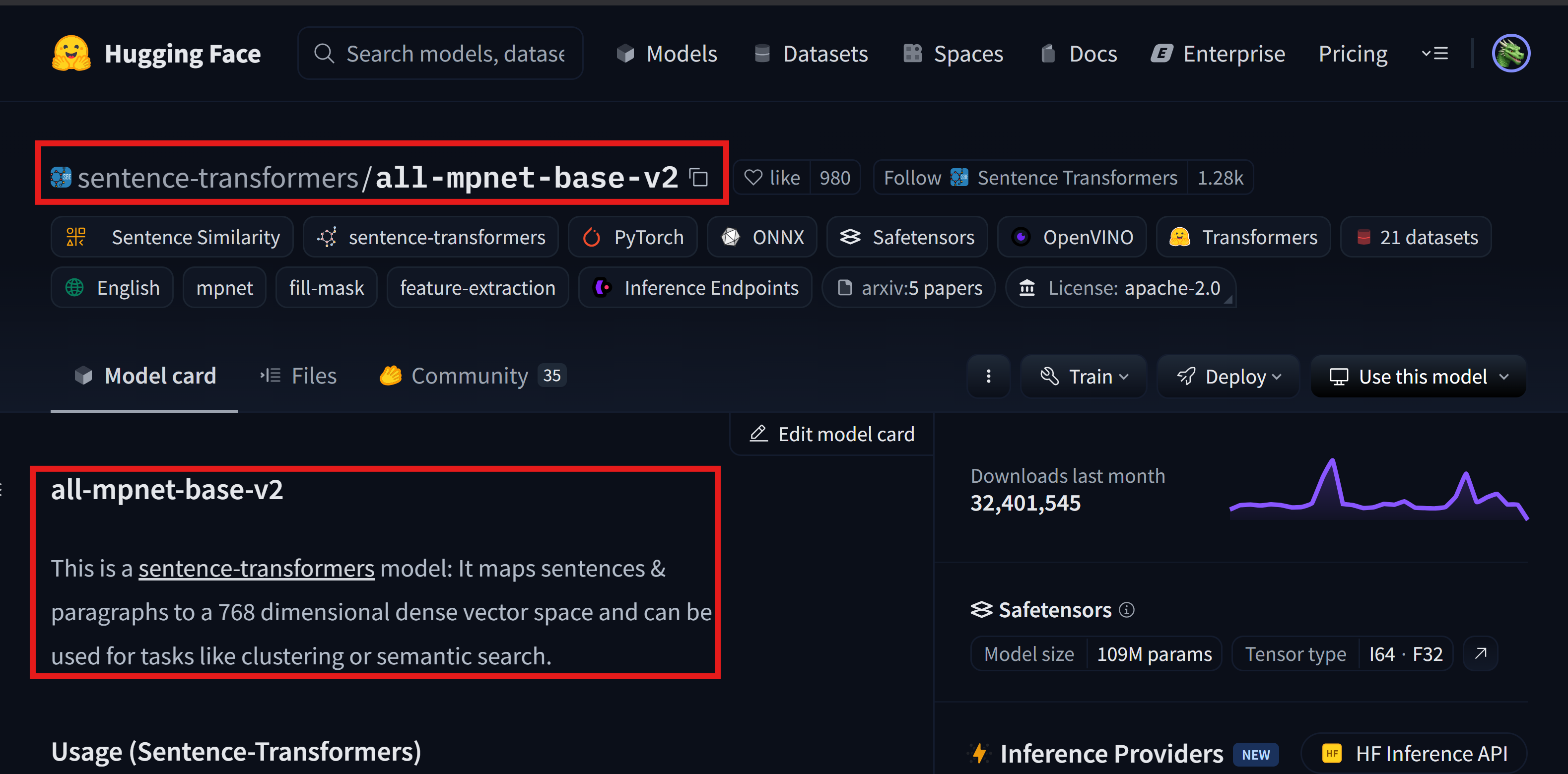
**Project Report: Data Science and Machine Learning Q&A Chatbot**

**1. Project Overview:**

The goal of this project was to build a question-answering chatbot capable of providing information about data science and machine learning concepts, algorithms, and techniques. The chatbot was built using the Rasa Open Source framework, leveraging its NLU (Natural Language Understanding) and dialogue management capabilities. A key component of the chatbot is its ability to retrieve answers from a predefined knowledge base, using Sentence-BERT embeddings for semantic similarity matching. A basic web frontend was created using HTML, CSS, and JavaScript to allow users to interact with the chatbot.

