SI 206 Final Project Report:

Analyzing the Impact of Temperature on COVID-19 and Flu Trends in Michigan and Nationwide

[GitHub Final Project Repository]
Fall 2024 – Data-Oriented Programming
Tharron Combs and Ashley Xu

Original Research Question:
Research question: How does temperature affect flu or COVID-19 cases in Michigan and nationwide?

1. Goals (20 Points)

Original Goals Based on the Research Question:

1. **Overview of project goals:**
    - Retrieve data from three APIs: Delphi Epidata, **Covid Act Now API**, and Meteostat JSON API.
    - Analyze trends in flu (`num_ili` from Delphi API) and COVID-19 case counts alongside temperature data (TAVG fom Meteostat JSON API).
    - Create visualizations to demonstrate seasonal trends and relationships between temperature and illness.
2. **Data Collection:**
    - Retrieve and store data on weekly flu cases (measured by `num_ili`) in Michigan and nationally.
    - Retrieve and store data on weekly confirmed COVID-19 cases in Michigan and nationally.
    - Retrieve and store daily temperature data for Michigan, including average temperature (TAVG), maximum temperature (TMAX), and minimum temperature (TMIN).
3. **Data Integration and Analysis:**
    - Correlate flu cases (weekly `num_ili`) in Michigan with temperature trends (TAVG) over time.
    - Correlate COVID-19 cases with temperature data in Michigan over time.
    - Investigate relationships between national trends in flu cases and temperature data.
4. **Seasonal Trends:**
    - Identify seasonal trends in flu (`num_ili`) and COVID-19 case counts in Michigan.

    - Use visualizations to show whether temperature fluctuations have a correlation with flu and COVID-19 trends in reported cases.
5. **Visualization and Reporting:**
    - Create clear visualizations showing relationships between temperature and flu/COVID-19 case counts.
    - Highlight seasonal patterns for flu and COVID-19 case trends in Michigan and nationally.

Achieved Goals

1. **Overview of achieved goals**
    - Successfully retrieved, processed, and stored weekly flu, COVID-19, and temperature data in an SQLite database.
    - Analyzed weekly flu trends and correlated them with temperature in Michigan.
    - Generated visualizations for flu trends and temperature relationships, achieving clear insights into the impact of temperature on public health (measured by flu and COVID-19 case trends).
2. **Data Retrieval and Storage:**
    - Retrieved weekly flu case data (`num_ili` and `wili`) for Michigan and nationally from the Delphi Epidata API.
    - Retrieved daily COVID-19 case data for Michigan and nationally from the COVID-19 Statistics API.
    - Retrieved daily temperature data (TAVG, TMAX, TMIN) for Michigan from the Climate Data Online (CDO) API.
    - Stored all data in an SQLite database with shared keys for integration (`date` and `region`).
3. **Analysis and Calculations:**
    - Calculated weekly averages for flu (`num_ili`) and COVID-19 cases
    - Calculated average weekly temperature by averaging daily temperature
    - Identified seasonal trends in flu cases and COVID-19 case counts in Michigan and nationally.
4. **Visualizations:**

- Used **line charts** to represent data from March 2020 to March 2022:
    - Used shaded blue and red backgrounds to identify the summer and winter months
        - Created one line chart for weekly new flu cases trends in Michigan and nationally.
        - Created one line chart for weekly new COVID-19 cases trends in Michigan and nationally.
    - Created one line chart showing the average national weekly temperature
    - Created one line chart showing the average Michigan weekly temperature

5. **Reporting and Statistical Interpretation:**

    - Visually presented the results of the data compiled from APIs, including displaying the number of new flu and COVID-19 cases and pointing out trends during winter and summer months.

