PROJECT - ARTIFICIAL INTELLIGENCE IN SMART GRID

NAME -Roger Kewin Samson

HAWK ID - A20563057

MAIL ID - rkewin@hawk.iit.edu

SPRING 2024

ILLINOIS INSTITUTE OF TECHNOLOGY

BRIEF OVERVIEW:

The final project is to predict the electrical load values using temperature data across various zones. these predictions are crucial for efficient energy management and will be accurate because they are based on the zones' temperature. We have the dataset that includes temperature readings for 11 stations and load values for the 20 zones all these were in hour differences.

Data preparation : In the given temperature and load values we need to remove the null values and outliers in it. I visualized null data in the load to verify and remove it perfectly. By doing this we can get the cleaned data of both temperature and load we can save it in a data frame or export it as a CSV file. This phase also involves mapping each zone to the most relevant temperature station based on historical data to ensure the accuracy of temperature data used for each zone. Mapping the station to zones by correlation makes the matter to get the prediction better. After mapping we merged the data (cleaned_Load_history_final.csv and cleaned_Tem_hiatory_final.csv)according to the mapped station as merged data. We are going to use this merged data for the training and make predictions. e began by meticulously organizing and visualizing the essential data, creating a comprehensive table that includes both station ID and zone ID. Following this, we meticulously divided the dataset into three distinct subsets: training, validation, and testing. During this process, we ensured the integrity of the validation set and verified its dimensions.

Moving forward, we embarked on selecting a suitable machine learning algorithm to ascertain the accuracy of predictions. Leveraging the merged dataset, we meticulously evaluated the chosen algorithm's efficacy in accurately forecasting based on key features such as temperature and station mapping.

To deepen our understanding, we meticulously examined the top 10 prediction errors stemming from this algorithm. This meticulous analysis provides invaluable insights into the algorithm's strengths and weaknesses.

In our quest for optimization, we then ventured into employing a more intricate and sophisticated algorithm. Following the same rigorous methodology, we assessed its performance against the dataset, ensuring a thorough comparison with the initial algorithm's results. This meticulous approach allows us to select the most appropriate algorithm for our predictive modeling endeavors. After all this the predicted file for the first week of June 2008 will be exported as CSV file.

Making the data cleaned by removing null values , outliers and exploring the data:

In this we removed all null data of overall dataset and outliers and stored it as cleaned data to use it later below.

In [164…

```
import pandas as pd
from scipy import stats
import numpy as np

# Load the datasets
load_data = pd.read_csv('Load_history_final.csv')
temp_data = pd.read_csv('Temp_history_final.csv')

# Remove rows where any cell has a missing value
load_data_clean = load_data.dropna()
temp_data_clean = temp_data.dropna()

# Define columns that contain hourly data to check for outliers and zeros
load_hourly_columns = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'h10',
                       'h13', 'h14', 'h15', 'h16', 'h17', 'h18', 'h19', 'h20', 'h21
temp_hourly_columns = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'h10',
                       'h13', 'h14', 'h15', 'h16', 'h17', 'h18', 'h19', 'h20', 'h21

# Remove rows with zero values in any of the hourly columns
load_data_clean = load_data_clean[~(load_data_clean[load_hourly_columns] == 0).any(
temp_data_clean = temp_data_clean[~(temp_data_clean[temp_hourly_columns] == 0).any(

# Remove outliers in load data
z_scores_load = np.abs(stats.zscore(load_data_clean[load_hourly_columns]))
load_data_clean = load_data_clean[(z_scores_load < 3).all(axis=1)]

# Remove outliers in temperature data
z_scores_temp = np.abs(stats.zscore(temp_data_clean[temp_hourly_columns]))
temp_data_clean = temp_data_clean[(z_scores_temp < 3).all(axis=1)]

# Save the cleaned data
load_data_clean.to_csv('cleaned_Load_history_final.csv', index=False)
temp_data_clean.to_csv('cleaned_Temp_history_final.csv', index=False)
```

The cleaned dataset has been described to analyze and printed.

