Statistical Approach for Smart Home Energy Management System

Team Members:

Vivek Choudhry
Vanita
Anusha
Bhuvanesh

1. Data Collection and Preprocessing

Sources of Data:

- Energy meters, IoT sensors, weather APIs, and smart devices.

Key Variables:

- Energy consumption (kWh).
- Device usage patterns.
- Time of day/week (peak vs. non-peak).
- Environmental conditions (temperature, humidity).
- Energy tariffs (time-of-use pricing).

Techniques:

- **Data Cleaning:** Remove outliers and handle missing data.
- **Feature Engineering:** Create features like average consumption per hour, weekend/weekday indicators.

Example Code for Data Cleaning: import

pandas as pd

```
data = pd. read _csv (”energy _data . csv”) data .
dropna( inplace=True) data = data [ data [
’consumption ’ ] > 0] print( data . head ())
```

2. Descriptive Statistics

Analyze patterns and trends:

- **Mean, Median, Mode:** Average energy usage.
- **Variance and Standard Deviation:** Consumption variability.
- **Histogram Analysis:** Peak usage hours.
- **Correlation Analysis:** Relationship between variables (e.g., temperature vs. energy usage).

Example Code for Descriptive Statistics:

```
print("Mean: " , data [ 'consumption ' ] . mean()) print("Standard Deviation
: " , data [ 'consumption ' ] . std ()) print( data [ [ ' temperature ' ,
'consumption ' ] ] . corr ())
```

3. Predictive Modeling

Statistical Models:

- **Time Series Analysis:**
  - ARIMA (AutoRegressive Integrated Moving Average) for demand forecasting.
  - Seasonal Decomposition of Time Series (STL) to capture seasonal patterns.
- **Regression Analysis:**
  - Multiple Linear Regression to predict consumption based on weather, occupancy, etc.
  - Polynomial Regression for non-linear patterns.
- **Classification Models (Logistic Regression):** Identify devices likely to consume high energy.

Example Code for Regression Analysis:

```
from sklearn . linear _
model import LinearRegression

X = data [ [ ' temperature ' , ' humidity ' ] ] y = data
[ 'consumption ' ] model = LinearRegression ()
model . f i t (X, y)
print(" Coefficients : " , model . coef _ )
```

4. Optimization Techniques

Methods for Scheduling and Load Management:

- **Linear Programming (LP):** Minimize energy costs by scheduling appliance usage during off-peak hours.
- **Mixed-Integer Linear Programming (MILP):** Incorporate binary variables (e.g., ON/OFF state of devices).
- **Stochastic Optimization:** Account for uncertainties like variable renewable energy availability.

Example Code for Optimization: **from** scipy

. optimize **import** linprog

```
costs = [0.12 , 0.15 , 0.20] constraints = [[1 , 1 , 1] ,
[1 , 0 , 1]]
bounds = [(0 ,      5) ,   (0 ,   3) ,   (0 ,   4)]
result = linprog ( costs , A _eq=constraints ,          b _eq =[10,      7] , bounds=bounds)
print(”Optimal schedule : ” ,          result . x)
```

5. Energy Efficiency Metrics

Key Performance Indicators (KPIs):

- **Energy Usage Intensity (EUI):** kWh per square meter of home area.
- **Load Factor:** Average load divided by peak load.
- **Demand Response Effectiveness:** Reduction in peak demand during response events.

6. Statistical Anomaly Detection

Techniques:

- Z-Score for outlier detection.
- Moving averages to detect sudden spikes.
- Clustering (e.g., K-means) to group devices by usage patterns.

Example Code for Anomaly Detection: **from**

scipy . stats **import** zscore

data [' z _score '] = zscore (data ['consumption ']) anomalies = data [data [' z _score '] . **abs**() > 3] **print**(anomalies)

7. Demand Response and User Insights

Behavioral Analytics:

- Analyze how habits affect energy use.
- Develop energy-saving recommendations.

Energy Dashboards:

- Visualize statistics (e.g., daily, weekly trends).

8. Renewable Energy Integration

Modeling for Renewable Sources:

- Solar irradiation prediction using regression.
- Storage optimization for battery usage based on statistical load prediction.

Example Code for Renewable Integration: **from** sklearn .

ensemble **import** RandomForestRegressor

X = data [[' solar radiation ' , ' temperature ']] y = data [' solar output '] model = RandomForestRegressor () model . f i t (X, y)
print("Feature importance : " , model . feature _importances _)

9. Real-Time Monitoring and Feedback

Real-Time Statistical Analysis:

- Use moving averages or exponentially weighted averages for adaptive controls.
- Implement real-time alerts for abnormal consumption.