**2. Key Technologies Used:**

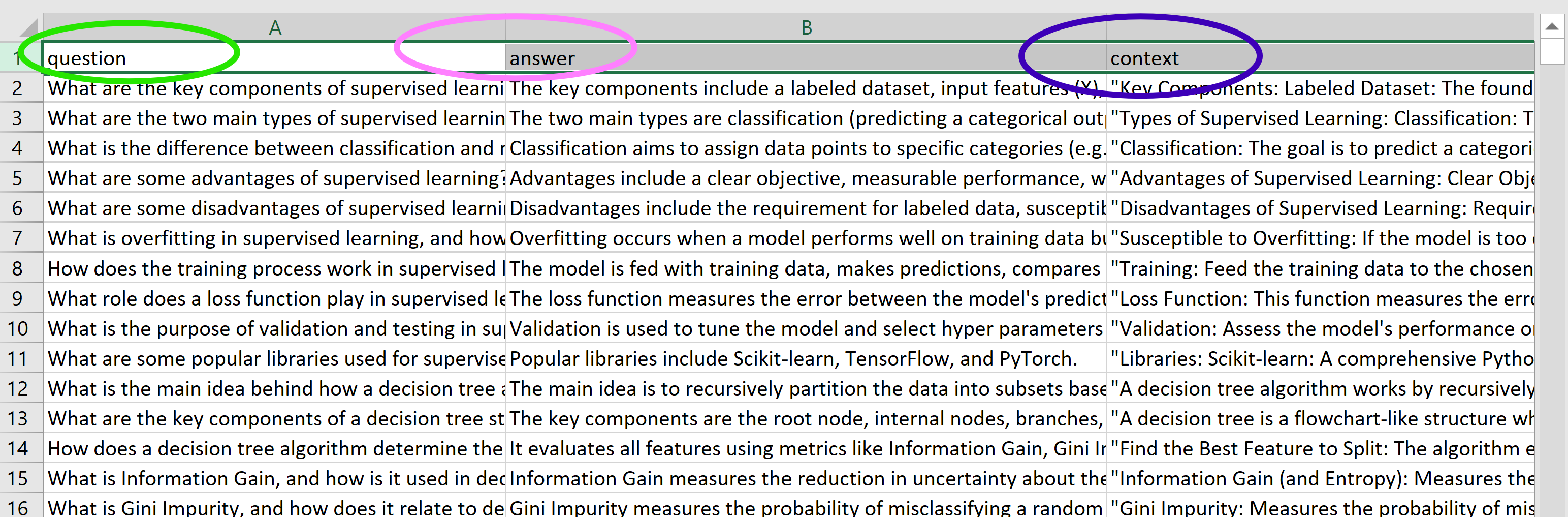
* **Rasa Open Source (3.x):** The foundation of the chatbot. Rasa handled intent classification, entity extraction, dialogue management, and action execution.
* **Python:** Used for custom actions (the core question-answering logic) and data processing scripts.
* **Sentence Transformers (all-mpnet-base-v2):** A pre-trained Sentence-BERT model was used to generate embeddings for both user questions and the questions in the knowledge base. Cosine similarity between these embeddings was used to find the most relevant answer.



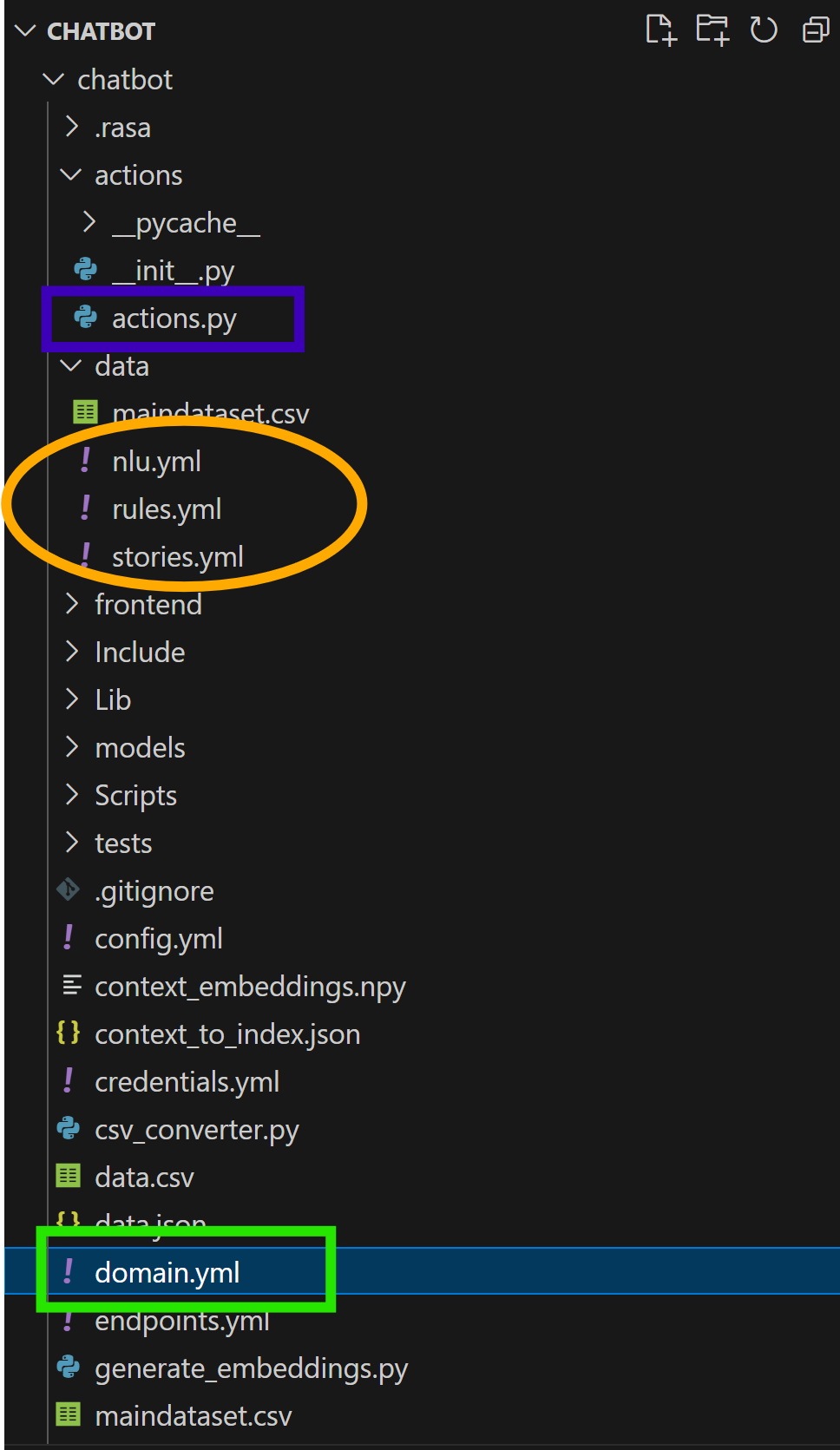
* **Pandas:** Used for reading and manipulating the knowledge base data from a CSV file.
* **NumPy:** Used for handling the numerical embedding data.
* **HTML, CSS, JavaScript:** Used to create a simple, user-friendly web frontend for interacting with the chatbot.
* **Rasa's REST API:** The communication between the frontend and the Rasa server was handled via Rasa's built-in REST API.
* **yq (YAML Processor):** Used for automated correction of YAML indentation.
* **VS Code (with YAML Extension):** Recommended development environment.
* **JSON:** Used for storing the mapping between normalized questions and their indices in the embedding array.

