## DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

**College Code:** 4106

**Project Domain:** Applied Datascience

**Project Title :** Electricity Price Prediction model

**Project Mentor:**

**Team Members:**


1) Mohammed Ismail I - 410621104064

2) Syed Ashraf S - 410621104105

3) Sivaramakrishnan R - 410621104097

4) Shafi Ahamed S - 410621104092


## ABSTRACT:


"The electricity market is a complex and dynamic system influenced by numerous factors, including supply and demand dynamics, weather conditions, regulatory policies, and market participants' behaviors. Accurate electricity price prediction is essential for various stakeholders, such as utility companies, consumers, and energy traders, to make informed decisions, optimize resource allocation, and mitigate financial risks".

# Day-Ahead Electricity Price Prediction

Carter Bouley

# Project Goals

Wholesale Electricity prices change over the course of the day

**This provides opportunity to optimize consumption according to prices**

**Price Prediction**

- Electricity prices follow patterns over ***daily***, ***weekly*** and ***seasonal*** timeframes.
- Machine learning techniques can be utilized to ***predict*** these patterns effectively

***Few electricity providers give their customers access to wholesale prices.***

*Octopus energy has designed an 'Agile Tariff' which does exactly that.*

## Data Collection

**Electricity**

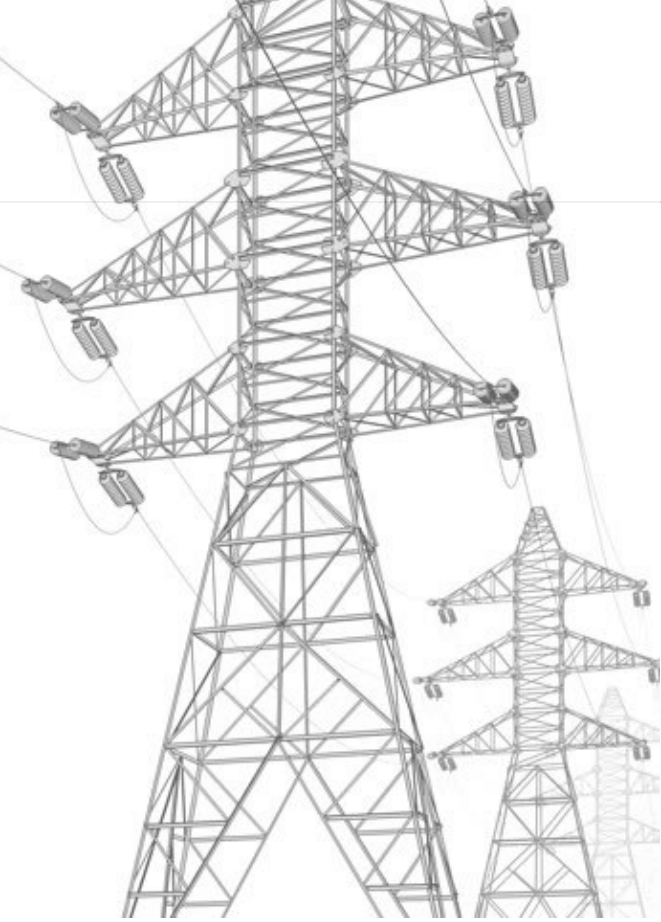
- Hourly electricity price data

**Commodity**

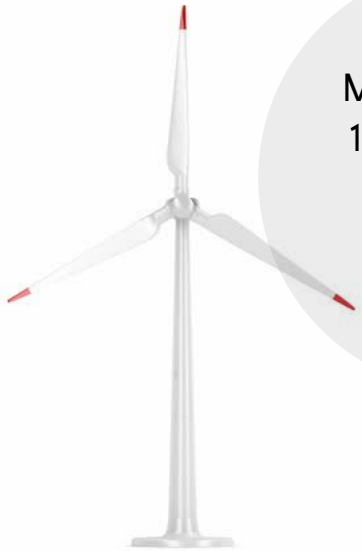
- Daily commodity prices

**Temperature**

- Hourly temperature data



## Data Shape



**Input**
Model can ‘see’ 168 hours into the past # of each of the variables

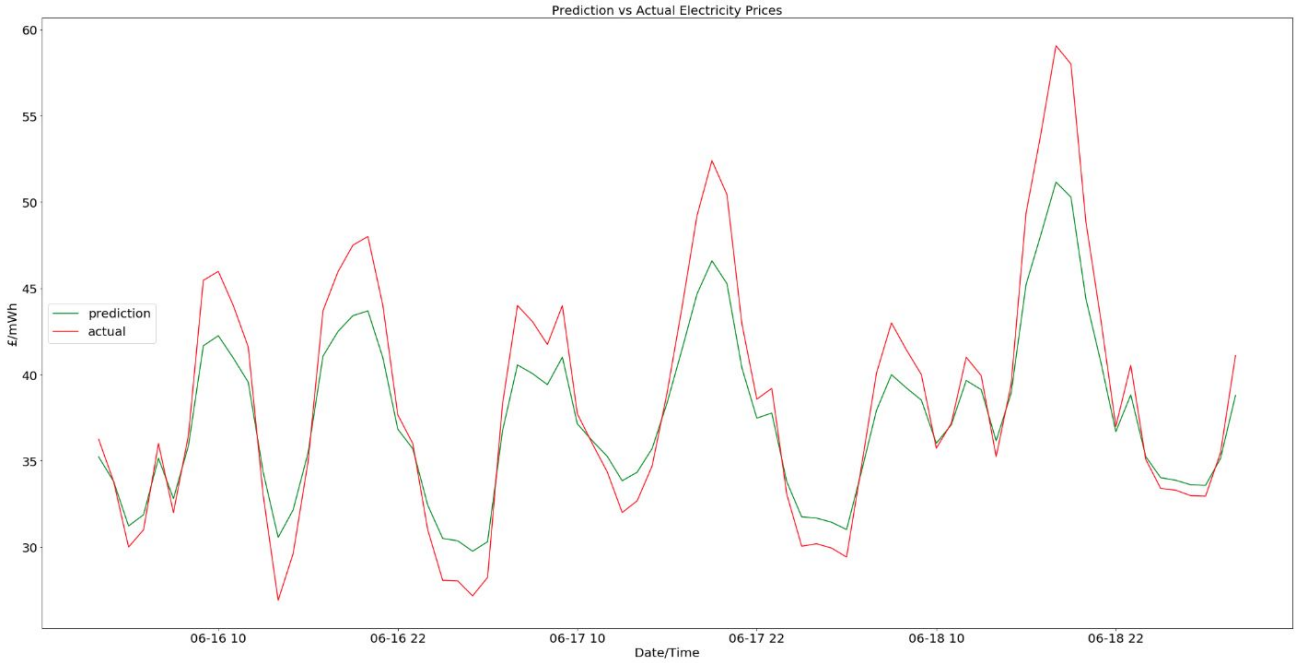
**Process**
Predicts hourly electricity price for 24 hours into the future

