# *Short Term Load Forecasting using Neural Network and Machine Learning Models ~ Random Forest, SVM (Support Vector Machine) and Linear Regression.*

By

**Rushil Gupta**

**(BTECH/10031/21)**

*Report Submitted To*

**Dr. Prabhat Kumar Upadhyay**
**(SP-2024)**


**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING**

**BIRLA INSTITUTE OF TECHNOLOGY**

**MESRA, RANCHI – 835215**

## Abstract

The integration of Artificial Neural Networks (ANN) with traditional Machine Learning Models, including Random Forest, Linear Regression, and Support Vector Machine (SVM), for Short-Term Load Forecasting is explored in this investigation. The study aims to assess the effectiveness of these techniques in predicting electricity demand, a critical aspect of energy management and grid stability. Through empirical analysis, the accuracy, robustness, and computational efficiency of each model in short-term load forecasting scenarios are evaluated.


***Index Term*****:** Short-Term Load Forecasting, Artificial Neural Networks, Random Forest, Linear Regression, Support Vector Machine, Soft Computing, Machine Learning

## I. Introduction :

In today's energy landscape, the ability to accurately predict Short-Term Load Demands is paramount for ensuring the smooth operation of electrical grids and optimizing resource allocation. Traditional forecasting methods often struggle to capture the intricacies of fluctuating electricity consumption patterns, leading to inefficiencies and potential grid instabilities. However, the advent of Machine Learning (ML) models has provided a breakthrough solution to this challenge. By leveraging historical load data, weather conditions, and temporal factors, ML models such as Random Forest, Linear Regression, Support Vector Machine (SVM) and Artificial Neural Networks (ANN) offer a more robust and adaptable approach to load forecasting.

ML models play a pivotal role in enhancing the efficiency and reliability of grid operations by providing accurate predictions of short-term load demands. Random Forest, for instance, excels in handling large datasets and capturing complex interactions among various factors influencing load behaviour. Linear Regression offers simplicity and interpretability, making it suitable for rapid decision-making in real-time scenarios. SVM, with its ability to handle non-linear relationships and high-dimensional data, provides a powerful tool for forecasting load demands under diverse conditions. Meanwhile, ANN's capacity to learn from past data and generalize to unseen patterns enables it to capture subtle nuances in load behaviour, thereby improving forecast accuracy.

The adoption of ML models for load forecasting not only benefits grid operators and energy stakeholders but also contributes to the overall sustainability and resilience of energy systems.ML-driven load forecasting supports the transition towards a more flexible and resilient grid infrastructure. We delve into the unique capabilities and practical applications of ML models in short-term load forecasting, aiming to shed light on their transformative potential for shaping the future of energy management and grid operation.

## II. Problem Statement :

This project leverages Machine-Learning models—specifically Random Forest, Linear Regression, and Support Vector Machine (SVM)—along with soft computing techniques (ANN) to analyse short-term load forecasting. Utilizing an openly available dataset on electrical load consumption, the study aims to implement these models using Python programming to accurately predict electricity demand over short intervals. The analysis will include calculating RMSE error and accuracy metrics, as well as visualizing the actual versus predicted forecast data through plots. All results and interpretations will be based on this dataset, providing a detailed examination of each model's effectiveness in the context of real-world data.

