Front-End Documentation for Streamlit Application

Overview

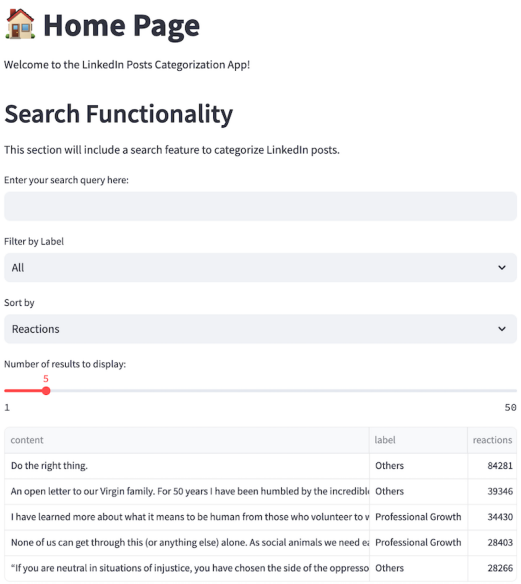
This web application is built using Streamlit to allow users to input corpus data and receive real-time results based on Python backend processing. The application automatically handles the frontend (HTML, CSS, JavaScript) via Streamlit, providing an interactive and user-friendly interface.

The application allows users to enter values through a clean interface and displays the corresponding results on the same page. The user interacts with the app by submitting queries that are processed by the backend.

Home Page

1. Key Features

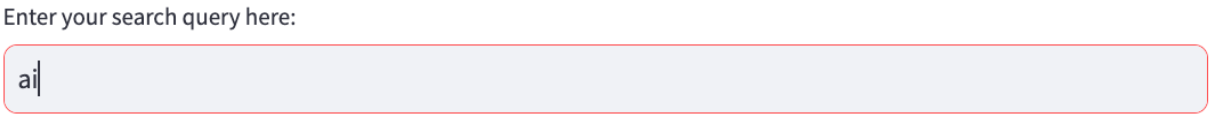
- Text Input: A field where users can enter a specific value (e.g., a number or text).
- Slider: Allows users to select a numeric value within a defined range.
- Filter and Sort by Menu: Triggers the backend processing the category when choosing the option.
- Dynamic Output: Displays the results or feedback based on user input.



2. Home Page User Interaction:

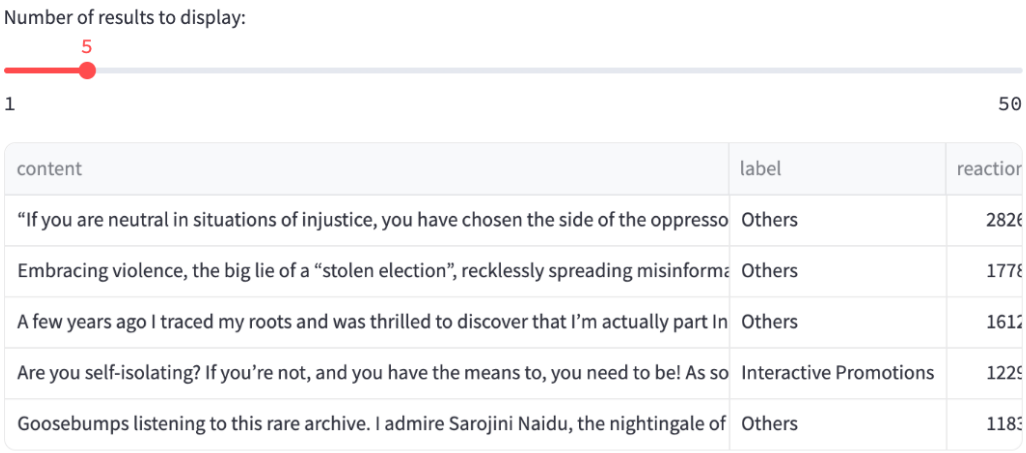
    1. Text Input Field:

- Users can enter a value in the text input box as the keyword, for example, a number.
- After the user presses the enter button, the app will dynamically reflect the entered value in real-time as it updates.