**Output**
Hour 24

# Machine Learning Methods

| | MAPE | Comparison Vs 'Dumb' |
|---|---|---|
| Arima | 14.43% | +20.09% |
| Pure Time Series Simple Neural Network | 18.85% | -8.77% |
| Multivariate Recurrent Neural Network | 8.13% | +46.91% |

# Results


Prediction vs Actual Electricity Prices
60
55
50
45
40
35
30
£/mWh
prediction
actual
06-16 10
06-16 22
06-17 10
06-17 22
06-18 10
06-18 22
Date/Time

# Why is this useful?

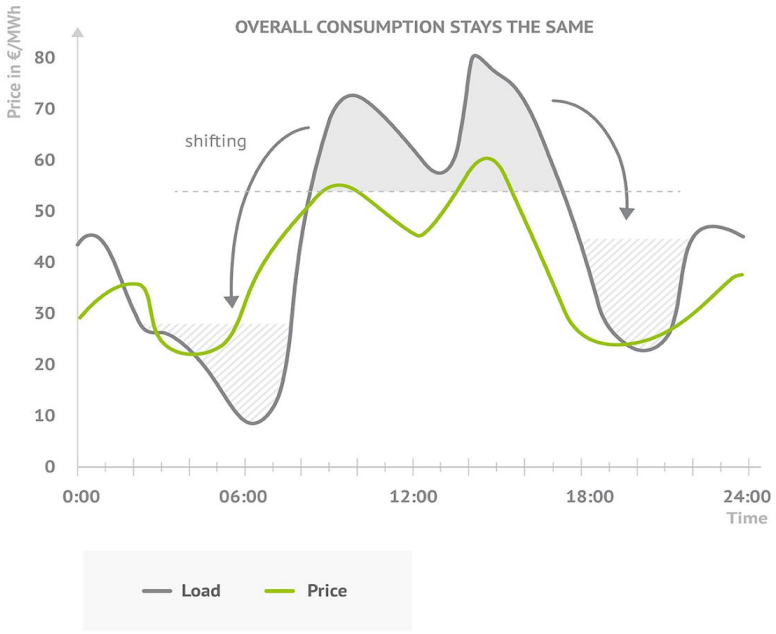
### Load Shifting

Optimizing building electricity ***demand*** can lead to both monetary and carbon emission savings without sacrificing overall consumption.

### Battery Simulation

A similar result can be achieved through storage. I have chosen to model this through a battery simulation.

Using the a popular home battery's parameters *(Tesla Powerwall 2)* I can attribute a value to the price prediction.



# DETAILED STEPS

- Download dataset from Kaggle using the below link: https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction
- Go through the dataset and filter as per needed.
- Extract the libraries that are needed to work with the dataset for electricity price prediction.
- Using **Pandas** library is the most helpful feature in python to handle with datasets.
- Using **Sklearn** library in python is a best library for prediction type machine learning models which are pre build in it.
- Split the data into train and test data so that the model uses certain data for training purposes and after training the model can be evaluated using the test data. This can be done through the **train_test_split()** function from **sklearn**
- To fit the model on the training data **model.fit()** function can be used**.**
- To make prediction on test data **model.test()** function can be used.
- After the model is trained, find it's accuracy using functions like **mean_squared_error**, accuracy or any other similar approaches.

I can guide you through the general steps to load and preprocess a dataset for stock price prediction, but I can't directly access external websites or download data. Here's a high-level overview of the process:

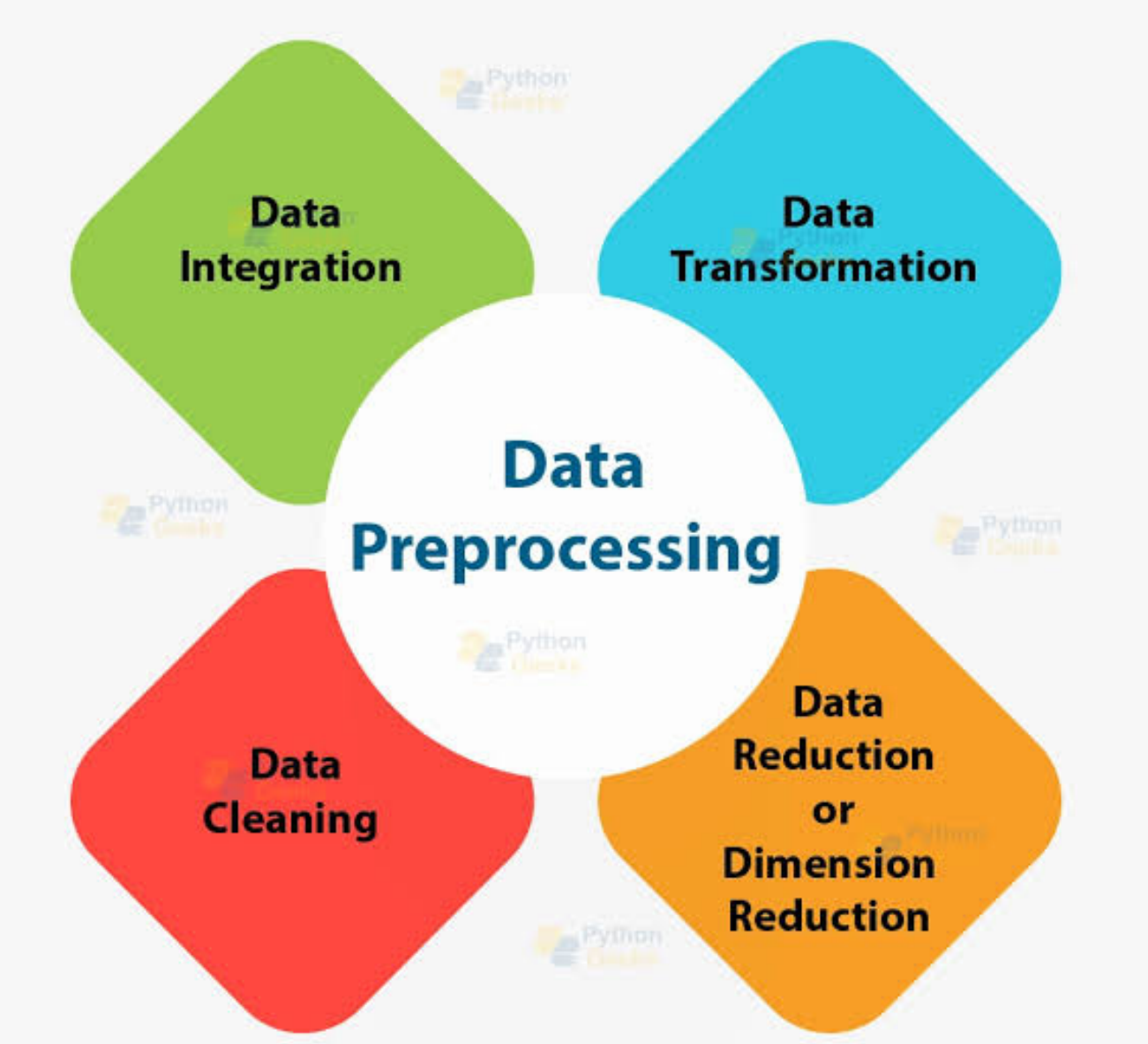
## 1. Data Collection:

- Download the dataset from Kaggle (https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset)

## 2. Data Inspection:

- Check the dataset for any missing values or anomalies.
- Examine the structure of the data to understand its features.

## 3. Data Preprocessing:

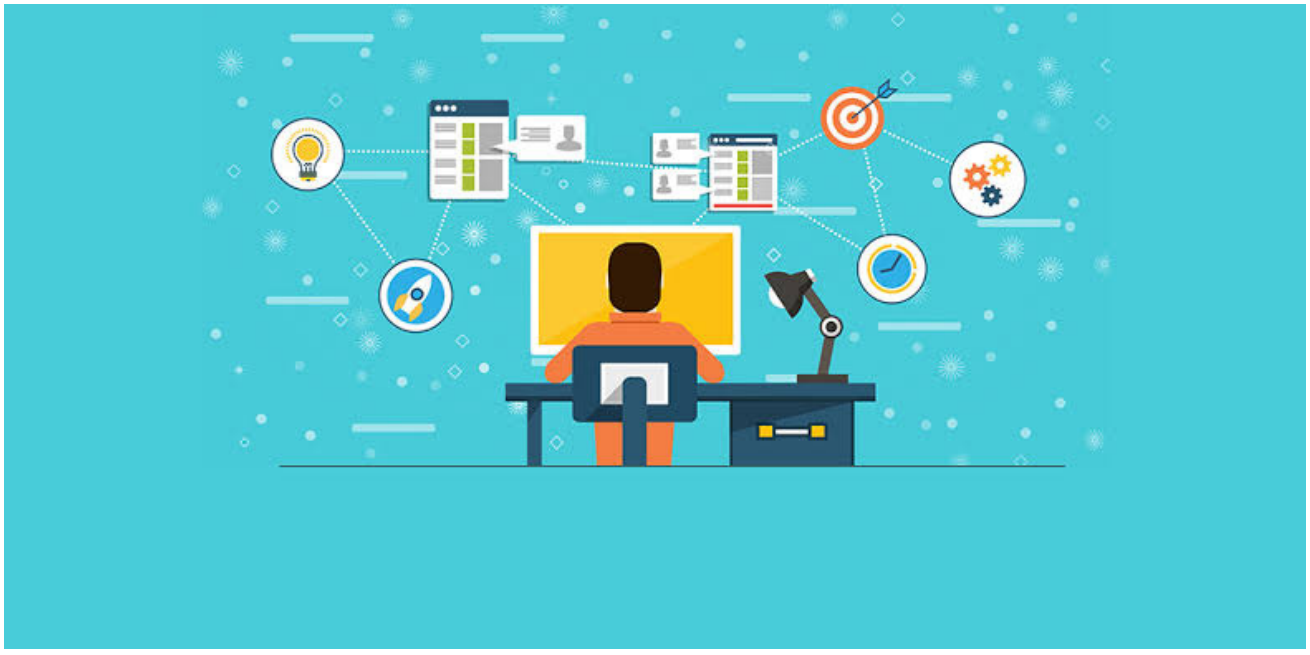


- Convert date columns to datetime objects.

- Sort the data by date in chronological order.

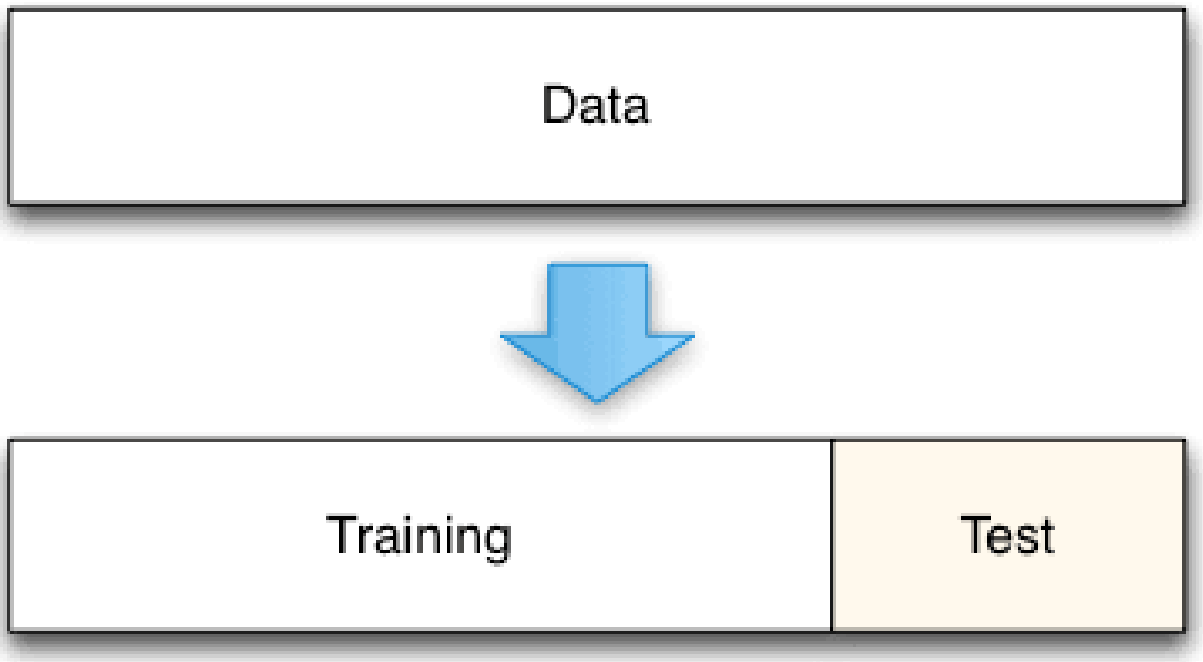
- Handle missing data, such as filling or removing missing values.

