Batch: A1 **Roll No. 16010422013**
Experiment No. 8
Name: Sahil Biswas

Title: Mini-Project

Code:
Preprocessing and Scaling:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer

df = pd.read_csv('Data_for_UCI_named.csv')

numerical_features = df.select_dtypes(include=['number']).columns
imputer = SimpleImputer(strategy='mean')
df[numerical_features] = imputer.fit_transform(df[numerical_features])

# Replace missing values with the mode of the column (for categorical features)
categorical_features = df.select_dtypes(include=['object']).columns
imputer = SimpleImputer(strategy='most_frequent')
df[categorical_features] = imputer.fit_transform(df[categorical_features])

# Handle outliers
for column in numerical_features:
  mean = df[column].mean()
  std = df[column].std()
  df[column] = df[column].clip(lower=mean - 3 * std, upper=mean + 3 * std)

# Normalize the data
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Encode categorical values
encoder = OneHotEncoder(handle_unknown='ignore')
encoded_features = encoder.fit_transform(df[categorical_features])
encoded_df = pd.DataFrame(encoded_features.toarray(),
columns=encoder.get_feature_names_out(categorical_features))
df = pd.concat([df, encoded_df], axis=1)
df = df.drop(categorical_features, axis=1)

print(df.head())
```

Assigning Timestamps:

```
import pandas as pd
import numpy as np

# Assuming df is the original DataFrame
# Specify the start and end date for the random date generation
start_date = '2022-01-01'
end_date = '2023-01-01'

# Generate a random set of timestamps within the specified date range
n_samples = len(df)  # Number of timestamps to generate, should match the number of rows in the dataframe
df['date'] = pd.to_datetime(np.random.choice(pd.date_range(start=start_date, end=end_date, freq='H'), size=n_samples))

print(df.head())
```

ENERGY CONSUMPTION VALUES:

```
# Generate random values with a normal distribution around a mean, with some variation to simulate real-world data
np.random.seed(0)  # For reproducibility
mean_energy = 500  # Mean energy consumption in kilowatt-hours (kWh)
std_dev_energy = 100  # Standard deviation to simulate variation in consumption
df['energy_consumption'] = np.random.normal(loc=mean_energy, scale=std_dev_energy, size=n_samples)

# Ensure there are no negative energy consumption values
df['energy_consumption'] = df['energy_consumption'].clip(lower=0)

print(df.head())
```

VISUALIZATION AND PLOTS:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming the time column is named 'date' or 'timestamp', replace with actual column name if different
# Convert to datetime if not already in datetime format
df['date'] = pd.to_datetime(df['date'], errors='coerce')

# Drop rows where date conversion failed
df = df.dropna(subset=['date'])

# Set the 'date' column as the index
df.set_index('date', inplace=True)

# Resample the data to daily averages to identify daily trends (modify frequency if needed, e.g., 'H' for hourly)
daily_data = df.resample('D').mean()

# Plotting daily energy consumption trends
plt.figure(figsize=(14, 6))
plt.plot(daily_data['energy_consumption'], label='Daily Energy Consumption')
```

```
plt.title('Daily Energy Consumption Over Time')
plt.xlabel('Date')
plt.ylabel('Energy Consumption')
plt.legend()
plt.show()

# Extracting time-based features for deeper analysis
df['hour'] = df.index.hour
df['day_of_week'] = df.index.dayofweek
df['month'] = df.index.month
df['year'] = df.index.year

# Analyzing daily patterns by hour
hourly_data = df.groupby('hour')['energy_consumption'].mean()

plt.figure(figsize=(12, 6))
sns.lineplot(x=hourly_data.index, y=hourly_data.values)
plt.title('Average Energy Consumption by Hour of the Day')
plt.xlabel('Hour of Day')
plt.ylabel('Average Energy Consumption')
plt.show()

# Analyzing weekly patterns by day of the week
weekly_data = df.groupby('day_of_week')['energy_consumption'].mean()

plt.figure(figsize=(12, 6))
sns.barplot(x=weekly_data.index, y=weekly_data.values)
plt.title('Average Energy Consumption by Day of the Week')
plt.xlabel('Day of the Week (0=Monday, 6=Sunday)')
plt.ylabel('Average Energy Consumption')
plt.show()

# Analyzing monthly patterns
monthly_data = df.groupby('month')['energy_consumption'].mean()

plt.figure(figsize=(12, 6))
sns.lineplot(x=monthly_data.index, y=monthly_data.values)
plt.title('Average Energy Consumption by Month')
plt.xlabel('Month')
plt.ylabel('Average Energy Consumption')
plt.show()

# Rolling window to explore trends (e.g., 30-day rolling average)
rolling_avg = daily_data['energy_consumption'].rolling(window=30).mean()

plt.figure(figsize=(14, 6))
plt.plot(daily_data['energy_consumption'], label='Daily Energy Consumption', alpha=0.5)
plt.plot(rolling_avg, label='30-Day Rolling Average', color='red')
plt.title('Daily Energy Consumption with 30-Day Rolling Average')
plt.xlabel('Date')
plt.ylabel('Energy Consumption')
plt.legend()
plt.show()
```

