Analyzing the Impact of Disasters on the U.S. Economy and Energy Sector

Group1: Yiming Sun, Emily Spector

Business Intelligence ISQS-6339-001

Table of Contents

Introduction

In recent years, the incidence and severity of disasters have raised concerns about their potential effects on economic stability and energy sustainability. Disasters can disrupt local economies, induce national economic fluctuations, and impact energy production and consumption patterns. This project seeks to understand the impact of disasters on various economic indicators and energy metrics. Disasters often have ripple effects on economies, impacting prices, employment rates, and energy production and consumption. By analyzing datasets from various domains, this project will provide insights into the potential consequences of disasters on the U.S. economy and energy sector.

Data Domain Discussion

1.Background of the Datasets:

- Disasters Dataset:

  This dataset provides insights into various disasters that have occurred, capturing the accident, types, date, affected areas and some useful binary flags. Such events can directly or indirectly influence economic parameters.

- US_CPI Dataset:

  The Consumer Price Index (CPI) measures the average change over time in the prices paid by urban consumers for goods and services, serving as a primary indicator of inflation.

- US_Gas Dataset:

  Fluctuations in gas prices can be a result of several factors, including geopolitical events, natural disasters, and supply-demand imbalances.

- Unemployment_rate Dataset:

  Employment figures, include men and women, different slots of ages, provide insights into the economic health of a nation.

- Energy_data Dataset:

Energy metrics provide a view of a country's production and consumption patterns.

2.Industry Directions:

The energy industry is increasingly focusing on green solutions. At the same time, measurements like the CPI and unemployment rate give us a snapshot of the nation's economic well-being. By combining these data, we can see if disasters play a crucial role in changing these numbers.

Analysis of Data:

1. Dataset Description:

- **Disasters Dataset:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64900 entries, 0 to 64899
Data columns (total 25 columns):
#   Column                    Non-Null Count  Dtype
---  ------                   --------------  -----
0   femaDeclarationString     64900 non-null  object
1   disasterNumber            64900 non-null  int64
2   state                     64900 non-null  object
3   declarationType           64900 non-null  object
4   declarationDate           64900 non-null  object
5   fyDeclared                64900 non-null  int64
6   incidentType              64900 non-null  object
7   declarationTitle          64900 non-null  object
8   ihProgramDeclared         64900 non-null  int64
9   iaProgramDeclared         64900 non-null  int64
10  paProgramDeclared         64900 non-null  int64
11  hmProgramDeclared         64900 non-null  int64
12  incidentBeginDate         64900 non-null  object
13  incidentEndDate           64365 non-null  object
 14  disasterCloseoutDate     48950 non-null  object
 15  tribalRequest            64900 non-null  int64
 16  fipsStateCode            64900 non-null  int64
 17  fipsCountyCode           64900 non-null  int64
 18  placeCode                64900 non-null  int64
 19  designatedArea           64900 non-null  object
 20  declarationRequestNumber  64900 non-null  int64
 21  lastIAFilingDate         17986 non-null  object
 22  lastRefresh              64900 non-null  object
 23  hash                     64900 non-null  object
 24  id                       64900 non-null  object
dtypes: int64(11), object(14)
memory usage: 12.4+ MB
None
```

- **US_CPI Dataset:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1303 entries, 0 to 1302
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Yearmon  1303 non-null   object
 1   CPI      1303 non-null   float64
dtypes: float64(1), object(1)
memory usage: 20.5+ KB
None
```

- **US_Gas Dataset**:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 2 columns):
 #   Column                                              Non-Null Count  Dtype
---  ------                                              --------------  -----
 0   Month                                               366 non-null    object
 1   U.S. All Grades All Formulations Retail Gasoline Prices Dollars per Gallon
366 non-null    float64
dtypes: float64(1), object(1)
memory usage: 5.8+ KB
None
```

- **Unemployment_rate Dataset**:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 887 entries, 0 to 886
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   date            887 non-null    object 
 1   unrate          887 non-null    float64
 2   unrate_men      887 non-null    float64
 3   unrate_women    887 non-null    float64
 4   unrate_16_to_17  887 non-null    float64
 5   unrate_18_to_19  887 non-null    float64
 6   unrate_20_to_24  887 non-null    float64
 7   unrate_25_to_34  887 non-null    float64
 8   unrate_35_to_44  887 non-null    float64
 9   unrate_45_to_54  887 non-null    float64
 10  unrate_55_over   887 non-null    float64
dtypes: float64(10), object(1)
memory usage: 76.4+ KB
None
```

- **Energy_data Dataset:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 10 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
```