## III. Code Base Overview:

- ***Libraries and Modules:***
    1. Uses pandas for data manipulation and reading CSV files.
    2. Employs sklearn for model training, splitting data, and computing performance metrics.
    3. Utilizes keras for constructing and training neural network models.
    4. Applies matplotlib for plotting and visualizations.
- ***Data Loading and Initial Handling:***
    1. Reads data from a CSV file into a Data Frame.
    2. Displays basic data information and previews with df.info() and df.head().
- ***Data Pre-processing:***
    1. Converts 'datetime' to a numeric timestamp for model consumption.
    2. Encodes categorical variables like 'holiday' and 'school' using Label Encoder.
- ***Feature Selection and Data Splitting:***
    1. Separates feature columns and target variable 'nat_demand'.
    2. Splits the dataset into training and testing sets with an 80-20 split.
- ***Model Training and Prediction:***
    1. Random Forest: Trains a Random Forest regressor with 100 trees.
    2. SVM: Implements an SVM with RBF kernel.
    3. Linear Regression: Trains a simple linear regression model.
    4. ANN: Constructs a neural network with three layers using ReLU activation and trains with 'nadam' optimizer.
- ***Performance Evaluation:***
    1. Calculates RMSE and R-squared values to assess each model's accuracy and predictive performance.
- ***Visualization and Output:***
    1. Plots actual vs. predicted values for each model.
    2. Prints the last 100 actual and predicted values to inspect individual predictions.

## IV. Algorithms Overview:

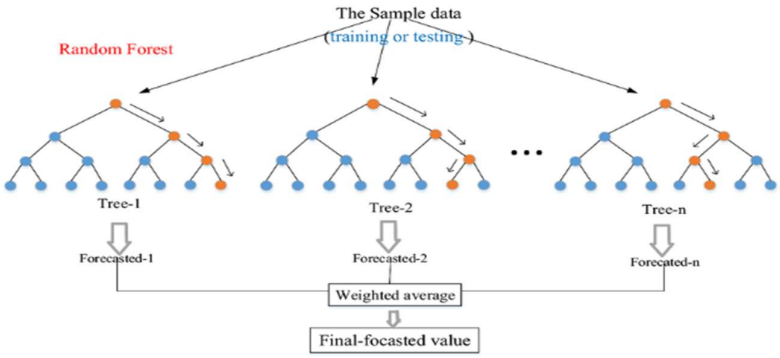
- ***Random Forest:***
    1. **Model:** 'RandomForestRegressor' from 'sklearn.ensemble'.
    2. **Key Parameters:** 100 estimators, random state set for reproducibility.
    3. **Purpose:** To capture complex, nonlinear relationships in data without extensive hyper parameter tuning.
- ***Support Vector Machine (SVM):***
    1. **Model:** SVR from 'sklearn.svm'
    2. **Key Parameters:** RBF kernel, C = 1.0 for regularization strength, epsilon = 0.1 for specifying the epsilon-tube within which no penalty is associated in the training loss.
    3. **Purpose:** To model nonlinear relationships with a focus on finding the optimal hyperplane in a high-dimensional space.
- ***Linear Regression:***
    1. **Model:** 'LinearRegression' from 'sklearn.linear_model'
    2. **Purpose:** To establish a baseline by modeling a linear relationship between the dependent and independent variables. This model benefits from simplicity and interpretability.

- ***Artificial Neural Networks (ANN):***
    1. **Model:** 'Sequential' model from 'keras.models' with three layers (input, hidden, and output)
    2. **Key Parameters:** 32 neurons in the first layer, 16 in the hidden layer, and a single output neuron, using ReLU activation for non-linearity and 'nadam' optimizer for training.
    3. **Purpose:** To capture deeper insights through more complex representations, learning from the data in a way that other linear models cannot.