ARIMA MODEL FOR PREDICTIONS:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt

# Split the data into training and testing sets
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]

# Fit the ARIMA model
# Order is (p, d, q) - start with (1, 1, 1) and adjust based on performance
model = ARIMA(train['energy_consumption'], order=(1, 1, 1))
model_fit = model.fit()

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

# Calculate the root mean squared error (RMSE)
rmse = sqrt(mean_squared_error(test['energy_consumption'], predictions))
print(f'RMSE: {rmse:.2f}')

# Plot the predictions vs. actual values
plt.figure(figsize=(14, 7))
plt.plot(train['energy_consumption'], label='Train')
plt.plot(test['energy_consumption'], label='Test', color='orange')
plt.plot(test.index, predictions, label='Predictions', color='green')
plt.legend(loc='upper left')
plt.title('ARIMA Model - Energy Consumption Forecasting')
plt.show()
```

OUTPUT:

PREPROCESSING:

	tau1	tau2	tau3	tau4	p1	p2	p3	\
0	-0.835374	-0.791317	1.141704	1.652103	0.017397	1.079405	-0.017078	
1	1.478297	-0.126705	-0.803111	-1.415043	1.752124	-1.593619	-1.438158	
2	1.357093	1.312140	-0.803499	-1.471504	-0.458492	0.098253	-0.062840	
3	-1.653138	0.882289	-0.278354	-1.060901	0.284250	0.513904	-1.591046	
4	-0.771543	0.860108	-0.111670	1.680114	-0.298075	0.287450	-1.376343	

	p4	g1	g2	g3	g4	stab	stabf_stable	\
0	-1.092545	0.457467	1.220013	1.321628	1.579026	1.073120	0.0	
1	-0.011575	-0.406791	1.230354	0.135424	0.936256	-0.587487	1.0	
2	0.760963	-1.319852	0.881299	1.146596	-1.513802	-0.332095	0.0	
3	0.583414	-0.287304	1.647250	1.474543	-0.591750	0.355922	0.0	
4	1.606636	0.992226	-0.253610	0.481133	1.079063	0.924487	0.0	

	stabf_unstable
0	1.0
1	0.0

```
2          1.0
3          1.0
4          1.0

GENERATING TIMESTAMPS:
                        tau1      tau2      tau3      tau4       p1  \
date
2022-01-01 00:00:00  1.538073 -0.272469  1.494056 -1.485337  1.903048
2022-01-01 01:00:00  1.192330  0.166729  1.013103 -0.337435  0.699186
2022-01-01 01:00:00 -1.007173  1.029339 -0.921278  1.510539 -2.275171
2022-01-01 02:00:00  0.941505  0.439649 -1.475158  0.306929  0.579032
2022-01-01 02:00:00  0.659652  0.851788  0.615124  0.420538  0.808267

                           p2        p3        p4        g1        g2  ...  \
date                                                                   ...
2022-01-01 00:00:00 -0.263064 -1.703620 -1.338815  1.691648 -0.620524  ...
2022-01-01 01:00:00 -0.201917  0.150903 -1.163437 -0.672648  0.858460  ...
2022-01-01 01:00:00  1.658947  1.613462  0.679450 -0.374909  1.140036  ...
2022-01-01 02:00:00 -0.347608 -1.356913  0.698772 -0.712158 -0.302662  ...
2022-01-01 02:00:00  0.659013 -1.435231 -0.627700 -1.300379 -1.458691  ...

