Phase 5 : FINAL PROJECT

Name : Gayathri S

Register Number : 312621243013

College Name : Thangavelu engineering college

Project 4: Electricity Prices Prediction

Objective :

➢. The electricity price prediction task is based on a case study where you need to predict daily price of electricity based on the daily consumption of heavy machinery used by businessess.

Problem Statement :

➢. Create a predictive model that utilizes historical electricity prices and relevant factors to forecast future electricity prices, assisting energy providers and consumers in making informed decisions regarding consumption and investment.

Problem Definition :

➢ The problem is to develop a predictive model that uses historical electricity prices and relevant factors to forecast future electricity prices. The objective is to create a tool that assists both energy providers and consumers in making informed decisions regarding consumption and investment by predicting future electricity prices. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

Data Source :

➢. Utilize a dataset containing historical electricity prices and relevant factors like date, demand, supply, weather conditions, and economic indicators.

CODING :

```
Load your electricity price dataset

import pandas as pd

data = pd.read_csv('Electricity.csv')
```

Design Thinking :

- ❖ Data Preprocessing
- ❖ Feature Engineering
- ❖ Model Selection
- ❖ Model Training
- ❖ Evaluation

Data preprocessing :

➢ .Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task. Data preprocessing is an important step in the data mining process. It refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data

preprocessing is to improve the quality of the data and to make it more suitable for the specific data mining task.

CODING :

```
print(data.describe())  # Summary statistics
print(data.isnull().sum())  # Check for missing values
# Handle missing values (if any)
data.fillna(data.mean(), inplace=True)
# Remove duplicate values (if any)
data = data.drop_duplicates()
```

FEATURE ENGINEERING :

Abstract :

- .The growing violation of the day ahead electricity market (DAM) and price forecast errors are attributed to changing market conditions, such as increased renewable energy production and intraday market trading. To improve forecast accuracies, new features must be identified to capture the effects of technical or fundamental price drivers. This paper focuses on identifying and engineering technical

features that capture the behavioral biases of DAM traders using technical indicators (Tis), such as Bollinger Bands Momentum indicators or exponential moving averages. The inclusion of TI features in DAM forecasting can significantly boost regression accuracies of machine learning models, reducing root mean squared errors by up to 4.50%, 5.42%, and 4.09%, respectively.

Materials and Methods :

- .While a plethora of Ts exists, none can guarantee the addition of explanatory power. This is because the success of individual Tis is domain and period-dependent. For instance, price-based trend-following indicators may be useful in times of high autocorrelation; however, they case being indicative of future price moves when autocorrelation vanishes and trends become spurious [19] We are mindful of this while selecting a list of Tis to examine, and as a result, choose Tis satisfying the following two criteria

    - I calculation only requises dose prices.

- T1 inclusion is likely to improve predictive performance by highlighting oscillations or trends in DAM prices

Criterion 1 :

➢ Necessitates from the properties of the DAM, neither high, low, not open prices are available. Other financial markets record security's open/clase/high/low prices, which are the initial/last/highest/lowest recorded trading prices over a given period, respectively . Together,these metrics can be used to convey primary information about a security's price movements over a specific period. Unlike other financial markets, however, the DAM only records a single clearing price for each hourly contract We consider this price to be the close price

Criterion 2 :

➢ . Stems from our desire to avoid examining overly intricate The which have been optimised for ran use cases and evaluate Ths which are well suited for the DAM DAM prices typically move between horizontal support and resistance

lines, behaving comparably to ranging markets whose prices move within a band. On occasion, however, DAM prices breakout from this price band, and establish a trend, behaving comparably to trending markets. Overall, such varying behaviour spurs un to evaluate both oscillators, which vary around a central line or within set levels,and trend following Ths, which measure the direction of a trend. The former Ts are well suited for ranging markets, while the latter Tis are optimised for trending markets Together, criteria 1 and 2 allow us to follow the suggestions of [1] by focusing our research on the simplest and most widely used Tls, which are likely to introduce more explanatory power than noise. Below, we introduce our list of chosen Tls and provide formulas for their calculation