```
 0   Year                                   32 non-null     int64
 1   Energy Related CO2missions (Gigatonnes)       31 non-null     float64
 2   Oil Production (Million barrels per day)       31 non-null     float64
 3   Natural Gas Production (Billion Cubic Metres)  30 non-null     float64
 4   Coal Production (million tons)                 30 non-null     float64
 5   Electricity Generation (Terawatt-hours)        31 non-null     object
 6   Hydroelectricity consumption in TWh            31 non-null     float64
 7   Nuclear energy consumption in TWh              31 non-null     float64
 8   Installed Solar Capacity (GW)                  22 non-null     float64
 9   Installed Wind Capacity in GW                  31 non-null     float64
dtypes: float64(8), int64(1), object(1)
memory usage: 2.6+ KB
None
```

2.Data Sources:

- CPI dataset
https://www.kaggle.com/datasets/varpit94/us-inflation-data-updated-till-may-2021
- Unemployment rate data
https://www.kaggle.com/datasets/axeltorbenson/unemployment-data-19482021
- Natural Disaster data
https://www.kaggle.com/datasets/headsortails/us-natural-disaster-declarations
- Gas Prices
https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=EMM_EPM0_PTE_NUS_DPG&f=M
- Energy data
https://www.kaggle.com/datasets/iannjuguna/energy-data-since-1990?select=Energy+Data1.csv

3.Beneficial Attributes:

- Disaster Frequency & Incident Type:

Categorizing disasters by type and severity can allow businesses to tailor their disaster response and recovery strategies.

- Unemployment Rate (and its sub-categories):

The unemployment rate is a direct reflection of the job market's health. Breaking it down further by gender and age provides a detailed picture of which demographic groups are most affected. Post-disaster, a rise in unemployment could suggest businesses are struggling to recover, while stability might indicate resilience.

- Gas Price:

The price of gas often influences consumer behavior, transportation costs, and even the price of goods (since transportation plays a role in goods pricing). Understanding its fluctuations in relation to disasters can help industries predict potential price hikes or drops post-disasters.

- Consumer Price Index (CPI):

CPI is a vital economic indicator, reflecting the average change in prices over time that consumers pay for a basket of goods and services. Understanding how it changes, especially in relation to disasters, can provide insights into economic resilience and the ability of the market to recover after catastrophic events.