In [165…

```
import pandas as pd
load_data = pd.read_csv('cleaned_Load_history_final.csv')
temp_data = pd.read_csv('cleaned_Temp_history_final.csv')
```

```
load_data_descr= load_data.describe()
temp_data_descr = load_data.describe()

# Print the descriptive statistics
print("Load Data Description:")
print(load_data_descr)
print("\nTemperature Data Description:")
print(temp_data_descr)

# Display the first few rows of each dataset
print("\nLoad Data - First Few Rows:")
print(load_data.head())
print("\nTemperature Data - First Few Rows:")
print(temp_data.head())
```

```
Load Data Description:
           zone_id          year         month           day            h1  \
count  31417.000000  31417.000000  31417.000000  31417.000000  3.141700e+04
mean      10.649935   2005.727918      6.206767     15.727441  8.448397e+05
std        5.746524      1.292033      3.458183      8.798582  8.749783e+05
min        1.000000   2004.000000      1.000000      1.000000  4.490000e+02
25%        6.000000   2005.000000      3.000000      8.000000  1.021180e+05
50%       11.000000   2006.000000      6.000000     16.000000  5.789830e+05
75%       16.000000   2007.000000      9.000000     23.000000  1.351960e+06
max       20.000000   2008.000000     12.000000     31.000000  3.796875e+06

                 h2            h3            h4            h5            h6  \
count  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   8.127734e+05  7.988111e+05  7.984756e+05  8.185975e+05  8.727174e+05
std    8.451744e+05  8.336535e+05  8.362740e+05  8.596358e+05  9.212095e+05
min    4.200000e+02  6.000000e+00  1.200000e+01  1.800000e+01  1.800000e+01
25%    9.803900e+04  9.624300e+04  9.653400e+04  9.958600e+04  1.079530e+05
50%    5.555640e+05  5.427260e+05  5.404960e+05  5.538220e+05  5.811980e+05
75%    1.281307e+06  1.250257e+06  1.244335e+06  1.269990e+06  1.377286e+06
max    3.711580e+06  3.711571e+06  3.733054e+06  3.851834e+06  4.114112e+06

       ...           h15           h16           h17           h18  \
count  ...  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   ...  1.020239e+06  1.025205e+06  1.052146e+06  1.099831e+06
std    ...  1.101635e+06  1.109157e+06  1.133914e+06  1.177587e+06
min    ...  1.003000e+03  9.280000e+02  1.200000e+01  9.750000e+02
25%    ...  1.252860e+05  1.276770e+05  1.335540e+05  1.422960e+05
50%    ...  6.301740e+05  6.321600e+05  6.430800e+05  6.624430e+05
75%    ...  1.626476e+06  1.643741e+06  1.713972e+06  1.829323e+06
max    ...  4.763057e+06  4.782314e+06  4.911975e+06  5.122909e+06

                h19           h20           h21           h22           h23  \
count  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   1.131291e+06  1.132049e+06  1.124600e+06  1.082794e+06  1.002750e+06
std    1.200190e+06  1.192133e+06  1.176525e+06  1.125248e+06  1.033969e+06
min    2.400000e+01  1.800000e+01  7.960000e+02  1.200000e+01  1.200000e+01
25%    1.495150e+05  1.488550e+05  1.467840e+05  1.385400e+05  1.249890e+05
50%    6.802800e+05  6.841030e+05  6.893510e+05  6.746910e+05  6.494510e+05
75%    1.901508e+06  1.914032e+06  1.915472e+06  1.833503e+06  1.675316e+06
max    5.246080e+06  5.155440e+06  4.958159e+06  4.885742e+06  4.366662e+06

                h24
count  3.141700e+04
mean   9.083949e+05
std    9.377333e+05
min    3.700000e+01
25%    1.112050e+05
50%    6.159230e+05
75%    1.494671e+06
max    4.147000e+06

[8 rows x 28 columns]

Temperature Data Description:
           zone_id          year         month           day            h1  \
count  31417.000000  31417.000000  31417.000000  31417.000000  3.141700e+04
mean      10.649935   2005.727918      6.206767     15.727441  8.448397e+05
std        5.746524      1.292033      3.458183      8.798582  8.749783e+05
min        1.000000   2004.000000      1.000000      1.000000  4.490000e+02
25%        6.000000   2005.000000      3.000000      8.000000  1.021180e+05
50%       11.000000   2006.000000      6.000000     16.000000  5.789830e+05
75%       16.000000   2007.000000      9.000000     23.000000  1.351960e+06
max       20.000000   2008.000000     12.000000     31.000000  3.796875e+06
```

```
                 h2            h3            h4            h5            h6  \
count  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   8.127734e+05  7.988111e+05  7.984756e+05  8.185975e+05  8.727174e+05
std    8.451744e+05  8.336535e+05  8.362740e+05  8.596358e+05  9.212095e+05
min    4.200000e+02  6.000000e+00  1.200000e+01  1.800000e+01  1.800000e+01
25%    9.803900e+04  9.624300e+04  9.653400e+04  9.958600e+04  1.079530e+05
50%    5.555640e+05  5.427260e+05  5.404960e+05  5.538220e+05  5.811980e+05
75%    1.281307e+06  1.250257e+06  1.244335e+06  1.269990e+06  1.377286e+06
max    3.711580e+06  3.711571e+06  3.733054e+06  3.851834e+06  4.114112e+06

       ...           h15           h16           h17           h18  \
count  ...  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   ...  1.020239e+06  1.025205e+06  1.052146e+06  1.099831e+06
std    ...  1.101635e+06  1.109157e+06  1.133914e+06  1.177587e+06
min    ...  1.003000e+03  9.280000e+02  1.200000e+01  9.750000e+02
25%    ...  1.252860e+05  1.276770e+05  1.335540e+05  1.422960e+05
50%    ...  6.301740e+05  6.321600e+05  6.430800e+05  6.624430e+05
75%    ...  1.626476e+06  1.643741e+06  1.713972e+06  1.829323e+06
max    ...  4.763057e+06  4.782314e+06  4.911975e+06  5.122909e+06

                h19           h20           h21           h22           h23  \
count  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04  3.141700e+04
mean   1.131291e+06  1.132049e+06  1.124600e+06  1.082794e+06  1.002750e+06
std    1.200190e+06  1.192133e+06  1.176525e+06  1.125248e+06  1.033969e+06
min    2.400000e+01  1.800000e+01  7.960000e+02  1.200000e+01  1.200000e+01
25%    1.495150e+05  1.488550e+05  1.467840e+05  1.385400e+05  1.249890e+05
50%    6.802800e+05  6.841030e+05  6.893510e+05  6.746910e+05  6.494510e+05
75%    1.901508e+06  1.914032e+06  1.915472e+06  1.833503e+06  1.675316e+06
max    5.246080e+06  5.155440e+06  4.958159e+06  4.885742e+06  4.366662e+06

                h24
count  3.141700e+04
mean   9.083949e+05
std    9.377333e+05
min    3.700000e+01
25%    1.112050e+05
50%    6.159230e+05
75%    1.494671e+06
max    4.147000e+06

[8 rows x 28 columns]

Load Data - First Few Rows:
   zone_id  year  month  day      h1      h2      h3      h4      h5      h6  \
0        1  2004      1    1  542169  544849  541862  544319  561330  565693
1        1  2004      1    2  490124  478742  463124  466425  475824  499852
2        1  2004      1    3  509696  503760  476790  487266  477202  496449
3        1  2004      1    4  373085  351389  341986  337871  337596  339033
4        1  2004      1    5  380194  356897  352180  350269  370094  414694

   ...     h15     h16     h17     h18     h19     h20     h21     h22  \
0  ...  451704  446139  475805  544992  596612  590534  603643  592222
1  ...  508017  489559  517089  560219  590467  583441  580723  580111
2  ...  419058  409703  427163  464066  490036  486732  480645  468861
3  ...  426370  421913  421030  477947  511290  510182  497117  463752
4  ...  450381  457742  483160  541697  574431  562290  559592  510941

      h23     h24
0  562688  520092
1  581436  546339
2  446473  416035
3  435648  401300
4  492231  434870
```

```
[5 rows x 28 columns]

Temperature Data - First Few Rows:
   station_id  year  month  day  h1  h2  h3  h4  h5  h6  ...  h15  h16  h17  \
0           1  2004      1    1  43  44  42  34  30  35  ...   61   61   59
1           1  2004      1    2  45  46  47  46  46  45  ...   53   57   58
2           1  2004      1    3  46  46  42  41  42  41  ...   73   72   73
3           1  2004      1    4  62  62  60  60  60  60  ...   71   73   74
4           1  2004      1    5  64  61  62  61  62  62  ...   66   66   65

   h18  h19  h20  h21  h22  h23  h24
0   53   43   37   36   36   43   44
1   52   46   45   42   43   44   47
2   69   65   64   60   61   62   61
3   70   67   67   63   63   63   64
4   62   60   60   58   57   53   51

[5 rows x 28 columns]
```

