HealthAccess documentation

This document provides a detailed overview of the entire functioning of HealthAccess. Specifically, it gives details on:

- The main program that the user would run
- The input/supporting programs for that main program
- Datasets used for HealthAccess
- APIs used for HealthAccess
- Scraping done for HealthAccess

High level summary of the main program

File name: - HealthAccess.py

- The user is asked to enter his/her street address
- The user is asked to choose one of the following 3 categories:
    - Emergency
    - Non-emergency
    - Know your well-being
- If emergency:
    - The user receives a document containing the following information about 8 hospitals, sorted in terms of closest to farthest:
        - Name of the hospital
        - Telephone number
        - Travel time from the user's address
- If non-emergency:
    - The user receives a document containing the following information about 8 hospitals, sorted in terms of closest to farthest:
        - Name of the hospital
        - Telephone number
        - Travel time from the user's address
        - Average google rating
        - Whether the facility is open now
        - Appointment page's url address
        - Whether the facility is operational or not
- If Know your well-being
    - The user gets 2 choices:
        - AQI information
        - User sleep hours comparison to national data

Supporting program files

Below is a list of all the supporting program files that enable this main program and their behind the scenes working

travel_time_finder.py

What it does: - Calculates the driving time between 2 locations
Data source:- Google maps API - distance matrix
Packages/Libraries required:- requests, json

Output:- A dict containing the following information: **{**‘text’: ‘The travel time in words’, ‘value’: ‘A numeric value for the travel time’**}**

Brief description of functioning:

- Takes 2 input parameters pertaining to 2 addresses/names of places
- Creates a search url using API_KEY (free, generated by developer)
- The query is equivalent to entering to text addresses/names in google maps interface
- Parses the dictionary returned from the query
- Returns the travel time in text and associated numeric value

hospital_website_finder_2.py

What it does: - Returns the website of a particular hospital along with some additional information
Data source:- Google maps API place text search + google maps API place details search
Packages/Libraries required:- requests, json, re

Output:- A dict containing the following information: **{‘**website’: ‘The hospital’s website’, ‘rating’: ‘The google rating for that healthcare provider’,‘open’: ‘Whether the place is open or not’,’business status’: ‘Whether the hospital is operational or not’**}**

Brief description of functioning:

- Takes a health care facility’s name as input
- Creates a search url using API_KEY (free, generated by developer)
- Runs the text search API using the input facility name
- Gets the official name (or close to official name) from the returned dict
- Runs the text search API again, using the official name this time
- Gets the google place ID for that facility
- Runs the details API using the place ID
- Retrieves website, open now, operational status and rating details from the returned dict
- Uses string matching to trim the returned website to contain the basic website name (removes stuff like https, extra /s)
- Constructs the output dict

appt_page_link.py

What it does: - Returns the appointment page link in a particular website (of a healthcare facility), if it exists
Data source:- Scraping the healthcare facility website
Packages/Libraries required:- requests, BeautifulSoup from bs4, re

Output:- A dict containing the following information: **{‘**appt_link’: ‘the appointment link available on that website’, ‘confirmation’: ‘whether that link leads to a valid web page or not’**}**

Brief description of functioning:

- Takes a health care facility’s website as input
- Constructs a url by adding ‘https://’
- Get’s the complete html code of that website as a text file
- Opens the text file, goes through each line and searches for the html tag <a> which denotes there’s a href link there, grabs all such links present and writes to an output file
- Opens the output file from the previous point, goes through each line, searches on 3 keywords denoting appointments and returns the first match.
- Performs some text cleaning on that match and constructs the appointment link by adding it to the website url
- Runs a query to check if that constructed link leads to a valid website
- Returns the dict output

hospitals_list.py

What it does: - Returns the hospital name, address, contact number along with its category
Data source:- National Plan & Provider Enumeration System National Provider Identifier API
Packages/Libraries required:- Requests, JSON, Pandas, Re

Output:- A csv containing a list of healthcare providers with specialisation categories.. The list will contain healthcare providers names, address, telephone number, and category.

Brief description of functioning:

- Takes NPPES NPI website as input
- Creates a search url using API_KEY and runs the text search API
- Gets a dataframe of hospitals with names, address, phone number, and category
- Uses string matching search to assign a simpler, user-friendly category keyword to each hospital based on its specific listed category
- Uses the dataframe to create a dict of categories with category keywords
- Returns a csv file with a list of hospital names, addresses, phone numbers, categories and category keywords.

aqi_info.py

What it does: - This program displays data on current air quality in Pittsburgh, shows a category of air quality along with a health warning and graphs monthly averages of Air Quality Index (AQI) for the past 6 months.

