SI206 Final Project

Visualizing Carbon Intensity vs. Electricity Costs and Environmental Implications in the UK

vincoding.vn
Huy Dung Lou, Jessical Tran

GitHub Repository: https://github.com/huylou/SI206_finalproj

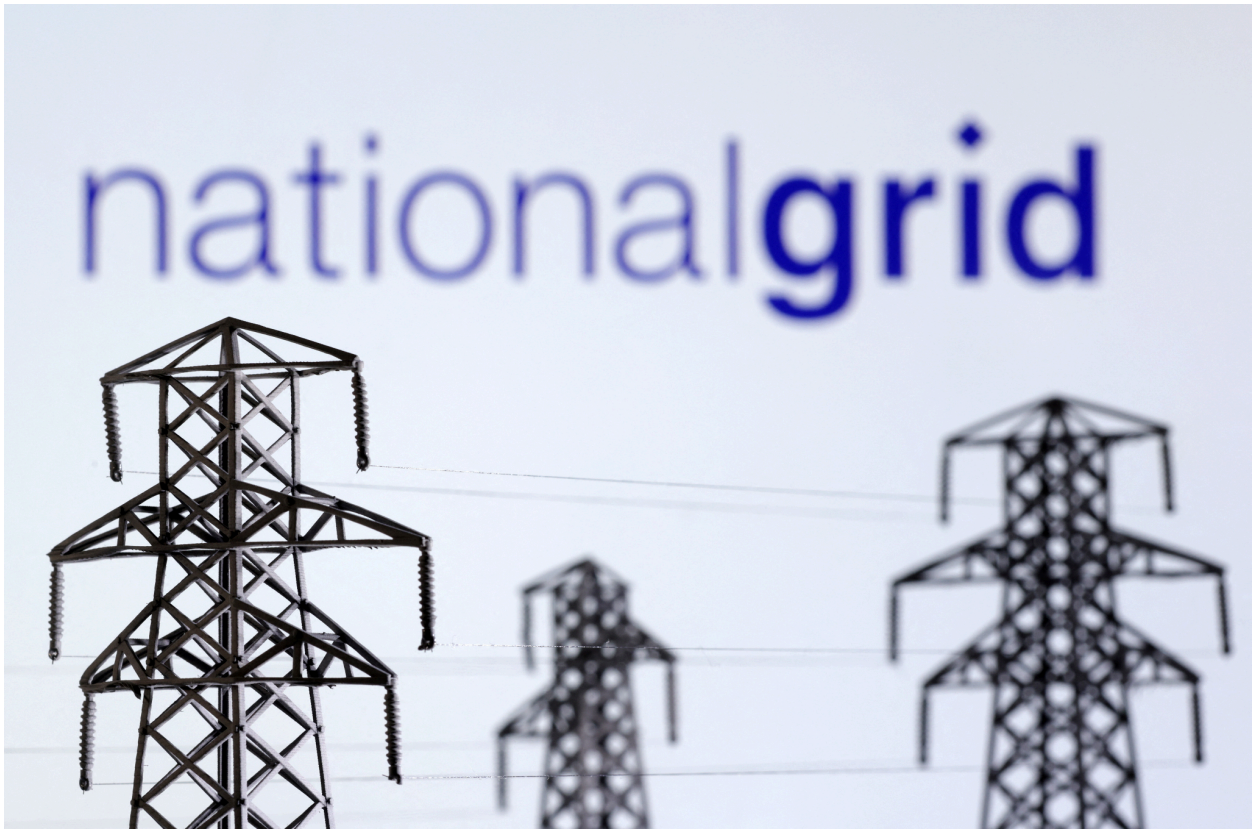


Table of Contents

Original Proposal

Our original plan proposal was to look at how investment in urban infrastructure may impact urbanization trends and socioeconomic development in Vietnam. We wanted to use the World Bank API to collect socioeconomic data, infrastructure expenditure, transportation projects, housing development, and utility provision. For our second API, we wanted to use either MapBox or OpenStreetMap to collect geospatial data to analyze urban infrastructure distribution and accessibility in different regions of Vietnam. During our initial start of the project, however, we ran into two main issues: 1) the planned APIs did not have the specific data we would have liked for countries, and 2) Vietnam is a developing country and lacked data for us to access in general. Therefore, we decided to pivot to a project that focused on a more developed region, the United Kingdom, and looked at commonly collected and accessible data, environmental metrics.

