Table of Contents

List of Figures

List of Tables

Project Overview

The final project focusses mainly on the entire life cycle of a machine learning task right from data preprocessing, data cleaning, data mapping, data correlation and then to training a couple of machine learning models to understand the correlation between the data files given.

This correlation is ultimately used to train a set of machine learning models for them to predict a week worth of load data using the learning from temperature correlation. The concept here is, that the data being input into the system is an instance from the real-world set of data available.

This adds a challenge to the process since there is a requirement to process the data and provide results within a certain timeframe, however, on the contrary, the size of the data is huge and to deal with big data in a certain limited timeframe by also providing a higher accuracy of prediction needs a machine learning model to be robust.

Data Preprocessing

The data from both files namely 'Load_history_final.csv' and 'Temp_history_final.csv' contain a lot of values that are zero. This would lead to misleading the correlation phase since these zero values add no practical significance to the raw data and the ratio of increase or decrease to the data in relation between the Load and Temperature data.

To avoid this, the zero values are ignored during the correlation phase and not discarded entirely however, to preserve the raw data integrity. The logic used for the preprocessing of the data is pictorially represented below with the explanation of each stage following the image.

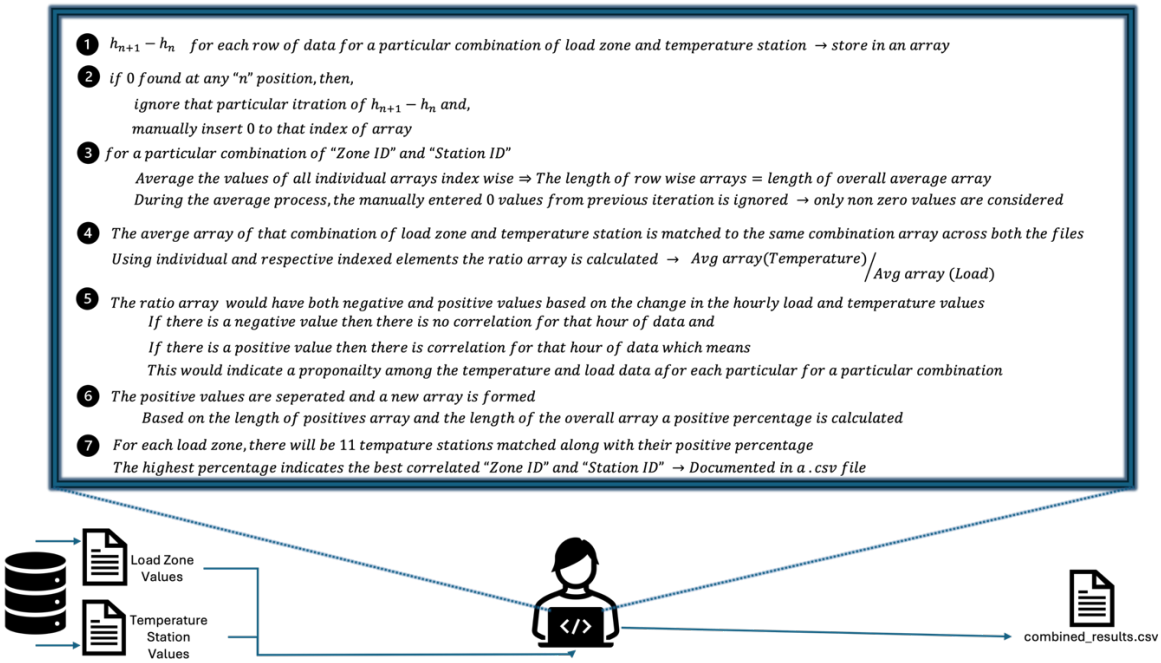

Figure 1: Logic behind the correlation between the Load Zones and Temperature Stations.

Data Correlation Logic

Stage 1

The correlation first begins by understanding how the values are varying each hour since that is the lowest level of the data available for prediction. The difference calculated for each hour of the data for every row there is an array created. These row wise arrays can be looked at as day wise hourly value differences. This is done for both the Temperature and Load data files.