## 4. Feature Engineering:



- Create additional features that could be relevant for your prediction, like moving averages, technical indicators, or sentiment analysis scores.

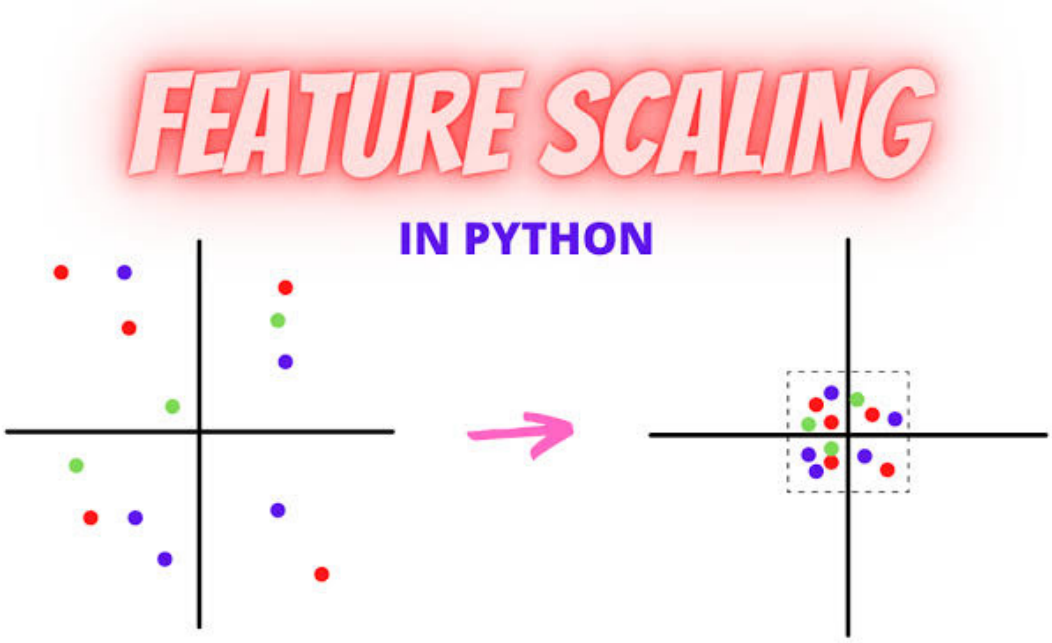
5. **Split Data:**




- Split the data into training, validation, and test sets for model

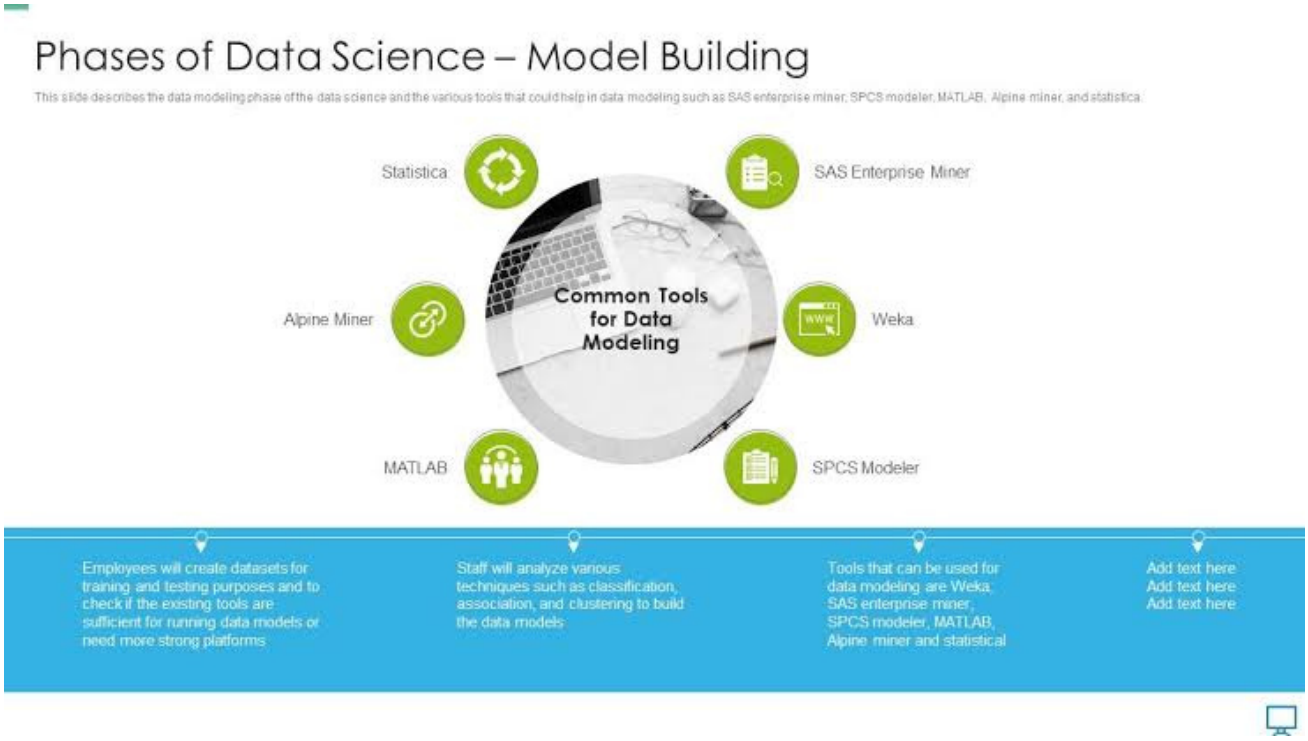
evaluation.