**3. Project Development Steps:**

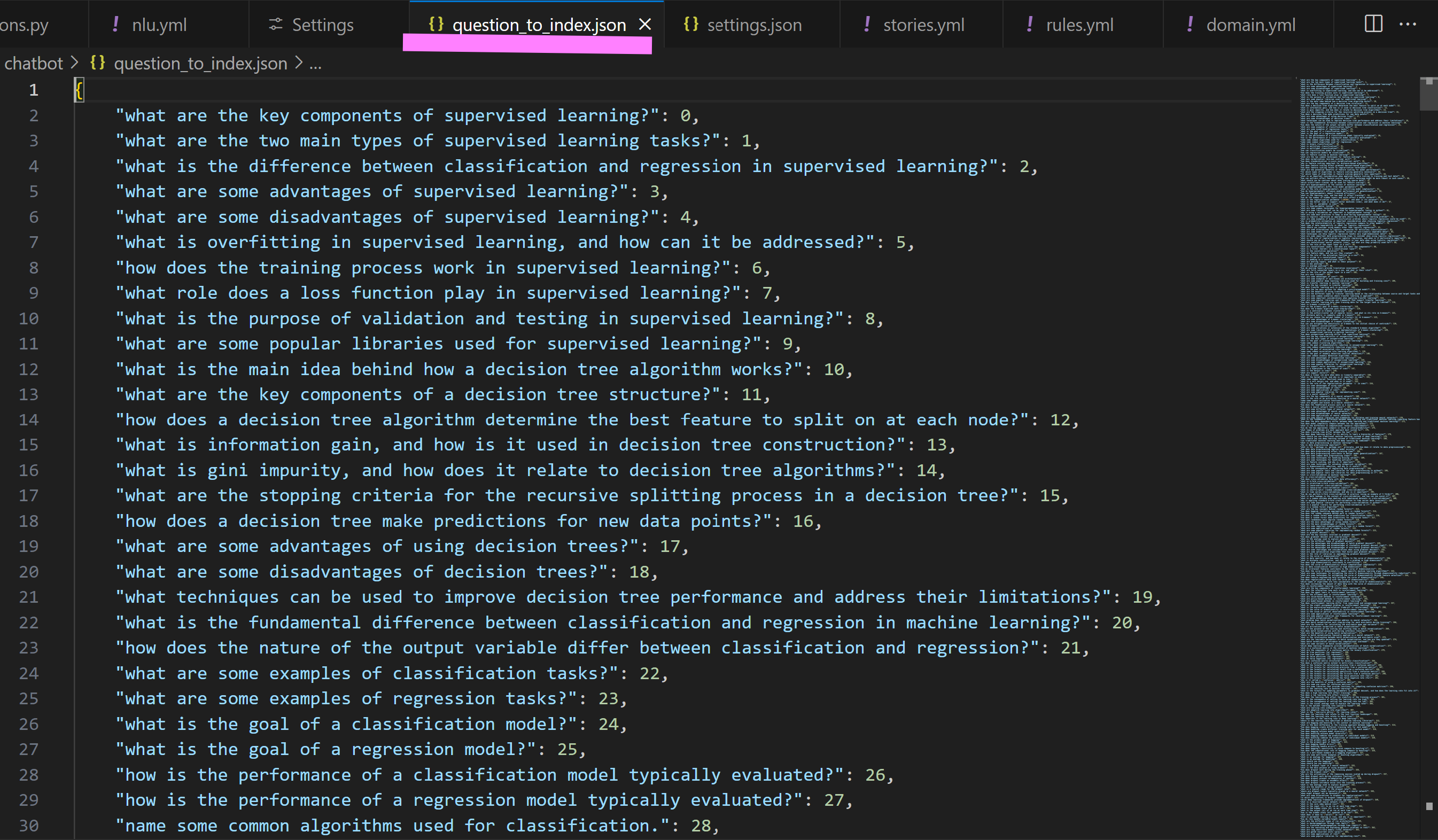
* **Data Collection and Preparation:**
  + A CSV file (data.csv) was created to serve as the knowledge base, containing columns for question, answer, and context.
  + A large set of questions related to data science and machine learning was compiled.
  + Corresponding answers and contextual information were added to the dataset.