2. Problems Faced (10 Points)

Data Retrieval Challenges:

- Our initial chosen APIs for COVID-19 cases and weather data posed difficulties in retrieval and analysis:
    - The COVID-19 API we originally chose from was poorly documented, making data extraction and processing cumbersome.
    - The Climate Data Online (CDO) API we originally chose from the National Centers for Environmental Information (NCEI) presented challenges due to its aggressive rate limits, which significantly slowed the retrieval of Michigan temperature data.
        - This API also had an overly complicated structure and did not support geo-location-based data. It provided regional averages instead of state-specific data, which conflicted with our project's focus on Michigan.
- Determining a national average for weather data using the weather API was challenging. We addressed this by selecting major cities across NCEI-defined climatic regions and calculating an average temperature. These cities were chosen based on their large populations and regional representation to reflect national trends accurately.

Data Processing Challenges:

- Aligning dates across the datasets retrieved from different APIs (flu cases, COVID-19 cases, and weather) was a significant challenge. Each API used different date formats and time zones. To resolve this, we standardized all data to Coordinated Universal Time (UTC) format to ensure data uniformity in the database for accurate analysis.
    - For example, the Delphi Epidata API gave us data on the number of new flu cases in each epiweek (or epidemiological week which is typically measured/numbered by the first full week of the year)
- COVID-19 data was provided as cumulative totals, requiring preprocessing to convert the data into weekly counts for trend analysis.
- Aligning flu and temperature data required careful filtering to account for missing or incomplete temperature records, ensuring data accuracy for the analysis.

Visualization Challenges for Line Charts:

We encountered difficulties with deciding on the best methods to visually represent data from different APIs (e.g., flu trends, COVID-19 trends, and temperature correlations)

To best represent our data in a digestible, intuitive format, we addressed these challenges by:

- Adjusting x-axis tick labels to focus on key months (March, June, September, December)
- Using commas for thousands (of flu or COVID-19 cases) on the y-axis to improve clarity and readability.
- Using dots to represent the number of cases or distinct temperature from each week of the year
- Choosing appropriate colors for line plots (e.g., flu and COVID-19 trends of new cases)
    - We used shaded background bars to highlight the winter and summer months of the year, using red to represent the summer (warmer) months and blue to represent winter (colder) months
    - We used these colors to illustrate an intuitive and readable understanding of our data

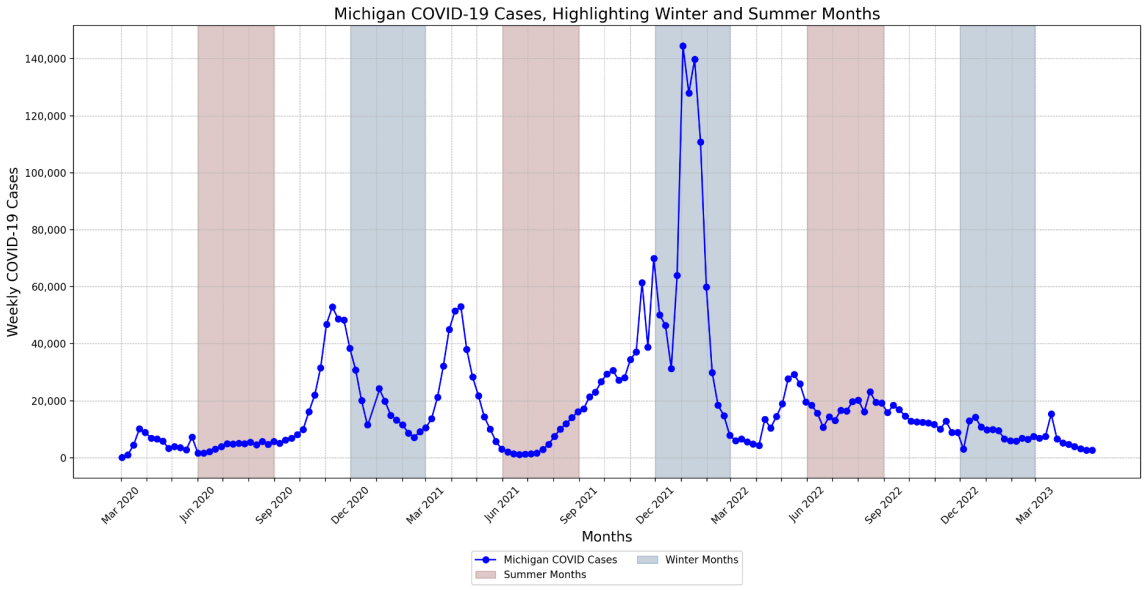
3. Calculation File (10 Points)