Note that throughout the paper the following notations are used ,

- Time
- Price at time
- Lag-factor for $\forall n \in$ Z
- s span for $\forall n \in$ Z

where and are hyperparameters tuned using grid-search during the modelling of DAM prices

CODING :

```
# Time-based features

data['Date'] = pd.to_datetime(data['Date'])

data['Year'] = data['Date'].dt.year

data['Month'] = data['Date'].dt.month

data['DayOfWeek'] = data['Date'].dt.dayofweek

# Lagged variables

data['ElectricityPrice_Lag1'] = data['ElectricityPrice'].shift(1)

data['ElectricityPrice_Lag7'] = data['ElectricityPrice'].shift(7)
```

MODEL SELECTION :

Abstract :

- .This research proposes a hybrid model for forecasting university buildings' electricity load using clustering and the autoregressive integrated moving average (ARIMA) model. The approach uses K-means clustering to cluster data from the entire year and forecast the electricity peak load. The

combination of clustering and ARIMA model improves forecasting performance, allowing management authorities to design strategies for peak load reduction. This method can also be implemented in demand response to reduce electricity bills by avoiding usage during high electricity rate hours. Understanding energy consumption patterns in buildings is crucial for optimizing resources and conserving energy.

ARIMA model :

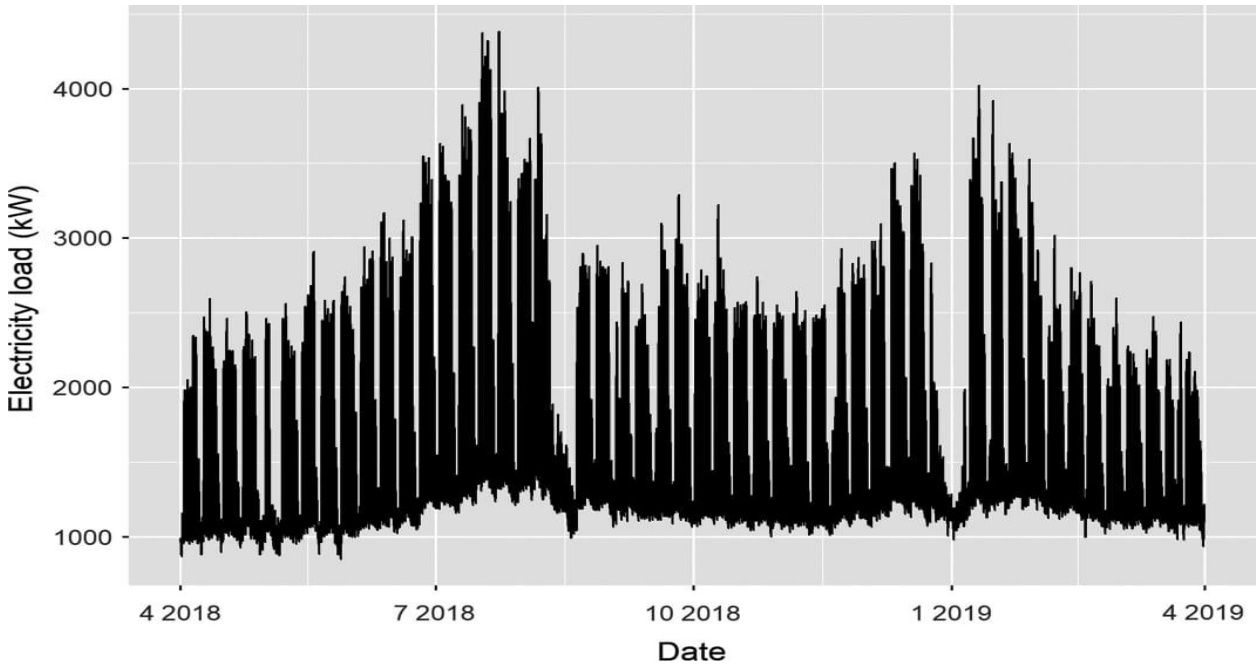
- .Box and Jenkins developed a mathematical model for forecasting a time series by fitting it with data and using the fitted model for forecasting (ie, ARIMA model). In general, the ARIMA process is written with the notation ARIMA(p,d,q), where p denotes the number of autoregressive orders in the model. Autoregressive orders specify the previous values from the series which are used to predict current values; difference (d) specifies the order of differencing applied to the series before estimating models; and moving average (q) specifies how deviations from the series mean for previous values are used to predict current values.