Final Project Plan and Goals

The goals of the project we settled on aimed to investigate sustainable uses of energy in the United Kingdom. In order to do so, we looked at recorded carbon intensity and electricity costs at regional levels based on Distribution Network Operator (DNO) regions. We planned to use the National Grid Carbon Intensity API to obtain the carbon intensity levels (gCO2/kWh) and the Electricity Cost API to obtain information about the cost of electricity (£). Both APIs had data grouped by timestamps in 30 minute intervals, allowing us to label timestamps as a shared integer key.

Additionally, we saw that the National Grid Carbon Intensity API included energy generation mixes including gas, coal, biomass, nuclear, hydro, wind, solar, imported sources, and others on a national scale. We decided to use that data as well for a visualization of energy generation mixes in the UK. This further helped us understand our original question of how does the UK utilize energy sustainably. Where we originally aimed to look at carbon intensity levels correlated with electricity use, now we additionally see the breakdown of energy sources for the electricity use.

Problems Faced

In coding our project, we faced a few main issues:

1. **Finding appropriate APIs that are aligned with our interests**: We had an overarching goal of looking at a country and finding data that would allow us to create geospatial visualizations. This is why we had to switch from our original topic of urbanization in Vietnam to find data that was more readily accessible and could be used for visualization.
2. **Mass amounts of data because of the way data was collected from the APIs:** We were collecting data from a time range of one week from 23 regions in the UK and the data provided was organized in 30 minute intervals. This meant when we originally

pulled the data, we had data for each region at 30 minute intervals for a full week, estimating to around 4700 lines of data. Hence, we narrowed our search to look at London specifically for calculations.

3. **Using JOIN**: We originally had a separate database for carbon intensity and electricity costs when we realized that JOIN only works for different tables in the same database. Therefore, we had to create a new database with carbon intensity and electricity costs as separate tables. From there, there was slight confusion on how to use JOIN with timestamps as the related column that established the relationship between tables but only grabbing data for the London region. Eventually, we figured out how to use INNER JOIN, ON, and WHERE clauses to join the carbon intensity and electricity costs for only the London region data.
4. **Standardizing data formats and keeping variables consistent:** The data we pulled from the two APIs were similar in content, but varied in structure. For example, they differed in time range formats and region notations that we had to standardize in our database. Furthermore, many functions were dependent on the same variable names, so we made sure to keep variables consistent throughout.