Link:
https://github.com/duckgandalfsaxophone/SI206FinalProject/blob/4a7ea00fc7c6e71e46d5b25f75e0c47e59ab4def/data_collection.py

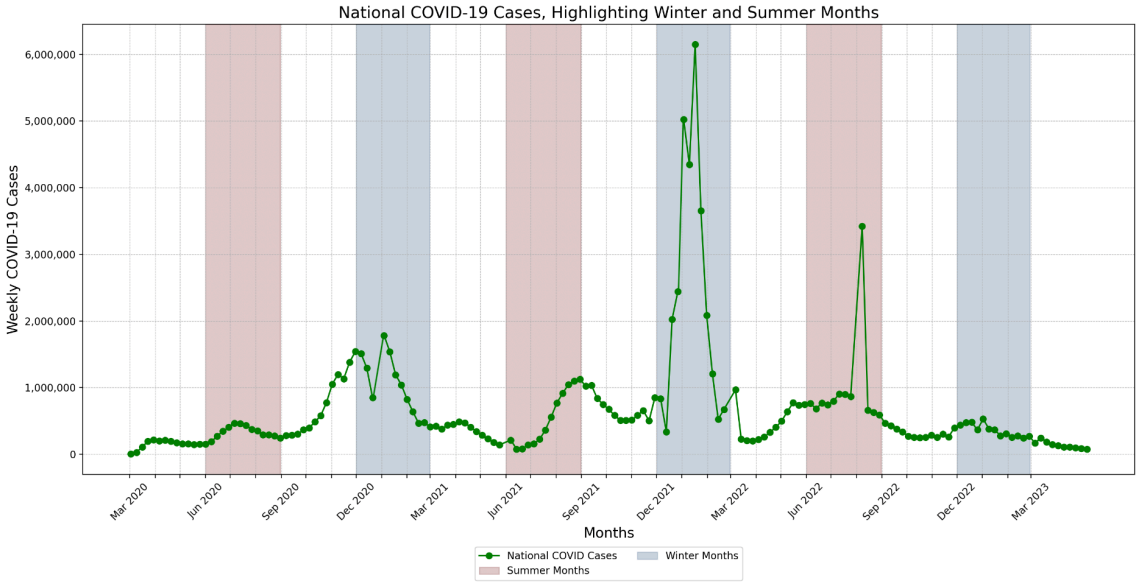
The file contains all calculations, including weekly flu trends, COVID-19 case averages, and averages calculated for temperature data.