An ARMA consists of two parts, an autoregressive (AR) part and a moving (MA) part. The model is usually then referred to as the ARMA (p, q) model where p is the order of the autoregressive part and q is the order of the moving average part.

$$X_t = c + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p X_{t-P} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

An ARIMA(p,d,q) model is a generalization of an ARMA model where p, d, and q are non-negative integers that refer to the order of the autoregressive, integrated, and moving parts of the model, respectively.

To deal with the seasonality of the ARIMA model, the generalized ARIMA model with the seasonal differencing of an appropriate order is used to remove non-stationary items from the series. For a monthly time series s = 12, and for quarterly time series s = 4. The model is generally termed as SARIMA(p,d,q) × (P,D,Q)s model.



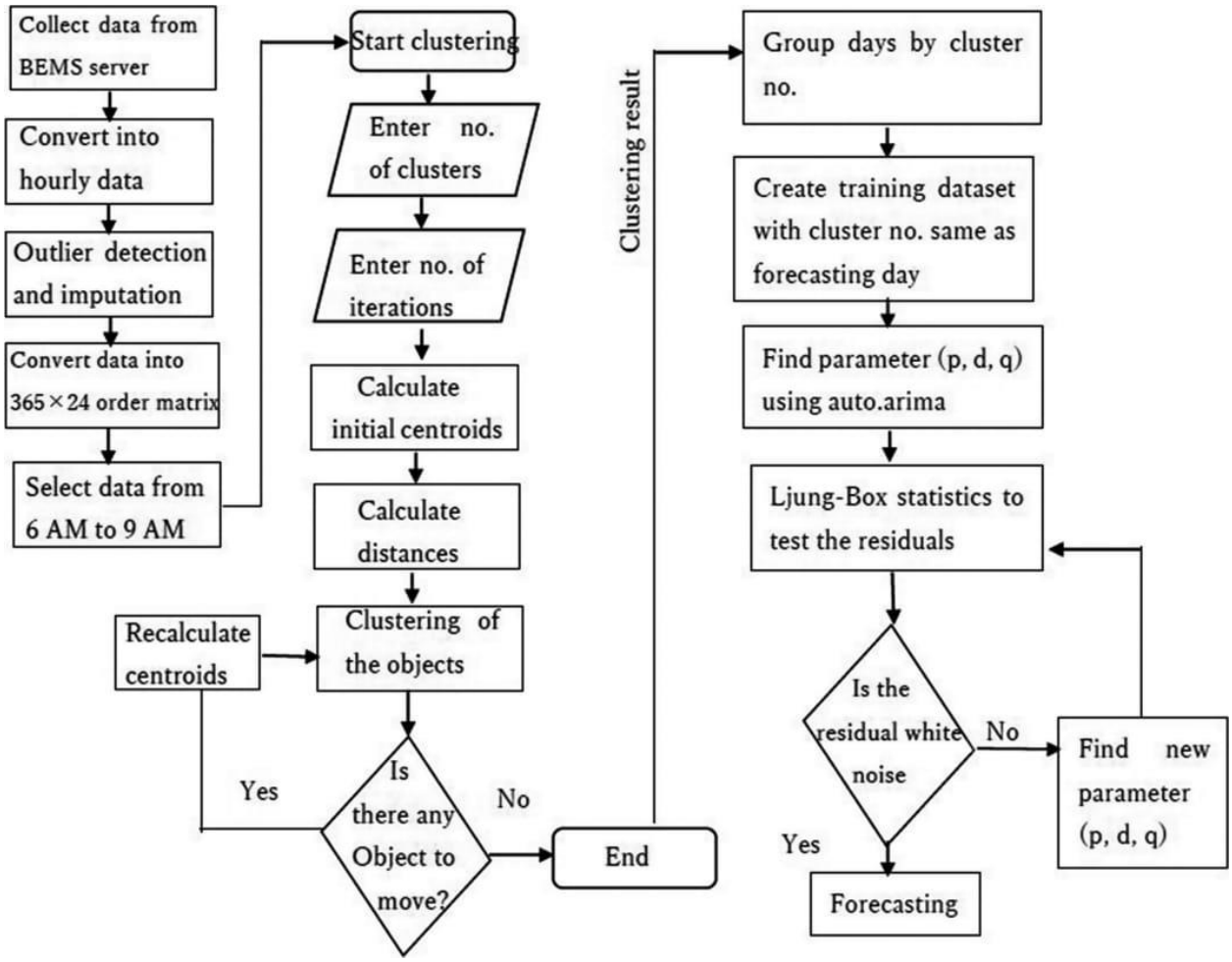
Methodology :