Calculations

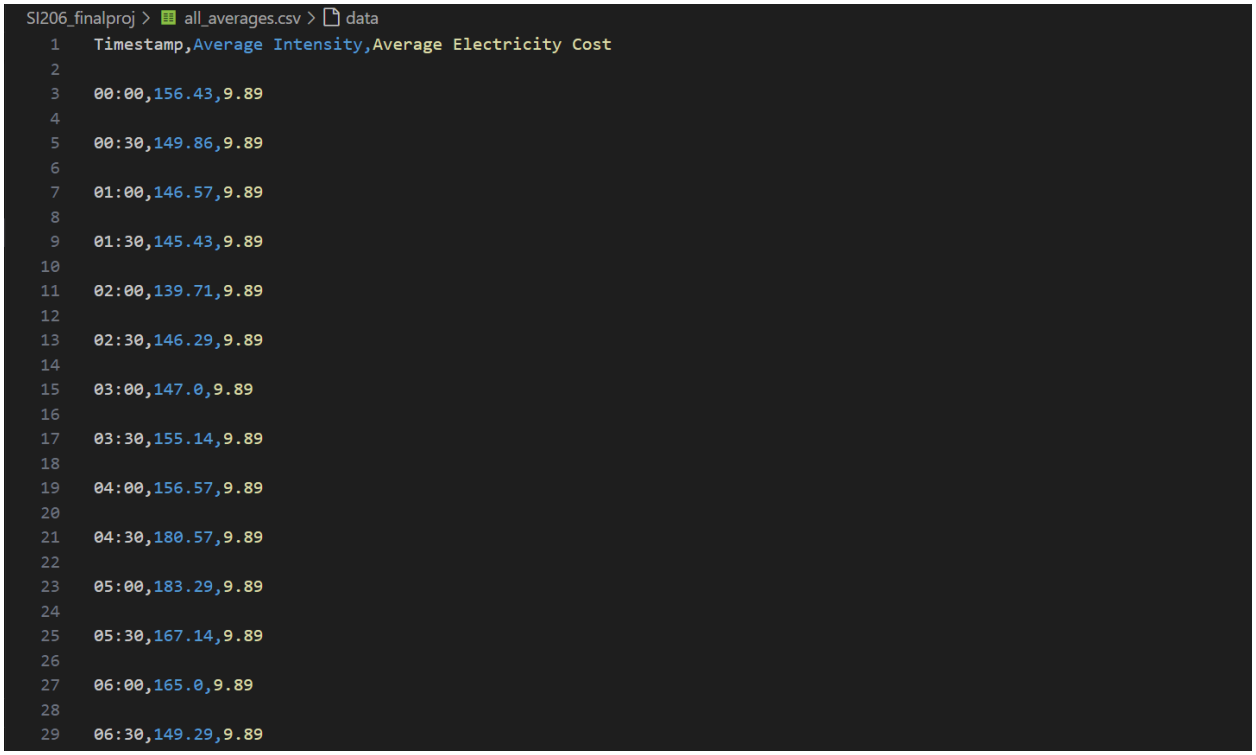
Carbon Intensity API

```
def calculate_average_carbonintensity_region(cur, dnoregion):
    cur.execute(f"SELECT Intensity_Forecast, DNO_Region FROM Carbon_Intensity_Data WHERE DNO_Region = ?", (dnoregion,))
    lst = cur.fetchall()
    total = 0
    count = 0
    for tup in lst:
        total += tup[0]
        count += 1
    avg = round(total / count, 2)
    return (tup[1], avg)
```

```
def avg_cost_intensity_calculation_region(cur, dnoregion):
    cur.execute(f'''
                SELECT Carbon_Intensity_Data.Time, Carbon_Intensity_Data.Date, Carbon_Intensity_Data.Intensity_Forecast, Electricity_Costs.Price_per_KWh
                FROM Carbon_Intensity_Data
                INNER JOIN Electricity_Costs ON Carbon_Intensity_Data.Timestamp = Electricity_Costs.Timestamp
                WHERE Electricity_Costs.DNO_Region = '{dnoregion}'
                ''')
    lst = cur.fetchall()
    d = {}
    avg_dict = {}
    for tup in lst:
        if tup[0] not in list(d.keys()):
            d[tup[0]] = {'intensity_forecast': tup[2], 'price': tup[3], 'count': 1}
        else:
            d[tup[0]]['intensity_forecast'] += tup[2]
            d[tup[0]]['price'] += tup[3]
            d[tup[0]]['count'] += 1

    for timestamp, accumdict in d.items():
        avgintensity = round(float(accumdict['intensity_forecast'] / accumdict['count']),2)
        avgprice = round(float(accumdict['price'] / accumdict['count']),2)
        avg_dict[timestamp] = {'average_intensity': avgintensity, 'average_price': avgprice}

    return avg_dict
```

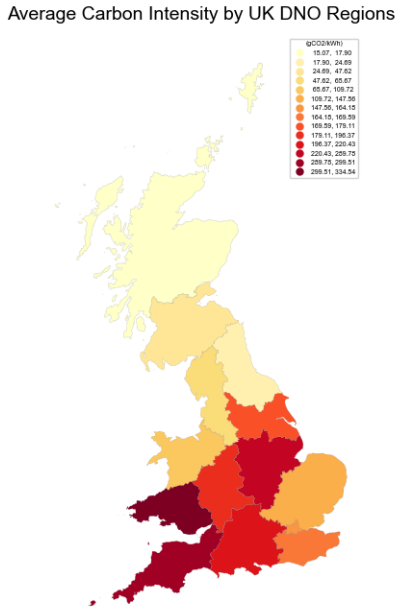
Output of All Averages in London in CSV File

