Goodthink Report

Jordan Miller, Joseph King, Taylor Woodard

10/06/2024

Contents

1 Introduction **3**

2 Data Analysis **4**

2.1 Preliminary pesticide Concentration Map 4

2.2 Butterfly Sighting Data 4

2.3 EPA Data 5

2.4 Correlation: Butterfly Sightings v Air quality 5

2.5 Pesticide Data Program inferencing 6

2.6 Correlation: Butterfly Sightings v Pesticide Use 6

3 Results **7**

4 Conclusion **12**

5 Appendix **13**

Chapter 1

Introduction

We were tasked with analyzing the possible correlation between Monarch Butterfly populations, based on public sighting data and air quality levels and pesticide used in agriculture. From the given data, we see possible trends to be explored in regards to the fluctuation in Butterfly sightings which may lead to a more sustainable approach to how we, as humans, interact with the environment around us. This report is not meant to give any definitive answers as to the effects that we are having on our environment, specifically the Monarch Butterfly population, but rather to provide a starting point for further research and analysis.

Chapter 2

Data Analysis

2.1 Preliminary pesticide Concentration Map

Using codap.concord.org we were able to create a map of the pesticide concentration in the United States. This map is a preliminary look at the data and will be further analyzed in the results section. (see Figure 3.1)

2.2 Butterfly Sighting Data

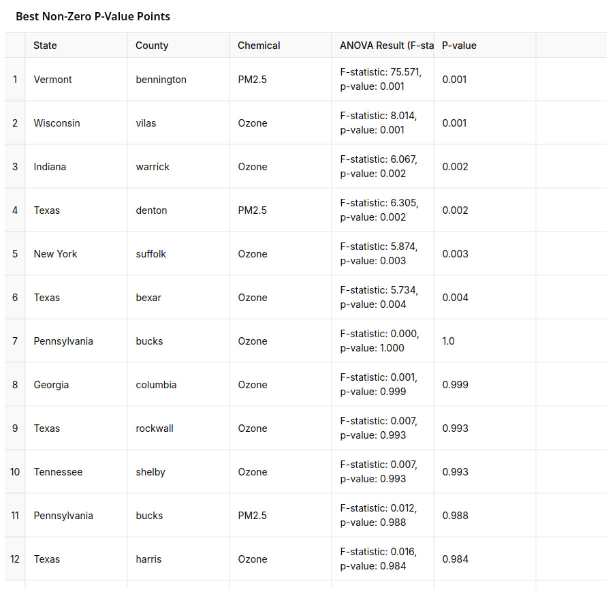
Due to the nature of the data, we decided to break the data down into smaller, more digestible pieces of data. This allowed us to compare the data within states to other states and to the overall data set. Furthermore, given the mapping of butterfly sightings, given by Journey North, we decided to narrow our analysis down into the following states: Alabama, Arkansas, Connecticut, Delaware, Florida, Georgia, Illinois, Indiana, Iowa, Kentucky, Louisiana, Maine, Maryland, Massachusetts, Michigan, Minnesota, Mississippi, Missouri, New Hampshire, New Jersey, New York, North Carolina, Ohio, Pennsylvania, Rhode Island, South Carolina, Tennessee, Vermont, Virginia, West Virginia, and Wisconsin. This is due to the fact that these states had the greatest density of sightings. As we progressed, we noticed a trend in the data provided by Journey NorthFigure 3.3 Figure 3.4. As the histograms we see, that the data appears to be heavily left skewed. Our claim is that this particular wesbite gained more popularity, likely due to abundant access to technology, in persuing spotting Monarch Butterflies in the wild, and thus more data is available for the later years of the data set. The next intuition into this data was the behavior of the sightings, relative to the time of the year, specifically the month. We noticed that the northern states only had one periodFigure **??** of sigtings during the middle to later months of the year, whille the southern statesFigure 3.6 seem to get two visits from our colorful winged friends. This is likely due to the migration patterns of the Monarch Butterflies. Lastly, we were able to establish a baseline for finding a trend in the sightings of the butterflies over the years. This was done by creating a time series plotFigure 3.7 Figure 3.8, which further solidified the claim that the number of visits depends on your longitudinal location in the United States.

2.3 EPA Data

Using the EPA's preconfigured Air Quality Index (AQI) data, we were able to analyze the air quality in the United States. Using the same states as before, we were able to create a time series plot of the AQI dataFigure 3.9 Figure 3.10. This data showed us that there was actually a decline in the AQI data over the years, which is a good sign for the environment. Our assumptions for this decline are that the EPA has been able to enforce stricter regulations on the emissions of pollutants into the air. This is a good sign for the environment and the Monarch Butterflies, as they are very sensitive to the environment around them.

2.4 Correlation: Butterfly Sightings v Air quality

Unfortunately, we were unable to produce any graphical representation of the correlation between the Butterfly sightings and the AQI data. This is due to the fact that the the EPA data showed a decline and therefore, graphically, made no indication that the quality of the air had any effect on the Monarch Butterfly population.(Figure 3.11) To further investigate this, we looked at the concentration of the chemicals found in the EPA data and performed ANOVA tests and used the p-score to determine if there was a significant difference in the concentration of the chemicals in the air and the number of sightings of the Monarch Butterflies. This was done for each state and the overall data set, but for the sake of brevity, we chose results from both ends of the spectrum: Given