6. **Scaling:**



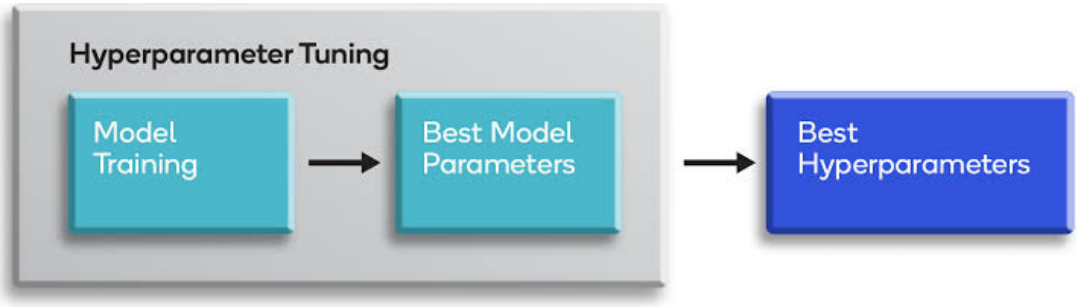

- Normalize or scale the numerical features if needed. This is often crucial for deep learning models.

# 7. **Model Building:**



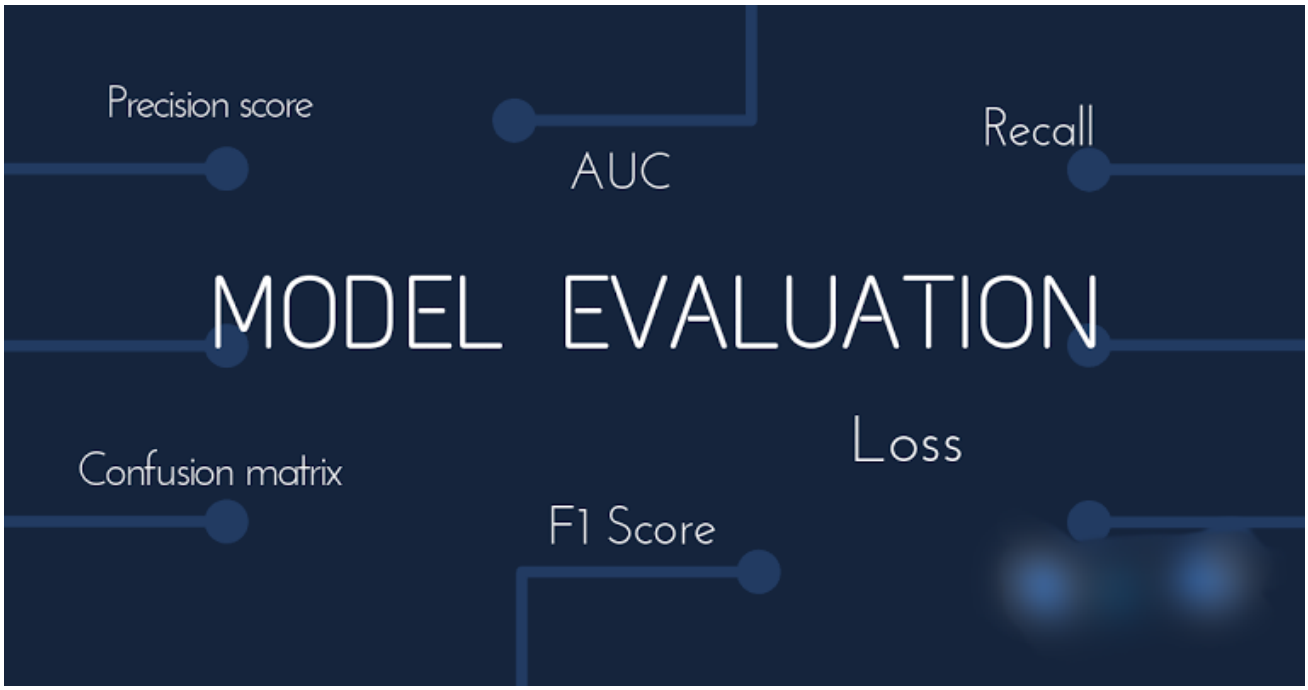

**- Start building your stock price prediction model, such as a time series forecasting model (e.g., LSTM, GRU) or other regression models (e.g., linear regression).**

