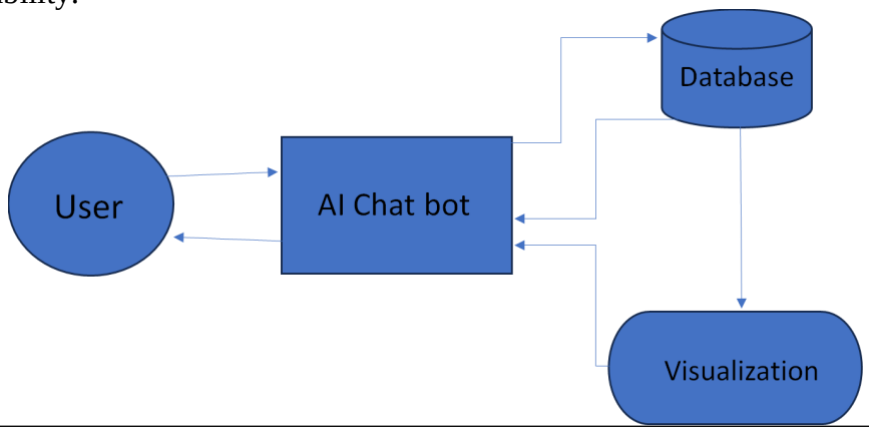
Project Paper – CAPP122 Final Project
Group Name: Planet status code 404
Group Members: Gayathri, Gregory, Kiran, Onur
CnetIDs: gmitchelljr, ksjivnani, gayathrij, onurcan
GitHub userids: gregthemitch, gayathrij-hub, KiranJivnani, onurcan-b

Project Abstract: Exploring Climate Change's Effects Through an Equity Lens

This project seeks to explore the multifaceted impacts of climate change and its distributional effects on communities across the United States. We aim to understand how climate-induced factors such as floods, heatwaves, and hurricanes disproportionately affect vulnerable populations particularly in redlined communities. Leveraging data from the EPA, FEMA, Richmond Redlining dataset, Census, and a well-designed Climate Vulnerability Index, we will provide a tool to explore the ways climate change will affect communities differently and how those impacts may (or may not) disproportionately affect marginalized groups. This project focuses on Chicago, New Orleans, Los Angeles, Seattle, Dallas, and Houston, selected for their diversity in climate vulnerability.



The image above shows the structure of how our data interacts with the Chatbot to create visualizations.

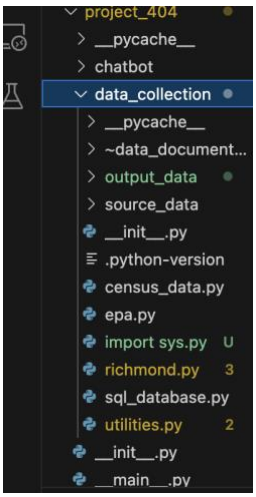
Overall structure of the project:

This project is structured in two sections:

1. **Data collection.**

The data collection folder is split into three sub folders. The first one is data_documentation. This folder contains the following pyscripts:

- richmond.py: This pyscript contains functions for data processing, cleaning, and editing for a FEMA dataset, two Climate Vulnerability Index datasets, and the Richmond Redlining dataset. It outputs 3 CSV files.



- The FEMA dataset is a national risk index that shows natural hazards and social risks faced by different communities across the United States.
  - Because this dataset is so large it cannot be uploaded to GitHub. Therefore, you can find it in this Dropbox: https://www.dropbox.com/home/Planet%20Status%20Code%20404
  - To get the outputted CSV for only this dataset, import richmond.py and run the function "def clean_fema_data". It can also be run from app.py. The output of this function can be found on Github.
- The Richmond Redlining dataset displays redlining maps for most cities across the United States.
  - To get the outputted CSV for this dataset, import richmond.py and run the function, "def clean_redlined_with_tract_data". It can also be run from app.py. Both the dataset and the output can be found on GitHub.
- The Climate Vulnerability Index Datasets contain data showing communities facing climate impact challenges. One dataset (master) is on GitHub and the other was too large to be uploaded to GitHub and is in the Dropbox.
  - To get the outputted CSV for only this dataset, import richmond.py and run the function "def combine_cvi_df". It can also be run from app.py. The output of this function is not on GitHub because it is too large but can be found in the Dropbox.