- Energy-related Metrics:

Energy is the backbone of any economy. By monitoring energy production and consumption, businesses can assess the disaster's impact on energy infrastructure. This can inform decisions related to energy investments, environmental strategies, and even governmental lobbying for infrastructure rebuilding funds.

Data Cleaning

1.Data Exploration:

Before any cleaning or transformation, the ***".info()"*** method was called on each dataset to understand its basic structure, such as column names, non-null counts, and datatypes.
The datasets are of great quality, only energy dataset and disasters dataset have missing values.

2.Missing Values:

In energy_data.csv, the last row is empty, we just dropped it.
Columns including Energy Related CO2 emissions (Gigatonnes), Oil Production (Million barrels per day), Natural Gas Production (Billion Cubic Metres), Coal Production (million tons), Installed Solar Capacity (GW), Installed Solar Capacity (GW) have missing values.

For columns including Energy Related CO2 emissions (Gigatonnes), Oil Production (Million barrels per day), Natural Gas Production (Billion Cubic Metres), Coal Production (million tons), Installed Solar Capacity (GW), we cannot find other sources to fill the missing values, so we used forward fill.
For Installed Solar Capacity (GW), since solar capacity is not popular at that time period, we fill it with 0.

3.Standardizing Date Formats:

- A recurring theme across all the datasets was the need to standardize date formats for merging.
- For *"employment_df", "cpi_df",* and *"disaster_df",* the date columns were converted to datetime objects and a new column "*MonthYear"* was created to represent the date in the format "MM/YYYY".
- *"gas_price_df"* had a unique challenge due to its date string format (e.g., 'Jan-19'). A custom function was employed to parse and convert these strings into

datetime objects, which were then used to derive the *"MonthYear"* column. Invalid dates were also identified and printed for review.

- For *"energy_df",* the *"Year"* column was treated as a datetime object, and a corresponding *"MonthYear"* column was created.
- Change the yearly values of *"energy_df"* into monthly.

4.Data Integrity Check:

After processing the date formats, each dataset was saved to a CSV file (e.g., 'emp_test.csv', 'cpi_test.csv' etc.) to manually verify the transformation results.

Data Merging

1.Method of merging:

1. Preparing:

Standardizing Date Formats during data cleaning.
The *"cpi_df"* was chosen as the starting point for merging, mainly because the Consumer Price Index (CPI) is a primary economic indicator. The other data frames were compiled into a list ("dfs_to_merge").

2. Iterative Merging:

Each data frame from *"dfs_to_merge"* was successively merged with the growing merged data frame using the *"MonthYear"* column. Inner joins were utilized to ensure that the merged dataset only contained rows where *"MonthYear"* values existed in all participating datasets.

3. Dropping Redundant Columns:

Post merging, several columns that were deemed unnecessary or redundant (such as original date columns, and other specific columns like "fyDeclared", "hash", "id", etc.) were dropped from the merged data frame.

4. Removing Duplicates:

To ensure data integrity, duplicates were identified and removed, resulting in the *"unique_df"* data frame. This data frame represents the cleaned and merged dataset.

5. Saving the Cleaned and Merged Data:

Finally, *“unique_df”* was saved to a CSV file (“merged_data2.csv”) to facilitate subsequent analyses.

2.Issues related to different granularity between the datasets:

There are disparities in granularity between the datasets:

- The disaster dataset is very granular with multiple entries for each month.
- Economic data like CPI and unemployment rates and gas price are provided on a monthly basis.
- Energy-related data is provided on an annual basis.

Addressing Granularity Issues:

1. Aggregating Disaster Data:

   Since the disaster dataset is vast with multiple entries for each month, and we're focusing on January, we can aggregate this data to produce summarized metrics for January of each year. Metrics could include the total number of disasters, average severity, or total areas affected.
2. Temporal Alignment for Energy Data:

   For the annually recorded energy data, convert the yearly data into monthly data, assigning that month value to January. This creates a consistent reference point, allowing us to correlate the energy data with the aggregated disaster metrics and economic data for January of each year.

3.Creation of Macro-Variables:

Disaster-Economic Impact Index for January: By comparing the aggregated January disaster data with January’s CPI and unemployment rate, we can create an index to gauge the economic impact of disasters in the beginning of each year. This index can provide insights into how disaster-heavy Januaries might signal economic trends for the year.