4. Visualizations Created (10 Points)

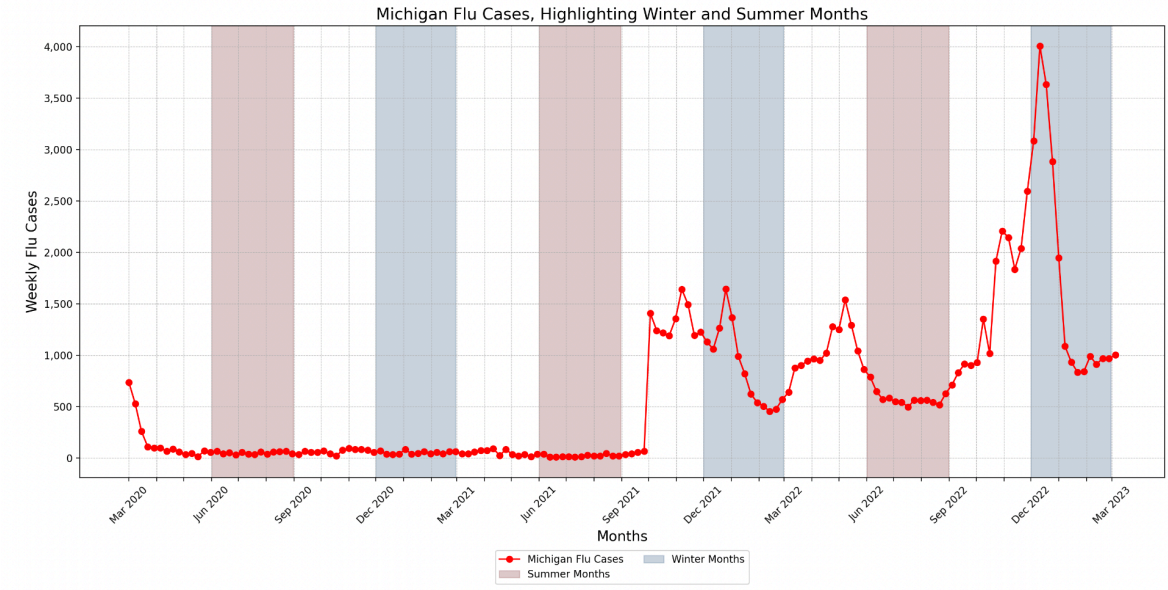
Visualization 1: Michigan Weekly New COVID-19 Cases, Highlighting Winter and Summer Months



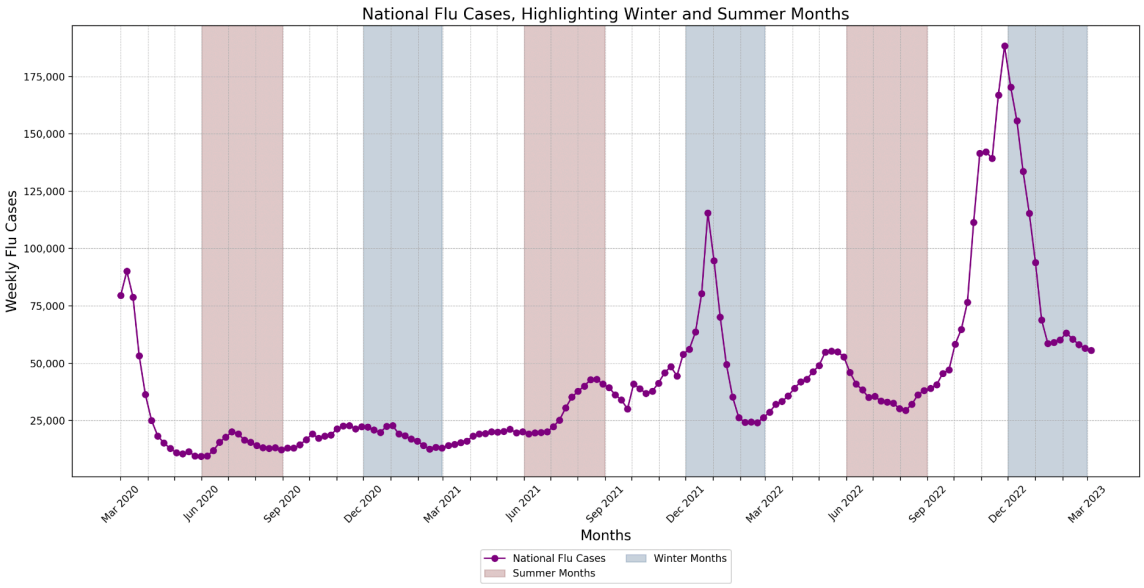
Visualization 2: Michigan Weekly New Flu Cases, Highlighting Winter and Summer Months



Visualization 3: National Weekly New COVID-19 Cases, Highlighting Winter and Summer Months



Visualization 4: National Weekly New Flu Cases, Highlighting Winter and Summer Months



5. Instructions for Running the Code (10 Points)

Dependencies:

```
pip install pandas matplotlib sqlite3 epiweeks requests meteostat

from meteostat import Point, Daily

from epiweeks import Week
```

Steps to Run the Code:

- **Open the main.py file**
- Ensure final_project.db is in the same directory as the notebook.
- Run the files in sequential order: data_collection.py, then data_visualization.py, then main.py.
- **Expected Outputs:**
- SQLite tables for flu, COVID-19, and temperature data.
- Line charts for flu and COVID-19 trends alongside temperature correlations.