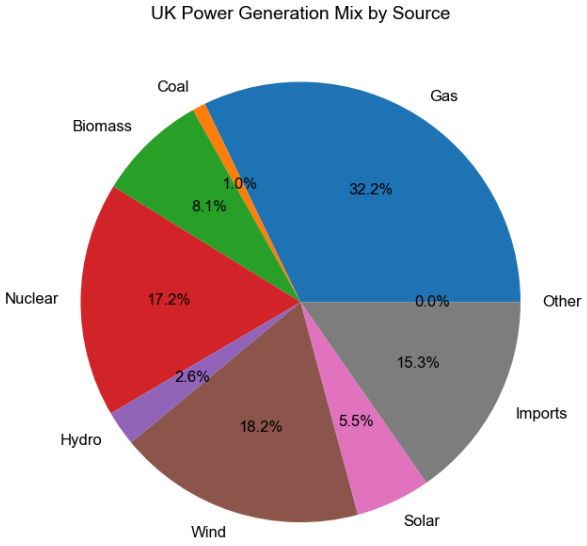
SI206_finalproj > all_averages.csv > data

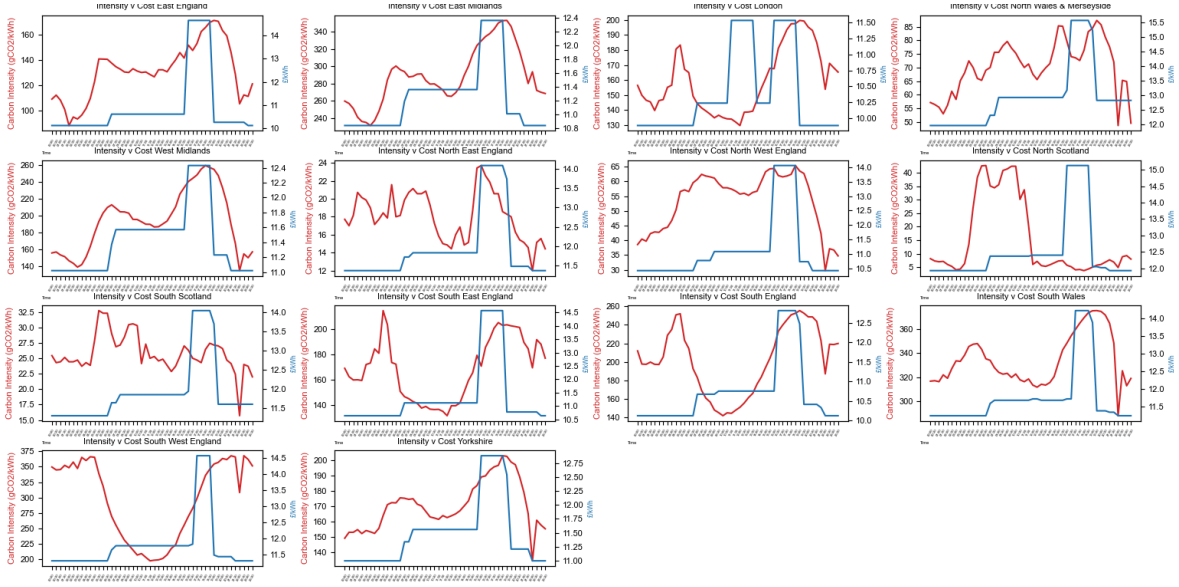


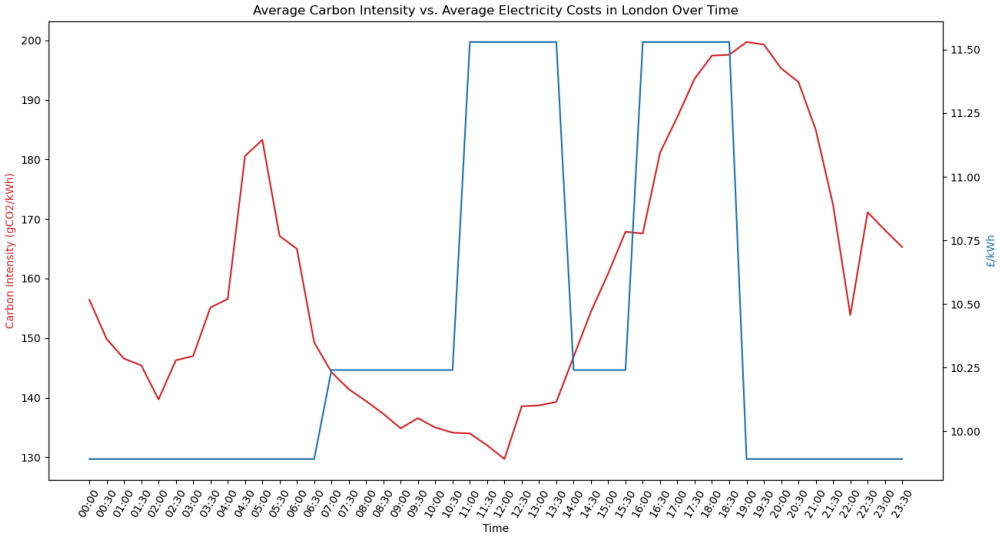
```
Timestamp,Average Intensity,Average Electricity Cost

00:00,156.43,9.89

00:30,149.86,9.89

01:00,146.57,9.89

01:30,145.43,9.89

02:00,139.71,9.89

02:30,146.29,9.89

03:00,147.0,9.89

03:30,155.14,9.89

04:00,156.57,9.89

04:30,180.57,9.89

05:00,183.29,9.89

05:30,167.14,9.89

06:00,165.0,9.89

06:30,149.29,9.89
```

Visualizations









Instructions for Running Code

Step One: Press Run on visualizations.py… That's it!

We have three coding files in total: carbon_intensity.py, electricity_costs.py, and visualizations.py. The first two, carbon_intensity.py, electricity_costs.py, set up the databases for each API, but they do not need to be run separately because visualizations.py imports them as modules. We also set up a loop to repeatedly run the code until all data points have been obtained for the database. Therefore, to run the code and capture all data points even with the 24 item limit, the code only needs to be run on visualizations.py once!

Function Documentation

carbon_intensity.py

def api_request(startdate, enddate, regionid):
Arguments: Takes a start and end date in string form in the format YYYY-MM-DD and a regional id based on:

1 North Scotland
2 South Scotland
3 North West England
4 North East England
5 Yorkshire
6 North Wales
7 South Wales
8 West Midlands