Best Non-Zero P-Value Points

	State	County	Chemical	ANOVA Result (F-sta	P-value	
1	Vermont	bennington	PM2.5	F-statistic: 75.571, p-value: 0.001	0.001	
2	Wisconsin	vilas	Ozone	F-statistic: 8.014, p-value: 0.001	0.001	
3	Indiana	warrick	Ozone	F-statistic: 6.067, p-value: 0.002	0.002	
4	Texas	denton	PM2.5	F-statistic: 6.305, p-value: 0.002	0.002	
5	New York	suffolk	Ozone	F-statistic: 5.874, p-value: 0.003	0.003	
6	Texas	bexar	Ozone	F-statistic: 5.734, p-value: 0.004	0.004	
7	Pennsylvania	bucks	Ozone	F-statistic: 0.000, p-value: 1.000	1.0	
8	Georgia	columbia	Ozone	F-statistic: 0.001, p-value: 0.999	0.999	
9	Texas	rockwall	Ozone	F-statistic: 0.007, p-value: 0.993	0.993	
10	Tennessee	shelby	Ozone	F-statistic: 0.007, p-value: 0.993	0.993	
11	Pennsylvania	bucks	PM2.5	F-statistic: 0.012, p-value: 0.988	0.988	
12	Texas	harris	Ozone	F-statistic: 0.016, p-value: 0.984	0.984	

Figure 2.1: ANOVA test results

our understanding of the p-score, we can see that the chemicals in the air showed signs of possible correlation with the number of sightings of the Monarch Butterflies, but due to the lack of consistency in the data, per state per county per year, further analysis of the data is needed to make any definitive claims.

2.5 Pesticide Data Program inferencing

The PDP data does not include any information on county in its pesticide testing observations. While it is very easy to extract the collection state from the first two characters of sample ID, it is much more difficult to infer the county from this data alone. However, one crucial clue lies in the commodity column, which tells us which crop was tested for pesticides.

We do have county data in the form of Agricultural Census Data since 1840. We wondered if this data could be harnessed to make inferences about the county of origin in the PDP data set. That is, could we use the historical proportions of crops grown by county to make a statistical guess about which specific county a particular crop being tested for pesticides may have originated?

We tested this idea on the Agricultural Census data from Texas. We filtered out animal data and used only data from 1992 and later, since the earliest available PDP analyses are from 1994. We then grouped data across 4 censuses to calculate the average proportion of the county area being used to grow each crop (1992-2017), and adjusted this likelihood by county size.

Next, we created a function that creates a sampling distribution for a given crop based on the relative likelihoods among all counties.(Figure 3.12) The likelihoods are normalized and then sampled with replacement to create a sampling distribution. A second function bootstraps this process, repeating it a user-specified number of times and aggregating a list of counties by frequency of appearing in sampling distributions. In this way, we can generate a probabilistic list of where a crop may have been grown while still retaining uncertainty.

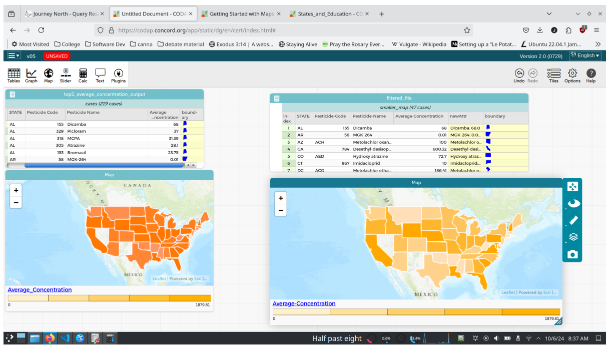
However, this does not solve the county inference problem. The number of crops tested in the PDP far exceeds the number of crops in the census file. More complete commodity information in the census file could inform county inferences in the PDP data set via likelihoods by crop, but even then, imputation still comes with heavy caveats.

2.6 Correlation: Butterfly Sightings v Pesticide Use

Due to the lack of county data in the PDP data set, we were unable to perform any correlation tests between the pesticide concentration and the number of sightings of the Monarch Butterflies. This is due to the fact that the data is not granular enough to make any definitive claims, but given the preliminary data, we can infer that there would be correlation between specific pesticides used in agriculture and the number of sightings of the Monarch Butterflies, rather than a general conclusion that ALL pesticides are harmful to the Monarch Butterfly population.

Chapter 3

Results



top5_average_concentration_output

cases (219 cases)

STATE	Pesticide Code	Pesticide Name	Average ...ncentration	bound- ary
AL	155	Dicamba	68	■
AL	329	Picloram	37	■
AL	318	MCPA	31.39	■
AL	305	Atrazine	28.1	■
AL	153	Bromacil	23.75	■
AR	58	MGK-264	0.01	■

filtered_file

smaller_map (47 cases)

in- dex	STATE	Pesticide-Code	Pesticide-Name	Average-Concentration	newAttr	boundary
1	AL	155	Dicamba	68	Dicamba: 68.0	■
2	AR	58	MGK-264	0.01	MGK-264: 0.0...	■
3	AZ	ACH	Metolachlor oxan...	100	Metolachlor o...	■
4	CA	784	Desethyl-desisop...	600.32	Desethyl-desi...	■
5	CO	AED	Hydroxy atrazine	72.7	Hydroxy atraz...	■
6	CT	967	Imidacloprid	10	Imidacloprid:...	■
7	DC	ACG	Metolachlor etha...	186.41	Metolachlor e...	■