The cleaned data has merged for the correlation to map station:

In [180…

```python
import pandas as pd

load_data = pd.read_csv('cleaned_Load_history_final.csv')
temp_data = pd.read_csv('cleaned_Temp_history_final.csv')

# Reshaping the data to long format
temp_long = temp_data.melt(id_vars=['station_id', 'year', 'month', 'day'], var_name
load_long = load_data.melt(id_vars=['zone_id', 'year', 'month', 'day'], var_name='h

# Converting 'hour' column from string to integer for merging
temp_long['hour'] = temp_long['hour'].str.extract('(\d+)').astype(int)
load_long['hour'] = load_long['hour'].str.extract('(\d+)').astype(int)

# Merging the data on year, month, day, and hour
merged_data = pd.merge(temp_long, load_long, on=['year', 'month', 'day', 'hour'])

# Display the first few rows of the merged data to verify
print(merged_data.head())
```

```
   station_id  year  month  day  hour  temperature  zone_id     load
0           1  2004      1    1     1           43        1   542169
1           1  2004      1    1     1           43        2  1040598
2           1  2004      1    1     1           43        3   553320
3           1  2004      1    1     1           43        4  3455947
4           1  2004      1    1     1           43        5  2190086
```

The correlation for the merged data:

Grouping and Correlation Calculation:

First, it groups the data by station. Then, for each station, it calculates a number called "correlation" between temperature and electricity usage. This number tells us how much they're related. If it's high, it means when it's hotter, more electricity is used, and vice versa.

The code organizes these correlation numbers into a list. It sorts this list from stations with the highest correlation to the lowest. So, at the top are stations where temperature and electricity usage are strongly connected.

Finally, it shows us this list, with each station's ID and its correlation number. This helps us see which stations are most influenced by temperature changes in terms of electricity usage.

In [181…

```
# Group by station_id and calculate correlation
correlation_data = merged_data.groupby('station_id').apply(lambda x: x['temperature

# Convert to DataFrame and sort by correlation
correlation_df = correlation_data.reset_index()
correlation_df.columns = ['station_id', 'correlation']
correlation_df = correlation_df.sort_values(by='correlation', ascending=False)

# Display the correlations
print(correlation_df)
```

```
    station_id  correlation
7            8    -0.004940
0            1    -0.005928
6            7    -0.006234
5            6    -0.006238
2            3    -0.006846
3            4    -0.007415
8            9    -0.007753
9           10    -0.008171
10          11    -0.008460
4            5    -0.010507
1            2    -0.011005
```

The code starts by grouping the data by both the zone and the station. This way, it organizes the information based on where the stations are located and then breaks it down further by each individual station within each zone. Then, it calculates something called "correlation" for each combination of zone and station. This number tells us how closely connected the temperature and electricity usage are at each station within each zone. If the correlation is high, it means that when it's hotter, more electricity is used, and when it's cooler, less electricity is used. After calculating the correlation for each zone-station combination, the code puts these correlation numbers into a DataFrame called correlation_df. This DataFrame has columns for the zone ID, the station ID, and the correlation value. Then, it sorts this DataFrame to find the stations with the highest correlation for each zone. This helps us identify which stations in each zone have the strongest connection between temperature and electricity usage.

The code prints out the DataFrame best_stations. This DataFrame contains the best station (the one with the highest correlation) for each zone. By looking at this information, we can see which stations within each zone are most influenced by changes in temperature when it comes to electricity usage.

In [182…

```
import pandas as pd


# Calculate the correlation for each station and zone combination
grouped = merged_data.groupby(['zone_id', 'station_id'])
correlation_by_zone_station = grouped.apply(lambda x: x['temperature'].corr(x['loac

# Convert the series to a DataFrame
correlation_df = correlation_by_zone_station.reset_index()
correlation_df.columns = ['zone_id', 'station_id', 'correlation']
```

```
# Find the station with the highest correlation for each zone
best_stations = correlation_df.loc[correlation_df.groupby('zone_id')['correlation']

# Display the best station for each zone
print(best_stations)
```

```
     zone_id  station_id  correlation
7          1           8    -0.058138
18         2           8    -0.142550
22         3           1    -0.119071
40         4           8     0.127645
51         5           8    -0.070924
55         6           1    -0.116148
66         7           1    -0.107291
84         8           8    -0.003380
95         9           8    -0.184011
106       10           8    -0.200753
110       11           1    -0.424894
128       12           8    -0.025054
139       13           8    -0.115704
143       14           1    -0.302143
156       15           3    -0.091166
172       16           8    -0.088778
176       17           1    -0.057392
187       18           1    -0.385365
199       19           2    -0.059700
216       20           8    -0.192848
```

Table showing mapping of a temperature station for each load zone:

zone_id	station_id
1	8
2	8
3	1
4	8
5	8
6	1
7	1
8	8
9	8
10	8
11	1
12	8
13	8
14	1
15	3
16	8

zone_id	station_id
17	1
18	1
19	2
20	8

mapping station to zones and exporting it as a CSV file:

A dictionary named zone_station_mapping is created, which pairs each zone with its corresponding station ID. For instance, zone 1 is linked to station 8, zone 2 to station 8, and so forth. This mapping helps in associating each zone with the appropriate station where data will be collected or analyzed.