- .This research uses hourly electricity load data from the East Campus of Chubu University from 2017-2018, collected through the Building Energy Management System (BEMS). The data is summed to produce hourly data, which consists of 8760 data for each building. However, technical issues have caused outliers and missing values in the data, potentially leading to errors in forecasting output. To address this, the outliers are converted into not available (NA) values and imputed with appropriate values using linear interpolation using the zoo package of R programming language.

The K-means clustering method is used to analyze data free from missing values and outliers, dividing a year's electricity load data into six clusters. The main objective is to forecast peak energy consumption in university buildings and develop an automatic process to reduce peak load. Factors such as holidays, lectures, air-conditioning use, events, and experimental facilities affect energy load, causing high errors in forecasting. To address this, a hybrid model with clustering and ARIMA model is created, which clusters data to determine the cluster number of the forecasting day. The ARIMA model is selected automatically using the auto.arima function of the R programming language. This system aims to improve energy efficiency in university buildings.

This paper presents a process for electricity load forecasting using building energy data from the East campus of Chubu University. The data is converted from 30-minute interval data to hourly data, and then into a 365 × 24 order matrix. The electricity load data from 6 to 9 am is extracted and clustered. The data from 6 to 9 am of the forecasting day

is used for clustering, not electricity load forecasting. Six clusters are identified, and the initial centroids are calculated using the percentile method. The clustering results classify all 365 days into six clusters with similar electricity load characteristics. A dataset with days belonging to the same cluster as the forecasting day is created as the training data for the forecasting model. The parameters of the best model are calculated using the auto.arima function of the R programming language. The residuals of the model are tested using Ljung-Box statistics, and if they are white noise, the model is ready for forecasting, otherwise a new model is selected.



Clustering and accuracy measures :

