# Web Scraping and Modeling Project Documentation

## I. Introduction

### Project Overview

- This project aims to create a model top 50 hospitals in the in world and create a gpt on various factors such as quality of care, patient satisfaction, and medical services.

### Project Scope

- We will scrape data related to hospital rankings, specialties, services, and locations from the websites of the top 50 hospitals.

### Data Sources

- The data will be collected from URLs obtained from  News & World Report.

## II. Web Scraping

### Website Selection

- We will scrape data from the following websites:
  - Hospital 1
  - Hospital 2
  - ...
  - Hospital 50

### Scraping Tools and Libraries

- We will use Python libraries, such as `requests` and `BeautifulSoup`, for web scraping.

Data Collection Process

- The data will be scraped using custom scripts, following the structure of the hospital websites.

Data Cleaning

- We will perform data cleaning by removing duplicate entries and handling missing values.

Data Storage

- The cleaned data will be saved in CSV format.

## III. Data Analysis and Preparation

Data Exploration

- Data exploration revealed correlations between hospital size and the range of services they offer.

Feature Engineering

- We created a feature for the number of specialties each hospital offers.

Data Preprocessing

- Data was preprocessed to scale and normalize features for modeling.

## IV. Model Building

Model Selection

- We chose a regression model to rank the hospitals based on a weighted combination of features.

Model Training

- The model was trained on the preprocessed data.

Model Evaluation

- The model achieved an R-squared value of 0.85, indicating a strong fit.

## V. Results and Insights

Ranking of Top Hospitals

- The top hospitals based on the model are as follows:

    Hospital A
    Hospital B
    ...

Key Findings

- Hospitals with a higher number of specialties tend to rank higher in the model.

## VIII. References

Data Sources

- Data was sourced from U.S. News & World Report.

Libraries and Tools

- Python libraries used in the project include requests, BeautifulSoup, and scikit-learn.

## IX. Appendices

Code

- Code snippets for web scraping, data cleaning, and modeling can be found in the project repository.

Data Dictionary

- Data dictionary explaining the meaning of each variable is available in the project documentation.

**Beautiful Soup is a Python library that provides tools for web scraping HTML and XML documents. It creates a parse tree from page source code that can be used to extract data in a hierarchical and more readable manner. Beautiful Soup makes it easy to navigate and search through the structure of an HTML or XML document, allowing you to extract specific data, such as text, links, and tags, from web pages. Here's a more detailed explanation of Beautiful Soup:**

HTML/XML Parsing: Beautiful Soup parses HTML and XML documents, creating a structured representation of the document. This allows you to navigate and manipulate the document's elements.

Installation: To use Beautiful Soup, you need to install it first. You can install it via pip using `pip install beautifulsoup4`.

Creating a BeautifulSoup Object: To start working with Beautiful Soup, you typically begin by creating a BeautifulSoup object. You pass in the HTML or XML document as a string and specify the parser to use (e.g., 'html.parser').

Navigating the Parse Tree:

- Beautiful Soup provides methods to navigate the parse tree, such as `find()`, `find_all()`, `select()`, and `find_parents()`, which allow you to search for specific elements and extract data.
- You can access elements and their attributes directly, like a dictionary, using dot notation, or using the `find()` method.

Searching and Filtering:

- Beautiful Soup makes it easy to search for elements based on tag names, attributes, and more. You can use methods like `find()`, `find_all()`, and `select()` for this purpose.
- You can filter elements by their attributes or text content.

Modifying the Parse Tree:

- You can modify the parse tree by adding, modifying, or removing elements and attributes.
- This is useful when you want to clean up or customize the HTML content.

Output and Encoding:

- Beautiful Soup can pretty-print the parse tree to produce a more human-readable version of the document.
- You can also encode the parse tree as a string using different encoding formats.

Beautiful Soup is a powerful tool for web scraping and parsing HTML and XML documents, and it's widely used in web scraping and data extraction projects. It simplifies the process of working with complex and messy web data, allowing you to focus on extracting the information you need.

In HTML, the h3 tag is used to define a subheading or a third-level heading within a document. h3 tags are part of the HTML structure that organizes content into a hierarchy of headings. When you want to extract h3 elements and their associated links (usually within the a tags), you typically use Beautiful Soup's methods for parsing and navigating the HTML document. Here's a detailed explanation of how to extract h3 elements and their associated links from HTML using Beautiful Soup:

- Creating a BeautifulSoup Object: First, you need to create a BeautifulSoup object by parsing the HTML content. You can use the html.parser or any other supported parser, depending on your needs.
-
- Navigating and Extracting h3 Elements: Beautiful Soup provides methods to navigate the parse tree and extract specific elements. In this case, you can use the find_all() method to find all h3 elements in the document.
- python
- Iterating Over h3 Elements: Once you've extracted the h3 elements, you can iterate over them to extract the associated links, usually within adjacent a tags. You can use Beautiful Soup's navigation methods to find the adjacent elements.
- In this example, we first find all h3 elements in the HTML content using find_all(). Then, for each h3 element, we search for the adjacent a tag to extract the associated link using find_next('a'). We extract the text within the h3 tags and the href attribute of the a tags. This allows you to capture h3 elements and their associated links in an organized manner from the HTML document.

# Llm gpt output::::

**Entering new AgentExecutor chain...**

***Thought:*** ***I need to find out which hospitals have doctors who specialize in Rheumatology***

***Action: python_repl_ast***

***Action Input: df[df['Specialty'] == 'Rheumatology']['Hospital Name'].unique()***

Observation: ***['Providence Holy Cross Medical Center' 'Sutter Roseville Medical Center'***

***'UCLA Santa Monica Medical Center' 'Poudre Valley Hospital'***

***'Christiana Hospital' 'Emory University Hospital Midtown']***

Thought:/usr/local/lib/python3.10/dist-packages/langchain/tools/python/tool.py:137: LangChainPendingDeprecationWarning: On 2023-10-27 this module will be be deprecated from langchain, and will be available from the langchain-experimental package.This code is already available in langchain-experimental.See https://github.com/langchain-ai/langchain/discussions/11680.

warn_deprecated(

***I now know the final answer***

***Final Answer:*** ***Providence Holy Cross Medical Center, Sutter Roseville Medical Center, UCLA Santa Monica Medical Center, Poudre Valley Hospital, Christiana Hospital, and Emory University Hospital Midtown.***

**> Finished chain.**

'Providence Holy Cross Medical Center, Sutter Roseville Medical Center, UCLA Santa Monica Medical Center, Poudre Valley Hospital, Christiana Hospital, and Emory University Hospital Midtown.