Figure 3.1: Pesticide Concentration Map via Codap

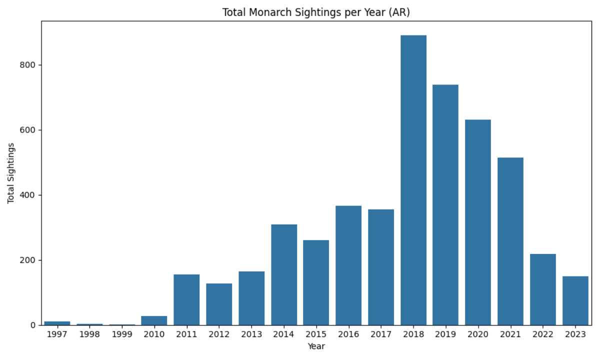


Figure 3.2: Monarch Histogram for Arkansas

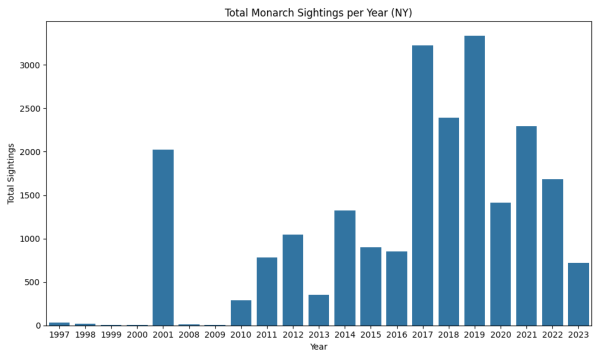


Figure 3.3: Monarch Histogram for New York

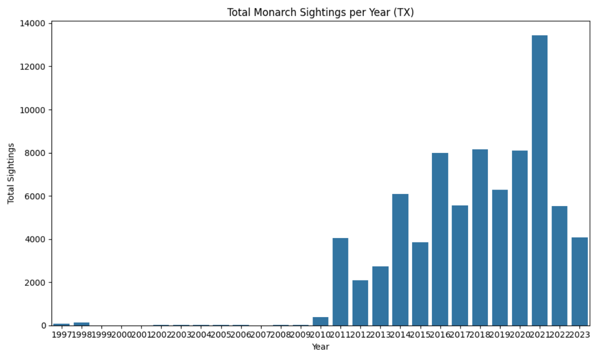


Figure 3.4: Monarch Histogram for Texas

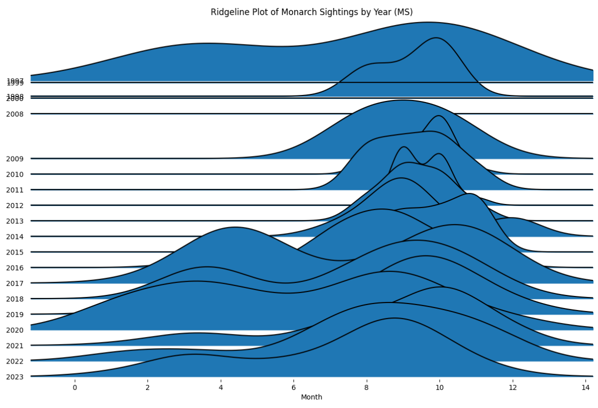


Figure 3.5: Monarch Histogram for Minnesota

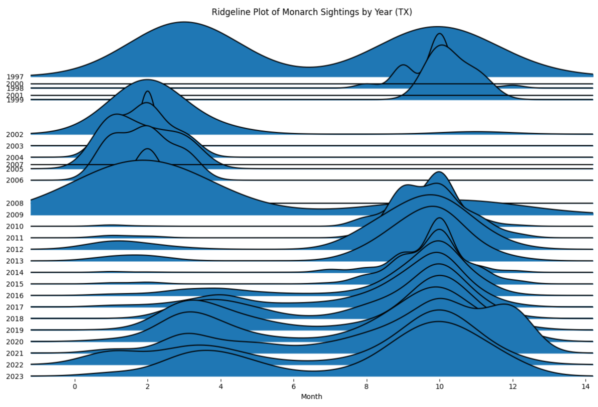


Figure 3.6: Ridgeline for Texas

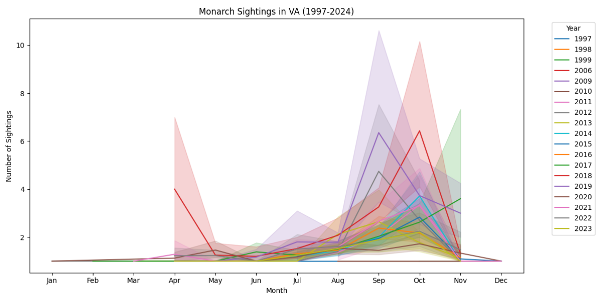


Figure 3.7: Stacked Time series for Virginia

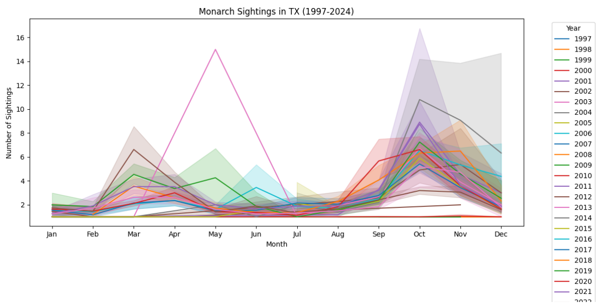


Figure 3.8: Stacked Time series for Texas

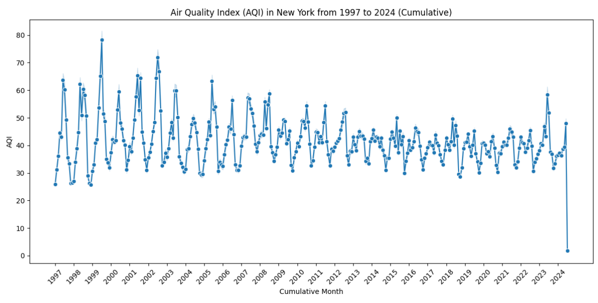


Figure 3.9: Time series for New York, 1997-2024

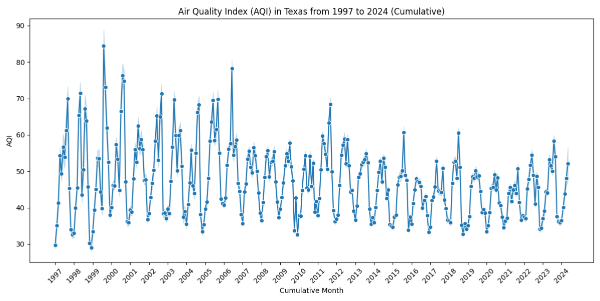
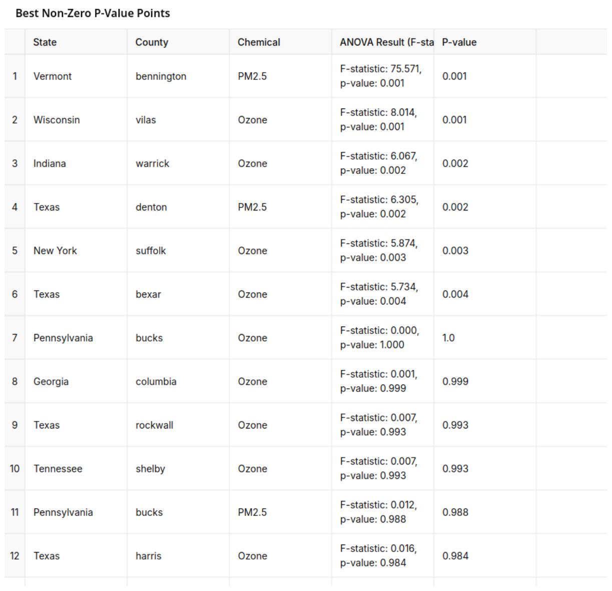