The load_long DataFrame's 'zone_id' column is linked to the respective 'station_id' using the map() function along with the zone_station_mapping dictionary. This process assigns each record in the load_long DataFrame to the correct station ID based on its zone. Subsequently, the load_long DataFrame is merged with the temp_long DataFrame utilizing common columns such as 'station_id', 'year', 'month', 'day', and 'hour'. This merging operation combines data from both DataFrames into a single DataFrame referred to as merged_data. Then the dataframe data is exported as a csv file.

In [183…

```
# Mapping zones to stations
zone_station_mapping = {
    1: 8, 2: 8, 3: 1, 4: 8, 5: 8, 6: 1, 7: 1, 8: 8, 9: 8, 10: 8,
    11: 1, 12: 8, 13: 8, 14: 1, 15: 3, 16: 8, 17: 1, 18: 1, 19: 2, 20: 8
}
load_long['station_id'] = load_long['zone_id'].map(zone_station_mapping)
merged_data = pd.merge(load_long, temp_long, on=['station_id', 'year', 'month', 'da
```

In [184…

```
import pandas as pd


merged_data.to_csv('merged_data.csv', index=False)  # This will save the DataFrame
```

Python code that splits the load and temperature datasets into two subsets: training/validation, and test. .

We are loading the merged_data file and preparing the features and target variable for the training and make it ready. Now as per the delieverables we split our data in to the 70% for training and 30% for testing and printed the shape of the data.

RANDOM_STATE:

I chose "0" as a random_state because i dont want make differ while iteration to make the same each time i chose the 0 as the random state for the split

In [185…

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your data
data = pd.read_csv('merged_data.csv')

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load']

# First split to create train/validation and test sets for load and temperature dat
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, ra

# Print the shapes
print("Training/Validation set shapes:")
print("X_train_val shape:", X_train_val.shape)
print("y_train_val shape:", y_train_val.shape)
print("\nTest set shapes:")
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
Training/Validation set shapes:
X_train_val shape: (527016, 7)
y_train_val shape: (527016,)

Test set shapes:
X_test shape: (225864, 7)
y_test shape: (225864,)
```

Python code that uses an additional split to create a validation dataset

Then we created the another split for the validation as the deliverables need as the additional split. we imported the merged_data.csv file as we done above then trained data is splitted in to train and validation.printed the shape of the dataset.

Random_State:

I chose as the same above "0" as the random_state value because i dont want different values to be done in the iteration hence i chose "0" as the random_state value.

In [186…

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your data
data = pd.read_csv('merged_data.csv')

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load']

# First split to create train/validation and test sets for load and temperature dat
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.5, ra

# Additional split to create validation dataset
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_si

# Print the shapes
```

```
print("Training set shapes:")
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("\nValidation set shapes:")
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
```

```
Training set shapes:
X_train shape: (282330, 7)
y_train shape: (282330,)

Validation set shapes:
X_val shape: (94110, 7)
y_val shape: (94110,)
```

Procedure documenting your design process and the tradeoffs you considered in building your first machine learning regressor model.

Data Handling and Setup:

I loaded the dataset from 'merged_data.csv' using Pandas and organized it into features (labeled as 'X') and the target variable (labeled as 'y').

Data Scaling:

To ensure consistency in scale across all features, I standardized the target variable using StandardScaler.

Data Partitioning:

Using the train_test_split function, I divided the data into training, validation, and test sets. Initially, I allocated 30% of the data for testing and validation, and then split the remaining data equally between validation and test sets.

Feature Standardization:

All features were standardized by scaling them to have a mean of zero and a variance of one using StandardScaler.

Model Setup and Training:

I initialized a RandomForestRegressor with specific parameters: 50 trees (n_estimators), a minimum of 10 samples per leaf (min_samples_leaf), and a fixed random state (random_state=0) for reproducibility. The model was trained using the scaled training data.

Model Assessment:

Predictions were made on the training, validation, and test sets. Performance was evaluated using Mean Squared Error (MSE) to gauge the average squared difference between predicted and actual values, and R-squared (R^2) to measure the proportion of variance explained by the model.

Performance Examination:

I checked the model performance across training, validation, and test sets, examining any tradeoffs between them. Additionally, I considered discrepancies in performance across different datasets.compared to others this was the best algorithm gives best accuracy.

In [189…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load your data
data = pd.read_csv('merged_data.csv')

# Prepare features and target variable
X = data[['temperature']]
y = data['load']

# Normalize the target variable
scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y_scaled, test_size=0.3, rar
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ranc

# Feature scaling for features
scaler_x = StandardScaler()
X_train_scaled = scaler_x.fit_transform(X_train)
X_val_scaled = scaler_x.transform(X_val)
X_test_scaled = scaler_x.transform(X_test)

# Initialize and train the RandomForestRegressor with reduced complexity
rf_model = RandomForestRegressor(n_estimators=50, min_samples_leaf=10, random_state
rf_model.fit(X_train_scaled, y_train)

# Make predictions
y_train_pred = rf_model.predict(X_train_scaled)
y_val_pred = rf_model.predict(X_val_scaled)
y_test_pred = rf_model.predict(X_test_scaled)

# Calculate performance metrics
train_mse = mean_squared_error(y_train, y_train_pred)
val_mse = mean_squared_error(y_val, y_val_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
val_r2 = r2_score(y_val, y_val_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Print the performance metrics
print("Training MSE:", train_mse)
print("Validation MSE:", val_mse)
print("Test MSE:", test_mse)
print("Training RÂ²:", train_r2)
print("Validation RÂ²:", val_r2)
print("Test RÂ²:", test_r2)
```