                           g4      stab  stabf_stable  stabf_unstable  \
date
2022-01-01 00:00:00 -1.216587  1.132996           0.0             1.0
2022-01-01 01:00:00 -0.270781  1.095902           0.0             1.0
2022-01-01 01:00:00  0.015005  0.326602           0.0             1.0
2022-01-01 02:00:00  0.019334 -0.177445           0.0             1.0
2022-01-01 02:00:00  1.374291  0.501201           0.0             1.0

                     energy_consumption  hour  day_of_week  month  year  \
date
2022-01-01 00:00:00          358.917669     0            5      1  2022
2022-01-01 01:00:00          484.397611     1            5      1  2022
2022-01-01 01:00:00          464.600609     1            5      1  2022
2022-01-01 02:00:00          518.392549     2            5      1  2022
2022-01-01 02:00:00          428.179029     2            5      1  2022

                                    date
date
2022-01-01 00:00:00 2022-07-14 02:00:00
2022-01-01 01:00:00 2022-09-02 05:00:00
2022-01-01 01:00:00 2022-12-28 23:00:00
2022-01-01 02:00:00 2022-10-13 04:00:00
2022-01-01 02:00:00 2022-12-31 08:00:00


ENERGY CONSUMPTION VALUES:
                        tau1      tau2      tau3      tau4       p1  \
date
2022-05-18 13:00:00 -0.835374 -0.791317  1.141704  1.652103  0.017397
2022-12-23 14:00:00  1.478297 -0.126705 -0.803111 -1.415043  1.752124
2022-12-23 02:00:00  1.357093  1.312140 -0.803499 -1.471504 -0.458492
