Welcome to our cardiovascular health project!
Authors: Beatrice Katsnelson, Elise Katsnelson, Maria Tamayo, and Summer Long

The main aim of our project is to provide the user with their heart disease risk score (as well as their BMI and risk factors), cardiovascular screening frequency recommendations, geographic risk based on their state, and educational and informative articles that they may be interested in based on their responses to a cardiovascular health survey that they take when they visit our website.

The project consists of many parts, which are explained below.
Skills used: Four web crawlers with beautiful soup, natural language processing, regular expressions, Geopandas (generating a map from CSV data), machine learning, Pandas, Numpy, statistical analysis (with P values), Django, and website design (HTML and CSS)

heart_health_site/risk_calculator.py
1. Machine learning model: Firstly, we created a random forest machine learning algorithm based, which we trained and tested on a publicly available CSV dataset of 70,000 patients, for which the target variable was (1) heart disease and (0) no heart disease, and the predictor variables included age, gender, height, weight, systolic and diastolic blood pressure, cholesterol, glucose, smoking habits, alcohol consumption habits, and activity level. We originally trained the model using all of the aforementioned factors in order to predict heart disease. However, we retrieved the top 5 predictors and then created a new ML model (which is the one that we kept) using only the top 5 predictors (which was even more accurate).
We also learned how to store and deploy the machine learning model using a pickle (pkl) file. This machine learning model is stored in the car_train.pkl file.

heart_health_site/risk_calculator.py
2. Predicting risk: We used both pandas and numpy to analyze the data from the 70,000 patients in order to create a weighted points system. We used the NumPy library to determine which clinical factors were indeed risk factors (as well as the weights of different risk factors) by performing two proportion z tests and calculating the p-value to perform correlation analysis. We then integrated this weighted points system with the output of the machine learning model (either 1 or 0), and based on this pipeline, we present a user with a risk score in one of the following categories: 'low', 'low-medium', 'medium', 'medium-high', and 'high'. The code for this is in the risk_calculator.py file.

heart_health_site/econ_model.py
3. Economic cost-benefit analysis model: We conducted a cost-benefit analysis to give the user a recommendation for how often they should be screened from an economic standpoint, based on their age, risk for severe heart disease (which we predicted), and a potential cost. Since we created this model at a population level, we used the average costs in the United States, such that the average cost of cardiac screening (which is most often done using a CT scan, but may also be done using an exercise stress test or an echocardiogram) is about $400,

while the average cost for treatment for severe heart disease (which is most notably a heart attack) for in-patient and out-patient care is $35,000. Our cost-benefit analysis created a frequency for screening that is still more financially beneficial than half of the cost of a heart attack (because the goal is to ultimately pay far less, although any less would be plausible). Thus, although there may be lost wages and other financial and personal impacts, the cost of screening (even quite frequently) is significantly cheaper than the direct costs from a heart attack, and screening any more frequently than this would not create any medical benefit to further prevent heart disease. Also, the average life expectancy in the United States is 80 years, so we calculated the screening frequency for the remainder of a person's lifetime (for users who are aged 65 and below). For users above 65, it is not reasonable to assume that someone has less than 15 years left to live (especially if they are already over 80 years old), so we automatically set this assumption to be 15 years.

heart_health_site/pubmedcrawler.py
heart_health_site/googlescholarcrawler.py
heart_health_site/mayocliniccrawler.py
heart_health_site/webmdcrawler.py
heart_health_site/searchterms.py

4. Web crawlers:

Our project implements four web crawlers: PubMed, Google Scholar, Mayo Clinic, and WebMD. Constructing a series of search terms utilizing the user-inputted responses on the form, each crawler is instructed to search their respective websites for articles with relevant information for the user.

