Electric Load & Weather Data Analysis - Report

Step 1: Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Explanation:
We begin by importing the most common Python libraries for data analysis and visualization:

- pandas: for working with data in tabular format.
- numpy: for numerical operations.
- matplotlib.pyplot: for plotting graphs.
- seaborn: for enhanced visualization.

Step 1.1: Reading the Datasets

```
weather_data =
pd.read_csv(r"C:\Users\hp\OneDrive\Desktop\Assignment_\weather_data.csv")


load=pd.read_csv(r"C:\Users\hp\OneDrive\Desktop\Assignment_\load_data.csv"
)
```

Explanation:
We load both datasets into pandas DataFrames using the read_csv() function. This function reads CSV files and turns them into structured data tables (DataFrames).

Step 1.2: Previewing the Data

print(load_data.head())

print(load_data.describe())

print(load_data.info())

print(weather_data.head())

print(weather_data.describe())

print(weather_data.info())

Explanation:
We use:

- .head() to view the first few rows of the dataset. Useful for getting a quick glimpse.
- .describe() to view statistical summaries like mean, standard deviation, min, and max.
- .info() to understand column types and count of missing/non-null values.
  This helps in identifying if there are issues such as missing values or incorrect datatypes.

Step 2: Convert from Wide to Long Format (Load Data)

```
load_data=[]
hour_cols=[col for col in load if col.startswith('h')]

for index,row in load.iterrows():
    for hour_col in hour_cols:
        load_data.append({
            'zone_id':row['zone_id'],
            'year':row['year'],
            'month':row['month'],
            'day':row['day'],
            'hour':hour_col,
            'load_MW':row[hour_col],
        })


load_data_loop=pd.DataFrame(load_data)
load_data_loop
```

Explanation:
The original dataset stores each hour's data (h1 to h24) in separate columns (wide format). We need to analyze hourly data, so we reshape it into a long format where each row represents one hour. This makes it easier for time-series analysis.

Step 2.1: Convert from Wide to Long Format (Weather Data)

```
weather_data=[]
weather_hour_cols=[col for col in load if col.startswith('h')]

for index,row in weather.iterrows():
    for hour_col in hour_cols:
        weather_data.append({
            'station_id':row['station_id'],
            'year':row['year'],
            'month':row['month'],
            'day':row['day'],
            'hour':hour_col,
            'temperature':row[hour_col],
        })


weather_data_loop=pd.DataFrame(weather_data)
weather_data_loop
```

Explanation:
We apply the same transformation to the weather dataset so it aligns with the hourly load data. This makes it easier to merge and compare load with weather data for the same datetime values.

Step 2.2: Check Resulting Shapes and Data

print(load_long.shape)

print(weather_long.shape)

print(load_long.head())

print(weather_long.head())

Explanation:
After reshaping, we check:

- .shape to verify the new number of rows and columns.
- .head() to confirm that the long-format structure looks correct and all columns are intact.

Step 3: Clean Hour Columns and Create Datetime

```
load_hour_list=[]

for h in load_data_loop["hour"]:
    if isinstance(h,str) and h.startswith("h"):
        load_hour_list.append(int(h[1:]))
    else:
        load_hour_list.append(None)

load_data_loop["hour"]=load_hour_list
```

```
weather_hour_list=[]

for h in weather_data_loop["hour"]:
    if isinstance(h,str) and h.startswith("h"):
        weather_hour_list.append(int(h[1:]))
    else:
        weather_hour_list.append(None)

weather_data_loop["hour"]=weather_hour_list
```

```
load_data_loop["datetime"] = pd.to_datetime(load_data_loop[["year",
"month", "day"]]) + pd.to_timedelta(load_data_loop["hour"] - 1, unit="h")

weather_data_loop["datetime"] = pd.to_datetime(weather_data_loop[["year",
"month", "day"]]) + pd.to_timedelta(weather_data_loop["hour"] - 1,
unit="h")
```

Explanation:
We:

- Remove the “h” from hour values and convert them to integers.
- Combine the date and hour values into a single datetime column to allow accurate time-based analysis and merging.

Step 4: Merge Datasets

```
df = pd.merge(load_data_loop, weather_data_loop[["datetime",
"temperature"]], on="datetime")
```

Explanation:
We merge both datasets on the datetime column. This lets us compare electric load and weather for each hour.

Step 5: Add Time-Based Features

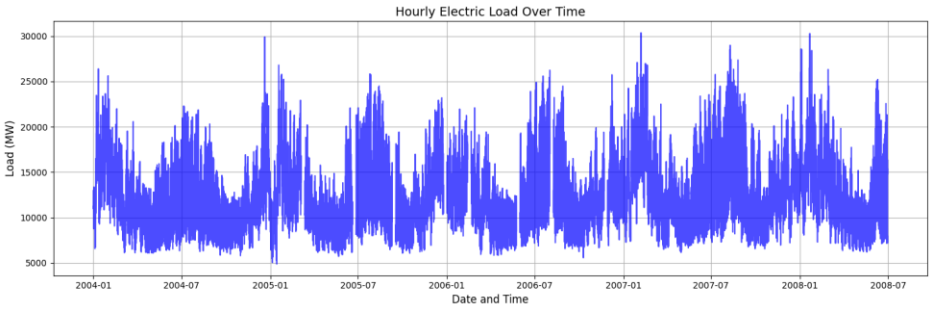
```
df["weekday"] = df["datetime"].dt.day_name()
df["hour"] = df["datetime"].dt.hour
df["month"] = df["datetime"].dt.month
```