Stage 2

The data files however need cleaning since it is found that there are abrupt 0 values in the data which can hinder the model's understanding of the correlation and mislead the model during the training phase, leading to poor efficiency of model's prediction capability.
To counter this, the logic is designed to ignore that particular iteration of the difference calculation and to fill the place of that indexed position of the array, the code automatically fills that value of the array with a 0 (NaN).

Stage 3

For each unique value of "Zone ID" and "Station ID", the entire list of arrays is averaged to each indexed value of the array, i.e. the length of the individual arrays is the same and thus, the average array consisting of averaged values in each of its indices of values is of the same length of each of the row wise arrays.
During this process, the 0 values from each row wise arrays is ignored to avoid loss of significance in the data. So, the average is calculated for all the non-zero elements only, so the ratio of change remains rich without the interference of manipulated and abrupt data. This will ensure a certain amount of robustness assistance to the model through a cleaned database.

Stage 4

At this stage, the individual and separate calculations of overall average arrays from Load values are linked with Temperature values. The ratio between each combination is extracted, i.e., The individual values from the overall average temperature array for the particular "Station ID" is divided by each particular indexed element from the overall average load array for the particular "Zone ID".
The resultant array is stored for that combination of "Zone ID" and "Station ID". Since we know that there are 20 load zones and 11 temperature stations, there will be a total of 220 arrays.

Stage 5

The increase or decrease in each hourly consecutive value ensures either a positive or negative value in the initial array. This sign is carried through the entire process as well. If there is a similar increase or decrease in the hourly data across both the Temperature and Load data files, then the ratio array for a set combination would be positive. Otherwise, it would indicate a negative value which means that the data for that set combination of "Zone ID" and "Station ID" is inversely proportional.

Stage 6

The positive values from this each ratio are copied onto another positive array for that set combination and similarly the negative values are copied onto another array for that set combination. The percentage of positives and negatives for each set combination is calculated based on the length of the positive and negative array to the length of the overall ratio array of that set combination. This will indicate the correlation from each Load Zone to Temperature Station.

Stage 7

There are 20 Load Zones in total along with 11 Temperature stations. Since we look at ratios of each load zone to each temperature station to find the best correlation. The result is the combination of 220 ratios and 220 sets of positive and negative percentages.
The code, also find the highest correlation positive percentage from a zone ID to station ID and documents that that Load Zone affects that particular Temperature Station. This signifies the mapping between the combination, which will further aid the model training and prediction thereafter.

Load and Temperature Mapping

The Table 1 below provides direct mapping explaining the mapping of Zone ID to the Station ID.

Zone ID	Station ID
1	5
2	1
3	1
4	5
5	7
6	3
7	5
8	10
9	5
10	4
11	5
12	5
13	5
14	5
15	5
16	2
17	5
18	1
19	4
20	5

Table 1: Zone ID and Station ID mapped matched values.