- census_data.py: The census_data.py script is responsible for pulling three datasets from the census.gov platform using API calls. Since the data retrieved from the census site is less than 500 calls per day, there is no API key required to run the file. The file additionally processes the dataset to make it uniform as per the requirements of the project by reordering columns and obtaining the geo_id for primary key.
  - The data fetched from the census.gov comprises of 3 datasets that are as follows:
    - Demographic and Housing Characteristics (Decennial Census 2020): This dataset captures demographic and housing characteristics of people tract-wise in the target states. Two datasets are collected from the Decennial Census 2020 API, providing detailed information about the population and housing in various geographical areas.
    - Community Resilience Estimates 2022: The third dataset captures Community Resilience Estimates using American Community Survey microdata and Population Estimates Program data. This dataset measures the capacity of individuals and households to absorb the external stresses of the impacts of a disaster.
  - The file can be executed by the running the app.py file's data_collection function. After the dataset is obtained through the API call, the data is cleaned and converted to a concatenated csv file and exported to the output_data subfolder under data_collection.
- epa.py: The epa pyscript gets environmental justice data from the EPA. This pyfile pulls data from: Chicago, Dallas, Houston, New Orleans and LA only. Each city has data from a sample of 450 tracts each and the API calls take about 2 hours each city. There are no special instructions to run collect_epa_data_from(city, max_rows, data_file). If running for Chicago, make sure it is properly spelled.

    - The data files are already included in the output_data folder so that you don't have to run it and wait about 8 hours.
    - If you run this, the API produces successful calls only about 80% of the time. This is due to some tracts not having data. If you want 500 rows, it is a good idea to call the function with a parameter of 600.
- sql_database.py: The sql_database.py script is responsible for traversing through the output_data subfolder, where all the data to be used by the chatbot is exported and creating a climate_database.db file. This file serves as the database repository for the chatbot.
    - Functionality:
        - Traversing through Subfolder: The script navigates through the output_data subfolder to access the exported data.
        - Database Creation: It creates a climate_database.db file to store the data retrieved from the CSV files.
    - Table Creation: For each CSV file in the subfolder, the script reads the data and creates a corresponding table in the database. The tables are mapped along the 11-digit geo_id, which serves as a unique identifier and primary key in all datasets accessed by the chatbot.
- Utilities.py: The utilities pyscript contains functions for preliminary data cleaning, reformatting, and visualizing tools for the EPA dataset.

The data collection folder also houses a source file and an output file. All data sources (except for one Climate Vulnerability Index dataset and the FEMA dataset) can be found in the source file. All CSVs can be found in the output file. Datasets and outputs that are too large to be uploaded to GitHub can be found in the Dropbox.

2. Chatbot:

The chatbot is the front-end, user-facing part of the project. By taking in and parsing user requests, the chatbot can provide general summary statistics and interactive maps. To do this, we relied on Mistral's 7B parameter model hosted on Google Colab—the project technically can be run locally, but for speed and testing purposes Google's GPUs were very helpful. The task to construct the chatbot was broken into 3 major tasks:

1. Coerce the chatbot to produce consistent output that can be easily parsed.
2. Write functions for the chatbot to follow that rely on the inputs pared from the user's input.
3. Respond back to the user with the results of the function calls.

To accomplish this, the prompt_prefix.py, json_responses.py, and agents.py python files were created.

- prompt_prefix.py: The script includes a class that generates a pre-user prompt to inform the chatbot about the available functions, available variables, and their respective formats. It also gives guidance on how to deal with ways to restrict the data, as well as the format that the chatbot should output: JSON.
- json_responses.py: This script defines a class that parses the user's input into parameters and conditions (way to restrict the data), that can be easily drawn upon by later functions. Each user request is translated into an individual "json_response". The naming

convention here is because it parses the JSON output from the LLM into an easier to manage and more useful class.

- agents.py: the script defines 3 classes, the agent_functions class, the function_calling_agent, and the response_agent.
    - The agent_functions class contains the list of available functions and the ability to call on them. It contains connections to our SQL lite databse as well as the shapefiles for the maps.
    - The function_calling_agent is the "bot" that is responsible for taking the initial user input, converting it into a "json_response" object, and running the functions available to it. This agent is responsible for obtaining the answer to the user's prompt. This includes opening maps that were generated by the requests.
    - The response_agent class takes the answers given by the function_calling_agent and converts them into a more chat friendly answer for the user.

It is important to note that the use of LLMs, especially "small" ones like Mistral 7B have issues and are often not consistent. Because of this, the chatbot can often produce errors in parsing the user's requests or in properly formatting its output to JSON[1]. One strategy to aid in the chatbot's consistency was the standardization of variable names. The variable names across the datasets followed no singular pattern, and although this normally would not cause problems for data analysis, the lack of a formal pattern seemed to cause issues for the chatbot in recognizing variable names and using them in functions. We labelled each variable "psc_" followed by an integer.