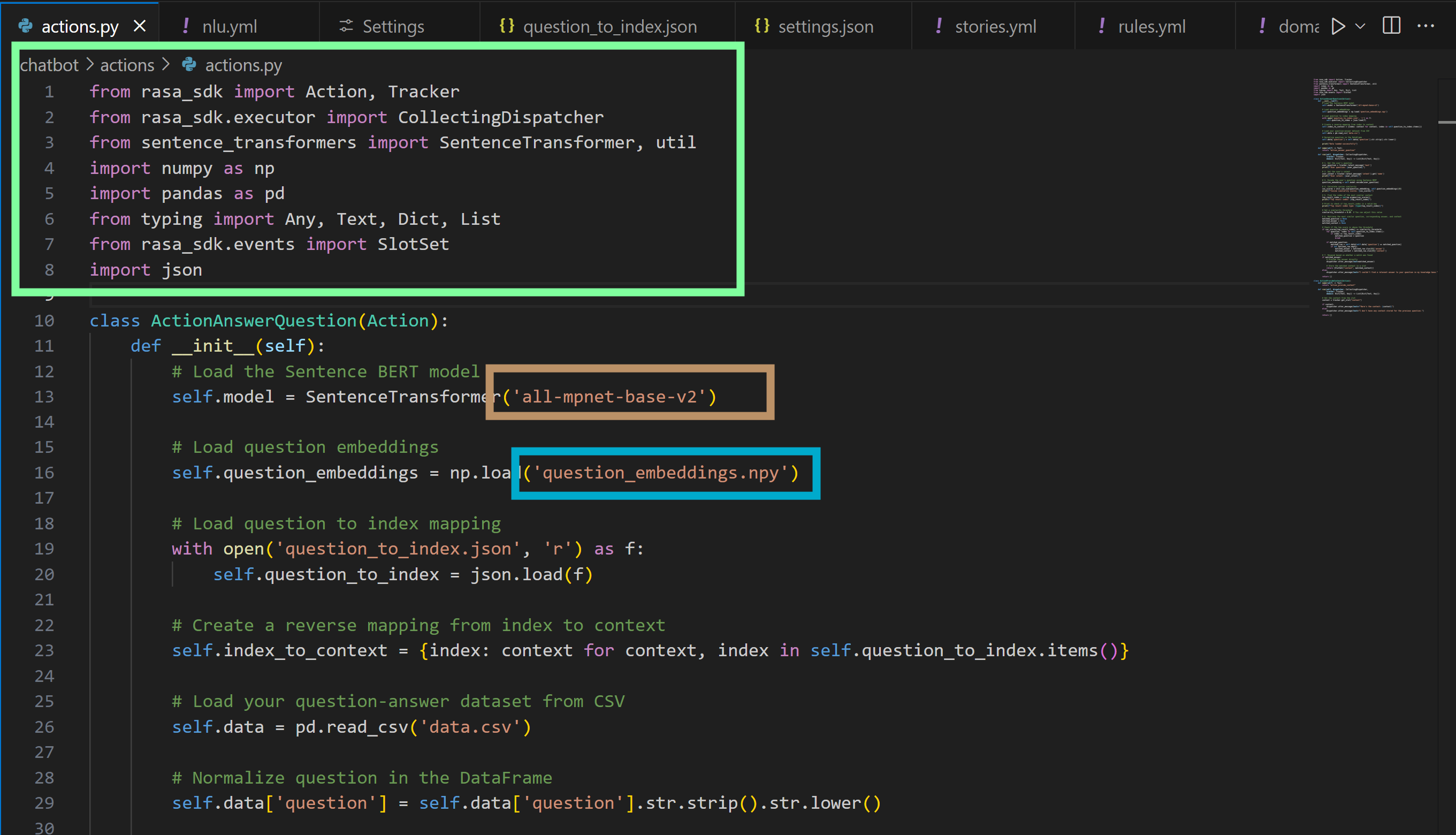
* **Rasa Project Setup:**
  + A new Rasa project was initialized using rasa init.
  + The necessary files (config.yml, domain.yml, data/nlu.yml, data/stories.yml, data/rules.yml, actions/actions.py) were created and populated.



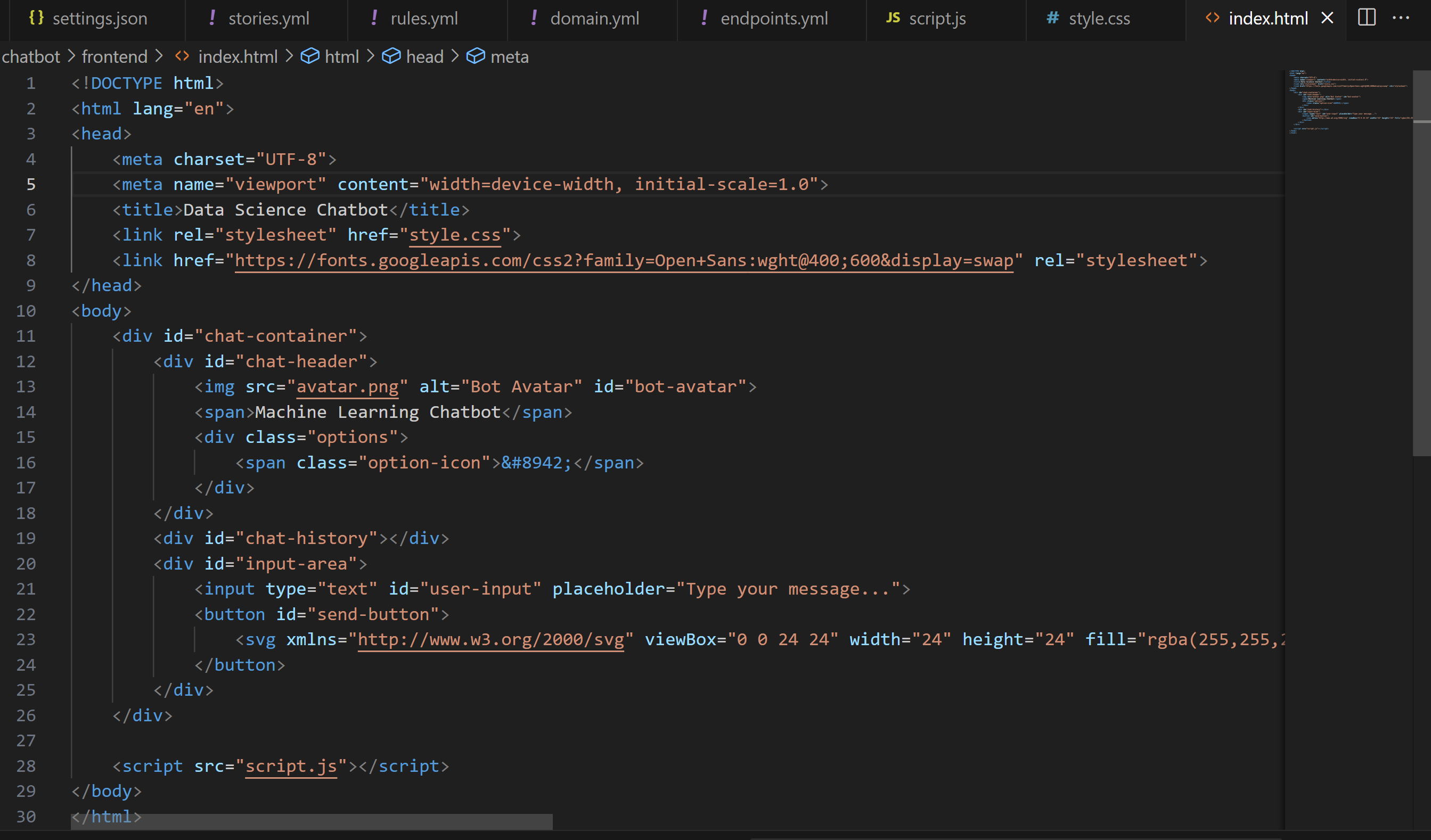
* **Embedding Generation:**
  + A Python script (generate\_embeddings.py) was created to:
    - Load the data.csv file.
    - Extract the question column.
    - Normalize the questions (lowercase, remove extra whitespace).
    - Use the all-mpnet-base-v2 Sentence-BERT model to generate embeddings for each question.
    - Save the embeddings to a NumPy file (question\_embeddings.npy).
    - Create and save a mapping of normalized question strings to their indices in the embeddings array (question\_to\_index.json).