# 8. **Model Training:**



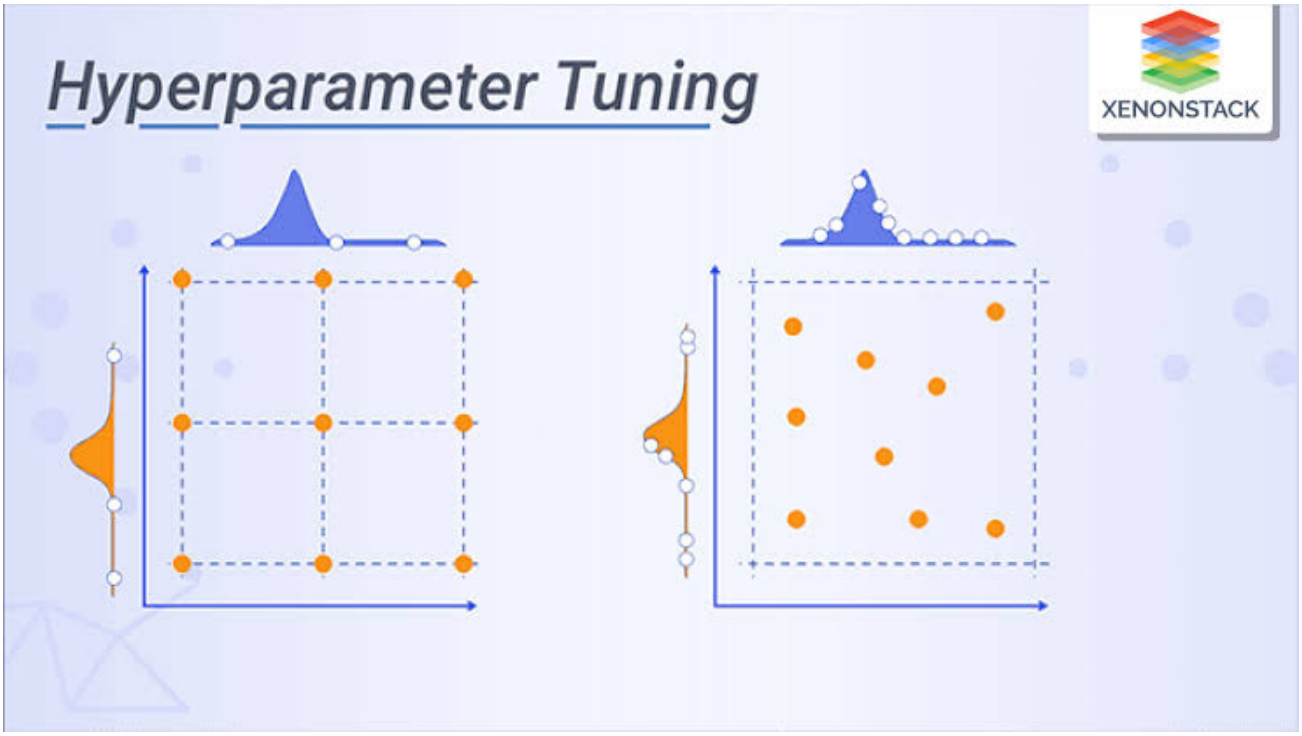

**- Train your model on the training data.**

## 9. **Model Evaluation:**



- Evaluate the model's performance on the validation set using appropriate metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or others.

10. **Hyperparameter Tuning:**



- Fine-tune your model by adjusting hyperparameters for better performance.

11. **Testing:**


- Assess the model's performance on the test set to see

how well it generalizes to unseen data.

12. **Deployment (if applicable):**


- If you plan to deploy the model, prepare it for production use.

# Electricity Price Prediction

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error

# Load the dataset
data = pd.read_csv("your_dataset.csv")

# Data Preprocessing
# Convert the 'daytime' column to datetime
data["daytime"] = pd.to_datetime(data["daytime"])

# Sort the data by date
data = data.sort_values(by="daytime")

# Check for missing values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)

# Handle missing data (you can choose to fill or remove them)
data = data.fillna(method="ffill")

# Split data into features (X) and the target variable (y)
X = data[
    [
        "Holiday",
        "Holiday flag",
        "day of the week",
        "week of year",
        "day",
        "month",
        "year",
        "period of day",
        "forecast wind production",
        "system load EA",
        "SMPEA",
        "ORKTemperature",
        "ORKwindspeed",
        "CO2INTENSITY",
        "actual wind production",
        "system load EP2",
        "SMPEP2",
    ]
]
y = data["Electricity_Price"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.2, random_state=42
)

# Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create an XGBoost model
model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

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

# Predict electricity prices
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# Now you have a basic electricity price prediction model using XGBoost.
```

```
df.describe()
```

| | HolidayFlag | DayOfWeek | WeekOfYear | Day | Month | Year | PeriodOfDay |
|---|---|---|---|---|---|---|---|
| count | 38014.000000 | 38014.000000 | 38014.000000 | 38014.000000 | 38014.000000 | 38014.000000 | 38014.000000 |
| mean | 0.040406 | 2.997317 | 28.124586 | 15.739412 | 6.904246 | 2012.383859 | 23.501105 |
| std | 0.196912 | 1.999959 | 15.587575 | 8.804247 | 3.573696 | 0.624956 | 13.853108 |
| min | 0.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 2011.000000 | 0.000000 |
| 25% | 0.000000 | 1.000000 | 15.000000 | 8.000000 | 4.000000 | 2012.000000 | 12.000000 |
| 50% | 0.000000 | 3.000000 | 29.000000 | 16.000000 | 7.000000 | 2012.000000 | 24.000000 |
| 75% | 0.000000 | 5.000000 | 43.000000 | 23.000000 | 10.000000 | 2013.000000 | 35.750000 |
| max | 1.000000 | 6.000000 | 52.000000 | 31.000000 | 12.000000 | 2013.000000 | 47.000000 |

MODEL BUILDING:

Building a model for a dataset involves a series of steps and considerations. Here's a general outline of the process:

- **Understand the Problem**
- **Data Collection**
- **Data Preprocessing**
- **Feature Engineering**
- **Model selection**
- **Model Training**
- **Model Evaluation**
- **Model Testing**
- **Model Interpretation**
- **Model Deployement**

# PROGRAM:

```
x=df[['HolidayFlag','DayOfWeek','WeekOfYear','Day','Month','Year']]
y=df['PeriodOfDay']
```

+ Code + Mark

```
x_train, x_test, y_train, y_test=train_test_split(x,y, test_size=0.2, random_state=42)
```

```
x_train
```

| | HolidayFlag | DayOfWeek | WeekOfYear | Day | Month | Year |
|---|---|---|---|---|---|---|
| 15238 | 0 | 3 | 37 | 13 | 9 | 2012 |
| 20071 | 0 | 6 | 51 | 23 | 12 | 2012 |
| 14654 | 0 | 5 | 35 | 1 | 9 | 2012 |
| 3964 | 0 | 6 | 3 | 22 | 1 | 2012 |
| 2855 | 0 | 4 | 52 | 30 | 12 | 2011 |
| ... | ... | ... | ... | ... | ... | ... |
| 16850 | 0 | 2 | 42 | 17 | 10 | 2012 |
| 6265 | 0 | 5 | 10 | 10 | 3 | 2012 |
| 11284 | 0 | 5 | 25 | 23 | 6 | 2012 |
| 860 | 0 | 4 | 46 | 18 | 11 | 2011 |
| 15795 | 0 | 1 | 39 | 25 | 9 | 2012 |

30411 rows × 6 columns

```
x_test
```

| | HolidayFlag | DayOfWeek | WeekOfYear | Day | Month | Year |
|---|---|---|---|---|---|---|
| 35833 | 0 | 5 | 46 | 16 | 11 | 2013 |
| 198 | 0 | 5 | 44 | 5 | 11 | 2011 |
| 36547 | 0 | 6 | 48 | 1 | 12 | 2013 |
| 26373 | 0 | 4 | 18 | 3 | 5 | 2013 |
| 21156 | 0 | 0 | 3 | 14 | 1 | 2013 |
| ... | ... | ... | ... | ... | ... | ... |
| 13927 | 0 | 4 | 33 | 17 | 8 | 2012 |
| 16926 | 0 | 3 | 42 | 18 | 10 | 2012 |
| 24520 | 0 | 0 | 13 | 25 | 3 | 2013 |
| 9059 | 1 | 0 | 19 | 7 | 5 | 2012 |
| 9786 | 0 | 1 | 21 | 22 | 5 | 2012 |

7603 rows × 6 columns

```
y_train
```

```
15238    24
20071     9
14654    16
3964     28
2855     23
         ..
16850     4
6265     25
11284     6
860      44
15795     5
Name: PeriodOfDay, Length: 30411, dtype: int64
```