The main portion of information is provided by crawling PubMed. This outputs the title, authors, link, and abstract. To decrease run time, we limited the number of retrieved articles to nine (which we then sort to feed the user of the program the five most relevant articles, in order). Further reading is provided in the form of links from Google Scholar, MayoClinic, and WebMD.

heart_health_site/nlpgradelevel.py

5. Natural language processing

Our project, in filling out the form, invites the user to identify their "literacy level" as Basic, Intermediate, or High. This is used to provide articles in an optimized order to suit the user's comprehension. Using natural language processing, our project computes the Flesch Reading Ease Score of each abstract.

heart_health_site/nlpgradelevel.py

6. Regular expressions and Article sorting

We used regular expressions to find the number of times that the search terms (determined in the searchterms.py file) appeared in an abstract. This was done in the count_key_words function in the nlpgradelevel.py file. To sort the articles, we integrated the results from the natural language processor (which output a literacy sophistication score) and the number of search terms to generate a "relevancy score" by which we sorted the articles based on the user's literacy level input of 'basic', 'intermediate', and 'high'.

map/mortality_data.csv
map/tl_2021_us_state/tl_2021_us_state.shp

7. Geopandas / map with mortality data: We built a Geopandas map to show users their geographic risk of heart disease based on their state. We used three files in our implementation of this map:

1) map/map.py: This file creates a geopandas map based on mortality data, outlines the state, and saves map to .png file. We chose to pre-save all of the images rather than generating them every time a user visits the website to decrease runtime and make the site as user friendly as possible.
2) map/mortality_data.csv: CDC dataset that records the number of deaths and death rate (number of deaths per 100,000 total population) by state by year. We focused on 2020. URL to website we extracted dataset from: https://www.cdc.gov/nchs/pressroom/sosmap/heart_disease_mortality/heart_disease.htm
3) map/tl_2021_us_state/tl_2021_us_state.shp: shapefile for US states, provided by the US Census Bureau. URL to website we extracted files from: https://www2.census.gov/geo/tiger/TIGER2021/STATE/

We used pandas to load mortality_data.csv into a pandas DataFrame and GeoPandas to load tl_2021_us_state.shp into a GeoPandas DF. We then merged the dataframes, using STUSPS as our key, into a new dataframe and used the matplotlib.pyplot package to plot our new dataframe into a US map that a) is colored based on the mortality rates column, such that states with higher death rates appear darker, and b) highlights (borders in blue) the state the user lives in. Lastly, to optimize efficiency, we iterated through all the states, generated their respective maps and saved the outputs (50) as png files in our repository. When a user inputs their state when they fill out the cardiovascular health survey, our software finds and pulls up the image with the map corresponding to that state, versus having to run the create_map function every time a new user comes in, which would increase run time and decrease usability of our program.

heart_health_site/heart_health/forms.py
heart_health_site/heart_health/views.py

9. Django:

- Forms.py: We built out a comprehensive Django form (forms.py) and used a variety of field types to ensure that the user cannot input the wrong data type or move on without filling out all the fields. We then assigned all of the user inputs to variables to use in all the other aspects of our site.
- Views.py: We stored the user inputs from the form when a post request is made in a dictionary, where we also clean our data to store all the necessary variables in a pkl file. Additionally, we render our views in this file.
- Data cleaning: In the dataset of 70,000 people (and thus in the machine learning model), age was given in number of days. Thus, we used the DateTime package to calculate the difference between today (the day that you are using the program) and their birthdate to

determine their age in days. We also determined and stored the BMI by dividing the weight by the height squared.

heart_health_site/heart_health/templates/heart_health/index.html
heart_health_site/heart_health/templates/heart_health/survey.html
heart_health_site/heart_health/templates/heart_health/outcome.html
heart_health_site/heart_health/templates/heart_health/education.html

10. HTML and CSS: We learned the basics of HTML and CSS to build out a clean, professional, and consistent design. Additionally, we integrated Django variables into the HTML files in order to present a sophisticated and interactive site.

11. Extraneous files

- extra_ML directory: We test many machine learning algorithms in Python and R, as well as performed feature selection based on importance metrics. Our code for some of these models and for feature selection is here.