User Guide

This application should be run through the terminal. **Prior to running the data_collection code, please make sure you have added "master_cvi_data_indicators.csv" and "fema_nri_census_tracts.csv" to your directory**. Both files are in the Drobox. To run the data collection code, run:

```
$ poetry run python -m project_404 data_collection
```

To run the climate chatbot, run:

```
$ poetry run python -m project_404 climate_bot
```

The user will then be prompted to input a Ngrok tunnel key. The chatbot relies on an LLM self-hosted on Google Colab, and the key is generated every time the server is started. The key is located at the bottom of the file. To get access to the Google Colab Jupiter notebook, contact Gregory Mitchell (gmitchelljr@uchicago.edu).

Once, the tunnel key has been inputted, the user can now make requests for the chatbot to provide maps or other data summaries. For best results, please refer to the "standardized"

[1] We used json_repair to help coerce Mistral's output to proper JSON, but this too is imperfect.

variable name documentation in GitHub (link)--these are the vairable names that follow the pattern psc_###.

The main functionality of the chatbot is to use SQL to help summarize data and to generate maps. You can add restrictions to the data as well (e.g., where life expectancy is less than 60). Again, because of the nature of LLMs, the ability for the chatbot to correctly parse input from the user is often inconsistent. While our code has been written around this fact, some trial and error with prompts may be needed. Also, due to time constraints, map making was the primary feature focused on and tested, but the bot can otherwise summarize the data. The details of the LLMs function calling abilities are below:

Data summaries:

i. Available function names
    i. Find top k
    ii. Sum
    iii. Average
    iv. Count
    v. Median
ii. Map making
    i. To make a map, you must specify the data to be mapped, and can specify the color of the map and the location/city of focus
    ii. The color can be a hexcode, a word, or a two-color name separated by hyphen (e.g., red-green).
    iii. The maps are programmed to open up in your browser automatically, however, if this doesn't happen, you may have to open it up manually. To do this go to "Final-Project/project_404/chatbot/maps" and open the desired map. Once opened, the map will be interactive in the browser.

Examples:

If a user wanted to learn about average earthquake risk in Los Angeles, an example of a data summary request to the chatbot would be:

>>> What is the average psc_83 in Los Angeles?

An example of a map request would be:

>>> Make me a green map of psc_83 in Los Angeles

To add a condition, for example you were interested in mapping earthquake risk in tracts that have a homeless population greater than 500:

>>> Make me a green map of psc_83 in Los Angeles if psc_30 is greater than 500

What the project tried to accomplish and what it accomplished

Our group started this project with the intention of examining if climate change would have an outsized impact on vulnerable populations across the United States-particularly in redlined communities. We intended to do this via visualizations and data summarizations generated by an AI Chatbot. For the most part, our group accomplished most of these goals. We were able to

collect significant amounts of data and feed said data into a chatbot which is programmed to summarize the data and create visualizations. However, we were not able to fully gauge if climate change impacts redlined communities more than other ones in the same city due to time constraints.

Workload Allocation:

Name	Module	Tasks	Files
Kiran	Data cleaning/processing /editing	Pulled and edited data the Richmond Redlining dataset. Pulled, cleaned, and edited data from two Climate Vulnerability Index datasets. Pulled, cleaned, and edited data from the FEMA National Risk Index	richmond.py
	Geospatial analysis	Wrote a function to conduct a geospatial join which added tract information to the Richmond Redlining data.	richmond.py
	Research	Conducted a substantial amount of research and read multiple documents to determine which indicators would work best with the LLM- particularly for the Climate Vulnerability Index and the FEMA National Risk Assessment.	N/A
	Def GO	Contributed to writing functions to make the entire program run	app.py
	Document writing/cleaning	Wrote and edited the project paper and collaborated on the ReadMe.	Project paper/ReadMe
Gayathri	Data cleaning/processing /editing	Pulled, cleaned, and edited 3 datasets from the Census.gov API.	census_data.py
	SQL database for the entire project	Created the SQL database that incorporated all the CSV files and data for the Chatbot.	sql_database.py
	Def GO	Contributed to writing functions to make the entire program run	app.py
Onur	Data cleaning/processing /editing	Pulled, cleaned, and edited data from the EPA API	epa.py
	SQL database for EPA	Functions to create/clean/merge and visualize a SQL database for EPA data	utilities.py
	Def GO	Contributed to writing functions to make the entire program run	app.py
	Document writing/ cleaning	Collaborated on the project paper.	Project paper
Gregory	LLM creation	Research and reviewed documentation on chatbot design. Wrote multiple a combination of classes and functions to create the Chatbot. This Chatbot uses the main SQL database to help summarize data and to generate visualizations	chatbot.models.agents.py
	Visualizations	Wrote multiple functions to connect the SQL database to create visualizations primarily in the form of maps.	chatbot.models.agents.py
	Def GO	Contributed to writing functions to make the entire program run	app.py
	Document writing/ cleaning	Wrote the ReadMe	ReadMe