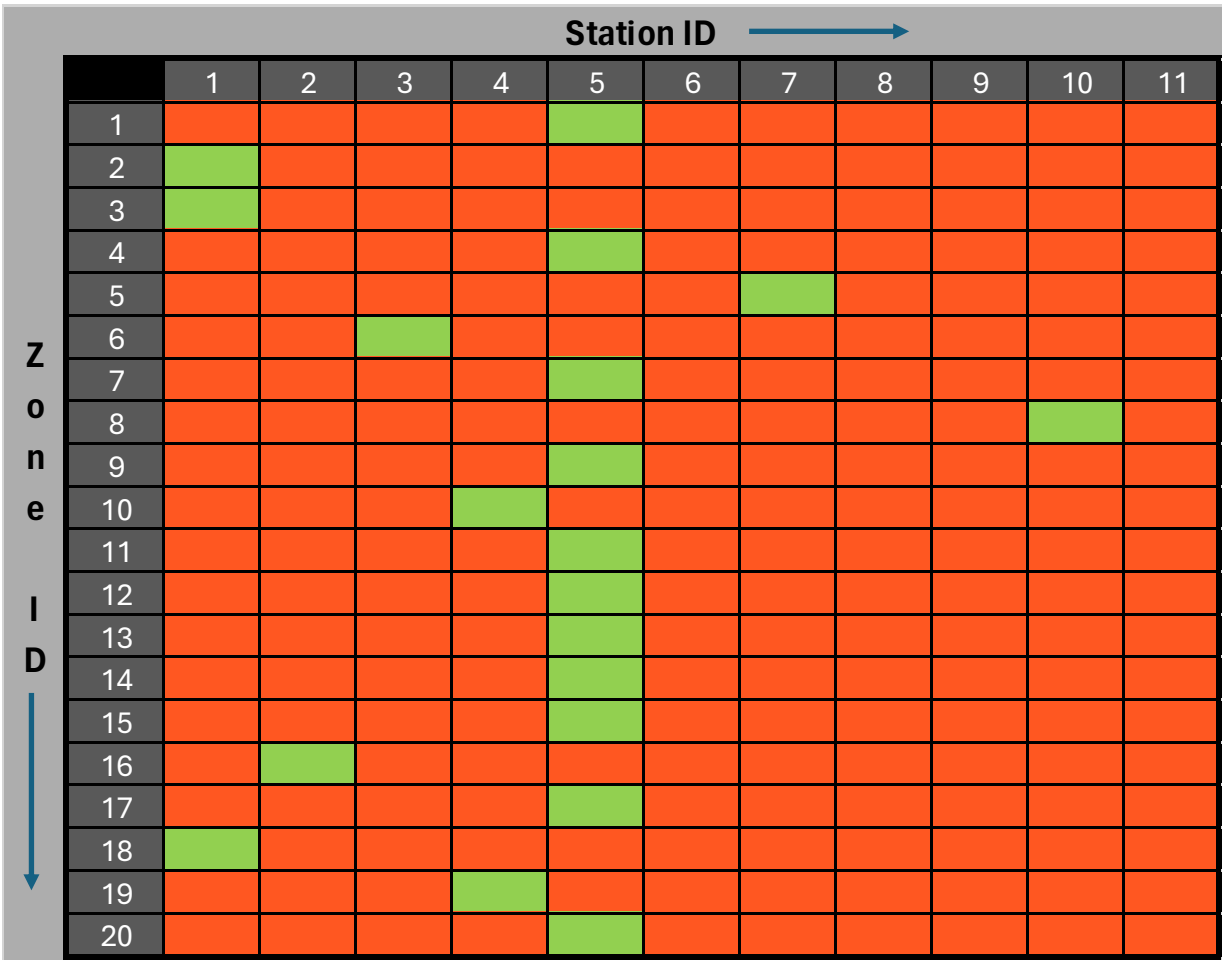

Table 2: Correlation between Load Zones and Temperature Stations.

Table 2 indicates an interactive way to describe the correlation mapping between the Zone ID and Station ID. The Green box indicates that there is a correlation between the two nodes (ex. There is a correlation between Zone ID 6 and Station ID 3), and the red boxes indicate that there is no significant correlation between the two nodes (ex. Zone ID 10 and Station ID 5).

Machine Learning

Design Process and Trade Offs

The following are the Design Processes:

1. Time Counter and Suppress Warnings:

At the start of the program, a start timer is initiated to keep track of the time elapsed from start to end of the program execution.
The program would also throw many warnings owing to the multiple dimensions in the big data.

2. Data Loading:

The data is then loaded onto the data frame that is then used to perform operations. This data come from the initial files for Load Values and Temperature Stations. There is also an addition of a third file, which would be an input that is the mapped correlated data file.

3. Data Preprocessing:

Since, the data contains a lot of 0 values abruptly, this can affect the learning curve of any training model. Thus, preprocessing is conducted to ensure that these values are ignored during the data processing.

