**AI\_Phase4**

**Development Part  - 2**

**Analyzing the energy consumption data Creating visualizations**

**Analyzing energy consumption data and creating visualizations can provide valuable insights and help in decision-making and resource management. Here are the steps you can follow to analyze energy consumption data and create visualizations:**

**1. Data Collection:**

**- Gather energy consumption data from reliable sources, such as smart meters, sensors, or utility bills. Ensure the data is well-structured and includes relevant information like date, time, location, and energy consumption values.**

**2. Data Cleaning and Preprocessing:**

**- Check for missing or erroneous data and handle them appropriately, either by imputing missing values or removing outliers.**

**- Convert data types, ensure consistency, and format dates and times correctly.**

**3. Data Exploration:**

**- Calculate basic statistics to understand the data's characteristics, such as mean, median, standard deviation, and percentiles.**

**- Create summary tables to get an overview of energy consumption patterns.**

**4. Time-Series Analysis (if applicable):**

**- If your data involves time series, perform time-series analysis, including trend analysis, seasonality decomposition, and autocorrelation to identify patterns.**

**5. Data Visualization:**

**- Use data visualization tools and libraries (e.g., Python with Matplotlib, Seaborn, or R with ggplot2) to create various types of charts and plots to represent the data. Some common visualizations for energy consumption data include:**

**- Line charts to show consumption trends over time.**

**- Bar charts for comparing consumption among different locations or periods.**

**- Heatmaps to display consumption patterns across time and locations.**

**- Box plots to identify outliers and distribution characteristics.**

**6. Geospatial Visualizations (if applicable):**

**- If you have location data, create maps or geospatial visualizations to display energy consumption across different geographic regions.**

**7. Interactive Dashboards (optional):**

**- Consider building interactive dashboards using tools like Tableau, Power BI, or Plotly to allow users to explore and analyze the data themselves.**

**8. Insights and Analysis:**

**- Analyze the visualizations to draw conclusions and gain insights into energy consumption patterns. Look for trends, seasonality, anomalies, and correlations.**

**9. Hypothesis Testing (optional):**

**- If you have specific questions or hypotheses, conduct statistical tests to validate your findings.**

**10. Reporting and Communication:**

**- Create a report or presentation summarizing your findings and insights. Make your visualizations and analysis easily understandable for stakeholders.**

**11. Future Recommendations:**

**- Based on your analysis, provide recommendations for energy conservation, optimization, or other relevant actions.**

**12. Data Updates and Monitoring:**

**- If your data is continuously collected, set up a process for regular updates, monitoring, and real-time visualization if needed.**

**Remember that the choice of visualization types and analysis techniques should be based on the specific goals and nature of your energy consumption data. The process may vary depending on the scale and complexity of the data and the tools available to you.**

**CODE :**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import warnings**

**​warnings.filterwarnings("ignore", category=UserWarning)**

**​from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.svm import SVR**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**​**