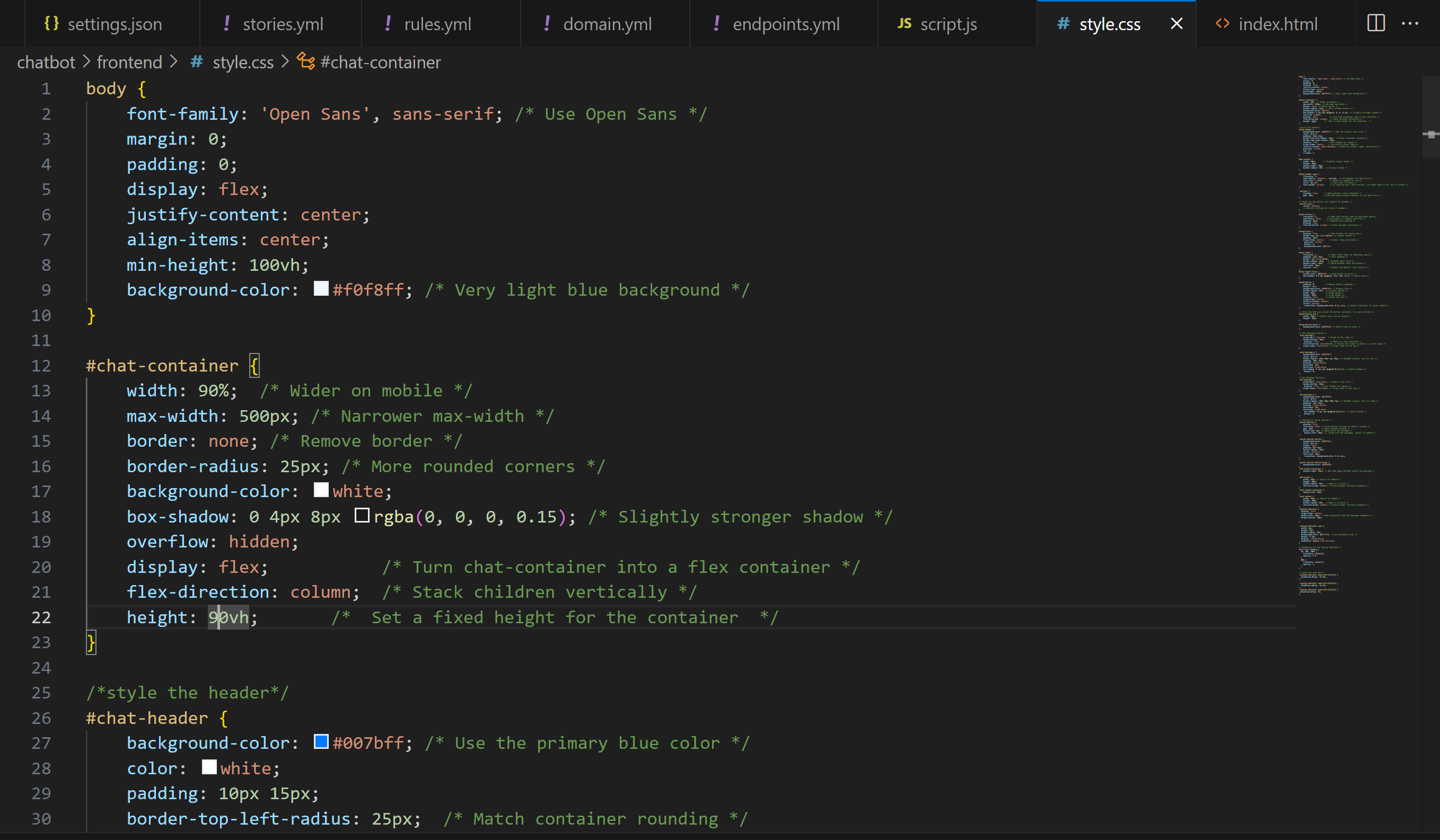
* **Custom Action Development (actions.py):**
  + A custom Rasa action (action\_answer\_question) was created to handle question answering. This action:
    - Receives the user's question from Rasa.
    - Encodes the user's question using the same Sentence-BERT model.
    - Calculates the cosine similarity between the user's question embedding and all pre-generated question embeddings.
    - Finds the question with the highest similarity score.
    - Retrieves the corresponding answer (and optionally, context) from the data.csv file.
    - Returns the answer to the user.
    - Stores the context in a slot, so it can be retrieved if requested.
  + A custom action (action\_provide\_context) was created.
  + A welcome message was implemented.