4. Data Correlation:

At this stage, the correlated and mapped Load Zones along with Temperature Stations is initiated and enabled for further operations upon the raw dataset.

5. Data Merging:

The correlated data is merged from both the Load Values file and the Temperature Stations file onto a merged data frame for easier data operations.

6. Sample Validation

At this stage, to cross verify the correctness of the functioning of the mapping from the mapped file by the code, a sample portion of the mapping to be used/used is displayed to check manually about the rightfulness of the mapping.

7. Train-Test-Split and Model Training

The model is now trained and used for testing based on a certain ratio of percentage of split of the data. The model is thus trained, thereafter.

8. Model Evaluation:

A model's effectiveness is measured based on any of the scoring metrics that are used. This will help grade the model's effectiveness.

Correlation Model

Code

THE CODE FOR THE CORRELATION MODEL IS ATTACHED IN THE MODEL 1-CODE PORTION OF THE APPENDIX OF THIS DOCUMENT.

Model 1

The choice of model 1 for this task is chosen to be performed by the Logistic Regression algorithm. This is to provide a linear model that is better than the basic yet un-complex enough to test its performance upon the real-world large dataset.

Code

THE CODE FOR THE MODEL 1 LEARNING AND PREDICTION IS ATTACHED IN THE MODEL 1-CODE PORTION OF THE APPENDIX OF THIS DOCUMENT.

Model 2

The choice of model 2 for this task is chosen to be performed by the Random Forest algorithm. This is to provide a dedicated complex approach in enabling the algorithm to approach the big data through multiple depths and thus aiding in prediction to greater accuracies.

Code

THE CODE FOR THE MODEL 2 LEARNING AND PREDICTION IS ATTACHED IN THE MODEL 2-CODE PORTION OF THE APPENDIX OF THIS DOCUMENT.

Models Comparison

Logistic Regression	Random Forest Regressor
Advantages	
Simple and Linear	Versatility
Interpretability	Reduces Overfitting
Efficient for Binary Classification	Handles both categorical and continuous data
Disadvantages	
Assumes Linearity	Computational Complexity
Sensitive to outliers	May overfit noisy data
Limited to Binary Classification	Difficulty in tuning hyperparameters

Project Conclusion

This project enabled the opportunity to explore real-world problems and to think through a perspective that approaches the problem statement from the logical reasoning to ultimately obtaining solution through learning systems.

The real-world problem posed an incredible problem in terms of the big data being presented. This data needed cleaning to begin with and then a correlation had to be established based on limited fields. Constructing a fool-proof logic to establish the correlation and to use this correlation to perform operations on the data through multiple models, ensures that stern dedication to quality of work is of highest importance to this entire project.

Through the work done upon this project, the key takeaways were that the sensitivity towards failure of a big data model sway very slightly when data is smoothened to a small extent. A model, irrespective of its simplicity, the big data upon which this model works define the model's effectiveness in terms of efficiency and time complexity.

Overall, the entire project and the course collectively hones and deepens the depth of understanding and enables the opportunity to create and work hands on real-word big data problems.