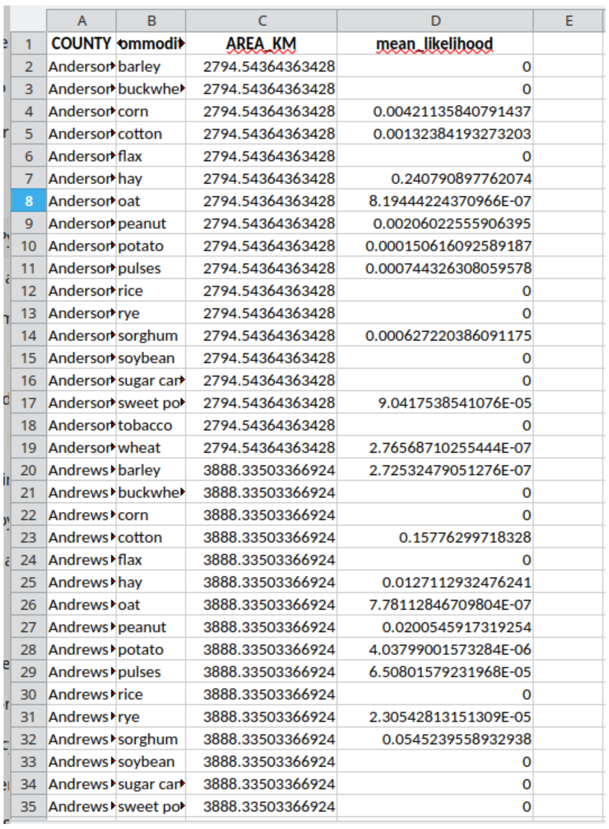
Figure 3.10: Time series for Texas, 1997-2024



Best Non-Zero P-Value Points

	State	County	Chemical	ANOVA Result (F-sta	P-value
1	Vermont	bennington	PM2.5	F-statistic: 75.571, p-value: 0.001	0.001
2	Wisconsin	vilas	Ozone	F-statistic: 8.014, p-value: 0.001	0.001
3	Indiana	warrick	Ozone	F-statistic: 6.067, p-value: 0.002	0.002
4	Texas	denton	PM2.5	F-statistic: 6.305, p-value: 0.002	0.002
5	New York	suffolk	Ozone	F-statistic: 5.874, p-value: 0.003	0.003
6	Texas	bexar	Ozone	F-statistic: 5.734, p-value: 0.004	0.004
7	Pennsylvania	bucks	Ozone	F-statistic: 0.000, p-value: 1.000	1.0
8	Georgia	columbia	Ozone	F-statistic: 0.001, p-value: 0.999	0.999
9	Texas	rockwall	Ozone	F-statistic: 0.007, p-value: 0.993	0.993
10	Tennessee	shelby	Ozone	F-statistic: 0.007, p-value: 0.993	0.993
11	Pennsylvania	bucks	PM2.5	F-statistic: 0.012, p-value: 0.988	0.988
12	Texas	harris	Ozone	F-statistic: 0.016, p-value: 0.984	0.984

Figure 3.11: US Correlation



	A	B	C	D	E
1	COUNTY	ommodi	AREA_KM	mean_likelihood	
2	Andersor	barley	2794.54364363428	0	
3	Andersor	buckwhe	2794.54364363428	0	
4	Andersor	corn	2794.54364363428	0.00421135840791437	
5	Andersor	cotton	2794.54364363428	0.00132384193273203	
6	Andersor	flax	2794.54364363428	0	
7	Andersor	hay	2794.54364363428	0.240790897762074	
8	Andersor	oat	2794.54364363428	8.19444224370966E-07	
9	Andersor	peanut	2794.54364363428	0.00206022555906395	
10	Andersor	potato	2794.54364363428	0.000150616092589187	
11	Andersor	pulses	2794.54364363428	0.000744326308059578	
12	Andersor	rice	2794.54364363428	0	
13	Andersor	rye	2794.54364363428	0	
14	Andersor	sorghum	2794.54364363428	0.000627220386091175	
15	Andersor	soybean	2794.54364363428	0	
16	Andersor	sugar car	2794.54364363428	0	
17	Andersor	sweet po	2794.54364363428	9.0417538541076E-05	
18	Andersor	tobacco	2794.54364363428	0	
19	Andersor	wheat	2794.54364363428	2.76568710255444E-07	
20	Andrews	barley	3888.33503366924	2.72532479051276E-07	
21	Andrews	buckwhe	3888.33503366924	0	
22	Andrews	corn	3888.33503366924	0	
23	Andrews	cotton	3888.33503366924	0.15776299718328	
24	Andrews	flax	3888.33503366924	0	
25	Andrews	hay	3888.33503366924	0.0127112932476241	
26	Andrews	oat	3888.33503366924	7.78112846709804E-07	
27	Andrews	peanut	3888.33503366924	0.0200545917319254	
28	Andrews	potato	3888.33503366924	4.03799001573284E-06	
29	Andrews	pulses	3888.33503366924	6.50801579231968E-05	
30	Andrews	rice	3888.33503366924	0	
31	Andrews	rye	3888.33503366924	2.30542813151309E-05	
32	Andrews	sorghum	3888.33503366924	0.0545239558932938	
33	Andrews	soybean	3888.33503366924	0	
34	Andrews	sugar car	3888.33503366924	0	
35	Andrews	sweet po	3888.33503366924	0	