9 East Midlands
10 East England
11 South West England
12 South England
13 London
14 South East England
15 England
16 Scotland
17 Wales

Returns: The data from the website for that day in 30 minute intervals in json format

def create_carbon_intensity_table(r, cur, conn):
Arguments: Takes the json output from the API request
Returns: None (creates Carbon_Intensity_Data Database)

def calculate_average_carbonintensity_region(cur, dnoregion):
Arguments: Takes in the UK region name
Returns: A tuple of (region name, average carbon intensity)

def create_generationmix_database(datadict, cur, conn):
Arguments: Takes the json output from the API request
Returns: None (creates Generation_Mix_Data Database)

def calculate_average_generationmix(cur):
Arguments: None, just the cursor connection
Returns: A list of tuples in the format of (energy source, national average percentage of generation mix)

electricity_costs.py

def get_electricity_costs_dict(dnonum, voltagelevel, start, end):
Goal: Calls UK Electricity Costs API, returns json dictionary containing timestamp and price/kWh data based on a half-hourly (HH) interval
Arguments: Takes DNO Region Numbers based on:

DNO Region Numbers (int):
- 10 – Eastern England
- 11 – East Midlands
- 12 – London
- 13 – North Wales
- 14 – West Midlands
- 15 – North East
- 16 – North West
- 17 – Northern Scotland
- 18 – Southern Scotland
- 19 – South East
- 20 – Southern
- 21 – South Wales
- 22 – South Western
- 23 – Yorkshire