**RED = "\033[91m"**

**GREEN = "\033[92m"**

**YELLOW = "\033[93m"**

**BLUE = "\033[94m"**

**RESET = "\033[0m"**

**​df = pd.read\_csv("/kaggle/input/hourly-energy-consumption/AEP\_hourly.csv")**

**​df["Datetime"] = pd.to\_datetime(df["Datetime"])**

**​**

**# DATA CLEANING**

**print(BLUE + "\nDATA CLEANING" + RESET)**

**# --- Check for missing values**

**missing\_values = df.isnull().sum()**

**print(GREEN + "Missing Values : " + RESET)**

**print(missing\_values)**

**# --- Handle missing values**

**df.dropna(inplace=True)**

**# --- Check for duplicate values**

**duplicate\_values = df.duplicated().sum()**

**print(GREEN + "Duplicate Values : " + RESET)**

**print(duplicate\_values)**

**# --- Drop duplicate values**

**df.drop\_duplicates(inplace=True)**

**​**

**# DATA ANALYSIS**

**print(BLUE + "\nDATA ANALYSIS" + RESET)**

**# --- Summary Statistics**

**summary\_stats = df.describe()**

**print(GREEN + "Summary Statistics : " + RESET)**

**print(summary\_stats)**

**​**

**# SUPPORT VECTOR MODELLLING**

**print(BLUE + "\nMODELLING" + RESET)**

**# Reduce the dataset size for faster training**

**df = df.sample(frac=0.2, random\_state=42)**

**# Split the data into features (Datetime) and target (AEP\_MW)**

**X = df[["Datetime"]]**

**y = df["AEP\_MW"]**

**# 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**

**)**

**# Preprocess the features (Datetime) to extract the day of the year**

**X\_train["DayOfYear"] = X\_train["Datetime"].dt.dayofyear**

**X\_test["DayOfYear"] = X\_test["Datetime"].dt.dayofyear**

**# Convert X\_train and X\_test to NumPy arrays**

**X\_train = X\_train["DayOfYear"].values.reshape(-1, 1)**

**X\_test = X\_test["DayOfYear"].values.reshape(-1, 1)**

**# Standardize the data**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**# Create an SVR (Support Vector Regression) model with a linear kernel**