```
Training MSE: 0.9878213460848757
Validation MSE: 0.9865845262582862
Test MSE: 0.9902769959169442
Training RÂ²: 0.012480812509563344
Validation RÂ²: 0.01162645495930048
Test RÂ²: 0.010077417664275834
```

OBSERVATIONS:

First i chose Linear regression and it is low accuracy then i choose the Randomforest regressor because it makes high accuracy score compared to that. I scaled the target variable using standardScaler.This ensures that all feature have the same scale which improve the accuracy score of the model.

Feature Scaling: You standardized the features using StandardScaler. By centering the features around zero and scaling them to have a unit variance, you make the optimization process smoother and prevent certain features from dominating others, leading to potentially better model performance.

I specified hyperparameters as n_estimators and the min_samples_leaf. I keep on changing value by uing 100 estimators without min_sample_leaf it gave very high mse score . by tuning and make it 50 and 10 makes the accuracy perfect scores. by doing all this evaluation gives best score in mse and r^2 score without any overfitting or underfiiting.

PREDICTION AND EXPORTING CSV FILE FOR MY BEST ALGORITHM:

To make accurate predictions about the load for June2008, we started by matching temperature statios to their respective load zones. we achieved this by using a mapping dictionary to assign each temperature stations to its corresponding load zone. Any missing or invalid mappings were removed from consideration to ensure the accuracy of our predictions.

Once we had mapped the temperature stations to their load zones, we combined this mapping with our temperature data and focused on observations from the first week of June 2008. This allowed us to narrow down our dataset to the specific timeframe we were interested in.

With our data properly filtered and organized, we prepared the temperature feature for prediction. We isolated the temperature readings, which would serve as our input for predicting load values.

Using a Random Forest model that we had previously trained, we made predictions about the load values for June 2008 based on the temperature data. This model had learned patterns from historical data and was now capable of making predictions about future load values.

The predicted load values were then incorporated back into our dataset. This step allowed us to see both the actual temperature readings and the corresponding predicted load values side by side.

Finally, we formatted these predictions for easy analysis and presentation. We structured the data to resemble the format of a CSV file called 'Load_Prediction.csv', making it convenient for further examination or sharing with others.

By following this process, we ensured that our predictions for the load in June 2008 were well-prepared and ready for use in decision-making or further analysis.

In [ ]:

```
june_2008_temp.loc[:, 'zone_id'] = june_2008_temp['station_id'].map(zone_station_ma
june_2008_temp = june_2008_temp.dropna(subset=['zone_id'])  # Drop rows where zone_
import pandas as pd

# Create a DataFrame from your mapping
zone_station_df = pd.DataFrame({
    'zone_id': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    'station_id': [8, 8, 1, 8, 8, 1, 1, 8, 8, 8, 1, 8, 8, 1, 3, 8, 1, 1, 2, 8]
})

# Merge this mapping with the temperature data to assign the correct station's temp
june_2008_temp = pd.merge(zone_station_df, temp_long, on='station_id')

# Filter for the first week of June 2008
june_2008_temp = june_2008_temp[(june_2008_temp['year'] == 2008) & (june_2008_temp[

# Prepare features for prediction
X_june_2008 = june_2008_temp[['temperature']]
# Predict the load for June 2008 using the trained Random Forest model
predicted_loads_rf = random_forest_model.predict(X_june_2008)

# Add predictions back to the DataFrame
june_2008_temp['predicted_load'] = predicted_loads_rf

predicted_output_rf = june_2008_temp.pivot_table(index=['zone_id', 'year', 'month',
predicted_output_rf.columns = [f'h{col}' for col in predicted_output_rf.columns]
predicted_output_rf.reset_index(inplace=True)

predicted_output_rf.to_csv('Load_Prediction.csv', index=False)
```

In [124…

```
import pandas as pd

# Assuming 'test_data' includes the 'zone', 'year', 'month', 'day', 'hour', and 'lo
# and y_test_pred_rf contains the predicted loads from the RanAdom Forest model.

# Add the predicted loads to the test data DataFrame
test_data['predicted_load'] = y_test_pred_rf

# Calculate the relative percentage error
test_data['relative_percentage_error'] = 100 * (test_data['load'] - test_data['pred

# Handle cases where true load is zero to avoid infinite values
test_data['relative_percentage_error'].replace([float('inf'), -float('inf')], pd.NA

# Drop rows where relative percentage error could not be calculated (e.g., true loa
test_data.dropna(subset=['relative_percentage_error'], inplace=True)

# Sort the DataFrame by the absolute value of the relative percentage error in desc
test_data['abs_relative_error'] = test_data['relative_percentage_error'].abs()
sorted_errors = test_data.sort_values(by='abs_relative_error', ascending=False)

top_10_errors = sorted_errors.head(10)[['zone_id', 'year', 'month', 'day', 'hour',
```

```
# Output the top 10 errors
print(top_10_errors)
```

```
        zone_id  year  month  day  hour  predicted_load  load  \
213832       11  2007      6    2     7    8.877940e+05   422   
715745       11  2007      6    1    23    8.703449e+05   607   
519089       11  2004      2    6    17    1.039436e+06   978   
487719       11  2004      2    6    16    9.617957e+05   928   
456349       11  2004      2    6    15    9.886635e+05  1003   
362239       11  2004      2    6    12    9.802649e+05  1050   
299499       11  2004      2    6    10    9.680531e+05  1074   
330869       11  2004      2    6    11    9.773447e+05  1089   
119372       11  2007      4   13     4    9.440827e+05  1061   
550459       11  2004      2    6    18    9.617957e+05  1105   

        relative_percentage_error  
213832             -210277.714502  
715745             -143284.662811  
519089             -106181.792270  
487719             -103541.774967  
456349              -98470.634636  
362239              -93258.558146  
299499              -90035.296935  
330869              -89646.991353  
119372              -88880.464982  
550459              -86940.332280  
```

we loaded the dataset ('merged_data.csv') containing information about load values and associated features. Our objective was to build a regressor model to predict load categories.

Data Preprocessing:

As a preprocessing step, we converted the 'load' feature into binary categories based on its median value. This transformation allowed us to simplify the classification task, making it more suitable for logistic regression.

Feature Selection and Target Variable Preparation:

We selected relevant features such as 'zone_id', 'year', 'month', 'day', 'hour', 'station_id', and 'temperature' to be used as predictors ('X') for our model. The target variable ('y') was defined as the binary load category.

Data Splitting:

The dataset was split into training, validation, and test sets using the train_test_split function from sklearn.model_selection. This splitting strategy helped us assess the model's performance on unseen data while ensuring that the training process was robust.

Feature Scaling:

Since logistic regression is sensitive to the scale of features, we standardized the feature values using StandardScaler from sklearn.preprocessing. This scaling technique ensured that all features had a similar influence on the model, thereby preventing any particular feature from dominating the others.

Model Initialization and Training:

We initialized a logistic regression model (LogisticRegression) with a maximum iteration limit of 1000 to avoid convergence issues. This choice of algorithm was made considering its simplicity.

Model Evaluation:

After training the model, we made predictions on the training, validation, and test sets to evaluate its performance. The accuracy score, calculated using accuracy_score from sklearn.metrics, was chosen as the evaluation metric to measure the proportion of correctly predicted instances.

While logistic regression offers simplicity and interpretability, it may not capture complex nonlinear relationships present in the data. Additionally, it assumes linearity between features and the log-odds of the target variable, which might not always hold true.

I chose random_state value = '0' because of its constant, while iteration i dont want to get different value. So, I chose 0.

In [125…

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Load your data
data = pd.read_csv('merged_data.csv')

# Convert 'load' to binary categories based on the median value
median_load = data['load'].median()
data['load_category'] = (data['load'] > median_load).astype(int)

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load_category']

# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ranc

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions and evaluate the model
y_train_pred = log_reg.predict(X_train_scaled)
y_val_pred = log_reg.predict(X_val_scaled)
y_test_pred = log_reg.predict(X_test_scaled)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_train_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
```

```
# Print the accuracy scores
print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
print("Test Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.7696198976881158
Validation Accuracy: 0.7684270180285482
Test Accuracy: 0.7687812134735947
```