Figure 3.12: area

Chapter 4

Conclusion

In Conclusion, we were able to analyze the data provided to us and make some preliminary claims about the correlation between the Monarch Butterfly population, the air quality and pesticide concentration in the United States. We were able to show that the air quality in the United States has been improving over the years, which is a good sign for the environment and the Monarch Butterfly population. We were also able to show that there is a possible correlation between Green house pollutants and the number of sightings of the Monarch Butterflies, but further analysis is needed to make any definitive claims.
Given our analysis, we can make the following recommendations:

1. In regions of the country that are know for being high industial regions, such as fracking and oil drilling, mining, and manufacturing, we recommend that the EPA continue to enforce emmision regulations to ensure that the air quality in these regions continue to improve. Some recommendations would be to include more air scrubbers in the factories to ensure that the most amout of pollutants are removed from the air before being released into the atmosphere.

2. Establish more educational outreach programs to inform the public about the migrational patters of ALL migratory animals in the United States. This would aim to give the public a more personal connection to the animals that live around us and promote a more sustainable approach to how we interact with the environment.

3. Create more habitats for migritory animals along their migratory paths with the aim of drawing future generations of these animals away from the more industrialized regions of the country.

4. Call on our leaders to encourage pesticide companies to create more sustainable pesticides that are less harmful to the environment and the animals that live in it.

Chapter 5

Appendix

language=Python, basicstyle=, keywordstyle=, commentstyle=, stringstyle=, breaklines=tru

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import joypy

# List of states you want to generate plots for
states_of_interest = ['TX', 'ME', 'NY', 'RI', 'OK', 'NC', 'SC', 'GA', 'AL', 'VA',
                      'MS', 'FL', 'WI', 'IA', 'KY', 'WV', 'AR', 'MI', 'OH', 'TN',
                      'PA', 'CT', 'IN', 'IL', 'VT']

# Directory to save the output graphs
output_dir = "/home/theWizard_m/Desktop/Datathon2024/data/monarch_plots/bigboystuff"
os.makedirs(output_dir, exist_ok=True)  # Create the directory if it doesn't exist

# Loop through each state and generate the visualizations
for state in states_of_interest:
    # Load the data for the current state
    file_path = f"/home/theWizard_m/Desktop/Datathon2024/data/monarch_data/states/com

%     try:
%         df = pd.read_excel(file_path)

%         # Convert 'Date' column to datetime
%         df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

%         # Ensure 'Number' column is numeric (convert non-numeric values to NaN)
%         df['Number'] = pd.to_numeric(df['Number'], errors='coerce')

%         # Drop rows where 'Number' or 'Date' is NaN
%         df = df.dropna(subset=['Number', 'Date'])

%         # Extract 'Year' and 'Month' from 'Date'
%         df['Year'] = df['Date'].dt.year
%         df['Month'] = df['Date'].dt.month

%         # Create a new column 'Cumulative_Month' to create a continuous timeline
%         df['Cumulative_Month'] = (df['Year'] - 1997) * 12 + df['Month']

%         ### 1. Heatmap ###
%         heatmap_data = df.pivot_table(index='Month', columns='Year', values='Number
%         plt.figure(figsize=(12, 8))
%         sns.heatmap(heatmap_data, cmap='coolwarm', annot=True, fmt=".0f")
%         plt.title(f'Heatmap of Monarch Sightings ({state})')
%         plt.ylabel('Month')
%         plt.xlabel('Year')
%         plt.tight_layout()
%         plt.savefig(os.path.join(output_dir, f'heatmap_{state}.png'))
%         plt.close()

%         ### 2. Stacked Area Plot ###
%         df['Year-Month'] = df['Date'].dt.to_period('M')
%         area_data = df.pivot_table(index='Year-Month', columns='State/Province', va
%         plt.figure(figsize=(12, 6))
%         area_data.plot(kind='area', stacked=True, figsize=(12, 8))
%         plt.title(f'Stacked Area Plot of Monarch Sightings ({state})')
%         plt.ylabel('Number of Sightings')
%         plt.xlabel('Year-Month')
%         plt.tight_layout()
%         plt.savefig(os.path.join(output_dir, f'area_plot_{state}.png'))
%         plt.close()

%         ### 3. Ridgeline Plot ###
    plt.figure(figsize=(12, 8))
    joypy.joyplot(df, by='Year', column='Number', ylim='own', overlap=1.5, figsize=(1
                  title=f'Ridgeline Plot of Monarch Sightings by Year ({state})')
    plt.savefig(os.path.join(output_dir, f'ridgeline_{state}.png'))
    plt.close()

    ### 4. Bar Plot ###
    yearly_data = df.groupby('Year')['Number'].sum().reset_index()
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Year', y='Number', data=yearly_data)
    plt.title(f'Total Monarch Sightings per Year ({state})')
    plt.xlabel('Year')
    plt.ylabel('Total Sightings')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'barplot_{state}.png'))
    plt.close()

    ### 5. Bubble Plot ###
    plt.figure(figsize=(12, 6))
    sns.scatterplot(x='Date', y='State/Province', size='Number', data=df, sizes=(20, 
    plt.title(f'Bubble Plot of Monarch Sightings by State and Date ({state})')
    plt.xlabel('Date')
    plt.ylabel('State')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'bubble_plot_{state}.png'))
    plt.close()

    ### 6. Violin Plot ###
    plt.figure(figsize=(12, 6))
    sns.violinplot(x='Year', y='Number', data=df)
    plt.title(f'Violin Plot of Monarch Sightings Distribution ({state})')
    plt.xlabel('Year')
    plt.ylabel('Number of Sightings')
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f'violin_plot_{state}.png'))
    plt.close()

    print(f"All plots for {state} saved successfully.")