Example Code for Real-Time Alerts:

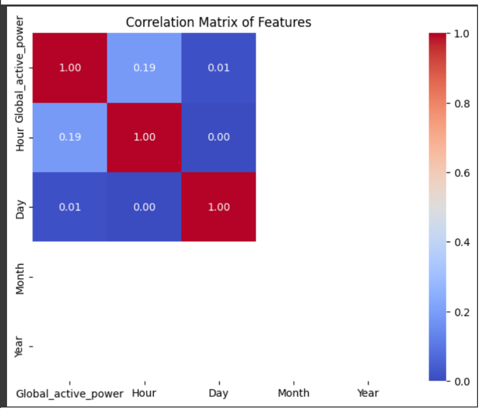
```
def check _abnormal _consumption (consumption ,          threshold ):
    if consumption > threshold :
        print("Alert : High consumption detected ! ") check _abnormal _

consumption (15 , 10)
```

Data-Analysis

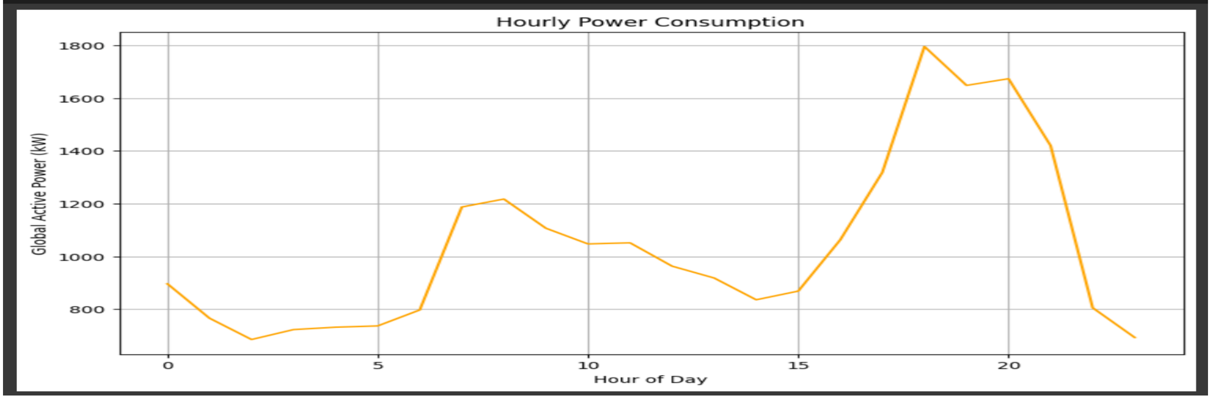
```
import pandas as pd

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

# Explore the first few rows
df.head()

# Check for missing values
df.isnull().sum()

# Check the data types and summary statistics
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20351 entries, 0 to 20350
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   index                  20351 non-null  int64
 1   Date                   20351 non-null  object
 2   Time                   20350 non-null  object
 3   Global_active_power    20350 non-null  object
 4   Global_reactive_power  20350 non-null  object
 5   Voltage                20350 non-null  object
 6   Global_intensity       20350 non-null  object
 7   Sub_metering_1         20350 non-null  object
 8   Sub_metering_2         20350 non-null  object
 9   Sub_metering_3         20349 non-null  float64
dtypes: float64(1), int64(1), object(8)
memory usage: 1.6+ MB
```

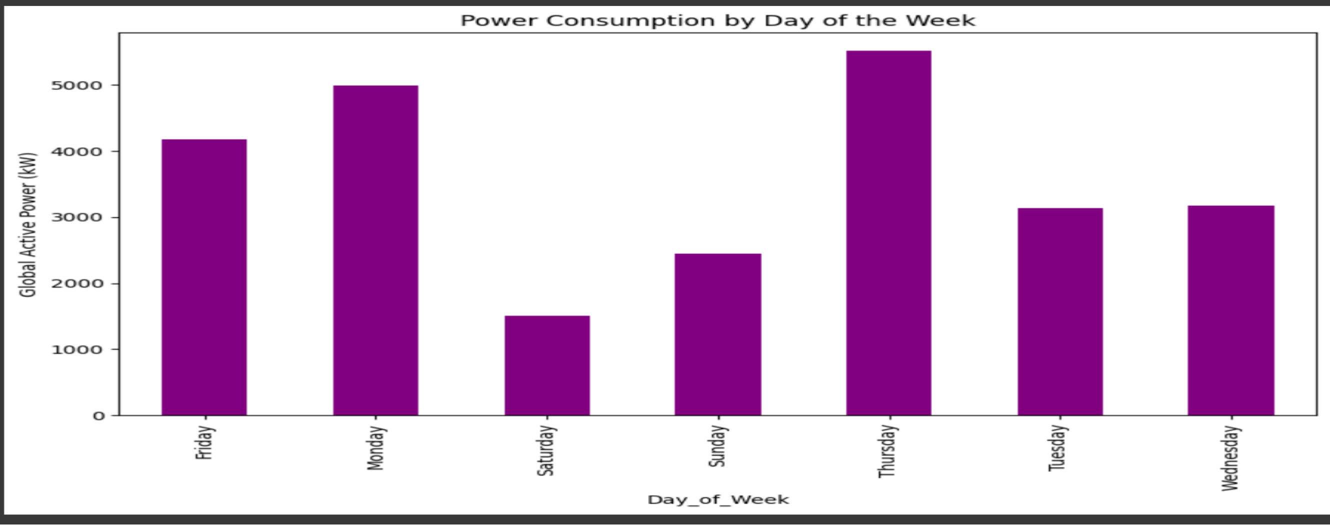
	index	Sub_metering_3
count	20351.000	20349.000000
mean	10175.000	6.873950
std	5874.972	8.527352
min	0.000	0.000000
25%	5087.500	0.000000
50%	10175.000	0.000000
75%	15262.500	17.000000
max	20350.000	19.000000