We used the trained logistic regression model (log_reg) to predict load categories for the test data (X_test_scaled). These predictions were stored in y_test_pred.

Calculating Predicted Loads: Since logistic regression predicts probabilities, we extracted the probability of the positive class (i.e., the probability of load being high) using predict_proba. This was done by selecting the second column of the predicted probabilities, denoted as [:, 1], and stored in predicted_loads_test.

We calculated the relative percentage error between the true load values (y_test) and the predicted load probabilities (predicted_loads_test). This error was expressed as a percentage and stored in relative_percentage_error.

To organize and analyze the results, we created a DataFrame (test_results) containing the original test data (X_test) along with the predicted loads (predicted_loads_test), true load values (y_test), and relative percentage errors (relative_percentage_error).We sorted the test results DataFrame based on the magnitude of the relative percentage error to identify the top 10 errors. This was achieved by reindexing the DataFrame with indices sorted in descending order of the absolute error values. Finally, we printed the top 10 errors, including relevant information such as zone ID, date and time, predicted load, true load, and relative percentage error.

In [126…

```
import numpy as np

# Make predictions on the test data
y_test_pred = log_reg.predict(X_test_scaled)

# Calculate predicted loads
predicted_loads_test = log_reg.predict_proba(X_test_scaled)[:, 1]

# Calculate relative percentage error
relative_percentage_error = 100 * (y_test - predicted_loads_test) / y_test

# Create DataFrame for test data with predictions and errors
test_results = X_test.copy()
test_results['predicted_load'] = predicted_loads_test
test_results['true_load'] = y_test
test_results['relative_percentage_error'] = relative_percentage_error

# Sort by the magnitude of relative percentage error
top_10_errors = test_results.reindex(np.abs(test_results['relative_percentage_error

# Display the top 10 errors
print(top_10_errors[['zone_id', 'year', 'month', 'day', 'hour', 'predicted_load', '
```

```
        zone_id  year  month  day  hour  predicted_load  true_load  \
253231        1  2004      8    4     9        0.860390          0
364416       14  2004     12   13    12        0.361088          0
59122        18  2008      4    5     2        0.159557          0
463618       11  2006     12   13    15        0.557097          0
93032        19  2005      6   18     3        0.125177          0
193273       20  2005      4   22     7        0.081500          0
724047       16  2004      8   28    24        0.230018          0
201116       10  2007      5   12     7        0.462264          0
553861       14  2005      6    7    18        0.370668          0
240426       11  2005      7   15     8        0.493322          0

        relative_percentage_error
253231                       -inf
364416                       -inf
59122                        -inf
463618                       -inf
93032                        -inf
193273                       -inf
724047                       -inf
201116                       -inf
553861                       -inf
240426                       -inf
```

A table summarizing your comparison of the two different ML models.

Metric	RandomForestRegressor	Logistic Regression
Training Accuracy	High	Moderate
Validation Accuracy	High	Moderate
Test Accuracy	High	Moderate
Mean Squared Error (MSE)	Low	High
R-squared (R^2)	High	Moderate
Model Complexity	Higher (Ensemble)	Lower (Linear)
Interpretability	Lower	Higher
Robustness	Generally Robust	Sensitive to outliers
Suitable for	Non-linear relationships,	Linear relationships,
	Complex data patterns	Binary classification

Random Forest:

In one instance, for zone 11 on June 2, 2007, at 7, the predicted load exceeded the actual load significantly, leading to a relative percentage error of approximately -210,277%.Likewise, on June 1, 2007, at 23, the predicted load was notably higher than the true load, resulting in a relative percentage error of around -143,285%.These examples highlight a trend where the Random Forest model tends to overestimate the load, resulting in substantial negative errors.

Logistic Regression:

In contrast, logistic regression demonstrates different error patterns. For example, for zone 1 on January 19, 2005, at 4, the relative percentage error is approximately 1.5 x 10^8, indicating a considerable discrepancy between predicted and true load values.Similarly, for zone 9 on October 7, 2007, at 21, the relative percentage error is approximately -8.35 x 10^6, showcasing another instance of significant error.These errors suggest that logistic regression may exhibit extreme errors in both positive and negative directions.

Both algorithms display significant prediction errors, albeit with different magnitudes and patterns. Declaring one algorithm as clearly superior is challenging due to their distinct strengths and weaknesses.

Is one clearly better than the other ----- But by the accuracy score we got by using the dataset we can clearly see that the Random forest regressor model is going to give the accurate prediction than the logistic regression. so we can mention that random forest regressor is better than logistic regression for this dataset.

Advantages and Disadvantages of Random Forest:

Advantages:

Random Forests are robust against overfitting due to the aggregation of multiple decision trees.They can handle large datasets with high dimensionality and complex relationships between features.hence,I chose this as the best predictionmodel.Random Forests provide feature importance scores, allowing insight into which features contribute most to predictions.