6. Code Documentation (20 Points)

get_db_connection

- Purpose: Establishes and returns a connection to the SQLite database.

- Input Parameters: None
- Returns: A SQLite connection object (sqlite3.Connection) to interact with the database.
- Explanation: This function uses the sqlite3 module to create a connection to the final_project.db database, enabling SQL queries and data manipulation.

get_week_id(date)

- Purpose: Converts a date into a unique identifier for the year and week number.
- Input Parameters:
    - date (str): A date string in the format YYYY-MM-DD.
- Returns: An integer (int) representing the week ID (e.g., 202301 for the first week of 2023).
- Explanation: The function parses the input date using datetime.strptime and formats it to extract the year and week number using strftime.

process_weather_data(location, start_date, end_date, table_name)

- Purpose: Fetches daily weather data for a specified location and stores weekly aggregated data in the database.
- Input Parameters:
    - location (meteostat.Point): Geographical coordinates of the location.
    - start_date (datetime): Start date for weather data retrieval.
    - end_date (datetime): End date for weather data retrieval.
    - table_name (str): Name of the database table where processed data will be stored.
- Returns: None
- Explanation: Uses the Daily class from the meteostat package to fetch weather data, calculates weekly averages (temperature min, max, and avg), and stores the results in a specified database table.

process_all_cities()

- Purpose: Processes and stores weather data for predefined cities into respective database tables.
- Input Parameters: None
- Returns: None
- Explanation: Iterates through the predefined NATIONAL_LOCATIONS, calls process_weather_data for each city, and stores the results in individual city tables.

calculate_national_weather_average()

- Purpose: Calculates and stores national average temperatures (min, max, and avg) from city weather data.
- Input Parameters: None
- Returns: None

- Explanation: Combines weather data from multiple city tables, calculates the average temperature values, and stores the results in the national_weather table.

fetch_and_store_michigan_covid()

- Purpose: Fetches Michigan COVID-19 timeseries data and stores weekly aggregated data in the database.
- Input Parameters: None
- Returns: None
- Explanation: Uses the COVID Act Now API to retrieve Michigan COVID-19 data, processes daily cases into weekly totals, and stores them in the database.

fetch_and_store_national_covid()

- Purpose: Fetches national COVID-19 timeseries data and stores weekly aggregated data in the database.
- Input Parameters: None
- Returns: None
- Explanation: Similar to fetch_and_store_michigan_covid, but retrieves data for the entire U.S.

store_covid_data(data, table_name)

- Purpose: Processes raw COVID-19 data and saves weekly aggregated data to a specified table in the database.
- Input Parameters:
    - data (dict): JSON data containing COVID-19 timeseries.
    - table_name (str): Name of the database table to store processed data.
- Returns: None
- Explanation: Extracts daily cases from the raw data, calculates weekly totals, and stores the results in the database.

fetch_and_store_flu_data()

- Purpose: Fetches weekly flu data from the Delphi Epidata API and stores it in the database.
- Input Parameters: None
- Returns: None
- Explanation: Retrieves flu data for Michigan and the U.S., processes epiweeks into dates, and stores the data in the database.

collect_all_data()

- Purpose: Coordinates the collection of weather, COVID-19, and flu data.
- Input Parameters: None
- Returns: None

- Explanation: Sequentially calls functions to process weather data for Michigan and cities, compute national averages, and fetch/store COVID-19 and flu data.

format_with_commas(x, pos)

- Purpose: Formats numbers with commas for better readability in plots.
- Input Parameters:
    - x (float): The number to format.
    - pos (int): The tick position on the axis (used internally by Matplotlib).
- Returns: A string (str) with the number formatted with commas.
- Explanation: Used as a helper function for axis formatting in plots.