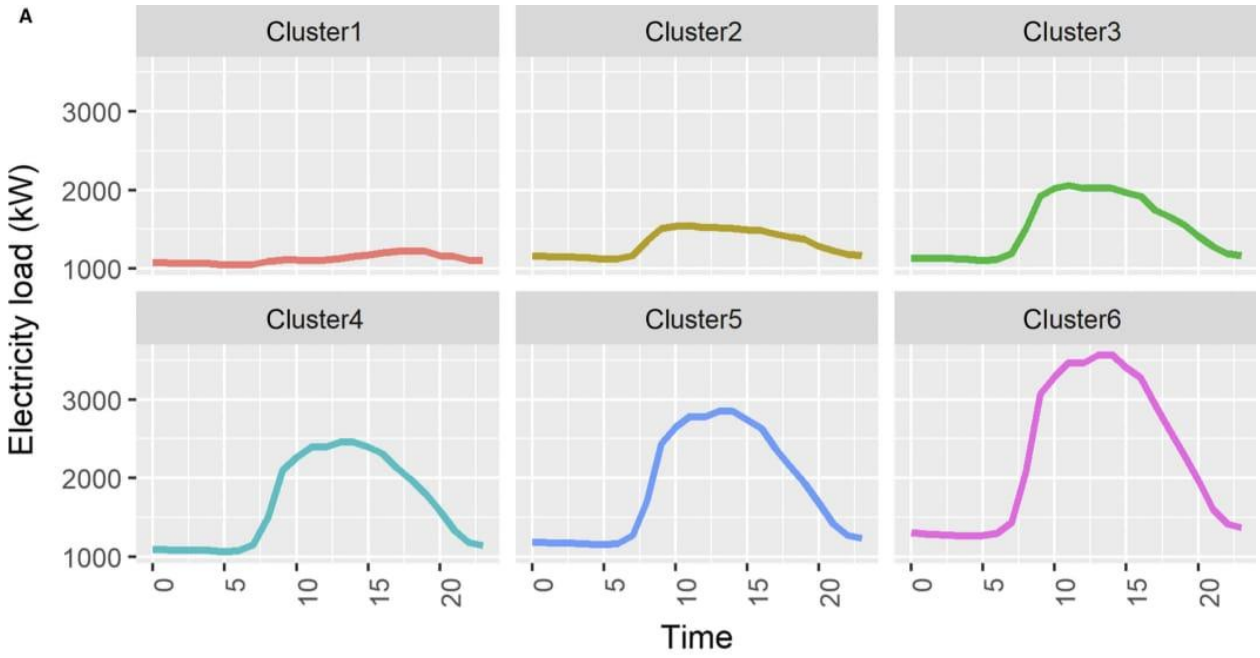
- .The study classified the electricity load of Chubu University into six clusters using K-means clustering based on the hourly distribution of one-year electricity load. The electricity load plots show a significant difference in energy load from cluster 1 to cluster 6, with base energy increasing with the number of clusters. The electric load from 10 pm to 6 am is almost similar, but from 8 am, it sharply rises until reaching its peak around 1 or 2 pm. Clustering the forecasting day and using its cluster members for electricity load

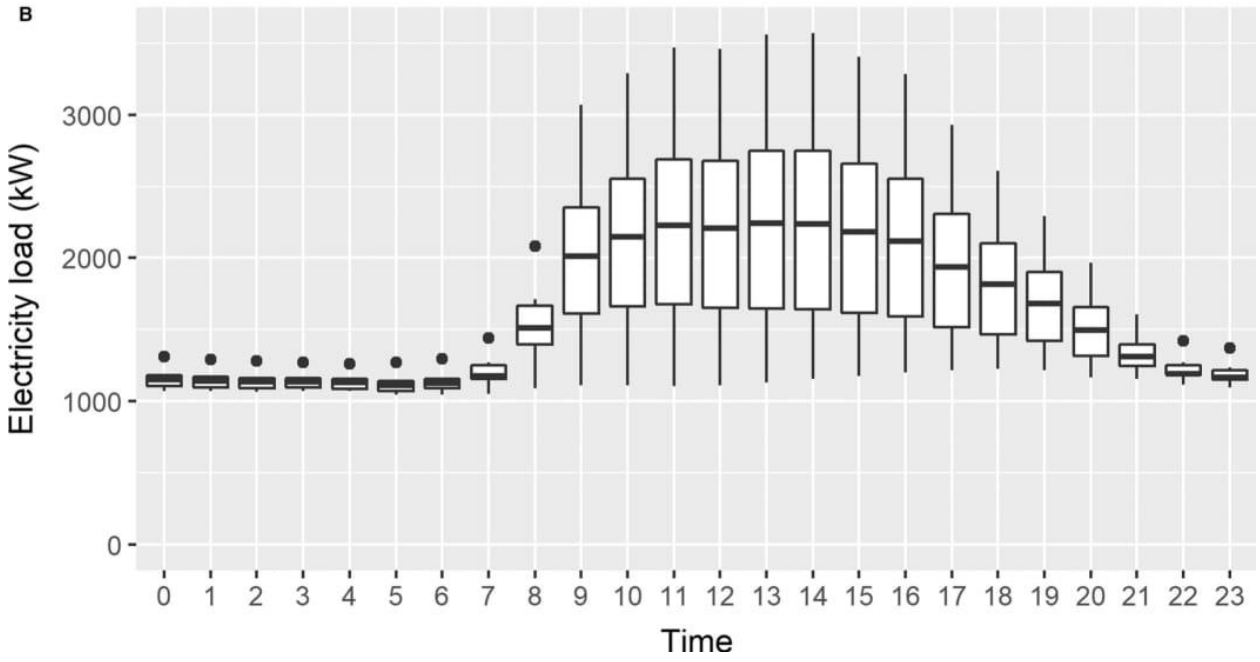
prediction can improve the peak load forecasting accuracy of any forecasting algorithm. Research findings indicate that the performance of forecasting methods varies according to the accuracy measure used, with root mean square error (RMSE), mean absolute percentage error (MAPE), and mean absolute error (MAE) being the early and most popular accuracy measures used in this paper.