Explanation:
We extract new columns from the datetime column for analysis:

- weekday: name of the day (e.g., Monday)
- hour: hour of the day (0–23)
- month: month of the year (1–12)

Step 6: Visualization - Load Over Time

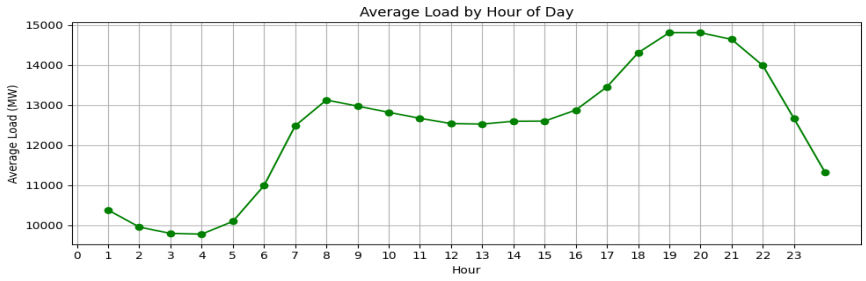
```
import matplotlib.pyplot as plt

# Plot hourly electric load over time
plt.figure(figsize=(15, 5))
plt.plot(load_data_loop["datetime"], load_data_loop["load_MW"],
color="blue", alpha=0.7)

plt.title("Hourly Electric Load Over Time", fontsize=14)
plt.xlabel("Date and Time", fontsize=12)
plt.ylabel("Load (MW)", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
Seasonal Patterns
Regular yearly cycles are clearly visible.

Higher loads during summer and winter months, possibly due to:

Air conditioning in summer

Heating in winter
Lower demand in milder months

There's a slight upward trend in base load over the years.

Indicates growing energy consumption
```

Step 7: Average Load by Hour

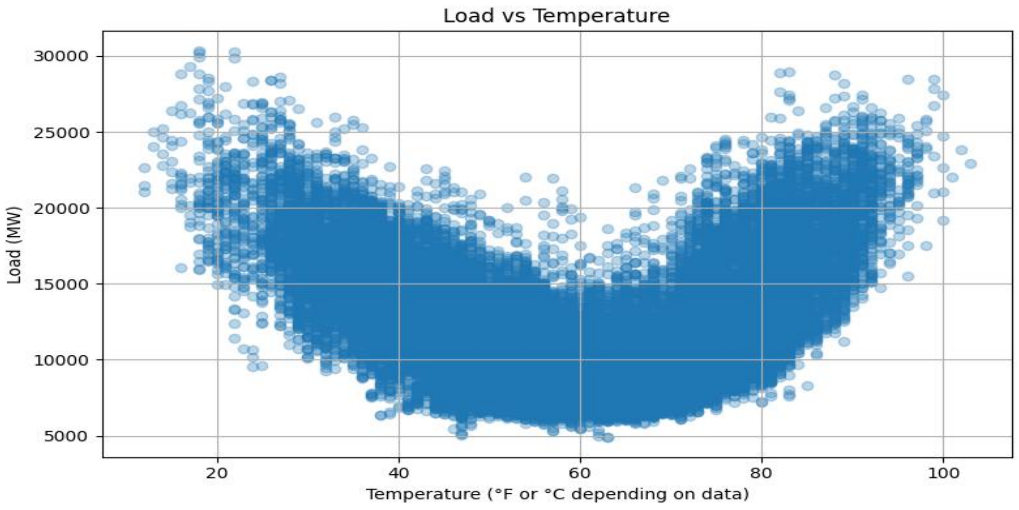
```
hourly_avg=load_data_loop.groupby("hour")["load_MW"].mean()
plt.figure(figsize=(10, 4))
plt.plot(hourly_avg.index, hourly_avg.values, marker='o',color='green')
plt.title("Average Load by Hour of Day")
plt.xlabel("Hour")
plt.ylabel("Average Load (MW)")
plt.grid(True)
plt.xticks(range(0, 24))
plt.tight_layout()
plt.show()
```



```
Pattern Observed:
Low demand during the early morning hours (midnight to 5 AM)

Sharp rise starts around 6 AM, likely as people wake up and businesses
begin operating

Two peaks observed:

Morning peak: Around 7–9 AM

Evening peak: Highest between 6–8 PM (around 18:00–20:00)

Drop after 9 PM as daily activity slows down
```