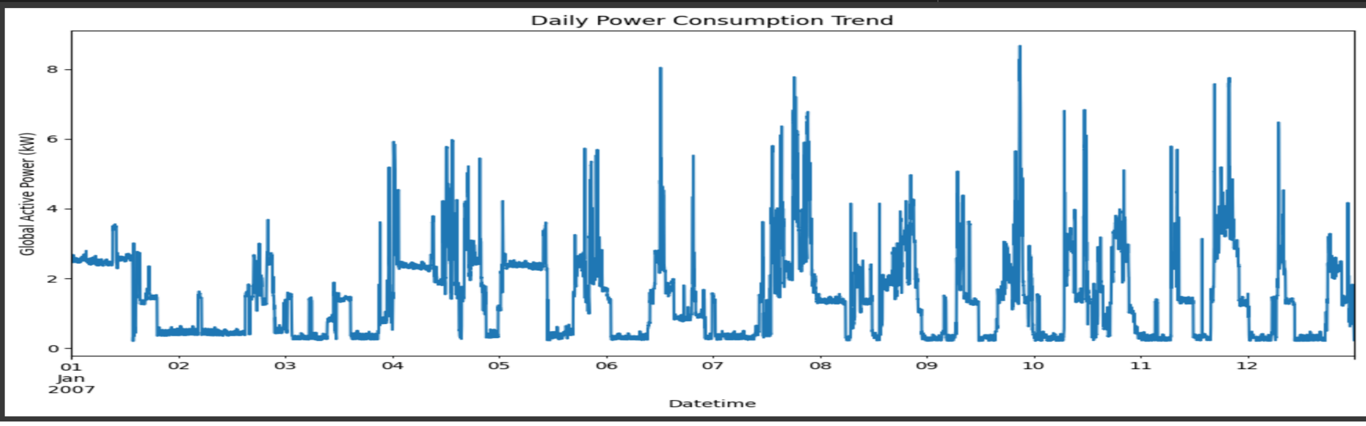

```
# Group by Hour to analyze daily patterns
hourly_consumption = df.groupby('Hour')['Global_active_power'].sum()

# Plot hourly power consumption
hourly_consumption.plot(kind='line', figsize=(10, 6), color='orange')
plt.title("Hourly Power Consumption")
plt.ylabel('Global Active Power (kW)')
plt.xlabel('Hour of Day')
plt.grid(True)
plt.show()
```



```python
# Group by Day of the Week to analyze weekly patterns
daily_consumption = df.groupby('Day_of_Week')['Global_active_power'].sum()

# Plot power consumption by day of the week
daily_consumption.plot(kind='bar', figsize=(10, 6), color='purple')
plt.title("Power Consumption by Day of the Week")
plt.ylabel('Global Active Power (kW)')
plt.show()
```




```python
# Convert 'Global_active_power' to numeric, handling errors
df['Global_active_power'] = pd.to_numeric(df['Global_active_power'], errors='coerce')

# Fill NaN values in 'Global_active_power' with the median
df['Global_active_power'] = df['Global_active_power'].fillna(df['Global_active_power'].median())

# Plot daily power consumption trend
df.groupby('Datetime')['Global_active_power'].sum().plot(figsize=(12, 6))
plt.title("Daily Power Consumption Trend")
plt.ylabel('Global Active Power (kW)')
plt.show()
```



Practical Implementation :

Linear Regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Assuming you're predicting 'Global_active_power' based on other features
X = df_model.drop('Global_active_power', axis=1)
y = df_model['Global_active_power']

y=y.dropna()

# Split the data into train and test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the imputer on the training data and transform both train and test data
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Create and train a model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

print(y_pred)
```

Model Evaluation

```
[3] from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

    mse = mean_squared_error(y_test, y_pred)

    rmse = mse**0.5  # or use np.sqrt(mse)

    r2 = r2_score(y_test, y_pred)

    ac= accuracy_score(y_test, y_train)

    print(f"RMSE: {rmse}")
    print(f"R-squared: {r2}")
    print(f"Accuracy: {ac}")
```