RMSE and MAE are scale-dependent measures since their values depend on the scale of the data. They are useful in comparing forecasting methods on the same data. For 24 hours ahead forecasting, if “et” represents the error in forecasting for each hour, RMASE and MAE can be defined as

$$MAE = \frac{1}{n}\Sigma_{t=1}^{n}|e_t|$$

$$MSE = \frac{1}{n}\Sigma_{t=1}^{n}e_t^2 and\ RMSE = \sqrt{MSE}$$

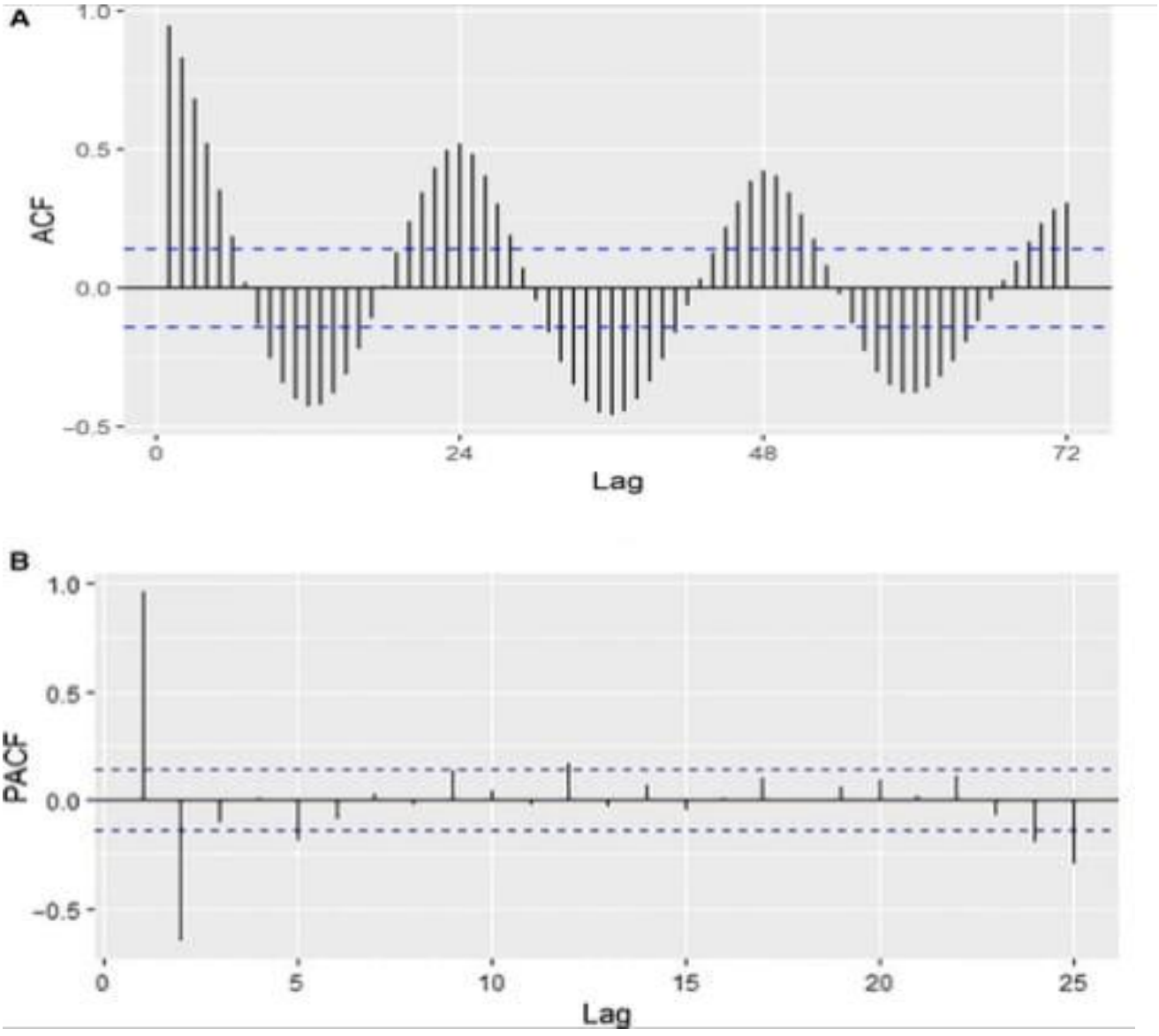




Fitting the ARIMA model :

- .The ARIMA model uses the autocorrelation function (ACF) and partial autocorrelation function (PACF) to estimate parameters p

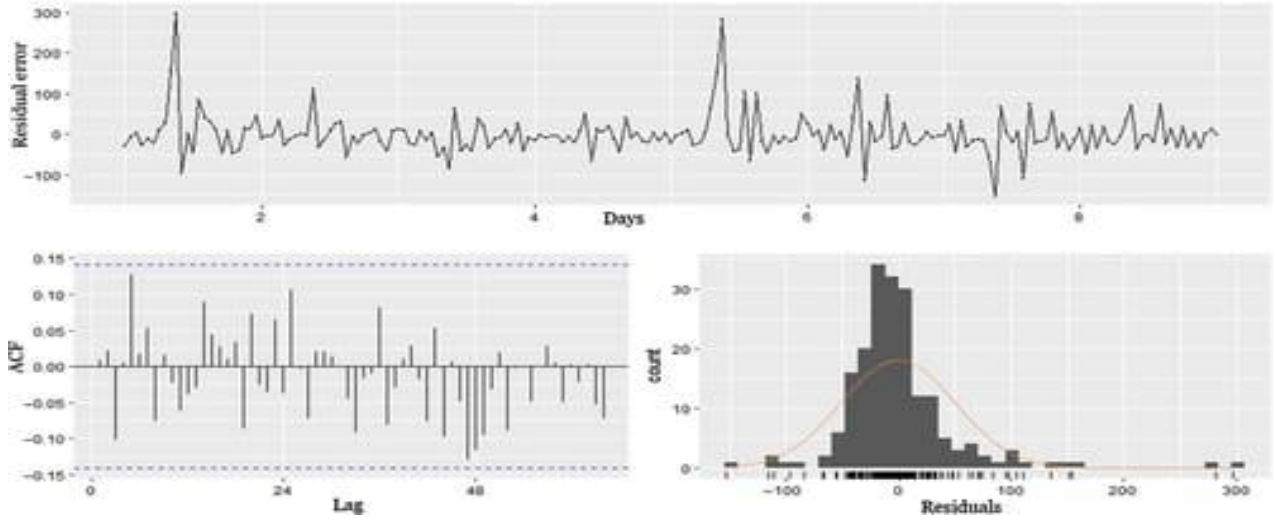
and q. The training dataset for electricity load forecasting on 2nd April 2018 shows the ACF and PACF. The augmented Dickey-Fuller test (ADF) is used to identify stationary/non-stationary processes in the time-series. The ADF test, using the adf.test function in R, found the series stationary, allowing the parameter d in the ARIMA model to be zero.



- .The study focuses on selecting the best model for a given data set by fitting different models and selecting the model with the minimum Akaike information criteria (AIC) value. AIC is an estimator of the relative quality of statistical models for a given data set, and under-fitting may not capture the true nature of the variability in the outcome variable, while over-fitting loses generality. The smallest AIC value for seasonal ARIMA is obtained with non-seasonal part (p,d,q) = (2,0,1) and seasonal part (P,D,Q) = (2,0,1). The parameters of the best model were calculated using the auto.arima function of R programming language, with the best model being "ARIMA(2,0,1)(2,0,1)" with an AIC value of 2096.4.