except FileNotFoundError:
    print(f"File not found for state {state}: {file_path}")

print("All plots for all states generated successfully.")

    import pandas as pd
import matplotlib.pyplot as plt
import os

# Function to calculate the overall correlation for the entire state
def calculate_state_correlation(state_df):
return state_df[['Number', 'AQI']].corr().loc['Number', 'AQI']

# Load each state file and process, returning the correlation for each state
def process_state_file(file_path, state_name):
try:
    state_data = pd.read_excel(file_path)
    correlation = calculate_state_correlation(state_data)
    return correlation
except Exception as e:
    print(f"Error processing {state_name}: {e}")
    return None

# List of all state files with their corresponding paths and names
state_files = {
'Texas': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/final_
'Maine': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/final_
'New York': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'Rhode Island': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged
'Oklahoma': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'North Carolina': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merg
'South Carolina': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merg
'Georgia': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fina
'Alabama': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fina
'Virginia': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'Mississippi': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/
'Florida': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fina
'Wisconsin': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fi
'Iowa': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/final_m
'Kentucky': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'West Virginia': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merge
'Arkansas': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'Michigan': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'Ohio': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/final_m
'Tennessee': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fi
'Pennsylvania': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged
'Connecticut': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/
'Indiana': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fina
'Illinois': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fin
'Vermont': '/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged/fina
}

# Dictionary to store the correlation results
state_correlations = {}

# Generalized loop to process each state file
for state_name, file_path in state_files.items():
correlation = process_state_file(file_path, state_name)
if correlation is not None:
    state_correlations[state_name] = correlation

# Creating a bar plot for the correlations
states = list(state_correlations.keys())
correlations = list(state_correlations.values())

plt.figure(figsize=(12, 6))
plt.bar(states, correlations, color='skyblue')
plt.title('Correlation Between Butterfly Counts and AQI Across States')
plt.xlabel('State')
plt.ylabel('Correlation Coefficient')
plt.axhline(0, color='black', linewidth=0.8)
plt.xticks(rotation=90)
plt.tight_layout()

# Save the plot as a file
```

```
output_file = '/home/theWizard_m/Desktop/Datathon2024/data/state_correlations_plot.pn
plt.savefig(output_file)
print(f"Saved correlation bar plot to {output_file}")

#plt.show()


import pandas as pd
import os

# List of states with abbreviations and full names
states = {
'AL': 'Alabama', 'TX': 'Texas', 'ME': 'Maine', 'NY': 'New York', 'RI': 'Rhode Island'
'OK': 'Oklahoma', 'NC': 'North Carolina', 'SC': 'South Carolina', 'GA': 'Georgia',
'VA': 'Virginia', 'MS': 'Mississippi', 'FL': 'Florida', 'WI': 'Wisconsin', 'IA': 'Iow
'KY': 'Kentucky', 'WV': 'West Virginia', 'AR': 'Arkansas', 'MI': 'Michigan',
'OH': 'Ohio', 'TN': 'Tennessee', 'PA': 'Pennsylvania', 'CT': 'Connecticut',
'IN': 'Indiana', 'IL': 'Illinois', 'VT': 'Vermont'
}

# Directory for the output merged data
output_dir = "/home/theWizard_m/Desktop/Datathon2024/data/merged_data/super_merged"
os.makedirs(output_dir, exist_ok=True)

# Loop through each state, load both datasets, and join by county name
for abbrev, state in states.items():
# Load the first dataset (for example: AL_AgCensus_data.xlsx)
ag_file_path = f"/home/theWizard_m/Desktop/Datathon2024/data/ag_parced/{abbrev}_AgCen
merged_file_path = f"/home/theWizard_m/Desktop/Datathon2024/data/merged_data/merged_{

try:
    ag_df = pd.read_excel(ag_file_path)
    merged_df = pd.read_excel(merged_file_path)

    # Convert county names to lowercase and strip any extra spaces for both dataset
    ag_df['COUNTY'] = ag_df['COUNTY'].str.lower().str.strip()
    merged_df['county Name'] = merged_df['county Name'].str.lower().str.strip()

    # Merge the datasets on 'county Name' and 'COUNTY' using an inner join
    final_merged_df = pd.merge(merged_df, ag_df, left_on='county Name', right_on='COU

    # Save the final merged dataframe
    output_file = os.path.join(output_dir, f'final_merged_{abbrev}_data.xlsx')
    final_merged_df.to_excel(output_file, index=False)

    print(f"Data for {state} ({abbrev}) merged and saved to {output_file}")

except FileNotFoundError:
    print(f"File not found for {state} ({abbrev}). Skipping...")
print("All state data merged successfully.")


import pandas as pd
import matplotlib.pyplot as plt
import os

# Load your ANOVA results
anova_by_county_df = pd.read_csv('/home/theWizard_m/Desktop/Datathon2024/data/anova_r

# Extract p-values from the 'ANOVA Result' column
anova_by_county_df['p-value'] = anova_by_county_df['ANOVA Result'].str.extract(r'p-va

# Function to visualize and save p-values per county and per chemical for a given s
def visualize_and_save_pscores_by_county_and_chemical(state_name, output_dir):
# Filter the data for the selected state
state_data = anova_by_county_df[anova_by_county_df['State'] == state_name]

# Pivot the data so that chemicals are columns and counties are rows
state_pivot = state_data.pivot_table(values='p-value', index='County', columns='Chemi

# Plot the p-values for each county and chemical
plt.figure(figsize=(14, 8))
state_pivot.plot(kind='bar', figsize=(14, 8), colormap='viridis')
plt.title(f'Average P-values (ANOVA) by County and Chemical in {state_name}')
plt.xlabel('County')
plt.ylabel('P-value')
plt.xticks(rotation=90)
plt.tight_layout()

# Save the plot
output_file = os.path.join(output_dir, f'{state_name}_pvalues_by_county_chemical.png'
plt.savefig(output_file)
print(f'Saved plot for {state_name} to {output_file}')
plt.close()