set_monthly_xticks(ax, start_date, end_date)

- Purpose: Sets custom x-axis ticks for a date range, labeling only specific months.
- Input Parameters:
    - ax (Axes): Matplotlib Axes object to modify.
    - start_date (str): Start of the date range (YYYY-MM-DD).
    - end_date (str): End of the date range (YYYY-MM-DD).
- Returns: None
- Explanation: Customizes x-axis labels for visual clarity in time-series plots.

plot_data(df, x_column, y_column, title, ylabel, label, color, start_date, end_date)

- Purpose: Creates a line plot of time-series data with formatted axes.
- Input Parameters:
    - df (DataFrame): DataFrame containing the data to plot.
    - x_column (str): Column for the x-axis.
    - y_column (str): Column for the y-axis.
    - title (str): Title of the plot.
    - ylabel (str): Label for the y-axis.
    - label (str): Legend label for the plot.
    - color (str): Color of the plot line.
    - start_date (str): Start date for the plot.
    - end_date (str): End date for the plot.
- Returns: None
- Explanation: Generates a simple time-series line plot with a formatted x-axis and labeled y-axis.

plot_cases_with_bars(df, x, y, label, color, title, ylabel, start_date, end_date, seasons)

- Purpose: Plots time-series data with seasonal highlight bars.
- Input Parameters:
    - df (DataFrame): DataFrame containing the data to plot.
    - x (str): Column for the x-axis.
    - y (str): Column for the y-axis.

    - label (str): Legend label for the line plot.
    - color (str): Line color.
    - title (str): Plot title.
    - ylabel (str): Label for the y-axis.
    - start_date (str): Start date for the x-axis.
    - end_date (str): End date for the x-axis.
    - seasons (list): List of tuples defining seasonal bars (start, end, label).
- Returns: None
- Explanation: Adds visual emphasis to specific seasonal periods on a time-series plot.

visualize_all_data()

- Purpose: Generates all visualizations, including COVID-19 and flu cases, with seasonal highlights.
- Input Parameters: None
- Returns: None
- Explanation: Iteratively generates plots for Michigan and national COVID-19 and flu data while incorporating seasonal highlights.

setup_database()

- Purpose: Ensures the database is correctly set up for the pipeline.
- Input Parameters: None
- Returns: Boolean (True if successful, False otherwise).
- Explanation: Verifies database connectivity and table existence.

collect_data()

- Purpose: Runs the data collection phase of the pipeline.
- Input Parameters: None
- Returns: Boolean (True if successful, False otherwise).
- Explanation: Orchestrates multiple data-fetching and processing functions.

create_visualizations()

- Purpose: Generates all visualizations in the analysis pipeline.
- Input Parameters: None
- Returns: Boolean (True if successful, False otherwise).
- Explanation: Calls the visualize_all_data function to generate insights from the collected data.

7. Documentation of Resources Used (20 Points)

Delphi Epidata API

- **Source:** FluView API
- **Purpose:** Collected weekly flu case data (num_ili, wili) for Michigan and nationally.

Covid Act Now API

- **Source:** COVID-19 API
- **Purpose:** Retrieved COVID-19 case data (daily confirmed cases) for Michigan and the nation.

Meteostat JSON API

- **Source:** Meteostat API
- **Purpose:** Gathered daily temperature data (`TAVG`, `TMIN`, `TMAX`) for Michigan.

Libraries Used:

- `pandas`: Data manipulation and analysis.
- `matplotlib`: Visualization creation.
- `sqlite3`: Database management.

8. Conclusion

The project successfully met its goals by collecting, processing, and analyzing data to investigate average temperatures and new flu and COVID-19 cases in Michigan and nationwide. The visualizations we created provided insights into how temperature fluctuations shifted with disease trends, achieving our original research objectives. Future expansions may include additional statistical tests to measure more specific correlations.