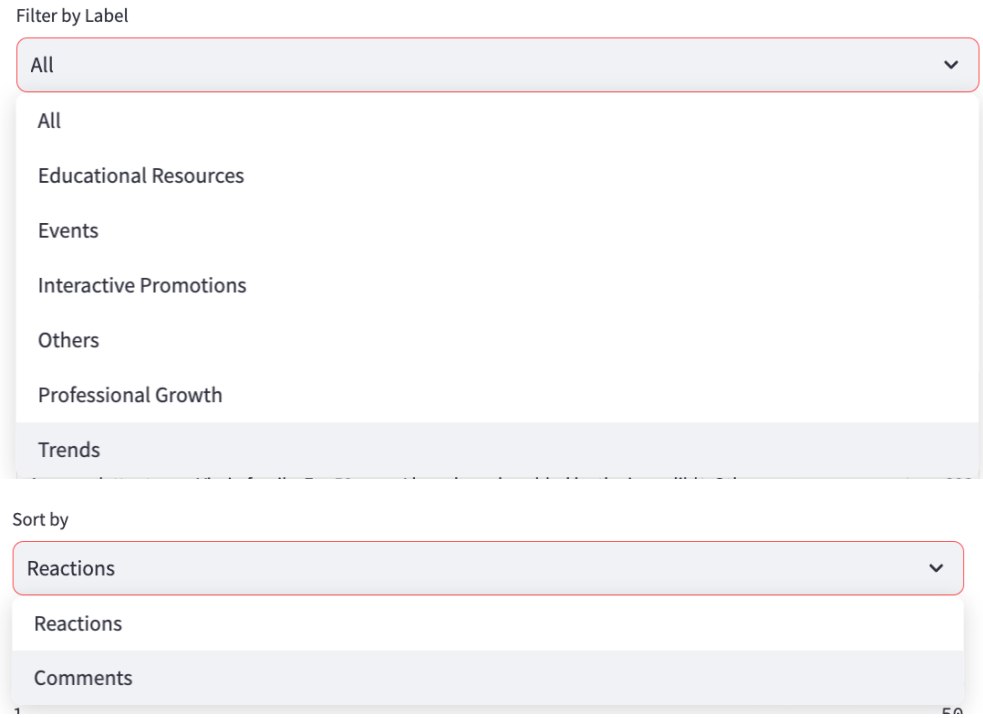
    2. Slider:

- A slider allows users to select a numeric value within a specified range.



3. Filter and Sort by Menu:

- The two menus allow users to filter out one of the labels and sort the data in specific order.



4. Result Display:

- The output dynamically reflects the results based on the inputs provided by the user.

The following Streamlit components are used in this application:

- st.text_input(): A simple text input box where the user enters data.

- st.slider(): A slider component that allows the user to select a numeric value.
- st.write(): Displays the results or any other outputs dynamically.

3. Frontend-Backend Communication

Streamlit automatically handles communication between the frontend and backend, which eliminates the need for explicit JavaScript coding. When a user submits data via the form (e.g., text input, slider), Streamlit passes the data to the backend (Python script), and the results are immediately reflected on the same page.

In the backend:

- The input data is retrieved from the frontend using Streamlit components such as st.text_input() or st.slider().
- After processing the data, the result is sent back to the frontend using st.write() or similar functions.

4. How to Use

Enter Input:

- In the input field, type a number or text that you'd like to use as the keyword for processing.

Select a Value:

- Use the slider to choose a numeric value for the length of displayed instances (or leave it at the default value).

Filter or Sort:

- Choose your preferred option to filter or sort the data.

View the Result:

- The application will dynamically display the results on the screen based on the inputs you provide.
- To view the full content, simply double-click the text.
- You can also freely download the data.