ARIMA(p,d,q)(P,D,Q)S	AIC
ARIMA(2,0,0)(0,0,0)24	2105.4
ARIMA(2,0,1)(2,0,0)24	2104.5
ARIMA(2,0,1)(2,0,1)24	2096.4
ARIMA(2,0,2)(2,0,1)24	2098.6
ARIMA(2,1,1)(2,0,1)24	2097.6
ARIMA(3,0,1)(2,0,0)24	2105.6
ARIMA(3,0,1)(2,0,1)24	2097.8
ARIMA(3,0,1)(3,0,1)24	2107.9

The residuals plot, ACF values, and histogram indicate that the auto.arima model is best fit for the training dataset. As new data is added, the model's parameters (p, d, q) must be selected for accurate forecasting. The Ljung-Box test confirms residuals as white noise.



CODING :

```
from statsmodels.tsa.arima_model import ARIMA
# Define the ARIMA order (p, d, q)
p = 1  # Example value
d = 1  # Example value
q = 1  # Example value
# Create the ARIMA model
model = ARIMA(data['ElectricityPrice'], order=(p, d, q))
# Fit the model to the data
```

```
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
```

MODEL TRAINING

➢ .It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

- There is no optimal split percentage
- You must choose a split percentage that meets your project's objectives with considerations that include:

❖ Computational cost in training the model.

❖ Computational cost in evaluating the model.

❖ Training set representativeness.

❖ Test set representativeness.

Nevertheless, common split percentages include:

❖ Train: 80%, Test: 20%

- Train: 67%, Test: 33%
- Train: 50%, Test: 50%

CODING :

```
# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train, test = data['ElectricityPrice'][:train_size],
data['ElectricityPrice'][train_size:]
# Initialize and fit the ARIMA model on the training data
model = ARIMA(train, order=(p, d, q))
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
```

EVALUATION

- **Raw_values** :

  Returns a full set of errors in case of multioutput input.

- **Uniform_average**:

  Errors of all outputs are averaged with uniform weight.

- **Squaredbool, default=True**

  If True returns MSE value, if False returns RMSE value

CODING :

```
# Make predictions on the test set
predictions = model_fit.forecast(steps=len(test)
```

```
# Calculate MAE, MSE, RMSE (import necessary libraries)
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
mae = mean_absolute_error(test, predictions)
mse = mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
# Print the evaluation results
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

BODY OF THE CODING:

```
Load your electricity price dataset
import pandas as pd
data = pd.read_csv('Electricity.csv')
print(data.describe())  # Summary statistics
print(data.isnull().sum())  # Check for missing values
# Handle missing values (if any)
data.fillna(data.mean(), inplace=True)
# Remove duplicate values (if any)
data = data.drop_duplicates()
# Time-based features
```

```
data['Date'] = pd.to_datetime(data['Date'])
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['DayOfWeek'] = data['Date'].dt.dayofweek
# Lagged variables
data['ElectricityPrice_Lag1'] = data['ElectricityPrice'].shift(1)
data['ElectricityPrice_Lag7'] = data['ElectricityPrice'].shift(7)
from statsmodels.tsa.arima_model import ARIMA
# Define the ARIMA order (p, d, q)
p = 1  # Example value
d = 1  # Example value
q = 1  # Example value
# Create the ARIMA model
model = ARIMA(data['ElectricityPrice'], order=(p, d, q))
# Fit the model to the data
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
# Split the data into training and testing sets
train_size = int(len(data) * 0.8)
train, test = data['ElectricityPrice'][:train_size],
data['ElectricityPrice'][train_size:]
```

```
# Initialize and fit the ARIMA model on the training data
model = ARIMA(train, order=(p, d, q))
model_fit = model.fit()
# Print the summary of the model
print(model_fit.summary())
# Make predictions on the test set
predictions = model_fit.forecast(steps=len(test))
# Calculate MAE, MSE, RMSE (import necessary libraries)
from sklearn.metrics import mean_absolute_error,
mean_squared_error
import math
mae = mean_absolute_error(test, predictions)
mse = mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
# Print the evaluation results
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
```

NOTE : Run the program with compiler with csv.file