4.Benefits to the Business:

1. Disaster Frequency & Economic Indicators:

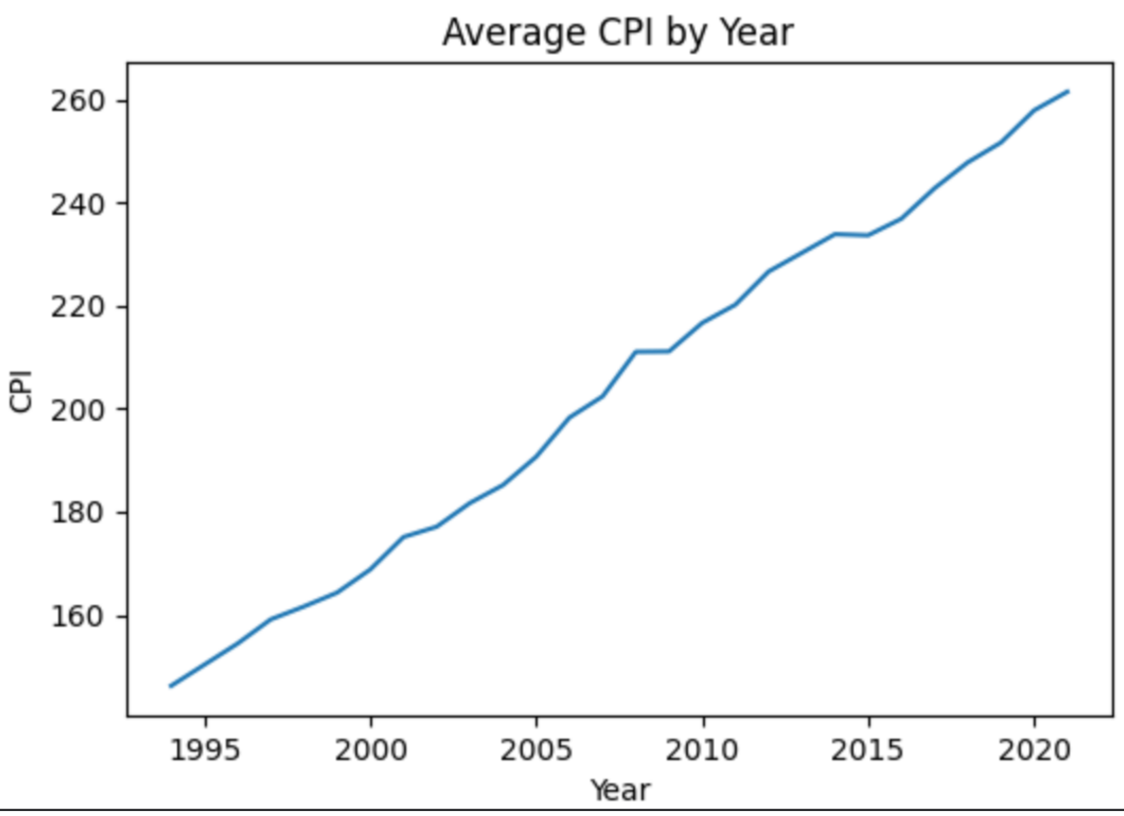
   By comparing the frequency of disasters in January with economic indicators, businesses can gauge the potential economic impact of disasters at the start of the year. This can help in budgeting, risk assessment, and resource allocation for potential disaster recovery or mitigation efforts.

2. Disaster Impact Area & Economic Indicators:
If certain regions are more disaster-prone in January and this correlates with economic downturns, regional businesses or government entities can employ preemptive measures, such as establishing stronger infrastructures or emergency funds.

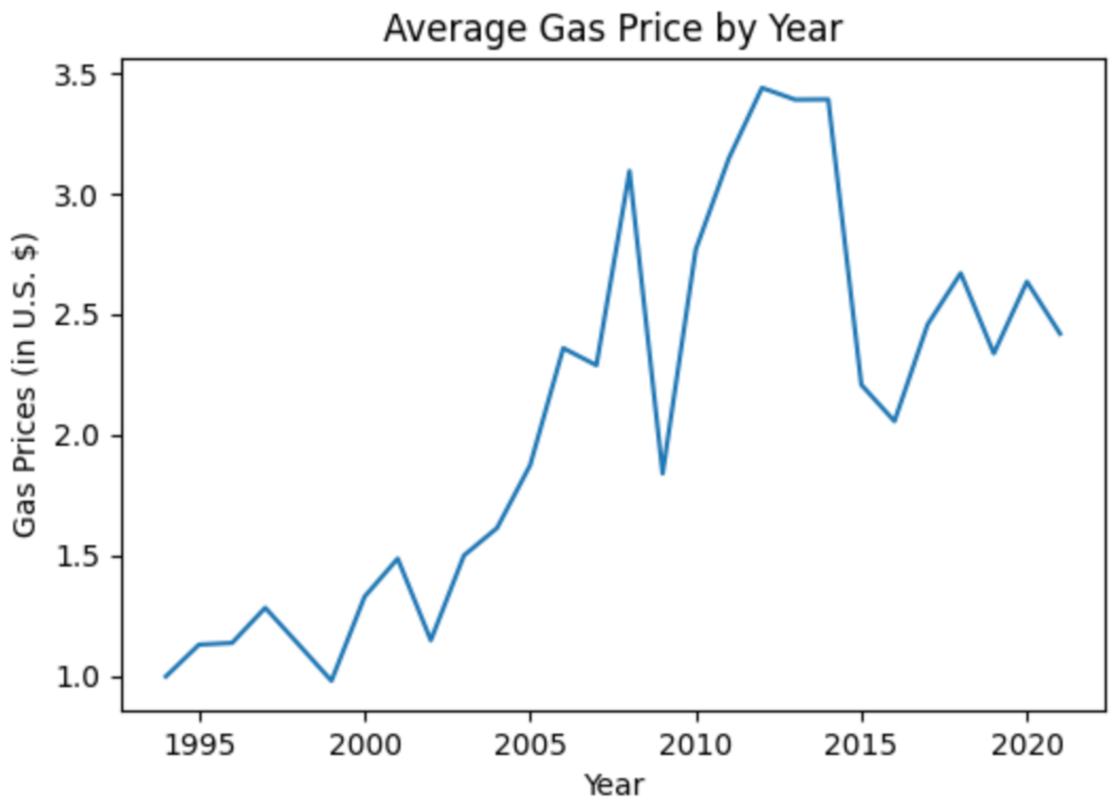
3. Unemployment Rate & Energy Production:
If a rise in unemployment consistently aligns with declines in certain energy production metrics, energy sector businesses can better predict labor market trends and adjust their hiring or training programs accordingly.

Visualization of Data

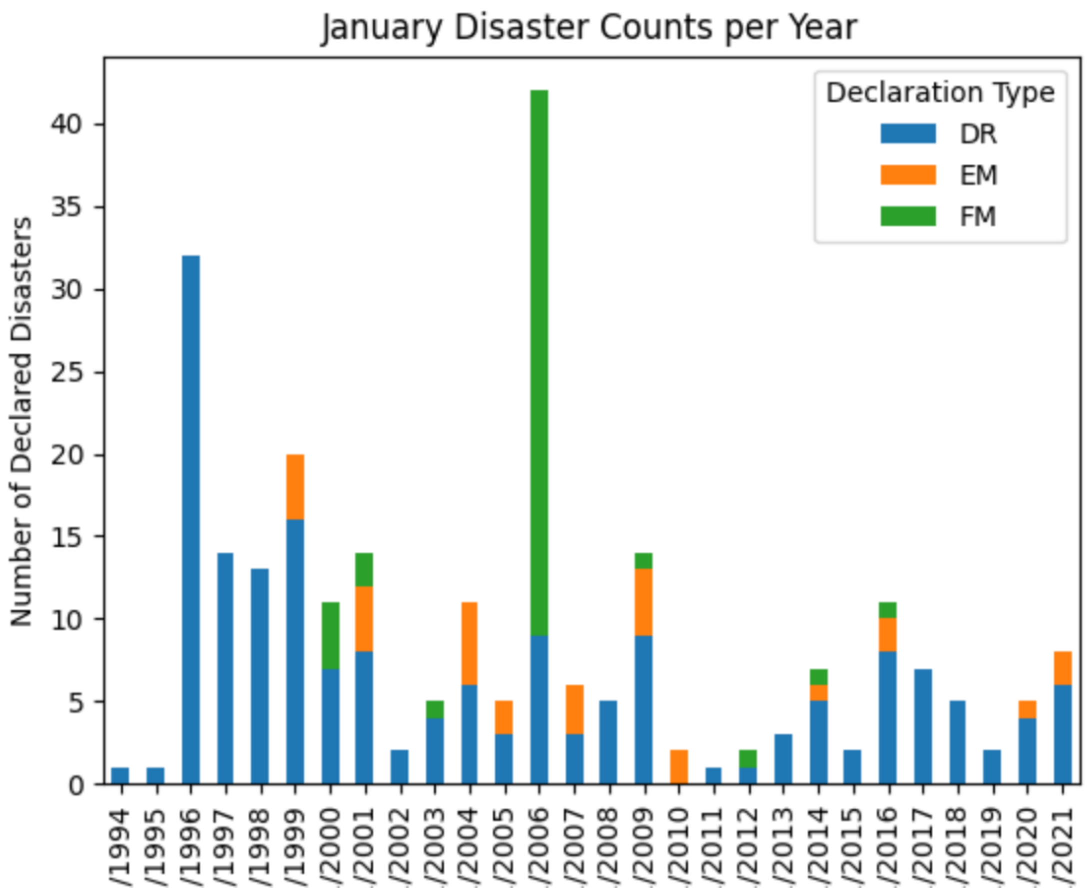
We made three visualizations.



The First one is Average CPI by Year. From this graph, we can find the CPI shows a clear upward trend.

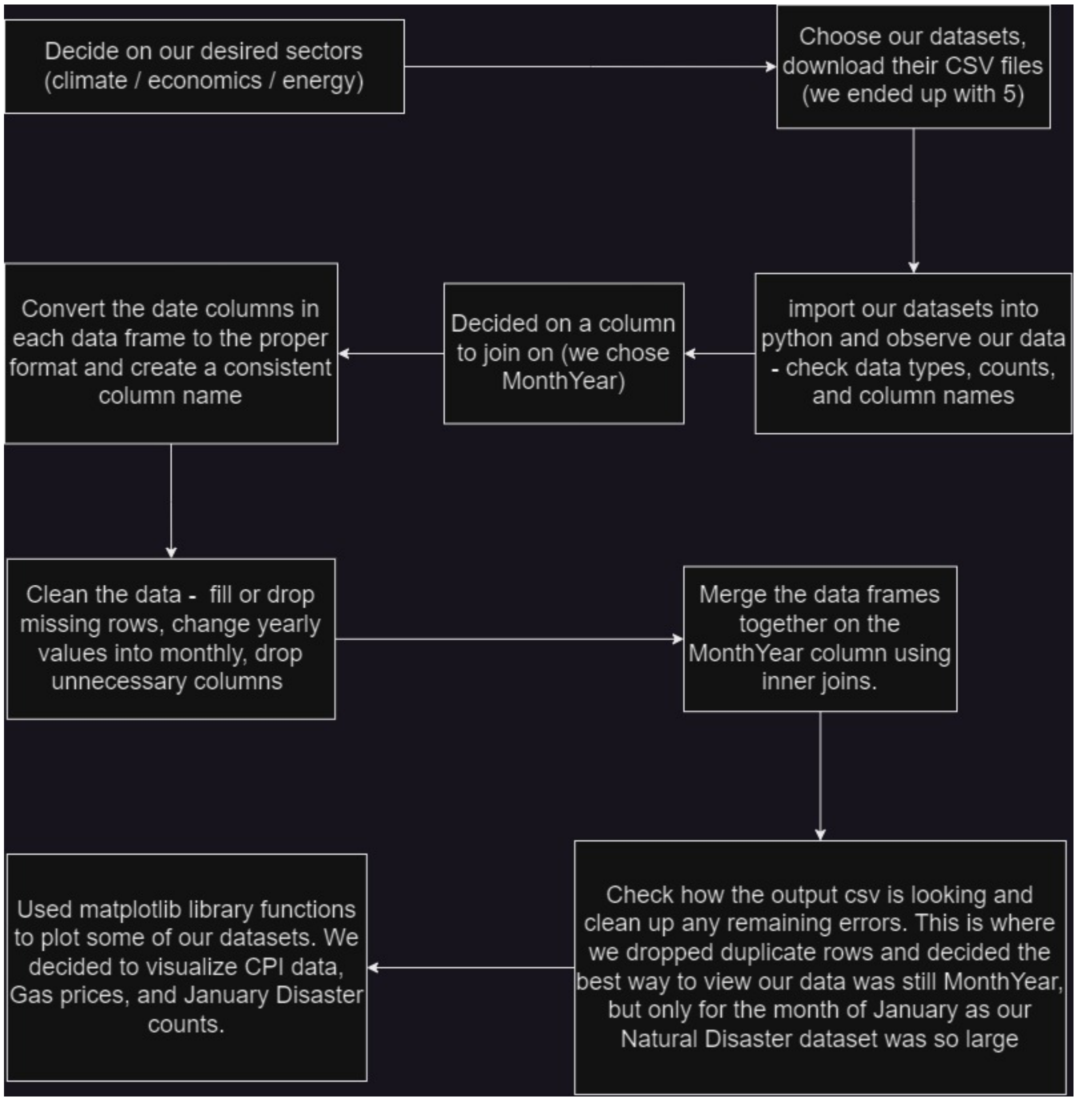


The second one is Average Gas Price by Year. The gas price has fluctuations, but still show an upward trend.



The third one is January Disaster Counts per Year. The disaster counts exhibits cyclical fluctuations, with the amplitude gradually diminishing.

Flow Diagram


Decide on our desired sectors (climate / economics / energy)
Choose our datasets, download their CSV files (we ended up with 5)
import our datasets into python and observe our data - check data types, counts, and column names
Decided on a column to join on (we chose MonthYear)
Convert the date columns in each data frame to the proper format and create a consistent column name
Clean the data - fill or drop missing rows, change yearly values into monthly, drop unnecessary columns
Merge the data frames together on the MonthYear column using inner joins.
Check how the output csv is looking and clean up any remaining errors. This is where we dropped duplicate rows and decided the best way to view our data was still MonthYear, but only for the month of January as our Natural Disaster dataset was so large
Used matplotlib library functions to plot some of our datasets. We decided to visualize CPI data, Gas prices, and January Disaster counts.

Instructions for Code