And voltage levels based on:

Voltage Levels (str):
- HV - High Voltage
- LV - Low Voltage
- LV-Sub - Low Voltage - Sub

And the start and end dates in the string format of DD-MM-YYYY

Returns: a json dictionary

def retrieve_price_and_timestamp(jsondict, dnodict):
Goal: Retrieve overall price per kWh and timestamp (XX:XX DD-MM-YYYY)
Arguments: Takes in the Json dictionary of electricity costs API and the dictionary containing {id number: region name}
Returns: A list of tuples (Price, Time, Date, Timestamp) containing the overall price of the timestamp and date (DD-MM-YYYY)

def create_electricitycost_table_with_limit(datalist, cur, conn):
Goal: Uses functions to pull data from electricity costs API and organize into data structure to create a database storing electricity costs data, with a data injection limit of 24
Arguments: Takes in a list of tuples (Price, Time, Date) from Electricity Costs API
Returns: None (Database table created or updated)

def calculate_average_electricity_costs(cur):

Goals: Calculates average electricity costs per timestamp
Arguments: None, just cursor connection
Returns: List of Tuples (Timestamp, Average Price per kWh)

visualizations.py

def set_up_database(db_name):
Goal: Sets up a SQLite database connection and cursor.
Arguments: The name of the SQLite database in string
Returns: A tuple containing the database cursor and connection.

def intensity_avg_csv_writer(cur, dnodict):
Goal: Write carbon intensity average by region to a CSV file.
Arguments: Takes in the dictionary containing {id number: region name}
Returns: None (creates avg_intensity_dnoregions.csv file)

def avg_cost_intensity_calculation_region(cur, dnoregion):
Goal: Calculate the averages for electricity cost and carbon intensity for reach timestamp
Arguments: Takes in the region name
Returns: A nested dictionary of {timestamp: {average_intensity: value, average_price: value}}

def averages_as_text_csv_file(cur, data):
Goal: Write averages for carbon intensity and electricity costs at each timestamp into a CSV file.
Arguments: Takes in the nested dictionary output of avg_cost_intensity_calculation_region()
Returns: None (creates all_averages.cvs file)

def avg_cost_to_intensity_linechart_dnoregion(cur, dnoregion):
Goal: Visualize averages of carbon intensity and electricity costs (y-axes) at each timestamp (x-axis) for London specific in a line chart
Arguments: Takes in the region name
Returns: None (creates graph)

def avg_cost_to_intensity_linechart_alldnoregion(cur, dnodict):
Goal: Visualize averages of carbon intensity and electricity costs (y-axes) at each timestamp (x-axis) for London specific in a line chart
Arguments: Takes in the dictionary containing {id number: region name}
Returns: None (creates graph)

def generation_mix_piechart(cur):
Goal: Visualize the national average of generation mixes into a pie chart
Arguments: None, just the cursor connection
Returns: None (creates pie chart)

def avg_intensity_uk_dnoregion_geospatial(dno_csv, geojson):

Goal: Visualize average carbon intensity by DNO regions through a choropleth geospatial map
Arguments: csv file, geojson file
Returns: None, creates a choropleth map

Resource Documentation

Date	Issue Description	Location of Resource	Result (did it solve the issue)
2024/04/13	Need Reference to API Documentation for Carbon Intensity	National Grid Carbon Intensity API	Very helpful in providing examples for API data structure output
2024/04/13	Need Reference to API Documentation for Electricity Costs	Imperial College London Electricity Costs API	Very helpful in providing examples for API structure output
2024/04/20	Need assistance for SQLite syntaxes	Stack Overflow	Slightly hard to find solutions tailored to our project, but was able to synthesize inputs for our problems
2024/04/27	Learning Matplotlib documentation	Matplotlib	Helpful documentation in producing line charts and pie charts
2024/04/28	Learning GeoPandas documentation	Geopandas	Helpful documentation in producing choropleth map