Step 8: Scatter Plot - Load vs Temperature

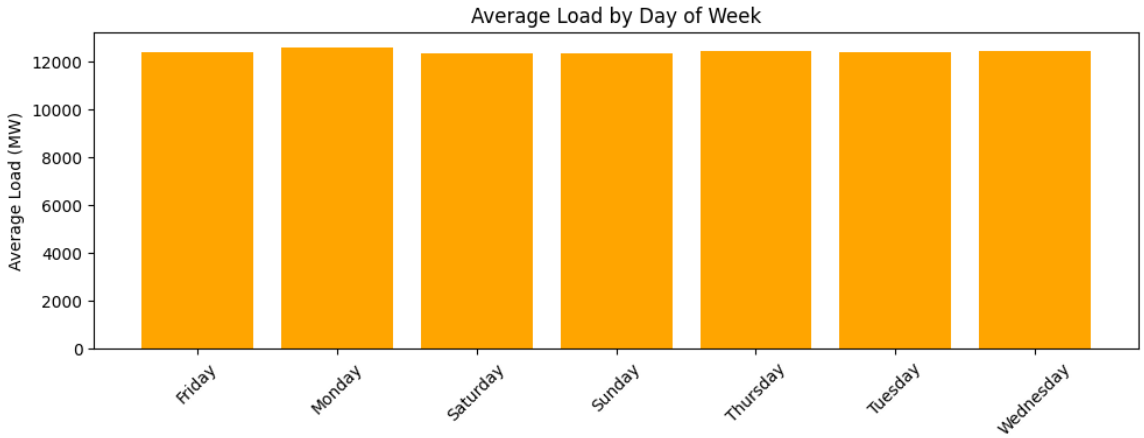
```
plt.figure(figsize=(8, 5))
plt.scatter(df["temperature"], df["load_MW"], alpha=0.3)
plt.title("Load vs Temperature")
plt.xlabel("Temperature (°F or °C depending on data)")
plt.ylabel("Load (MW)")
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
Higher load at low temperatures (left side of U): Increased heating demand

Higher load at high temperatures (right side of U):Increased cooling
demand (air conditioners, refrigeration)
```

Step 9: Average Load by day of week

```
weekday_avg = df.groupby("weekday")["load_MW"].mean()

plt.figure(figsize=(10, 4))
plt.bar(weekday_avg.index, weekday_avg.values, color='orange')
plt.title("Average Load by Day of Week")
plt.ylabel("Average Load (MW)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
Load values are very similar across all days of the week.

No significant drop on weekends (Saturday, Sunday), which is often seen in
residential-focused grids.
```

Step 10: Correlation

```
correlation = df[["load_MW", "temperature"]].corr()
print("Correlation between Load and Temperature:\n", correlation)
```

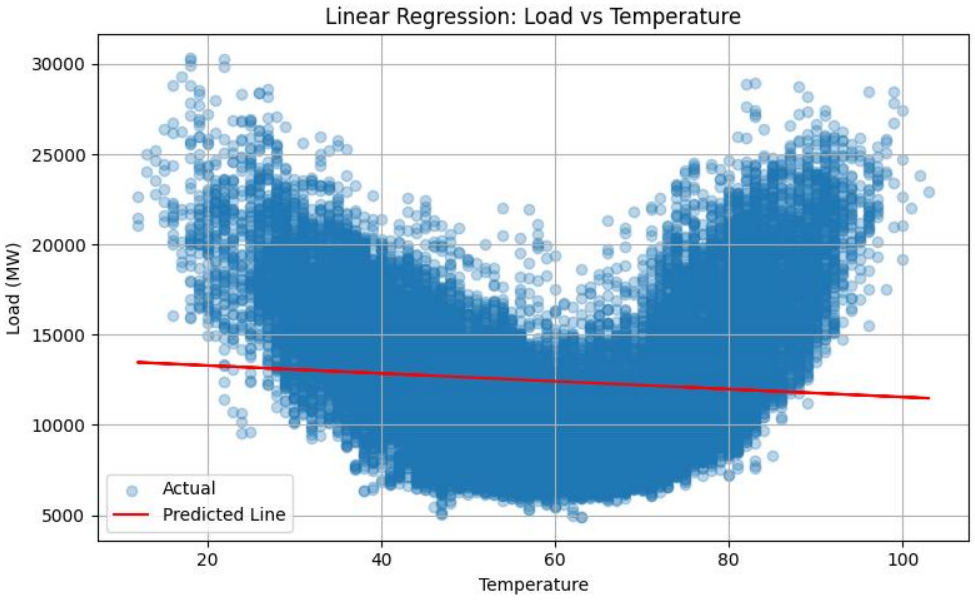
Explanation:
We calculate the correlation coefficient to quantify how closely load and temperature are related.

Step 11: (Optional) Basic Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

X = df[["temperature"]]
y = df["load_MW"]
```

Explanation:
We try a simple linear regression model to predict electric load based on temperature. This is optional and shows how machine learning can be applied to energy forecasting.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.scatter(df["temperature"], df["load_MW"], alpha=0.3, label="Actual")
plt.plot(df["temperature"], y_pred, color="red", label="Predicted Line")
plt.title("Linear Regression: Load vs Temperature")
plt.xlabel("Temperature")
plt.ylabel("Load (MW)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
I created a simple linear regression model using temperature to predict
electric load. The model captures basic trends, but since the relationship
is not linear, a more advanced model
```