```
y_test
```

```
35833    27
198       6
36547    21
26373    23
21156    38
         ..
13927     9
16926    32
24520    42
9059     37
9786     44
Name: PeriodOfDay, Length: 7603, dtype: int64
```

MODEL EVALUATION:

- Model evaluation is a critical step in the machine learning and data analysis process. It involves assessing how well a trained model performs on a given dataset. The goal of model evaluation is to determine the model's effectiveness, generalization capability, and suitability for a specific task. Here are some common techniques and metrics used for model evaluation.

  - **Splitting the Data**
  - **Training the Model**
  - **Model Evaluation Metrics**
  - **Classification Problems**

- Accuracy
- Precision, Recall, F1-Score
- ROC AUC
- Confusion Matrix

➢ **Regression Problems**

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R-squared (R2)

➢ **Error Analysis**

➢ **Deployment and Monitoring**

# PROGRAM:

```
from sklearn.model_selection import cross_val_score
num_folds = 5
def perform_cross_validation(model, X, y, num_folds):
    mse_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_mean_squared_error')
    rmse_scores = np.sqrt(mse_scores)
    mae_scores = -cross_val_score(model, X, y, cv=num_folds, scoring='neg_mean_absolute_error')
    r2_scores = cross_val_score(model, X, y, cv=num_folds, scoring='r2')

    return mse_scores, rmse_scores, mae_scores, r2_scores
```

##LINE REGRESSION##

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# Linear Regression
linear_model = LinearRegression()
linear_mse, linear_rmse, linear_mae, linear_r2 = perform_cross_validation(linear_model, x, y, num_folds)
print("Linear Regression:")
print(f"Average MSE: {np.mean(linear_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(linear_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(linear_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(linear_r2) * 100:.2f}%")
print("\n")
```

```
Linear Regression:
Average MSE: 816.63%
Average RMSE: 58.95%
Average MAE: 51.06%
Average R-squared: -0.01%
```

## ##RIDGE REGRESSION##

```
# Ridge Regression
ridge_model = Ridge(alpha=1.0)  # You can adjust alpha as needed
ridge_mse, ridge_rmse, ridge_mae, ridge_r2 = perform_cross_validation(ridge_model, x, y, num_folds)
print("Ridge Regression:")
print(f"Average MSE: {np.mean(ridge_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(ridge_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(ridge_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(ridge_r2) * 100:.2f}%")
print("\n")
```

```
Ridge Regression:
Average MSE: 816.63%
Average RMSE: 58.95%
Average MAE: 51.06%
Average R-squared: -0.01%
```

## ##LASSO REGRESSION##

```
# Lasso Regression
lasso_model = Lasso(alpha=1.0)  # You can adjust alpha as needed
lasso_mse, lasso_rmse, lasso_mae, lasso_r2 = perform_cross_validation(lasso_model, x, y, num_folds)
print("Lasso Regression:")
print(f"Average MSE: {np.mean(lasso_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(lasso_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(lasso_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(lasso_r2) * 100:.2f}%")
print("\n")
```

```
Lasso Regression:
Average MSE: 816.57%
Average RMSE: 58.95%
Average MAE: 51.06%
Average R-squared: -0.00%
```

## ##DECISION TREE##

```
from sklearn.tree import DecisionTreeRegressor

# Decision Trees
tree_model = DecisionTreeRegressor(max_depth=None, random_state=0)  # You can adjust parameters as needed
tree_mse, tree_rmse, tree_mae, tree_r2 = perform_cross_validation(tree_model, x, y, num_folds)
print("Decision Trees:")
print(f"Average MSE: {np.mean(tree_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(tree_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(tree_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(tree_r2) * 100:.2f}%")
print("\n")
```

```
Decision Trees:
Average MSE: 909.96%
Average RMSE: 62.04%
Average MAE: 53.06%
Average R-squared: -11.45%
```

##RANDOM FOREST##

```
from sklearn.ensemble import RandomForestRegressor

# Random Forest
forest_model = RandomForestRegressor(n_estimators=100, random_state=0)  # You can adjust parameters as needed
forest_mse, forest_rmse, forest_mae, forest_r2 = perform_cross_validation(forest_model, x, y, num_folds)
print("Random Forest:")
print(f"Average MSE: {np.mean(forest_mse) / np.mean(y) * 100:.2f}%")
print(f"Average RMSE: {np.mean(forest_rmse) / np.mean(y) * 100:.2f}%")
print(f"Average MAE: {np.mean(forest_mae) / np.mean(y) * 100:.2f}%")
print(f"Average R-squared: {np.mean(forest_r2) * 100:.2f}%")
```

```
Random Forest:
Average MSE: 834.38%
Average RMSE: 59.58%
Average MAE: 51.48%
Average R-squared: -2.18%
```

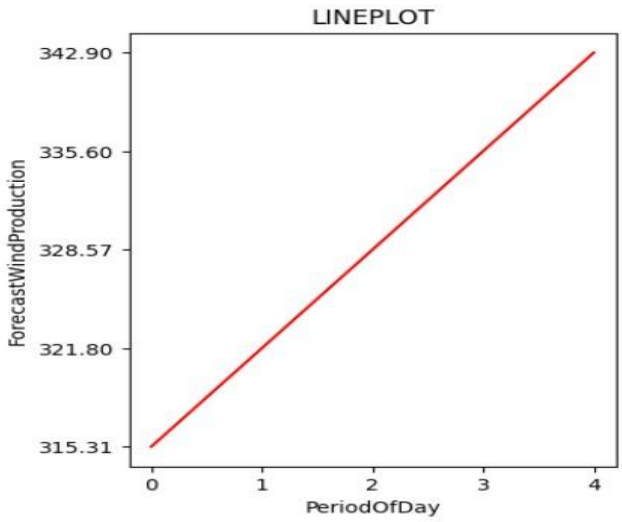
VISUALIZATION:

- Data visualization is a powerful way to represent and communicate information from data through visual elements like charts, graphs, and maps. Effective data visualization can make complex data more understandable and can help identify patterns, trends, and insights that might be hidden in raw data. Here are some key concepts and best practices for data visualization.