Appendix

Correlation Model

```
import pandas as pd
import numpy as np
import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")

# Step 1: Read Data and Clean
load_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024 Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Load_history_final.csv")
temp_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024 Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Temp_history_final.csv")

# Replace 0 values with NaN
load_data.replace(0, np.nan, inplace=True)
temp_data.replace(0, np.nan, inplace=True)

# Initialize list to store results
results = []
best_stations = []

# Step 2: Iterate Over Combinations
for zone_id in load_data['zone_id'].unique():
    best_station = None
    best_positive_percentage = 0

    for station_id in temp_data['station_id'].unique():
        # Step 3: Match Dates
        merged_data = pd.merge(load_data[load_data['zone_id'] == zone_id],
                               temp_data[temp_data['station_id'] == station_id],
                               on=['year', 'month', 'day'])

        if merged_data.empty:
            continue

        # Step 4: Calculate Differences and Ratios
        for i in range(1, 25):
            load_col = f'h{i}_x'
            temp_col = f'h{i}_y'
            diff_load_col = f'diff_load_{i}'
            diff_temp_col = f'diff_temp_{i}'
            ratio_col = f'ratio_{i}'

            # Calculate differences, considering NaN values
            merged_data[diff_load_col] = merged_data[load_col] - merged_data[load_col].shift(1)
            merged_data[diff_temp_col] = merged_data[temp_col] - merged_data[temp_col].shift(1)

            # Replace NaN differences with 0
            merged_data.loc[merged_data[diff_load_col].isnull(), diff_load_col] = 0
            merged_data.loc[merged_data[diff_temp_col].isnull(), diff_temp_col] = 0

            # Calculate ratio
            merged_data[ratio_col] = merged_data[diff_temp_col] / merged_data[diff_load_col]

        # Step 5: Aggregate Ratios
        merged_data['average_ratio'] = merged_data[[f'ratio_{i}' for i in range(1, 25)]].mean(axis=1, skipna=True)

        # Step 6: Separate Positive and Negative Ratios
        positive_ratios = merged_data[merged_data['average_ratio'] > 0]['average_ratio'].tolist()

        # Step 7: Calculate Positive Percentage
        total_count = len(merged_data)
        positive_percentage = (len(positive_ratios) / total_count) * 100

        # Store results for each combination of zone_id and station_id
        results.append({'Zone ID': zone_id,
                        'Station ID': station_id,
                        'Positive Percentage': positive_percentage})

        # Check if current station has higher positive percentage than the best so far
        if positive_percentage > best_positive_percentage:
            best_station = station_id
            best_positive_percentage = positive_percentage

    # Store best station for each zone
    best_stations.append({'Zone ID': zone_id,
                          'Best Station ID': best_station,
                          'Positive Percentage': best_positive_percentage})

# Convert results to DataFrame
results_df = pd.DataFrame(results)
best_stations_df = pd.DataFrame(best_stations)

# Merge results and best stations into a single DataFrame
combined_df = pd.merge(results_df, best_stations_df, on='Zone ID', suffixes=('_Result', '_Best'))

# Export combined results to a single CSV file
combined_df.to_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024 Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Code/combined_results.csv", index=False)

print("Combined results exported to combined_results.csv")
```

Model 1

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
import warnings
import time

#Measure overall execution time
start_time = time.perf_counter()

# Suppress warnings
warnings.filterwarnings("ignore")

# Step 1: Data Loading
load_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024 Semester/ECE563 (AI for
Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Load_history_final.csv")
temp_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024 Semester/ECE563 (AI for
Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Temp_history_final.csv")
combined_results = pd.read_csv("combined_results.csv")

# Step 2: Data Preprocessing
# Replace 0 values with NaN
load_data.replace(0, np.nan, inplace=True)
temp_data.replace(0, np.nan, inplace=True)

# Remove rows with NaN values
load_data.dropna(inplace=True)
temp_data.dropna(inplace=True)

# Step 3: Data Correlation
# Extract unique zone IDs and corresponding best station IDs from combined_results
zone_to_station = combined_results.drop_duplicates(subset=['Zone ID'])[['Zone ID', 'Best Station ID']]

# Step 4: Data Merging
# Merge load_data and temp_data based on zone_id and station_id
merged_data = pd.merge(load_data, zone_to_station, left_on='zone_id', right_on='Zone ID', how='inner')
merged_data.drop(columns=['Zone ID'], inplace=True)  # Drop redundant column