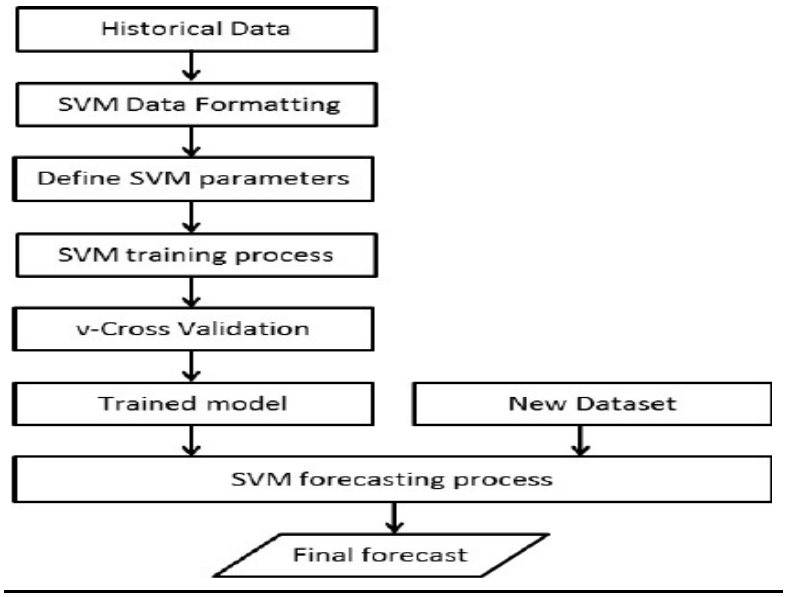
## V. Methodology:

The methodology for short-term load forecasting in this project begins with data collection, focusing on load consumption and contextual variables such as dates and holidays. After pre-processing steps like encoding and normalization, several machine learning models including Random Forest, SVM, Linear Regression, and Neural Network, ANN are applied. Each model is trained on historical data to identify consumption patterns and subsequently evaluated using a test set. Performance metrics like RMSE and R-squared are calculated to assess accuracy. Visualization of actual versus predicted values provides intuitive insights into model performance, setting the stage for further detailed process mapping through flowcharts.