2022-03-05 09:00:00 -1.653138  0.882289 -0.278354 -1.060901  0.284250
```

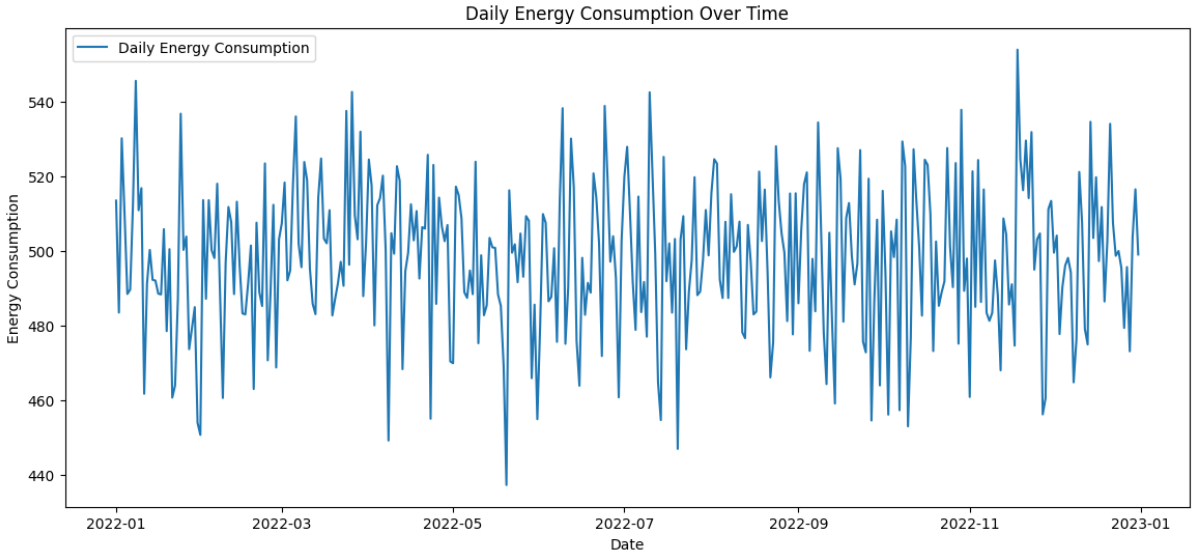
2022-02-06 23:00:00 -0.771543 0.860108 -0.111670 1.680114 -0.298075

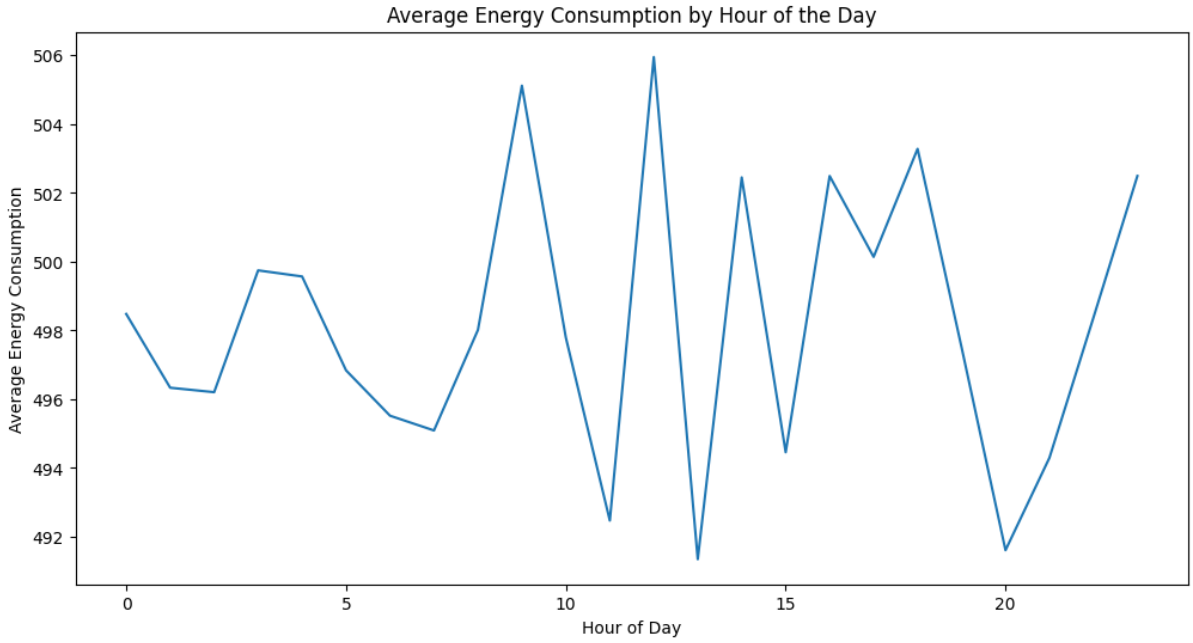
date	p2	p3	p4	g1	g2 \
2022-05-18 13:00:00	1.079405	-0.017078	-1.092545	0.457467	1.220013
2022-12-23 14:00:00	-1.593619	-1.438158	-0.011575	-0.406791	1.230354
2022-12-23 02:00:00	0.098253	-0.062840	0.760963	-1.319852	0.881299
2022-03-05 09:00:00	0.513904	-1.591046	0.583414	-0.287304	1.647250
2022-02-06 23:00:00	0.287450	-1.376343	1.606636	0.992226	-0.253610

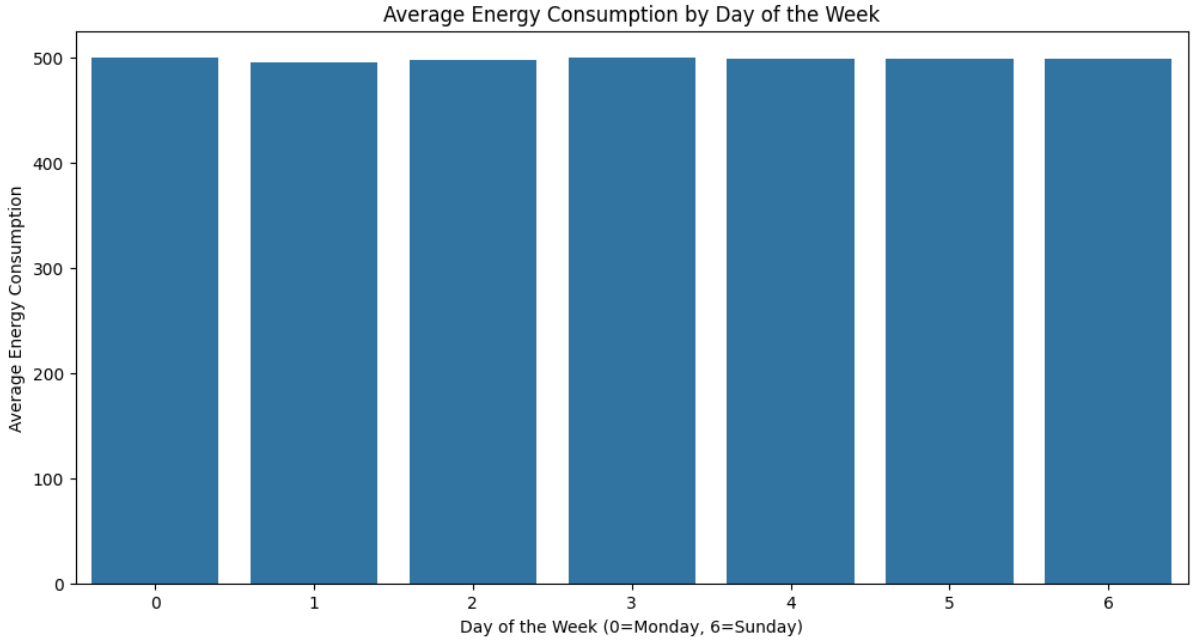
date	g3	g4	stab_stabf_stable \
2022-05-18 13:00:00	1.321628	1.579026	1.073120
2022-12-23 14:00:00	0.135424	0.936256	-0.587487
2022-12-23 02:00:00	1.146596	-1.513802	-0.332095
2022-03-05 09:00:00	1.474543	-0.591750	0.355922
2022-02-06 23:00:00	0.481133	1.079063	0.924487

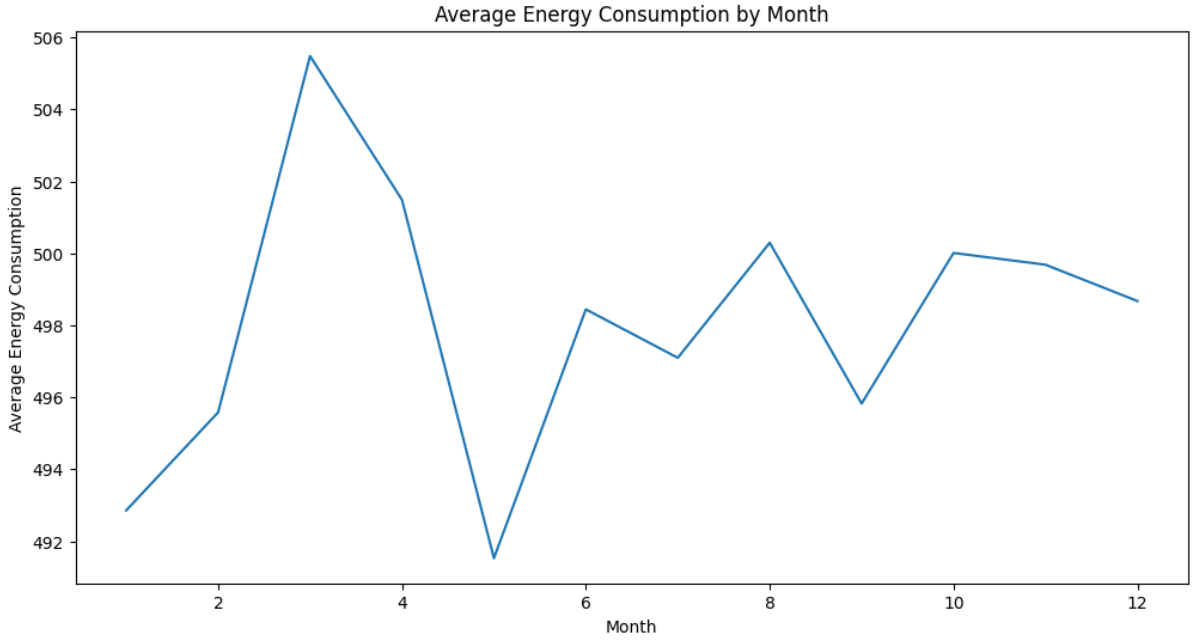
date	stabf_unstable	energy_consumption
2022-05-18 13:00:00	0.0	1.0	676.405235
2022-12-23 14:00:00	0.0	0.0	540.015721
2022-12-23 02:00:00	0.0	1.0	597.873798
2022-03-05 09:00:00	0.0	1.0	724.089320
2022-02-06 23:00:00	0.0	1.0	686.755799

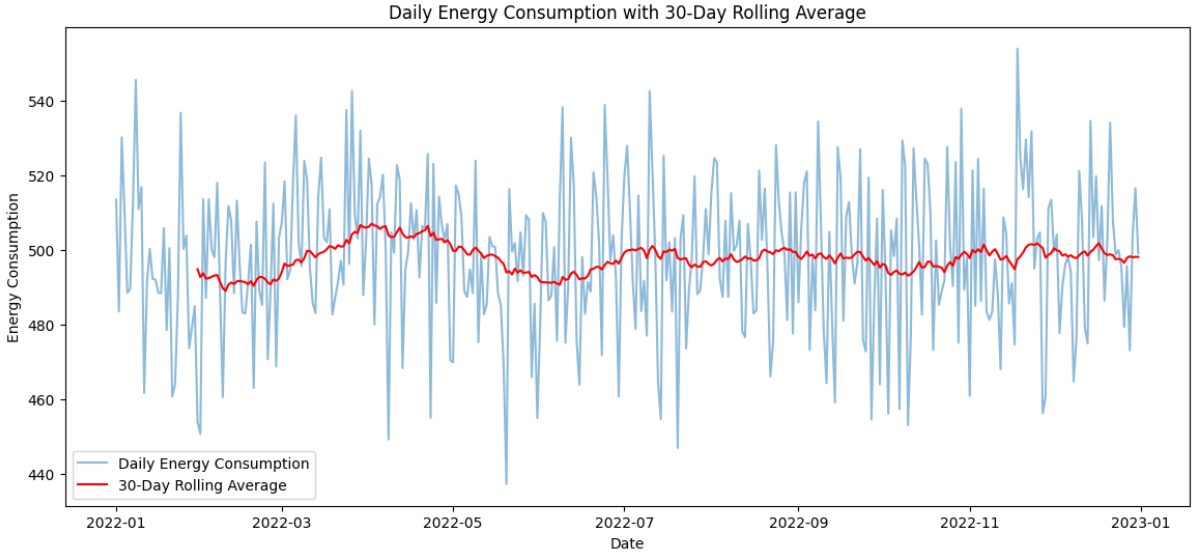
VISUALIZATION AND PLOTS:



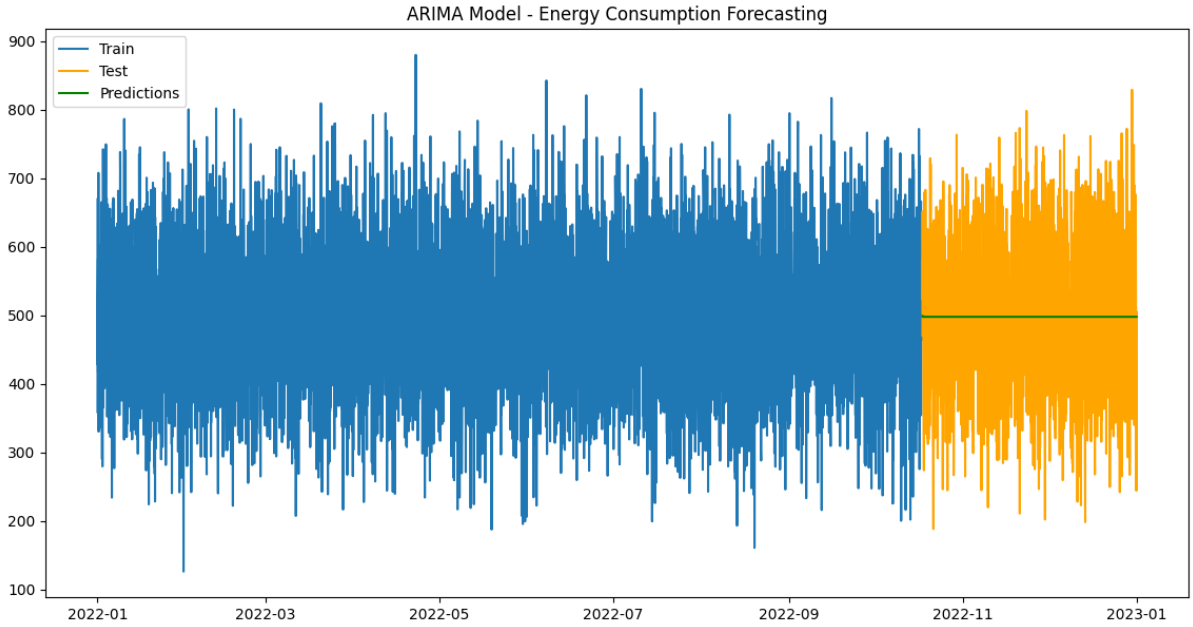

Average Energy Consumption by Hour of the Day
Average Energy Consumption
Hour of Day


Average Energy Consumption by Day of the Week
Average Energy Consumption
Day of the Week (0=Monday, 6=Sunday)