```
# Import pandas
import pandas as pd
# Import matplot
import matplotlib.pyplot as plt

# Read all csv files into respective dataframes
employment_df = pd.read_csv('unemployment_rate.csv')
```

```
cpi_df = pd.read_csv('US_CPI.csv')
disaster_df = pd.read_csv('disasters.csv')
gas_price_df = pd.read_csv('US_gas.csv')
energy_df = pd.read_csv('energy_data.csv')

# Print the info for each data frame to get counts and types
print("\n\nUnemployment Dataset:")
print(employment_df.info())

print("\n\nCPI Dataset:")
print(cpi_df.info())

print("\n\nDisaster Dataset:")
print(disaster_df.info())

print("\n\nGas Dataset:")
print(gas_price_df.info())

print("\n\nEnergy Dataset:")
print(energy_df.info())


## Get MonthYear format for the unemployment dataframe

# convert date column to datetime format
employment_df['date'] = pd.to_datetime(employment_df['date'], format=
'%m/%d/%Y')
# create MonthYear column
employment_df['MonthYear'] = employment_df['date'].dt.strftime('%m/%Y')
# make MonthYear column appear first for ease of checking
employment_df= employment_df[['MonthYear'] + [col for col in
employment_df.columns if col != 'MonthYear']]
# output to csv to check
```

```
employment_df.to_csv('emp_test.csv', index=False)


## Get MonthYear format for the CPI dataframe

# convert the provided date column to datetime
cpi_df['Yearmon'] = pd.to_datetime(cpi_df['Yearmon'], format='%d-%m-%Y')
# create MonthYear column
cpi_df['MonthYear'] = cpi_df['Yearmon'].dt.strftime('%m/%Y')
# make MonthYear column appear first for ease of checking
cpi_df= cpi_df[['MonthYear'] + [col for col in cpi_df.columns if col != 'MonthYear']]
# output to csv to check
cpi_df.to_csv('cpi_test.csv', index=False)


## Get MonthYear format for the disaster dataframe

# convert the provided date column to datetime
disaster_df['declarationDate'] = pd.to_datetime(disaster_df['declarationDate'],
format="%Y-%m-%dT%H:%M:%S.%fZ")
# create def function (trying out different methods)
def extract_year_month(date):
   return date.strftime('%m/%Y')
# apply the function to create a new column 'MonthYear'
disaster_df['MonthYear'] =
disaster_df['declarationDate'].apply(extract_year_month)
# make MonthYear column appear first for ease of checking
disaster_df= disaster_df[['MonthYear'] + [col for col in disaster_df.columns if col !=
'MonthYear']]

# output to csv to check
disaster_df.to_csv('disaster_test.csv', index=False)
```

```
## Get MonthYear format for the Gas dataframe

# create def function
def convert_to_datetime(date_string):
   try:
      month, year = date_string.split('-')
      month = pd.to_datetime(month, format='%b').month
      year = int(year)  # Convert year to an integer

