Defining the LSTM model
  model = Sequential()
  # Adding the first layer with 100 LSTM units and input shape of the data
  model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
  # Adding a dropout layer to avoid overfitting
  model.add(Dropout(0.2))
  # Adding a dense layer with 1 unit to make predictions
  model.add(Dense(1))
  # Compiling the model with mean squared error as the loss function and using Adam optimizer
  model.compile(loss='mean_squared_error', optimizer='adam')
```

```
# Plot the model
plot = plot_model(model, to_file='lstm_model.png')

print(" Model training with 50 epochs.")

print("********************************")

# Fitting the model on training data and using early stopping to avoid overfitting
history = model.fit(X_train, Y_train, epochs=50, batch_size=1240,
                    validation_data=(X_test, Y_test),
                    callbacks=[EarlyStopping(
                        monitor='val_loss', patience=4)],
                    verbose=1, shuffle=False)
# Displaying a summary of the model
print("Summary of the model. ")

print("********************************")
model.summary()

# make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

print("Prediction using test data ")

print("********************************")

# invert predictions
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform([Y_test])

print('Root Mean Squared Error (RMSE):', np.sqrt(
    mean_squared_error(Y_test[0], test_predict[:, 0])))
```

```
  print('Mean Absolute Error (MAE):',
      mean_absolute_error(Y_test[0], test_predict[:, 0]))

  print("********************************")

  return plot, Y_test, test_predict, history

def visualization(test_set, sliced_predictions, actual_plot, prediction_plot, history, Y_test,
test_predict):

  def plot_arima(test_set, sliced_predictions):
    print("ARIMA Model test data plot of Actual vs Predicted")
    # Plot actual vs. predicted values
    plt.figure(figsize=(20, 6))
    plt.plot(test_set.index,
         test_set['Global_active_power'], color='blue', label='Actual')
    plt.plot(test_set.index, sliced_predictions,
         color='red', label='Predicted')
    plt.title('Actual vs. Predicted Global Active Power')
    plt.xlabel('Date')
    plt.ylabel('Global Active Power')
    plt.legend()
    plt.show()

  def plot_xgboost(actual_plot, prediction_plot):
    print("XGBoost Model test data plot of Actual vs Predicted")
    plt.figure(figsize=(20, 6))
    plt.plot(actual_plot.index, actual_plot, color='blue', label='Actual')
    plt.plot(actual_plot.index, prediction_plot,
         color='red', label='Predicted')
    plt.xlabel('Date')
    plt.ylabel('Global Active Power')
    plt.title('Actual vs Predicted Global Active Power')
    plt.legend()
  plt.show()

def plot_lstm(history, Y_test, test_predict):
  print("LSTM Model loss plot")

  plt.figure(figsize=(8, 4))
  plt.plot(history.history['loss'], label='Train Loss')
  plt.plot(history.history['val_loss'], label='Test Loss')
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epochs')
  #plt.xticks(range(0,21))
  plt.legend(loc='upper right')
  plt.show()

  print("LSTM Model test data plot of Actual vs Predicted")

  aa = [x for x in range(len(Y_test[0]))]

  # Creating a figure object with desired figure size
  plt.figure(figsize=(20, 6))

  plt.title("Actual vs Predicted Global Active Power")

  # Plotting the actual values in purple with a dot marker
  plt.plot(aa, Y_test[0], marker='.', label="actual", color='blue')

  # Plotting the predicted values in red with a solid line
  plt.plot(aa, test_predict[:, 0], '-', label="prediction", color='red')

  # Removing the top spines
  sns.despine(top=True)
```

```
    # Adjusting the subplot location
    plt.subplots_adjust(left=0.07)

    # Labeling the y-axis
    plt.ylabel('Global_active_power', size=14)

    # Labeling the x-axis
    plt.xlabel('Time step', size=14)

    # Adding a legend with font size of 15
    plt.legend(fontsize=16)

    # Display the plot
    plt.show()

  plot_arima(test_set, sliced_predictions)
  plot_xgboost(actual_plot, prediction_plot)
  plot_lstm(history, Y_test, test_predict)

# Data Preprocessing
power_consumption                                                            =
data_preprocessing('/content/drive/MyDrive/household_power_consumption.txt')

# Perform exploratory data analysis
ed_analysis(power_consumption)

#Initializing ARIMA Mode
train_data, test_data, data1 = arima_initialize(power_consumption)

# Fit ARIMA model using the training data and evaluate its performance
test_set, sliced_predictions, arima_model_order = arima_fit(train_data, test_data)

# XGBoost model initialization and fitting
reg, actual_plot, prediction_plot = xgboost_fit(power_consumption)
```

```
# LSTM model initialization and training
plot, Y_test, test_predict, history = lstm_fit(power_consumption)

# Visualization of the three models.
visualization(test_set, sliced_predictions, actual_plot, prediction_plot, history, Y_test, test_predict)
```