# Example usage: Save the plots for all states
output_directory = '/home/theWizard_m/Desktop/Datathon2024/data/anova_graph'  # Repla
os.makedirs(output_directory, exist_ok=True)

# Loop through all the states in your dataset and save the plots
for state_name in anova_by_county_df['State'].unique():
visualize_and_save_pscores_by_county_and_chemical(state_name, output_directory)


# Load required libraries
library(dplyr)
library(tidyr)
library(writexl)
library(readxl)

# Step 1: Load the "TX" sheet from the Excel data
data <- read_excel("C:/Users/mille/Documents/Rowdy Datathon 2024/Datathon2024/AgChang

# Step 2: Convert "Year" to numeric
data <- data %>%
  mutate(Year = as.numeric(Year))

# Step 3: Select observations where the value of "Unit" is "proportion of county ar
filtered_data <- data %>%
  filter(Unit == "proportion of county area", Year > 1990)

# Step 4: Group the data by "COUNTY" and then by "Commodity", keeping AREA_KM
grouped_data <- filtered_data %>%
  group_by(COUNTY, Commodity, AREA_KM)

# Step 5: Calculate the likelihood of each crop in a given county
likelihood_data <- grouped_data %>%
  group_by(COUNTY) %>%
  mutate(total_proportion = sum(Value)) %>%
  ungroup() %>%
  mutate(likelihood = Value / total_proportion)

# Step 6: Average the likelihoods over the four years for each county and crop
averaged_likelihood_data <- likelihood_data %>%
  group_by(COUNTY, Commodity, AREA_KM) %>%
  summarize(mean_likelihood = mean(likelihood, na.rm = TRUE), .groups = 'drop')

# Step 7: Adjust likelihood by county size using "AREA_KM"
adjusted_likelihood_data <- averaged_likelihood_data %>%
  mutate(weighted_likelihood = mean_likelihood * AREA_KM)

# Step 8: Create a sampling distribution based on the relative likelihoods among al
create_sampling_distribution <- function(crop_name, num_samples = 100) {
  # Step 8.1: Filter for the specified crop and use all counties
  all_counties <- adjusted_likelihood_data %>%
    filter(Commodity == crop_name)  # Use all counties

  # Step 8.2: Normalize the weighted likelihoods to create a probability distributi
  all_counties <- all_counties %>%
    mutate(normalized_likelihood = weighted_likelihood / sum(weighted_likelihood))

  # Step 8.3: Create a sampling distribution by sampling from all counties based on
  sampled_counties <- sample(
    %all_counties$COUNTY,   # The counties to sample from
    size = num_samples,    # Number of samples
    replace = TRUE,        # Sampling with replacement
    %prob = all_counties$normalized_likelihood  # Probabilities based on normalized
  )

  return(sampled_counties)
}

# Step 9: Bootstrap to estimate where corn is most likely grown based on sampling d
bootstrap_sampling <- function(crop_name, num_bootstrap = 1000, num_samples = 100) {
  # Step 9.1: Initialize a table to store the count of appearances for each county
  %county_counts <- data.frame(COUNTY = unique(adjusted_likelihood_data$COUNTY), coun

  # Step 9.2: Perform bootstrapping
  for (i in 1:num_bootstrap) {
    # Generate a single sampling distribution
    sampled_counties <- create_sampling_distribution(crop_name, num_samples)

    # Count occurrences of each county in the sample and update the count in county_c
    county_counts <- county_counts %>%
      mutate(count = count + ifelse(COUNTY %in% sampled_counties, 1, 0))
  }

  # Step 9.3: Normalize the counts to get probabilities (divide by number of bootstra
  county_counts <- county_counts %>%
    mutate(probability = count / num_bootstrap)

  # Step 9.4: Return the counties sorted by their estimated probability
  county_counts %>%
    arrange(desc(probability))
}

# Example: Bootstrap to estimate where corn is most likely grown
# corn_bootstrap_results <- bootstrap_sampling("corn", num_bootstrap = 1000, num_sa
# print(corn_bootstrap_results)


# Step 9: Write the averaged likelihood data to a new Excel file
write_xlsx(averaged_likelihood_data, "C:/Users/mille/Documents/Rowdy Datathon 2024/Da

# Print the first 100 rows of the averaged likelihood data
print(head(adjusted_likelihood_data, 100))
```

```
import pandas as pd
from openai import  OpenAI
import re
from concurrent.futures import ThreadPoolExecutor, as_completed

# Set your OpenAI API key
client = OpenAI()
```

```
# Function to get the county from ChatGPT
def get_county_from_gpt(town, state):
    prompt = f"Answer only with the name of the county where the city of {town},{stat

    completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user", "content": prompt}
        ],
        max_tokens=10  # Keep token usage low
    )

    # Extract and return the county name
    county_name = completion.choices[0].message.content.strip()
    # Use regular expression to remove 'county' and anything after, case insensitiv
    cleaned_county_name = re.sub(r"\s*county.*", "", county_name, flags=re.IGNORECASE
    #print(cleaned_county_name)
    return cleaned_county_name

# Function to update a single row in the DataFrame
def update_row(row):
    town = row['Town']
    state = row['State/Province']
    #if pd.isna(row['County']) or row['County'] == '':
    county = get_county_from_gpt(town, state)
    return county

# Function to parallelize GPT requests and update the DataFrame
def parallel_update(df):
    with ThreadPoolExecutor(max_workers=10) as executor:  # Use 10 threads (adjust i
        futures = {executor.submit(update_row, row): index for index, row in df.iterr

        for future in as_completed(futures):
            index = futures[future]
            try:
                county = future.result()
                df.at[index, 'County'] = county
                if index % 100 == 0:
                    print(f"Processed {index} rows")
            except Exception as e:
                print(f"Error at row {index}: {e}")


# Read the CSV file
df = pd.read_csv("monarch_data/unique_town_state_us.csv")
if(not df):
    exit(f"monarch_data/unique_town_state_us.csv: not found")
```