      # adjust two-digit year to the correct century (assumes years between 1930
and 2029)
      if year < 30:
         year += 2000
      else:
         year += 1900

      return pd.to_datetime(f"{year}-{month}-01")
   except ValueError:
      return None

# convert the "Month" column to datetime objects
gas_price_df['Month'] = gas_price_df['Month'].apply(convert_to_datetime)

# check for missing or invalid date values
invalid_dates = gas_price_df[gas_price_df['Month'].isnull()]
if not invalid_dates.empty:
   print("Invalid dates found:")
   print(invalid_dates)

# create MonthYear column
gas_price_df['MonthYear'] = gas_price_df['Month'].dt.strftime('%m/%Y')
```

```
# reorder the columns for ease of checking
gas_price_df = gas_price_df[['MonthYear'] + [col for col in gas_price_df.columns if
col != 'MonthYear']]
# output to csv to check
gas_price_df.to_csv('gas_test.csv', index=False)



## Get MonthYear format for the energy dataframe

# Convert the 'Year' column of energy_df to datetime format using the specified
format '%Y'
energy_df['Year'] = pd.to_datetime(energy_df['Year'], format='%Y',
errors='coerce')
# create MonthYear column
energy_df['MonthYear'] = energy_df['Year'].dt.strftime('%m/%Y')
# reorder the columns for ease of checking
energy_df = energy_df[['MonthYear'] + [col for col in energy_df.columns if col !=
'MonthYear']]

# drop missing row
energy_df.drop(energy_df.tail(1).index, inplace=True)
# fill missing values in 'Energy Related CO2missions (Gigatonnes)'
# 'Oil Production (Million barrels per day)','Natural Gas Production (Billion Cubic
Metres)',
# 'Coal Production (million tons)' using ffill method
energy_df['Energy Related CO2missions (Gigatonnes)'].fillna(method='ffill',
inplace=True)
energy_df['Oil Production (Million barrels per day)'].fillna(method='ffill',
inplace=True)
energy_df['Natural Gas Production (Billion Cubic Metres)'].fillna(method='ffill',
inplace=True)
energy_df['Coal Production (million tons)'].fillna(method='ffill', inplace=True)
```

```
# fill missing values in 'Installed Solar Capacity (GW)' with 0
energy_df['Installed Solar Capacity (GW)'].fillna(0, inplace=True)


# remove ',' from numbers
energy_df['Electricity Generation (Terawatt-hours)'] = energy_df['Electricity
Generation (Terawatt-hours)'].str.replace(',', '', regex=True).astype(float)

# create a list of the columns contain values for an entire year
yearly_value_columns = ['Energy Related CO2missions (Gigatonnes)', 'Natural Gas
Production (Billion Cubic Metres)', 'Coal Production (million tons)',
                        'Electricity Generation (Terawatt-hours)','Hydroelectricity
consumption in TWh', 'Nuclear energy consumption in TWh',
                        'Installed Solar Capacity (GW)','Installed Wind Capacity in GW']

# divide each column by 12 to get monthly values
for column in yearly_value_columns:
    energy_df[column] = energy_df[column] / 12

# output to csv to check
energy_df.to_csv('energy_test.csv', index=False)



### Merge Dataframes ###

# set merged_df to the starting df
merged_df = cpi_df

# create a list of the remaining dataframes set equal to 'dfs_to_merge'
dfs_to_merge = [employment_df, disaster_df, gas_price_df, energy_df]
```

```
# create a for loop to iterate through the list and join on "MonthYear" column
# used inner joins so there were no large chunks of missing rows since some csv
files contained many years not in others
for df_to_merge in dfs_to_merge:
    merged_df = pd.merge(merged_df, df_to_merge, on='MonthYear', how='inner')

# drop columns
columns_to_drop = ['Yearmon',
'date','declarationDate','fyDeclared','incidentBeginDate',
                'incidentEndDate','disasterCloseoutDate','hash','id','Month','Year',

'fipsCountyCode','placeCode','designatedArea','lastIAFilingDate','lastRefresh']
merged_df.drop(columns=columns_to_drop, inplace=True)

# drop duplicates
unique_df = merged_df.drop_duplicates()
# output the merged df to a csv
unique_df.to_csv('merged_data.csv', index=False)


## CPI visualization ##

# create a year column in the data frame
unique_df['Year'] = pd.to_datetime(unique_df['MonthYear']).dt.year

# group the data by year and calculate the mean CPI for each year
unique_df_CPI_grouped = unique_df.groupby('Year')['CPI'].mean().reset_index()

# create a line graph of CPI against the year
plt.figure(figsize=(6, 4))
plt.plot(unique_df_CPI_grouped['Year'], unique_df_CPI_grouped['CPI'])

plt.xlabel('Year')
plt.ylabel('CPI')
plt.title('Average CPI by Year')

plt.show()