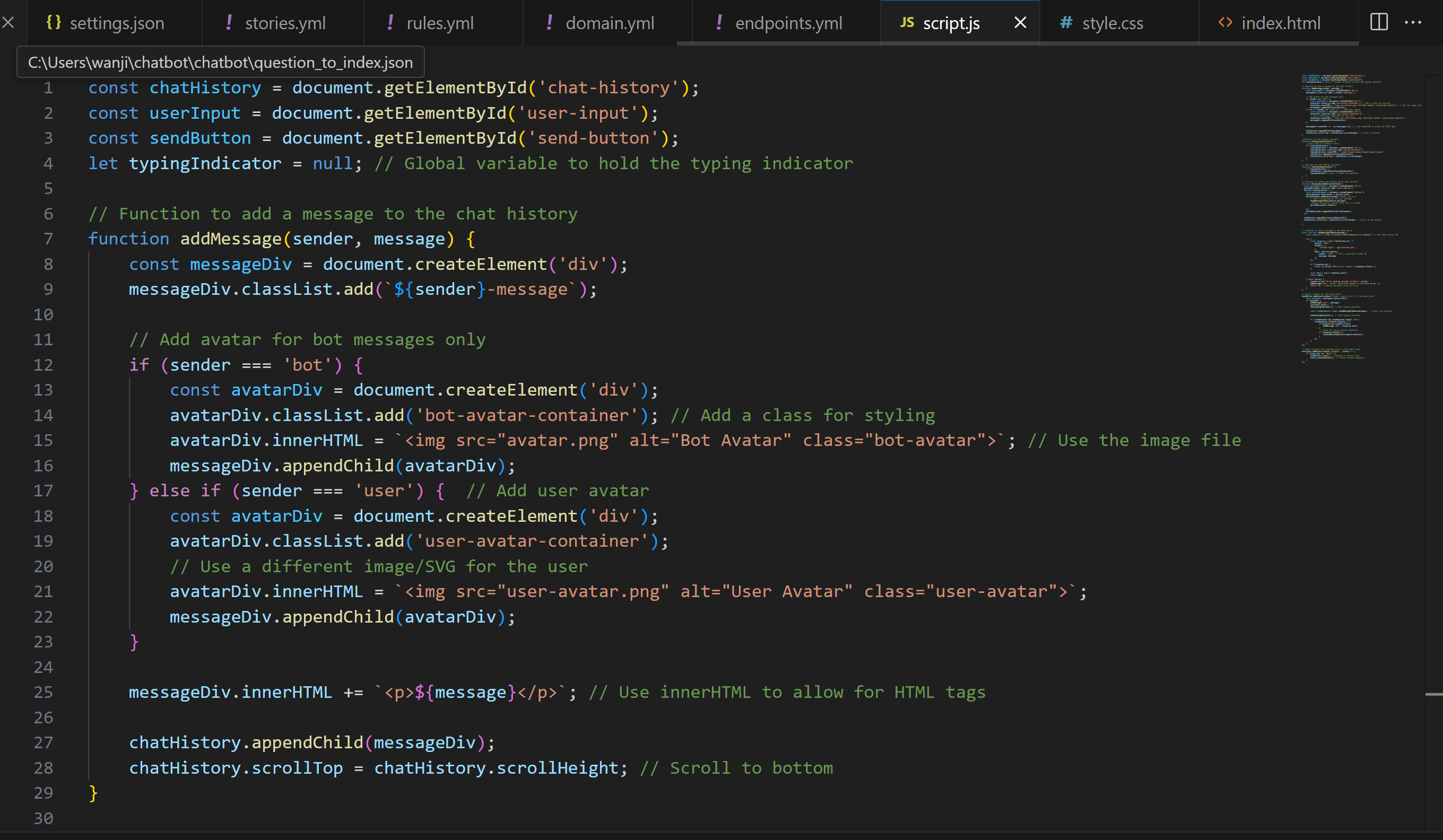
* **NLU Training Data (nlu.yml):**
  + Intents were defined to represent different types of user requests (e.g., ask\_definition, ask\_explanation, ask\_comparison, greet, goodbye, show\_context, etc.).
  + A large number of training examples were created for each intent, covering a wide range of phrasing variations.
  + The topic entity was defined and used to label relevant concepts within the examples.
* **Dialogue Management (Stories and Rules):**
  + stories.yml: Stories were created to define example conversation flows, including handling follow-up questions, user feedback (affirmation/denial), and requests for context.
  + rules.yml: Rules were defined to handle specific, predictable conversation turns, such as greetings, goodbyes, and triggering the custom action based on the detected intent. A rule was also added to handle nlu\_fallback.
* **Domain Definition (domain.yml):**
  + The domain.yml file was configured to define the chatbot's "universe":
    - All intents used in the NLU data, stories, and rules.
    - The topic entity.
    - The context slot (to store context information).
    - The custom actions (action\_answer\_question, action\_provide\_context, action\_default\_fallback).
    - **Configuration (config.yml):**
  + The Rasa NLU pipeline and dialogue management policies were configured. Key components include:
    - WhitespaceTokenizer
    - RegexFeaturizer
    - LexicalSyntacticFeaturizer
    - CountVectorsFeaturizer
    - DIETClassifier
    - EntitySynonymMapper
    - ResponseSelector
    - FallbackClassifier
    - MemoizationPolicy
    - RulePolicy
    - UnexpecTEDIntentPolicy
    - TEDPolicy
* **Frontend Development (index.html, style.css, script.js):**
  + A basic web-based chat interface was created using HTML, CSS, and JavaScript.
  + The JavaScript code handles:
    - Capturing user input.
    - Sending messages to the Rasa server via the REST API.
    - Receiving responses from the Rasa server.
    - Displaying messages in the chat history.
    - Displaying a typing indicator.
    - Handling quick replies (buttons).
    - Displaying a welcome message.
    - Display of user and bot avatars.

HTML

CSS



JAVA SCRIPT



* **Training and Testing**
  + The command rasa train was used to train the chatbot
  + The command rasa shell --debug was used to test the chatbot locally.