# PROGRAM:

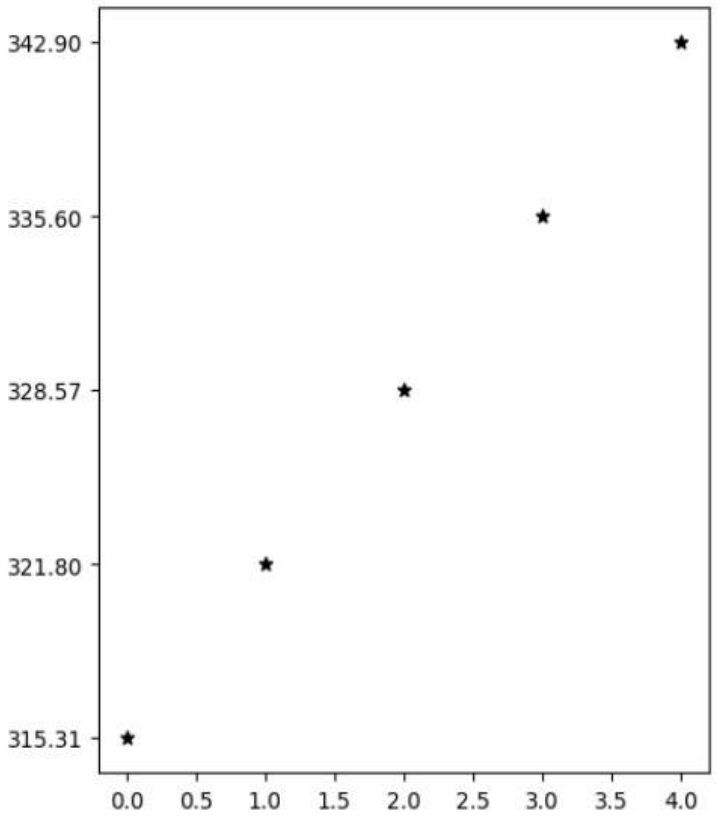
##LINE PLOT##

```
##Lineplot
a=df['PeriodOfDay'].head(5)
df1=df['ForecastWindProduction'].head(5)
fig = plt.figure(figsize =(4, 5))
plt.plot(a, df1,color='red')
plt.title("LINEPLOT")
plt.xlabel("PeriodOfDay")
plt.ylabel("ForecastWindProduction")
plt.show()
```

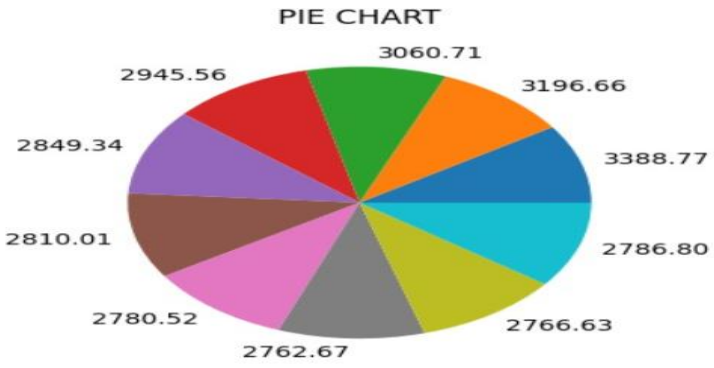


##SCATTER PLOT##

```
##Scatterplot
a=df['PeriodOfDay'].head()
df1=df['ForecastWindProduction'].head()
fig = plt.figure(figsize =(5, 6))
plt.scatter(a, df1,marker='*',color='black')
plt.show("SCATTERPLOT")
plt.show()
```

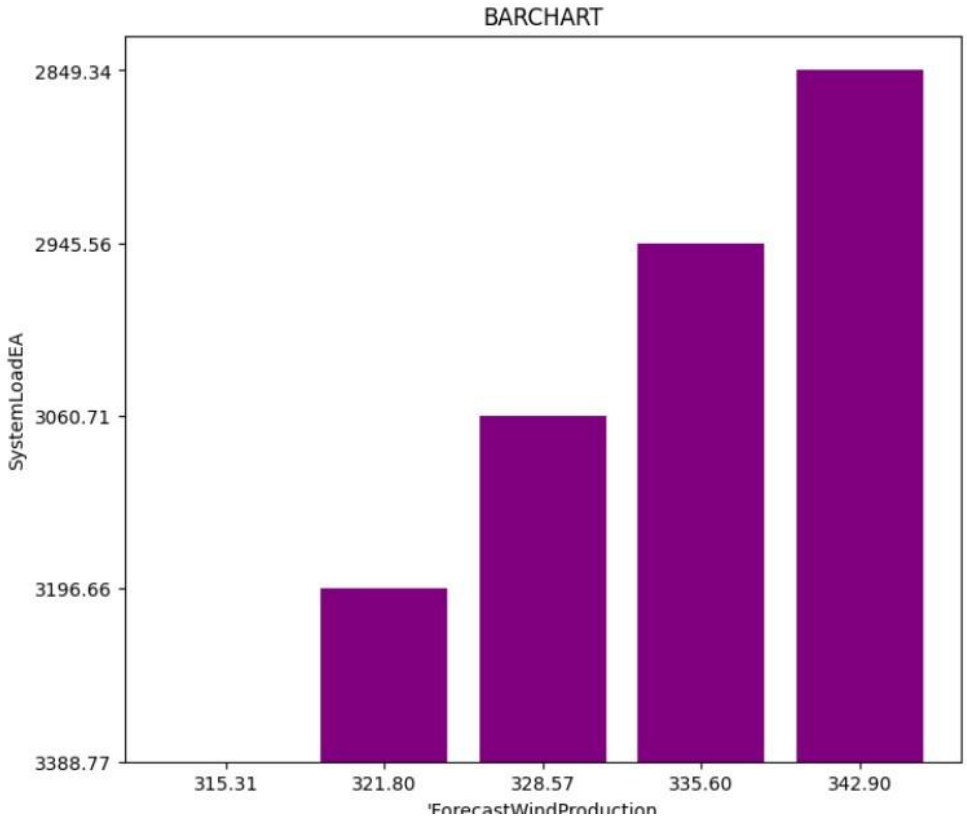



##PIE CHART##



##BAR CHART##

```
a=df['ForecastWindProduction'].head(5)
df1=df['SystemLoadEA'].head(5)
fig = plt.figure(figsize =(8, 7))
plt.bar(a, df1,color='purple')
plt.title("BARCHART")
plt.xlabel("'ForecastWindProduction")
plt.ylabel("SystemLoadEA")
plt.show()
```




## Conclusion:

**In this phase,The Model Building ,Model Evaluation and visualize the Dataset has been successfully verified and executed successfully.**