## Gas Prices Visualization ##

# create a year column in the data frame
unique_df_gas_grouped = unique_df.groupby('Year')['U.S. All Grades All
Formulations Retail Gasoline Prices Dollars per Gallon'].mean().reset_index()
plt.figure(figsize=(6, 4))

# group the data by year and calculate the mean gas price for each year
plt.plot(unique_df_gas_grouped['Year'], unique_df_gas_grouped['U.S. All Grades
All Formulations Retail Gasoline Prices Dollars per Gallon'])

plt.xlabel('Year')
plt.ylabel('Gas Prices (in U.S. $)')
plt.title('Average Gas Price by Year')

plt.show()


##  Natural Disaster Visualization ##

# count how many disaster occurances there were for each month
counts_per_month =
unique_df.groupby('MonthYear')['declarationType'].value_counts().unstack().fillna
(0)
# plot the counts
counts_per_month.plot(kind='bar', stacked=True)
```

```
plt.xlabel('Date')
plt.ylabel('Number of Declared Disasters')
plt.title('January Disaster Counts per Year')
plt.legend(title='Declaration Type')

plt.show()
```

Appendix

Detailed descriptions of the disaster dataset

- *fema_declaration_string*: Agency standard method for uniquely identifying Stafford Act declarations. Concatenation of declaration_type, disaster_number and state.
- *disaster_number*: Sequentially assigned number used to designate an event or incident declared as a disaster.
- state: US state, district, or territory.
- *declaration_type*: One of "DR" (= major disaster), "EM" (= emergency management), or "FM" (= "fire management")
- *declaration_date*: Date the disaster was declared.
- *fy_declared*: Fiscal year in which the disaster was declared.
- *incident_type*: Type of incident such as "Fire", "Flood", or "Hurricane". The incident type will affect the types of assistance available.
- *declaration_title*: Title for the disaster. This can be a useful identifier such as "Hurricane Katrina" or "Covid-19 Pandemic".
- *ih_program_declared*: Binary flag indicating whether the "Individuals and Households program" was declared for this disaster.
- *ia_program_declared*: Binary flag indicating whether the "Individual Assistance program" was declared for this disaster.
- *pa_program_declared*: Binary flag indicating whether the "Public Assistance program" was declared for this disaster.
- *hm_program_declared*: Binary flag indicating whether the "Hazard Mitigation program" was declared for this disaster.
- *incident_begin_date*: Date the incident itself began.

- *incident_end_date*: Date the incident itself ended. This feature has about 14% NA entries.
- *disaster_closeout_date*: Date all financial transactions for all programs are completed. This column has 98% NA entries.
- *fips*: 5-digit FIPS county code; used to identify counties and county equivalents in the United States, the District of Columbia, US territories, outlying areas of the US and freely associated states. Concatenated from the 2 source columns "fipsStateCode" and "fipsCountyCode".
- *place_code*: A unique code system FEMA uses internally to recognize locations that takes the numbers '99' + the 3-digit county FIPS code. There are some declared locations that don't have recognized FIPS county codes in which case a unique identifier was assigned.
- *designated_area*: The name or phrase describing the U.S. county that was included in the declaration. Can take the value "Statewide".
- *declaration_request_number*: Unique ID assigned to request for a disaster declaration.
- *hash*: MD5 Hash of the fields and values of the record.
- *last_refresh*: Date the record was last updated by FEMA.
- *id*: Unique ID assigned to the record. Those last 4 columns are primarily for bookkeeping.
- *disaster_number*: Sequentially assigned number used to designate an event or incident declared as a disaster.
- state: One of the three states "CA", "TX", or "WI".
- *declaration_type*: One of "DR" (= major disaster), "EM" (= emergency management), or "FM" (= "fire management")
- *declaration_date*: Date the disaster was declared.
- *incident_type*: Type of incident such as "Fire", "Flood", or "Biological". The incident type will affect the types of assistance available.
- *declaration_title*: Title for the disaster.
- *ih_program_declared*: Binary flag indicating whether the "Individuals and Households program" was declared for this disaster.
- *ia_program_declared*: Binary flag indicating whether the "Individual Assistance program" was declared for this disaster.

- *pa_program_declared*: Binary flag indicating whether the "Public Assistance program" was declared for this disaster.
- *hm_program_declared*: Binary flag indicating whether the "Hazard Mitigation program" was declared for this disaster.
- *incident_begin_date*: Date the incident itself began.
- *incident_end_date*: Date the incident itself ended. The M5 constraint is that the disaster must have begun before the start of the evaluation time period on 2016-05-23 and ended on or after the begin of the training data range on 2011-01-29. This feature has about 14% NA entries, which were considered under the assumption that start data = end date.
- *designated_area*: The name or phrase describing the U.S. county that was included in the declaration.