**4. Challenges and Solutions:**

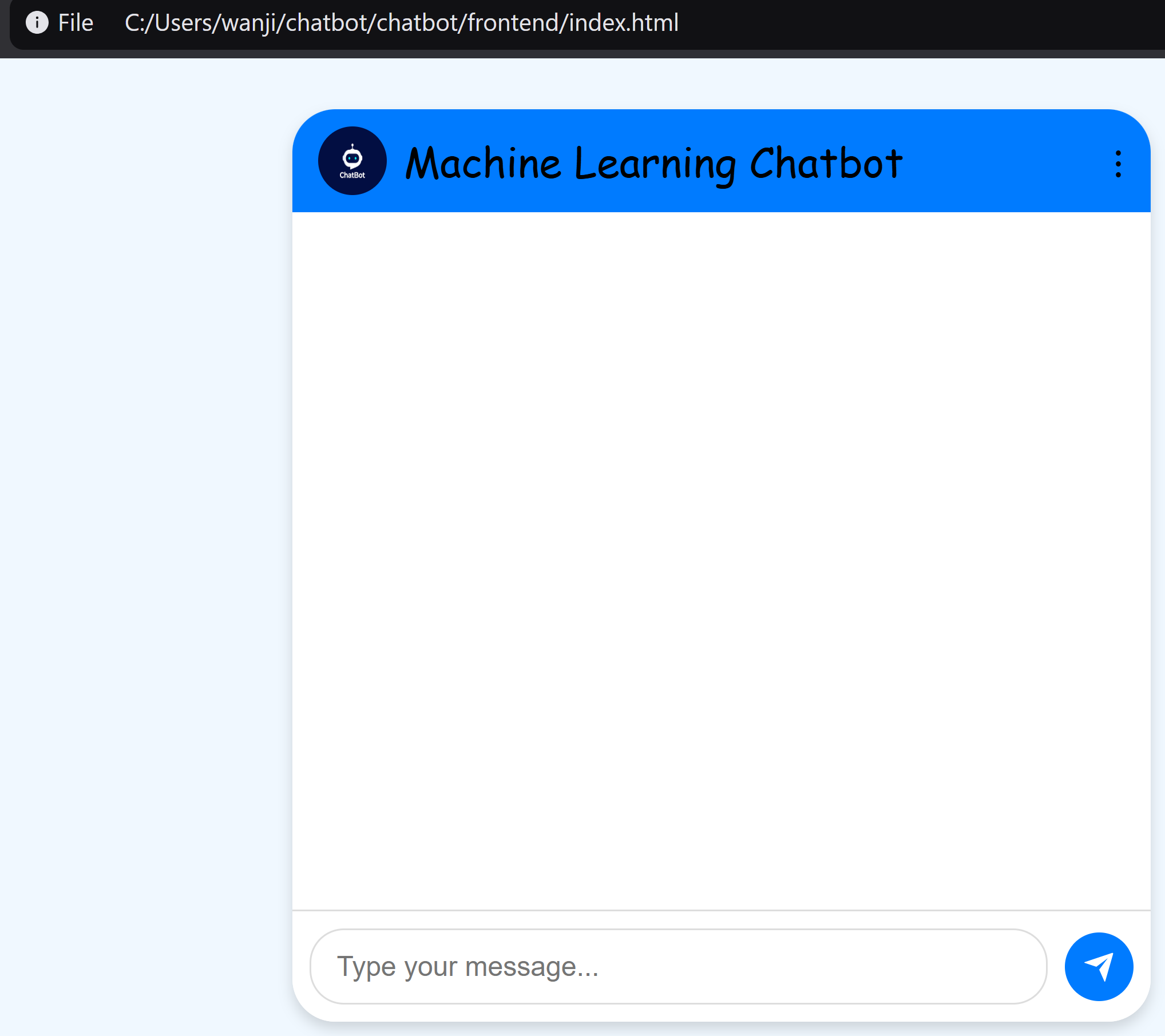
The development process involved troubleshooting several key issues:

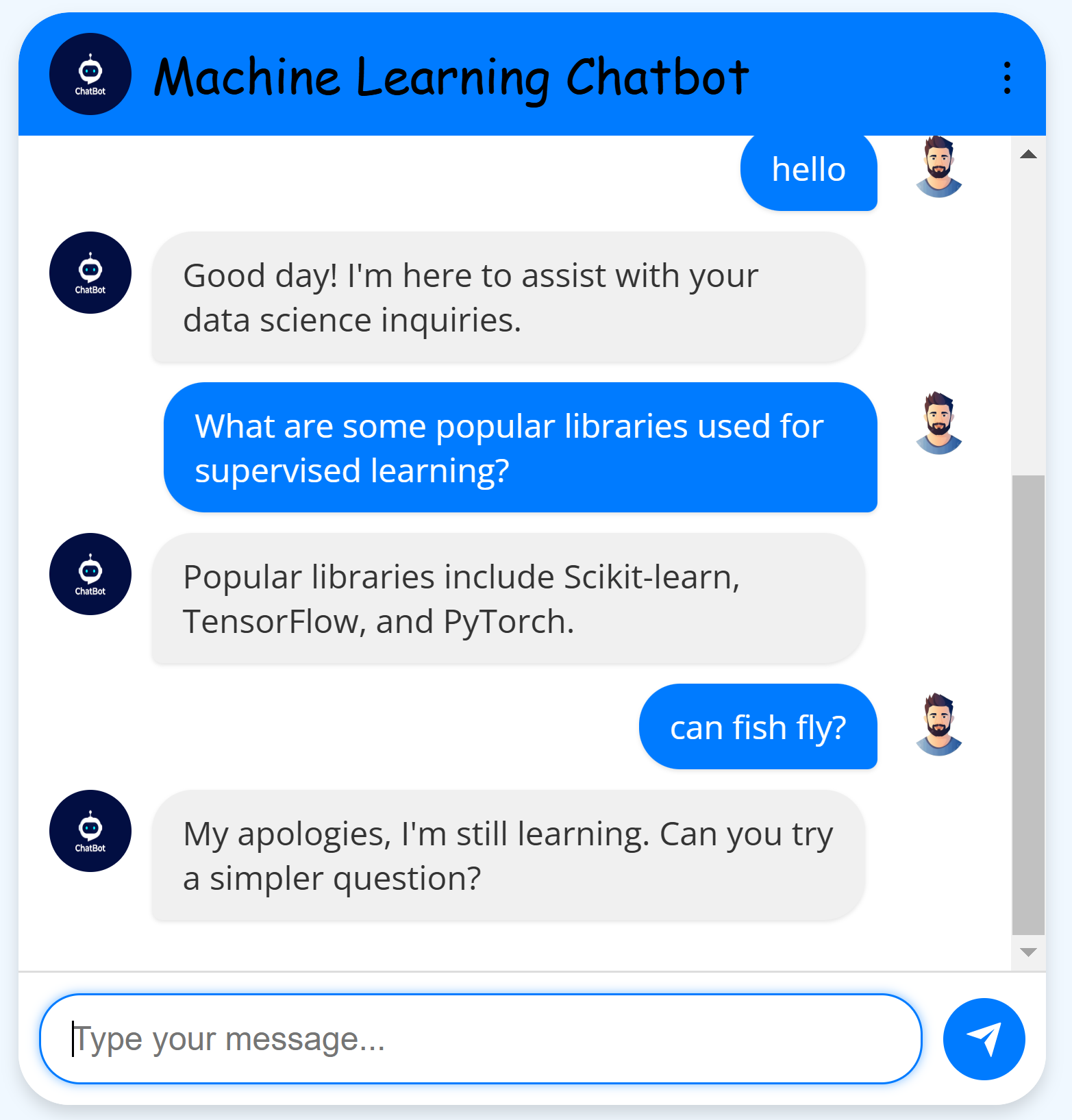
* **YAML Syntax Errors:** Indentation errors and incorrect YAML structure in the nlu.yml file prevented Rasa from training. These were resolved using a combination of:
  + **yq:** The yq command-line tool was used to automatically reformat the YAML file and correct indentation issues. This was the *primary* solution for the indentation problems.
  + **VS Code with YAML Extension:** Using a text editor with YAML syntax highlighting and automatic indentation helped prevent future errors.
  + **Online YAML Validators:** Online validators were used to check for YAML syntax errors.
  + **Manual Inspection:** Careful examination of the YAML file, guided by error messages.
* **KeyError: 'context' / AttributeError in actions.py:** These errors were caused by problems in the custom action code and inconsistencies in the data:
  + **Incorrect Key Names:** The code was initially trying to access data using incorrect key names (lowercase "context" instead of "Context").
  + **Data Inconsistencies:** There were mismatches between the context values in data.json and the keys in context\_to\_index.json. This was resolved by normalizing the context strings (lowercasing and stripping whitespace) during both embedding generation and data loading in the custom action.
  + **Incorrect Indexing Logic:** The initial code had some flaws in how it used the top\_result\_index to retrieve the correct data. The use of dictionary was to aid in data retrieval.
  + **Missing import:** The json and SlotSet were not imported in the file.
  + **Missing self:** There was an error of trying to use a variable before defining it.
* **Intent Misclassification (nlu\_fallback) and Incorrect Actions:** The chatbot was sometimes misclassifying user questions and triggering incorrect actions (like utter\_goodbye). This was addressed by:
  + **Improving NLU Data:** Adding *many* more training examples to the nlu.yml file, focusing on variety and covering different ways of asking questions.
  + **Adding Specific Intents:** Breaking down broader intents (like ask\_about\_algorithm) into more specific intents (e.g., ask\_about\_linear\_regression) *could* be a future improvement, but was not strictly necessary given the large number of examples.
  + **Adding a Fallback Rule:** Creating a rule in rules.yml to explicitly handle the nlu\_fallback intent and provide a helpful default response.
  + **Reviewing and Refining Stories:** Ensuring that the stories.yml file covered a wide range of conversation flows.
* **Domain inconsistencies:** These were caused by intents and actions not being defined in the domain file.
* **Frontend Issues**
  + The send button was not working because of an incomplete Javascript code.
  + The yaml syntax errors had to be fixed.

**5. Final Result:**

The final result is a functional Rasa chatbot that can answer questions about data science and machine learning, based on the provided knowledge base. The chatbot has a simple but user-friendly web interface. The key features include:

* **Semantic Similarity Search:** Uses Sentence-BERT embeddings for accurate question matching, even with variations in phrasing.
* **Custom Action Logic:** The core question-answering logic is handled efficiently by a Python custom action.
* **Fallback Mechanism:** Handles cases where the NLU model is not confident.
* **Basic Conversation Flow:** Supports greetings, goodbyes, and some basic chitchat.
* **Web Interface:** Provides a simple way for users to interact with the chatbot.
* **Typing indicator**
* **Quick replies**





**6. Future Improvements:**

This project provides a solid foundation. Here are some potential future improvements:

* **Expand Knowledge Base:** Add more data to data.csv to cover a wider range of topics and concepts.
* **Refine NLU Data:** Continuously improve the nlu.yml file by adding more examples, paraphrasing existing ones, and addressing any remaining misclassifications.
* **More Complex Dialogue Management:** Create more complex stories in stories.yml to handle multi-turn conversations and more nuanced user interactions.
* **Entity Extraction:** Train the NLU model to extract entities beyond just the topic (e.g., specific algorithm names, library names, etc.).
* **Frontend Enhancements:** Improve the frontend's design, add features like user history, message editing, and rich media support.
* **Fine-tuning Sentence-BERT:** For improved accuracy within the data science domain, consider fine-tuning the Sentence-BERT model on a larger, domain-specific dataset.
* **Integration with External APIs:** Connect the chatbot to external APIs to provide real-time data or perform calculations.
* **Deployment:** Deploy the chatbot to a server to make it accessible to users.
* **User Authentication:** Implement user authentication if you need to personalize the chatbot's responses or track user-specific data.
* **Analytics:** Track chatbot usage and performance metrics to identify areas for improvement.
* **Switch to a Frontend Framework:** Migrate the frontend to a framework like React, Vue.js, or Angular for better maintainability and scalability.

This project demonstrates the core principles of building a question-answering chatbot with Rasa and provides a strong starting point for further development. The iterative process of data refinement, model training, and testing is key to building a successful conversational AI.

Sincerely,

Lawrence Gitau.