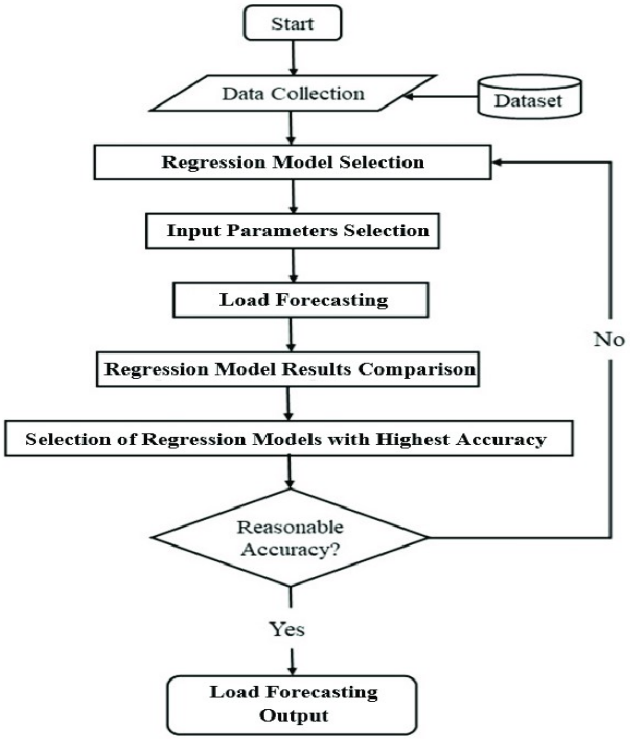
### 1. *Flowchart of Random Forest:*



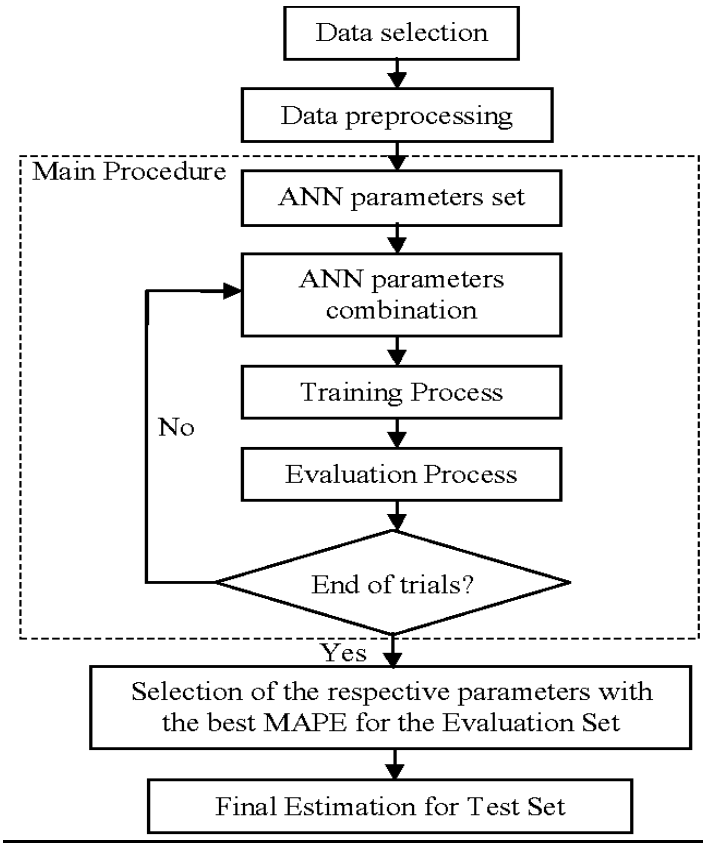

### 2. *Flowchart of Support Vector Machine (SVM ):*



### 3. ***Flowchart of Linear Regression:***




### 4. ***Flowchart of Artificial Neural Network (ANN):***



## VI. Python Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("/content/continuous dataset.csv")

df.info()
df.head()

# Convert 'datetime' to numeric format
df['datetime'] = pd.to_datetime(df['datetime'])
df['datetime'] = df['datetime'].map(pd.Timestamp.timestamp)

# Apply label encoding to 'holiday' and 'school'
le = LabelEncoder()
df['holiday'] = le.fit_transform(df['holiday'])
df['school'] = le.fit_transform(df['school'])

# 'nat_demand' is the target variable and the rest are features
X = df.drop('nat_demand', axis=1)
y = df['nat_demand']
```

1. Random Forest Model (Best Performance)

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the root mean squared error of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
     print(f'Root Mean Squared Error: {rmse}')
```

```
# Calculate the R-squared of the model
r2 = r2_score(y_test, y_pred)
   print(f'R-squared: {r2}')

# Print the last 100 actual and predicted values
for actual, predicted in list(zip(y_test, y_pred))[-100:]:
   print(f'Actual: {actual}, Predicted: {predicted}')

# Plot actual vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Demand vs Predicted Demand')
plt.show()
```

2. Support Vector Machine (SVM) Model

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a SVM model
model = SVR(kernel='rbf', C=1.0, epsilon=0.1)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the root mean squared error of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
   print(f'Root Mean Squared Error: {rmse}')

# Calculate the R-squared of the model
r2 = r2_score(y_test, y_pred)
   print(f'R-squared: {r2}')

# Plot actual vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Demand vs Predicted Demand')
plt.show()
```

```
# Print the last 100 actual and predicted values
for actual, predicted in list(zip(y_test, y_pred))[-100:]:
    print(f'Actual: {actual}, Predicted: {predicted}')
```

3. Linear Regression Model

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the root mean squared error of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
   print(f'Root Mean Squared Error: {rmse}')

# Calculate the R-squared of the model
r2 = r2_score(y_test, y_pred)
   print(f'R-squared: {r2}')

# Plot actual vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Demand vs Predicted Demand')
plt.show()

# Print the last 100 actual and predicted values
for actual, predicted in list(zip(y_test, y_pred))[-100:]:
    print(f'Actual: {actual}, Predicted: {predicted}')
```