**svr = SVR(kernel="linear", C=1.0)**

**# Train the SVR model**

**svr.fit(X\_train\_scaled, y\_train)**

**# Predict on the test set**

**y\_pred = svr.predict(X\_test\_scaled)**

**# Evaluate the model**

**mse = mean\_squared\_error(y\_test, y\_pred)**

**r2 = r2\_score(y\_test, y\_pred)**

**print(f"Mean Squared Error: {mse}")**

**print(f"R-squared: {r2}")**

**# Plot the actual vs. predicted values**

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

**plt.scatter(X\_test, y\_test, color="b", label="Actual")**

**plt.scatter(X\_test, y\_pred, color="r", label="Predicted")**

**plt.xlabel("Day of the Year")**

**plt.ylabel("Energy Consumption (MW)")**

**plt.title("SVR Model: Actual vs. Predicted")**

**plt.legend()**

**plt.grid()**

**plt.show()**

**​**

**# DATA VISUALIZATION**

**print(BLUE + "\nDATA VISUALIZATION" + RESET)**

**# --- Line plot**

**print(GREEN + "LinePlot : " + RESET)**

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

**sns.lineplot(data=df, x="Datetime", y="AEP\_MW")**

**plt.xlabel("Datetime")**

**plt.ylabel("Energy Consumption (MW)")**

**plt.title("Energy Consumption Over Year")**

**plt.grid()**

**plt.show()**

**# --- Histogram**

**print(GREEN + "Histogram : " + RESET)**

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

**plt.hist(**

**df["AEP\_MW"],**

**bins=100,**

**histtype="barstacked",**

**edgecolor="white",**

**)**

**plt.xlabel("AEPMW")**

**plt.ylabel("Frequency")**

**plt.title("Histogram of MEGAWATT USAGE")**

**plt.show()**

**​**

**# SAVING THE FILE**

**df.to\_csv("/kaggle/working/cleaned\_AEP\_hourly.csv", index=False)**

**print(BLUE + "\nDATA ANALYSIS" + RESET)**

**print(GREEN + "Data Cleaned and Saved !" + RESET)**

**DATA CLEANING**

**Missing Values :**

**Datetime 0**

**AEP\_MW 0**

**dtype: int64**

**Duplicate Values :**

**0**

**DATA ANALYSIS**

**Summary Statistics :**

Datetime AEP\_MW

count 121273 121273.000000

mean 2011-09-02 03:17:01.553025024 15499.513717

min 2004-10-01 01:00:00 9581.000000

25% 2008-03-17 15:00:00 13630.000000

50% 2011-09-02 04:00:00 15310.000000

75% 2015-02-16 17:00:00 17200.000000

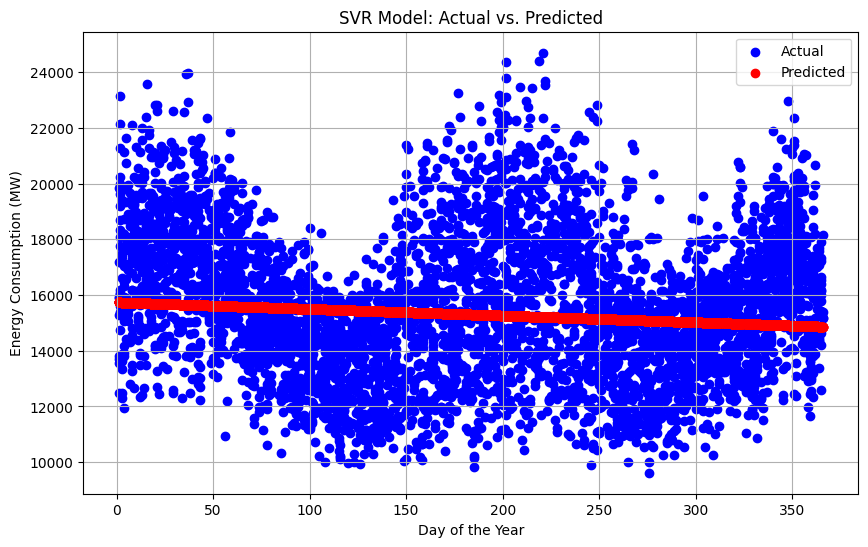
max 2018-08-03 00:00:00 25695.000000

std NaN 2591.399065

**MODELLING**

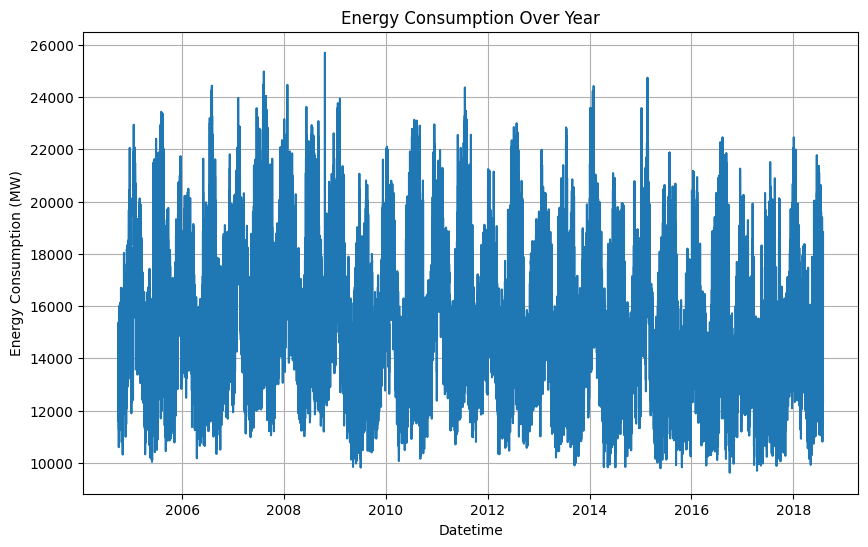
Mean Squared Error: 6758395.805638685

R-squared: 0.00270160624748228

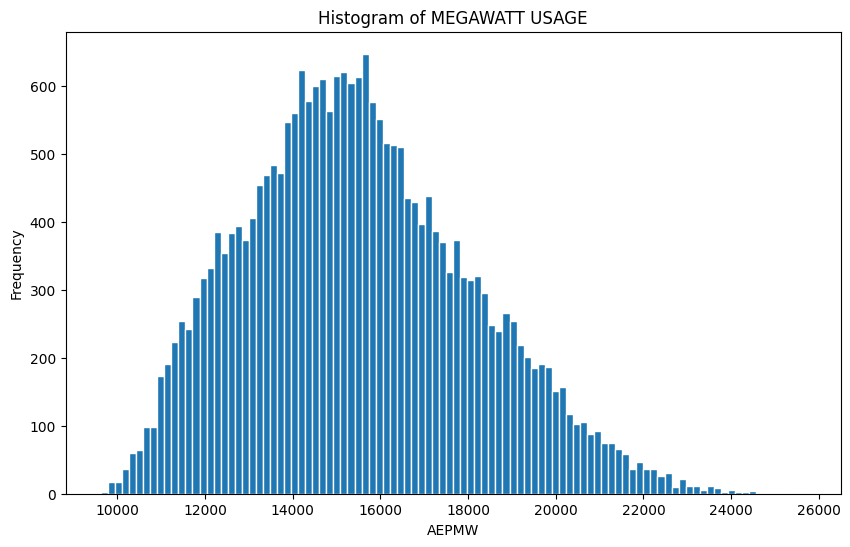


**DATA VISUALIZATION**

***LinePlot :***

******

***Histogram :***

******