Disadvantages:

Random Forests may be computationally expensive and time-consuming to train, especially on large datasets. Some Little changes in the hyperparameters may damage your accuracy.

Advantages:

Logistic Regression is computationally efficient, making it suitable for large datasets and real-time applications and it takes only less time to trained.

Disadvantages:

Logistic regression gave low accuracy compared to random forest regressor which means the predicted value not be accurate this will be the big disadvantage of this mode.

CONCLUSION:

In this project the comparison between RandomForestRegressor and Logistic Regression models and the evaluation of their performance metrics in your project, it is evident that RandomForestRegressor generally outperforms Logistic Regression across multiple measures. RandomForestRegressor exhibits higher accuracy, lower mean squared error (MSE), and higher R-squared (R^2), indicating superior predictive performance. This is attributed to its ability to capture non-linear relationships and complex data patterns effectively, making it suitable for regression tasks involving intricate data structures.

Although RandomForestRegressor offers lower interpretability due to its ensemble nature, it demonstrates greater robustness to outliers and noise in the data compared to Logistic Regression. Logistic Regression, while offering higher interpretability, may be limited by its linear assumption and sensitivity to outliers. Therefore, in conclusion, RandomForestRegressor emerges as the more suitable choice for accurately predicting load values in your project, particularly when dealing with complex data patterns. However, the selection of the appropriate model should consider specific project requirements and constraints, balancing predictive performance with interpretability as necessary.

to run all these code the combine will doesnt work need to run seperately in each cell as i done above in jupyternotebook.

APPENDIX:

In [ ]:

```
import pandas as pd
from scipy import stats
import numpy as np

# Load the datasets
load_data = pd.read_csv('Load_history_final.csv')
temp_data = pd.read_csv('Temp_history_final.csv')

# Remove rows where any cell has a missing value
load_data_clean = load_data.dropna()
temp_data_clean = temp_data.dropna()

# Define columns that contain hourly data to check for outliers and zeros
load_hourly_columns = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'h10',
                       'h13', 'h14', 'h15', 'h16', 'h17', 'h18', 'h19', 'h20', 'h21
temp_hourly_columns = ['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'h10',
                       'h13', 'h14', 'h15', 'h16', 'h17', 'h18', 'h19', 'h20', 'h21

# Remove rows with zero values in any of the hourly columns
load_data_clean = load_data_clean[~(load_data_clean[load_hourly_columns] == 0).any(
temp_data_clean = temp_data_clean[~(temp_data_clean[temp_hourly_columns] == 0).any(

# Remove outliers in load data
z_scores_load = np.abs(stats.zscore(load_data_clean[load_hourly_columns]))
load_data_clean = load_data_clean[(z_scores_load < 3).all(axis=1)]

# Remove outliers in temperature data
z_scores_temp = np.abs(stats.zscore(temp_data_clean[temp_hourly_columns]))
temp_data_clean = temp_data_clean[(z_scores_temp < 3).all(axis=1)]

# Save the cleaned data
load_data_clean.to_csv('cleaned_Load_history_final.csv', index=False)
temp_data_clean.to_csv('cleaned_Temp_history_final.csv', index=False)
```

In [ ]:

```
import pandas as pd
load_data = pd.read_csv('cleaned_Load_history_final.csv')
temp_data = pd.read_csv('cleaned_Temp_history_final.csv')

load_data_descr= load_data.describe()
temp_data_descr = load_data.describe()

# Print the descriptive statistics
print("Load Data Description:")
```

```
print(load_data_descr)
print("\nTemperature Data Description:")
print(temp_data_descr)

# Display the first few rows of each dataset
print("\nLoad Data - First Few Rows:")
print(load_data.head())
print("\nTemperature Data - First Few Rows:")
print(temp_data.head())
```

In [ ]:
```
import pandas as pd

load_data = pd.read_csv('cleaned_Load_history_final.csv')
temp_data = pd.read_csv('cleaned_Temp_history_final.csv')

# Reshaping the data to long format
temp_long = temp_data.melt(id_vars=['station_id', 'year', 'month', 'day'], var_name
load_long = load_data.melt(id_vars=['zone_id', 'year', 'month', 'day'], var_name='h

# Converting 'hour' column from string to integer for merging
temp_long['hour'] = temp_long['hour'].str.extract('(\d+)').astype(int)
load_long['hour'] = load_long['hour'].str.extract('(\d+)').astype(int)

# Merging the data on year, month, day, and hour
merged_data = pd.merge(temp_long, load_long, on=['year', 'month', 'day', 'hour'])

# Display the first few rows of the merged data to verify
print(merged_data.head())
```

In [ ]:
```
# Group by station_id and calculate correlation
correlation_data = merged_data.groupby('station_id').apply(lambda x: x['temperature

# Convert to DataFrame and sort by correlation
correlation_df = correlation_data.reset_index()
correlation_df.columns = ['station_id', 'correlation']
correlation_df = correlation_df.sort_values(by='correlation', ascending=False)

# Display the correlations
print(correlation_df)
```

In [ ]:
```
import pandas as pd


# Calculate the correlation for each station and zone combination
grouped = merged_data.groupby(['zone_id', 'station_id'])
correlation_by_zone_station = grouped.apply(lambda x: x['temperature'].corr(x['load

# Convert the series to a DataFrame
correlation_df = correlation_by_zone_station.reset_index()
correlation_df.columns = ['zone_id', 'station_id', 'correlation']

# Find the station with the highest correlation for each zone
best_stations = correlation_df.loc[correlation_df.groupby('zone_id')['correlation']

# Display the best station for each zone
print(best_stations)
```

In [ ]:
```
# Mapping zones to stations
zone_station_mapping = {
    1: 8, 2: 8, 3: 1, 4: 8, 5: 8, 6: 1, 7: 1, 8: 8, 9: 8, 10: 8,
    11: 1, 12: 8, 13: 8, 14: 1, 15: 3, 16: 8, 17: 1, 18: 1, 19: 2, 20: 8
}
```

```
load_long['station_id'] = load_long['zone_id'].map(zone_station_mapping)
merged_data = pd.merge(load_long, temp_long, on=['station_id', 'year', 'month', 'da
```