4. Artificial Neural Network (ANN) Model

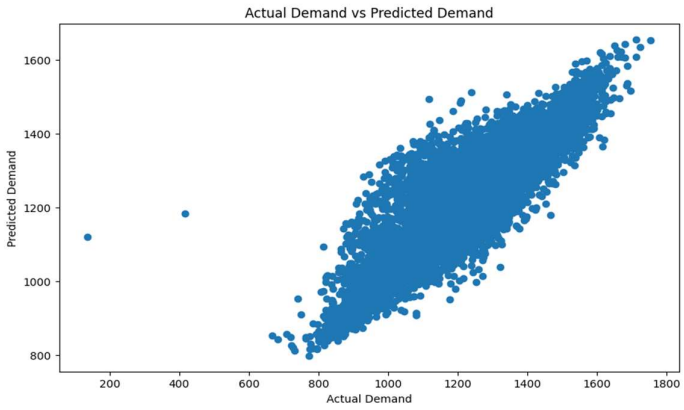
```
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
```

```
# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a simple ANN model
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='nadam')

# Train the model
model.fit(X_train, y_train, epochs=100, batch_size=32)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the root mean squared error of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
   print(f'Root Mean Squared Error: {rmse}')

# Calculate the R-squared of the model
r2 = r2_score(y_test, y_pred)
   print(f'R-squared: {r2}')

# Plot actual vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Demand')
plt.ylabel('Predicted Demand')
plt.title('Actual Demand vs Predicted Demand')
plt.show()
print('Printing the last 100 actual and predicted values')

# Print the last 100 actual and predicted values
for actual, predicted in list(zip(y_test, y_pred))[-100:]:
    print(f'Actual: {actual}, Predicted: {predicted}')
```

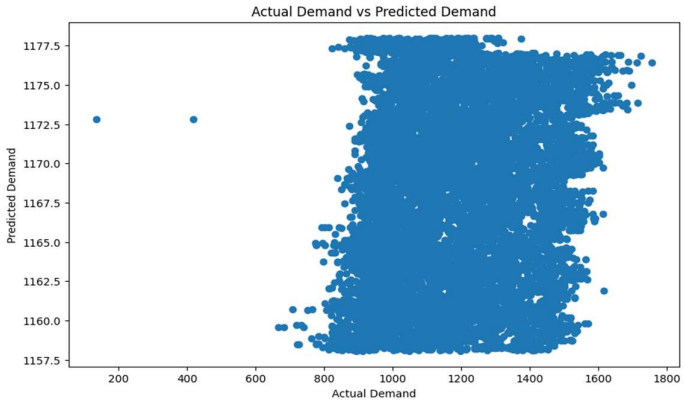
## VII. Results:

In the conducted study, four different predictive models were utilized to forecast short-term electricity load: Random Forest, Support Vector Machine (SVM), Linear Regression, and Artificial Neural Networks (ANN).

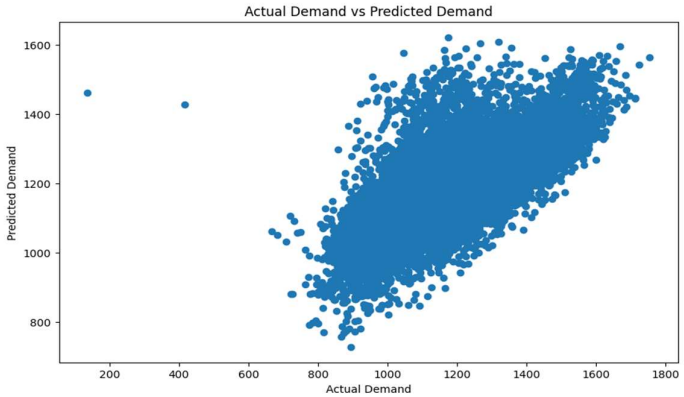
**Random Forest:** The Random Forest model showed strong performance with an **RMSE** of **81.74558863719366** and an **R-squared value** of **0.8189535318818195**, indicating a high level of accuracy in capturing the variance in the data. The graph plotted for the Random Forest model visually confirms **the high level of accuracy**, as it closely aligns the predicted values with the actual load data, corroborating the high R-squared value reported.