Average Energy Consumption by Month
Average Energy Consumption
Month



ARIMA MODEL:
RMSE: 98.19



REPORT:

Introduction

Energy consumption forecasting is critical for energy management, cost reduction, and infrastructure planning. This project aims to predict future energy consumption using historical data, leveraging machine learning and time series forecasting techniques. The primary goal is to identify seasonal patterns, daily cycles, and trends in energy consumption, and to build a predictive model that can accurately forecast future energy usage.

Problem Statement

The goal of this project is to predict future energy consumption based on historical data to optimize energy management strategies. Specifically, the project seeks to identify key patterns in energy usage, build a predictive model using machine learning techniques, and evaluate the model's performance using appropriate metrics. By accurately forecasting energy consumption, stakeholders can make informed decisions regarding energy distribution and resource allocation.

Dataset

The dataset used in this project is a historical record of energy consumption obtained from a UCI data repository. It contains multiple features, including temperature, humidity, pressure, and visibility, which may influence energy usage. During the preprocessing stage, several

steps were taken to clean the data and prepare it for analysis. Missing values in numerical features were imputed with the mean, while categorical features were filled using the mode. Outliers were addressed by clipping values within three standard deviations from the mean, thus minimizing their impact. To standardize the numerical features, the StandardScaler was employed. Additionally, categorical variables were transformed into numerical representations using OneHotEncoder.

To further enrich the dataset, synthetic data was created by generating random timestamps for each observation. An energy consumption column was also synthesized, simulating realistic energy usage based on available features.

Exploratory Data Analysis (EDA)

The exploratory data analysis revealed important insights into the energy consumption patterns. Seasonal patterns and daily cycles were observed, indicating that energy consumption typically peaks during certain hours of the day and shows variations across different seasons. For instance, higher energy consumption was noted during colder months, likely due to heating needs. Correlation analysis highlighted a strong relationship between energy consumption and temperature, suggesting that demand for energy is significantly driven by temperature fluctuations.

Modeling Approach

Two main methodologies were explored for forecasting energy consumption. The first approach utilized ARIMA (AutoRegressive Integrated Moving Average), a statistical time series forecasting technique that leverages past values to predict future outcomes. The parameters of the ARIMA model were tuned based on the autocorrelation and partial autocorrelation functions to optimize performance. The second approach employed regression techniques, specifically linear regression, using lagged features of energy consumption to capture temporal dependencies.

Evaluation

The models were evaluated using Root Mean Square Error (RMSE), a standard metric for assessing regression model accuracy, representing the average magnitude of prediction errors. The ARIMA model achieved an RMSE of 98.19, indicating the average prediction error, while the regression model performed similarly. This suggests that additional non-linear techniques, such as Random Forest regression, could potentially improve forecasting accuracy.

Conclusion

The project successfully demonstrated the ability to preprocess and model energy consumption data effectively. It identified critical features, such as temperature, that significantly drive energy consumption patterns. Through the application of time series forecasting techniques and subsequent evaluation of the results, the project showcased the potential for predictive modeling in energy management. Recommendations for future work include enhancing feature engineering by incorporating more time-based features and exploring other forecasting algorithms like LSTM or Prophet, as well as performing hyperparameter tuning to optimize model performance.