```
# for testing
#df = pd.read_csv("monarch_data/top200.csv")

df['County'] = df['County'].astype(str)

# Call the function to parallelize the update
parallel_update(df)

# Save the updated DataFrame back to a new CSV
df.to_csv('monarch_data/updated_towns_with_counties.csv', index=False)

#df = pd.read_csv('monarch_data/updated_towns_with_counties.csv')

## should probably take a sample of 50 random rows not the header and then ask chat

# Sample 200 random rows from the CSV
SAMPLE_SIZE = 200
sampled_df = df.sample(n=SAMPLE_SIZE, random_state=42)

# Initialize a counter for wrong predictions
wrong_predictions = 0

# Define a function to query GPT-4 for a second opinion
def query_gpt_for_correctness(town, state, county):
    prompt = f"In which county is the town of {town} located in {state}? The current

    completion = client.chat.completions.create(
        model="gpt-4",
        messages=[
            {"role": "user", "content": prompt}
        ],
        max_tokens=10  # Keep token usage low
    )

    # Extract and return the county name
    answer = completion.choices[0].message.content.strip()
    if "no" in answer:
            return False  # Indicating that the model thinks it's wrong
    return True

# Loop through each row in the sample and query GPT-4
for index, row in sampled_df.iterrows():
    town = row['Town']
    state = row['State/Province']
    county = row['County']

    is_correct = query_gpt_for_correctness(town, state, county)
    if is_correct is False:
        wrong_predictions += 1

# Calculate the error rate
error_rate = (wrong_predictions / SAMPLE_SIZE) * 100
print(f"Error rate: {error_rate:.2f}%")


import pandas as pd
from openai import  OpenAI

client = OpenAI()

df = pd.read_csv('monarch_data/monarch_data_us_with_counties.csv')

## should probably take a sample of some random rows not the header and then ask ch

# Sample 200 random rows from the CSV
SAMPLE_SIZE = 200
sampled_df = df.sample(n=SAMPLE_SIZE, random_state=42)

# Initialize a counter for wrong predictions
wrong_predictions = 0

# Define a function to query GPT-4 for a second opinion
def query_gpt_for_correctness(town, state, county):
    prompt = f"In which county is the town of {town} located in {state}? The current 

    completion = client.chat.completions.create(
        model="gpt-4",
        messages=[
            {"role": "user", "content": prompt}
        ],
        max_tokens=10  # Keep token usage low
    )

    # Extract and return the county name
    answer = completion.choices[0].message.content.strip()
    if "no" in answer:
            return False  # Indicating that the model thinks it's wrong
    return True

# Loop through each row in the sample and query GPT-4
for index, row in sampled_df.iterrows():
    town = row['Town']
    state = row['State/Province']
    county = row['County']

    is_correct = query_gpt_for_correctness(town, state, county)
    if is_correct is False:
        wrong_predictions += 1

# Calculate the error rate
error_rate = (wrong_predictions / SAMPLE_SIZE) * 100
print(f"Error rate: {error_rate:.2f}%")

import requests
from bs4 import BeautifulSoup
import pandas as pd
import csv
import os

def download_data(year, season):
    base_url = "https://journeynorth.org/sightings/querylist.html"
    if season == "fall":
        url = f"{base_url}?season=fall&map=monarch-adult-fall&year={year}&submit=View
    else:
        url = f"{base_url}?season=spring&map=monarch-adult-spring&year={year}&submit=

    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    return soup

def main():
    '''
    if not os.path.exists('monarch_data'):
        os.makedirs('monarch_data')

    filename = f"monarch_data_all.csv"
    csvfile = open(f"monarch_data/{filename}", 'w', newline='', encoding='utf-8')
    writer = csv.writer(csvfile)

    for year in range(1997, 2024):
        for season in ['spring', 'fall']:
            soup = download_data(year, season)

            table = soup.find('table')
            if table is None:
                print(f"No data found for {season} {year}")
                continue

            if(year==1997 and season=='spring'):
                headers = [th.text.strip() for th in table.find_all('th') if th.tex
                writer.writerow(headers)

            for row in table.find_all('tr')[1:]:
                columns = row.find_all('td')
                row_data = [col.text.strip() for col in columns[:-1]]
                writer.writerow(row_data)

            print(f"Data for {season} {year} has been saved to '{filename}'")
    '''
    #f"monarch_data/{filename} contains all monarch data, now we filter down only to
    # List of valid US state abbreviations
    us_states = [
        'AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL', 'GA', 'HI', 'ID', 'IL',
        'MD', 'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM', 'NY',
        'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY'
    ]
    print(f"Filtering down the data to us only")

    # Read the CSV file
    df = pd.read_csv(f"monarch_data/monarch_data_all.csv")

    # Filter the DataFrame to keep only rows where the 'State/Province' is a valid U
    filtered_df = df[df['State/Province'].isin(us_states)]

    # Save the filtered data to a new CSV file
    filtered_df.to_csv(f"monarch_data/monarch_data_us.csv", index=False)

#now making the smaller unique town/state list with blank spot for county
    print("Making unique town/state .csv from US data")
    # Read the filtered CSV file
    df = pd.read_csv("monarch_data/monarch_data_us.csv")

    # Convert Town to lowercase (case insensitive) and State/Province to uppercase f
    df['Town'] = df['Town'].str.lower()
    df['State/Province'] = df['State/Province'].str.upper()

    # Create a new DataFrame with unique Town and State/Province combinations
    df_unique = df.drop_duplicates(subset=['Town', 'State/Province'], keep='first',
                                   ignore_index=True)

    # Add a blank 'County' column
    df_unique['County'] = ""

    # Select only the required columns
    df_final = df_unique[['Town', 'State/Province', 'County']]

    # Save the new DataFrame to a CSV file
    df_final.to_csv('monarch_data/unique_town_state_us.csv', index=False)



if __name__ == "__main__":
```

```
main()
```