# Step 5: Sample Validation
# Choose a few random samples from merged_data and cross-check with combined_results.csv
# For example, select 5 random rows and validate the mapping
sample_rows = merged_data.sample(n=5, random_state=42)
print("\nSample Validation:")
for index, row in sample_rows.iterrows():
    # Check if 'station_id' is present in the row
    if 'station_id' in row:
        station_id = row['station_id']
    else:
        # Use the correct column name if it's different
        station_id = row['Best Station ID']

    zone_id = row['zone_id']
    # Ensure 'Zone ID' is used if 'zone_id' is not present in row
    if 'zone_id' in row:
        zone_id = row['zone_id']
    else:
        zone_id = row['Zone ID']

    best_station_id = zone_to_station[zone_to_station['Zone ID'] == zone_id]['Best Station ID'].values[0]
    print(f"Zone ID: {zone_id}, Station ID: {station_id}, Best Station ID: {best_station_id}")
print()

# Step 6: Train-Test Split
X = merged_data.drop(columns=['zone_id', 'year', 'month', 'day', 'Best Station ID'])
y = merged_data[['h1']]  # Select one of the hour columns for training

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 7: Model Training
model = LogisticRegression()
model.fit(X_train, y_train.values.ravel())  # Convert y_train to 1D array using ravel()

# Step 8: Model Evaluation
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
mean_cv_score = np.mean(cv_scores)
print("Mean Cross-Validation Score:", mean_cv_score)

y_pred = model.predict(X_test)
overall_deviation = mean_squared_error(y_test, y_pred)
print("Overall Deviation:", overall_deviation)
print()

#Measure the overall execution time
end_time = time.perf_counter()
execution_time = end_time - start_time
print(f"Overall Execution Time: {execution_time:0.4f} seconds")
print()
```

Model 2

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import warnings
import time
from itertools import product

# Measure overall execution time
start_time = time.perf_counter()

# Suppress warnings
warnings.filterwarnings("ignore", message="It seems that frozen modules are being used", category=UserWarning)
warnings.filterwarnings("ignore", message="Debugger warning", category=UserWarning)



# Step 1: Data Loading
load_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024
Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Load_history_final.csv")
temp_data = pd.read_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024
Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Temp_history_final.csv")
combined_results = pd.read_csv("combined_results.csv")

# Step 2: Data Preprocessing
# Replace 0 values with NaN
load_data.replace(0, np.nan, inplace=True)
temp_data.replace(0, np.nan, inplace=True)

# Remove rows with NaN values
load_data.dropna(inplace=True)
temp_data.dropna(inplace=True)

# Rename 'Zone ID' column to 'zone_id' in combined_results DataFrame
combined_results.rename(columns={'Zone ID': 'zone_id'}, inplace=True)

# Step 3: Data Merging
# Merge load_data and temp_data based on zone_id
merged_data = pd.merge(load_data, combined_results, on='zone_id', how='inner')

# Step 4: Train-Test Split
X = merged_data.drop(columns=['zone_id', 'year', 'month', 'day', 'Best Station ID'])
y = merged_data[['year', 'month', 'day', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'h7', 'h8', 'h9', 'h10', 'h11', 'h12',
'h13', 'h14', 'h15', 'h16', 'h17', 'h18', 'h19', 'h20', 'h21', 'h22', 'h23', 'h24']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Model Training for Each Hour
hourly_models = {}
for hour in range(1, 25):
    model = RandomForestRegressor(n_estimators=100, max_depth=7, n_jobs=-1)
    model.fit(X_train, y_train[f'h{hour}'])
    hourly_models[f'h{hour}'] = model

# Step 6: Prediction for Each Hour
hourly_predictions = {}
for hour, model in hourly_models.items():
    y_pred = model.predict(X_test)
    hourly_predictions[hour] = y_pred

# Step 7: Combine Hourly Predictions for Entire Population
population_predictions = pd.DataFrame(hourly_predictions)

# Step 8: Export Predictions to CSV
population_predictions.to_csv("/Users/Kashyap/Documents/Files/Academics/Institutions/Masters(USA)/IIT/Spring 2024
Semester/ECE563 (AI for Smart Grid)/AI-in-Smart-Grid/Projects/Final Project/Code/population_predictions1.csv",
index=False)

# Measure the overall execution time
end_time = time.perf_counter()
execution_time = end_time - start_time
print(f"Overall Execution Time: {execution_time:0.4f} seconds")
```