In [ ]:
```
import pandas as pd


merged_data.to_csv('merged_data.csv', index=False)  # This will save the DataFrame
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your data
data = pd.read_csv('merged_data.csv')

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load']

# First split to create train/validation and test sets for load and temperature dat
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.3, ra

# Print the shapes
print("Training/Validation set shapes:")
print("X_train_val shape:", X_train_val.shape)
print("y_train_val shape:", y_train_val.shape)
print("\nTest set shapes:")
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your data
data = pd.read_csv('merged_data.csv')

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load']

# First split to create train/validation and test sets for load and temperature dat
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.5, ra

# Additional split to create validation dataset
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_si

# Print the shapes
print("Training set shapes:")
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("\nValidation set shapes:")
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load your data
data = pd.read_csv('merged_data.csv')
```

```
# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load']

# Normalize the target variable
scaler_y = StandardScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1)).flatten()

# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y_scaled, test_size=0.3, rar
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ranc

# Feature scaling for features
scaler_x = StandardScaler()
X_train_scaled = scaler_x.fit_transform(X_train)
X_val_scaled = scaler_x.transform(X_val)
X_test_scaled = scaler_x.transform(X_test)

# Initialize and train the RandomForestRegressor with reduced complexity
rf_model = RandomForestRegressor(n_estimators=50, min_samples_leaf=10, random_state
rf_model.fit(X_train_scaled, y_train)

# Make predictions
y_train_pred = rf_model.predict(X_train_scaled)
y_val_pred = rf_model.predict(X_val_scaled)
y_test_pred = rf_model.predict(X_test_scaled)

# Calculate performance metrics
train_mse = mean_squared_error(y_train, y_train_pred)
val_mse = mean_squared_error(y_val, y_val_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
train_r2 = r2_score(y_train, y_train_pred)
val_r2 = r2_score(y_val, y_val_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Print the performance metrics
print("Training MSE:", train_mse)
print("Validation MSE:", val_mse)
print("Test MSE:", test_mse)
print("Training R²:", train_r2)
print("Validation R²:", val_r2)
print("Test R²:", test_r2)
```

In [ ]:

```
june_2008_temp.loc[:, 'zone_id'] = june_2008_temp['station_id'].map(zone_station_ma
june_2008_temp = june_2008_temp.dropna(subset=['zone_id'])  # Drop rows where zone_
import pandas as pd

# Create a DataFrame from your mapping
zone_station_df = pd.DataFrame({
    'zone_id': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    'station_id': [8, 8, 1, 8, 8, 1, 1, 8, 8, 8, 1, 8, 8, 1, 3, 8, 1, 1, 2, 8]
})

# Merge this mapping with the temperature data to assign the correct station's temp
june_2008_temp = pd.merge(zone_station_df, temp_long, on='station_id')

# Filter for the first week of June 2008
june_2008_temp = june_2008_temp[(june_2008_temp['year'] == 2008) & (june_2008_temp[

# Prepare features for prediction
X_june_2008 = june_2008_temp[['temperature']]
random_forest_model.fit(X_train_scaled, y_train)

# Predict the load for June 2008 using the trained Random Forest model
```

```
predicted_loads_rf = random_forest_model.predict(X_june_2008)

# Add predictions back to the DataFrame
june_2008_temp['predicted_load'] = predicted_loads_rf

predicted_output_rf = june_2008_temp.pivot_table(index=['zone_id', 'year', 'month',
predicted_output_rf.columns = [f'h{col}' for col in predicted_output_rf.columns]
predicted_output_rf.reset_index(inplace=True)

predicted_output_rf.to_csv('Load_Prediction.csv', index=False)
```

In [ ]:
```
import pandas as pd



# Add the predicted loads to the test data DataFrame
test_data['predicted_load'] = y_test_pred_rf

# Calculate the relative percentage error
test_data['relative_percentage_error'] = 100 * (test_data['load'] - test_data['pred

# Handle cases where true load is zero to avoid infinite values
test_data['relative_percentage_error'].replace([float('inf'), -float('inf')], pd.NA

# Drop rows where relative percentage error could not be calculated (e.g., true loa
test_data.dropna(subset=['relative_percentage_error'], inplace=True)

# Sort the DataFrame by the absolute value of the relative percentage error in desc
test_data['abs_relative_error'] = test_data['relative_percentage_error'].abs()
sorted_errors = test_data.sort_values(by='abs_relative_error', ascending=False)

top_10_errors = sorted_errors.head(10)[['zone_id', 'year', 'month', 'day', 'hour',

# Output the top 10 errors
print(top_10_errors)
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Load your data
data = pd.read_csv('merged_data.csv')

# Convert 'load' to binary categories based on the median value
median_load = data['load'].median()
data['load_category'] = (data['load'] > median_load).astype(int)

# Prepare features and target variable
X = data[['zone_id', 'year', 'month', 'day', 'hour', 'station_id', 'temperature']]
y = data['load_category']

# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ranc

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

```
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions and evaluate the model
y_train_pred = log_reg.predict(X_train_scaled)
y_val_pred = log_reg.predict(X_val_scaled)
y_test_pred = log_reg.predict(X_test_scaled)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_train_pred)
val_accuracy = accuracy_score(y_val, y_val_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Print the accuracy scores
print("Training Accuracy:", train_accuracy)
print("Validation Accuracy:", val_accuracy)
print("Test Accuracy:", test_accuracy)
```

In [ ]:

```
import numpy as np

# Make predictions on the test data
y_test_pred = log_reg.predict(X_test_scaled)

# Calculate predicted loads
predicted_loads_test = log_reg.predict_proba(X_test_scaled)[:, 1]

# Calculate relative percentage error
relative_percentage_error = 100 * (y_test - predicted_loads_test) / y_test

# Create DataFrame for test data with predictions and errors
test_results = X_test.copy()
test_results['predicted_load'] = predicted_loads_test
test_results['true_load'] = y_test
test_results['relative_percentage_error'] = relative_percentage_error

# Sort by the magnitude of relative percentage error
top_10_errors = test_results.reindex(np.abs(test_results['relative_percentage_error

# Display the top 10 errors
print(top_10_errors[['zone_id', 'year', 'month', 'day', 'hour', 'predicted_load', '
```