Data source:- https://aqicn.org

Packages/Libraries required:- requests, BeautifulSoup, pandas, re, matplot.lib, datetime

Output:- Displays current air quality indicator levels in Pittsburgh and associated category and health warning, returns a graph of monthly averages of Air Quality Index (AQI) and saves it as a PDF.

Brief description of functioning:

- Scrapes https://aqicn.org/city/usa/pennsylvania/allegheny/parkway-east/ to get current AQI levels of PM2.5, O_3 and NO_2 and the associated air quality category
- Stores every category with associated health warning (obtained from https://aqicn.org/scale/) in a dict
- Displays current AQI levels and associated category and health warning
- Reads csv file with historical AQI data downloaded from https://aqicn.org/data-platform/register/ and generates graph of monthly averages of Air Quality Index (AQI) for the past six months and plots current AQI level in the same graph
- Saves graph as PDF (AQI.pdf)

average_sleep.py

What it does: - This program shows the user that given his/her income, how different he/she is from the average sleeping hours

Data source:- American Time Use Survey of 2020 (https://www.bls.gov/tus/datafiles-2020.htm)
The user survey provides nationally representative estimates of how, where, and with whom Americans spend their time. It is the only federal survey providing data on the full range of non-market activities, from childcare to volunteering.

Packages/Libraries required:- pandas, seaborn, matplotlib

Output:-

- If working individual
    - An overlay of that user's income and sleeping hours against a regression line between sleeping hours and income (**sleep_hours_comparison.pdf**)
- If non-working individual
    - User's sleep, average sleep for similar category of people

Brief description of functioning:

- Takes the American time use survey data as input (csv)
- Asks user for 4 pieces of information
    - Age (between 5 to 70)
    - Gender (male or female)
    - Income (<=280,000)
    - Sleep hours (<=20)
- Filters the dataset to get a set of data points that are similar to the user on age, gender and employment status

- Plots a regression line between sleeping hours and income, using that filtered dataset
- Overlays a dot denoting where the user lies in that x-y plane
- If user is not working, it just reports the data on user's sleep hours and the average sleep hours from data for non working people (since regression not possible)

Main program's working

The following is a stepwise description of what the main program does.

Stage 1

- Imports the 6 supporting python files
- Imports the necessary libraries
- Puts a limit on the number of results a user sees
- Decides a random sample size (explained later)
- Runs the hospitals_list.py file to generate an input csv
- Asks the user to input their street address
- Asks the user to choose one of the 3 options this app provides

Stage 2

- Reads in the hospital_list csv
- Drops rows containing missing values
- Generates a random sample of hospitals
    - This is done because of the limited computing power and hence minimising search time. However, the logic of the code is the same and this step can be dropped if we had a larger computing power (like running on a cloud server)
- Generates a subset of the above csv containing the keywords that the user can choose. Keywords denote buckets of categories of the healthcare facilities (for example- pediatrics)
- Using the travel_time_finder program, adds 2 columns to the dataframe that contains the travel time (in text and numeric value) from the user's location to each of the hospitals in the sampled dataframe
- Since the dataset contains some individual providers whom google maps cannot search, we drop such invalid values

Stage 3 (if the user chooses emergency)

- The program just sorts based on travel time and displays the top 10 searches, along with the hospital name, telephone number and driving time
- The final result is a csv **(emergency_output.csv)**

Stage 4 (if the user chooses non-emergency)

- The program displays the list of category keywords for the user
- The user is asked to choose a particular category
- The program creates a dataframe that is a filtered list of hospitals matching the chosen category
- The program sorts that filtered list based on travel time, and keeps the top 10 results
- For each of those 10 results:

    - The program uses the hospital_website_finder_2 supporting file to query the hospital name and get the website, whether the place is open now, google ratings and operational status
    - These results are put in a list
- For each of those 10 websites retrieved in the previous step:
    - The program uses the appt_page_link supporting file to scrape and construct the appointment page url from that website, if available
    - The result is stored in a list
- After the above 2 steps are done, the relevant information is added to the data frame that is to be displayed for the user.
- Now, a selection of columns from this dataframe is exported to a csv, which forms the result of the non emergency part of the app **(non_emergency_output.csv)**

Stage 5 (if the user chooses 'Know your well-being')

- The program shows 2 choices:
    - AQI information
    - Sleep hours comparison
- If AQI information:
    - Runs the aqi_info file that scrapes current AQI, plots past trends and shows the AQI category (AQI.pdf)
- If sleep hours comparison
    - Runs the average average_sleep file that cleans the sleep dataset and plots the graph (sleep_hours_comparison.pdf)