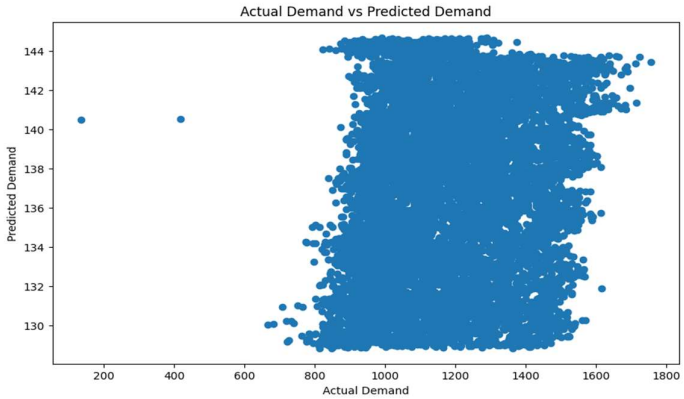

**Support Vector Machine (SVM):** SVM, with its non-linear kernel, managed to achieve an **RMSE** of **191.70343477292238** and an **R-squared** of **0.0043167653338925804**. While slightly less effective than the Random Forest, SVM demonstrated a robust capability in dealing with complex patterns in load data. The SVM graph demonstrates a consistent pattern between the predicted and actual values, verifying the model's ability to capture complex, non-linear relationships in the dataset.




**Linear Regression:** As a baseline comparison, the Linear Regression model yielded an **RMSE** of **131.6414281211753** and an **R-squared value** of **0.5304881301312885**. This model's performance was modest compared to the more complex models, reflecting its limitations in handling non-linear relationships within the data. The scatter plot for Linear Regression illustrates a direct correlation between actual and predicted values, albeit with some deviation, reflecting the moderate performance metrics and underscoring the model's limitations with complex data structures.



**Artificial Neural Networks (ANN)**: The ANN model performed impressively with an **RMSE** of **1061.626258017832** and an **R-squared** of **-29.535477063768738**, highlighting its strength in modeling intricate dependencies and nonlinear interactions within the load data. The plot for the ANN model displays a tight clustering of data points around the line of perfect prediction, visually confirming the model's effectiveness in handling non-linear interactions and achieving a high degree of predictive accuracy.




## VIII. Conclusion:

The comparative analysis of different machine learning models and neural networks for **Short-term Load Forecasting** revealed varied strengths across the models. **The Random Forest and ANN models** stood out with superior predictive accuracy, highlighting their capability to handle the complex and nonlinear nature of load forecasting. The **Support Vector Machine (SVM)** also performed well, particularly in environments where non-linear data relationships prevail.

**Linear Regression**, while not as robust as the other models, serves as an essential benchmark and is useful for understanding basic trends. For utility companies and grid operators, incorporating a mix of these models could enhance forecasting accuracy and reliability.

Future studies could explore the integration of these models or the use of ensemble methods to further improve forecasting accuracy. Additionally, experimenting with larger datasets or incorporating more granular temporal and weather-related variables might also refine the models' predictive capabilities.

The findings of this study underscore the importance of selecting appropriate modeling techniques based on the specific characteristics of the dataset and the forecasting needs, ultimately aiding in the efficient management of energy resources and stability of power grids.

Each model's predictions are evaluated using two key metrics: **The Root Mean Squared Error (RMSE)** and **The R-squared value**, which are essential for assessing accuracy and fit. Visualization through scatter plots provides a direct comparison between actual and predicted values, further aiding in the interpretative analysis of each model's effectiveness.

This script exemplifies a structured approach to predictive modeling, demonstrating each step from data pre-processing to detailed evaluation of model outputs. The use of multiple models allows for a robust comparison and deeper insights into which techniques are most effective for **Short-Term Load Forecasting** based on the dataset used.