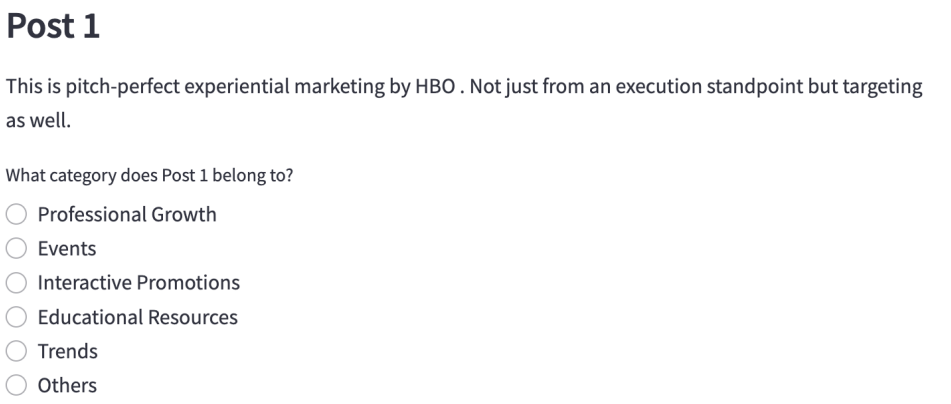
Game Page

1. Key Features

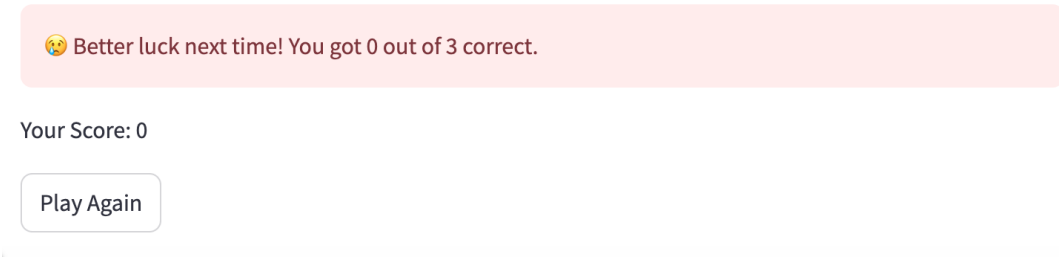
- Post Display: Displays three randomly sampled LinkedIn posts for categorization.
- Radio Buttons: Allows users to select a category for each post from a list of options.
- Submit Button: Triggers backend processing to evaluate guesses and calculate the score.
- Dynamic Feedback: Shows results with custom messages and emojis based on the number of correct guesses.
- Score Tracking: Maintains and displays a cumulative score in the sidebar.
- Play Again Option: Resets the game with new posts for repeated play.

2. User Interaction

1. Post Display and Category Selection:
- Three randomly sampled LinkedIn posts are displayed with their content.
- For each post, users select a category using a radio button interface.



2. Submit Button:
- After selecting categories for all posts, users click the "Submit" button to evaluate their guesses.
- The app processes the input and updates the interface with results in real-time.
3. Dynamic Feedback:
- Feedback is displayed based on the number of correct guesses:



3. Streamlit Components Used

- **st.subheader()**: Displays post numbers (e.g., "Post 1").
- **st.write()**: Shows post content, feedback messages, and the total score.
- **st.radio()**: Provides category selection options for each post.
- **st.button()**: Triggers submission ("Submit") or game reset ("Play Again").
- **st.success()**, **st.warning()**, **st.error()**: Displays styled feedback messages with emojis.
- **st.balloons()**: Adds a celebratory animation for perfect scores.
- **st.sidebar.write()**: Shows the cumulative score in the sidebar.
- **st.rerun()**: Reloads the app with new posts when "Play Again" is clicked.

4. Frontend-Backend Communication

Streamlit seamlessly manages communication between the frontend and backend:

- **Input Collection**: User guesses are collected via st.radio() and stored in a list.
- **Backend Processing**: Upon clicking "Submit," the calculate_score() function compares guesses to actual labels and updates the session state.
- **Output Display**: Results are rendered using st.write(), st.success(), etc., with no explicit JavaScript required.
- **Session State**: Persistent variables (e.g., score, attempted, posts) are managed via st.session_state to maintain game progress across interactions.

In the backend:

- Data is loaded and sampled using load_data() and pandas.
- The display_posts_and_collect_guesses() function handles UI rendering and input collection.
- The calculate_score() and display_results() functions process and display outcomes.

5. How to Use

View Posts:

- Three LinkedIn posts are automatically displayed with their content.

Select Categories:

- For each post, choose a category using the radio buttons.

Submit Guesses:

- Click the "Submit" button to evaluate your selections.

View Results:

- Check the feedback message and your updated score.

Play Again:

- Click "Play Again" to load a new quiz and continue playing.

Live Prediction Page

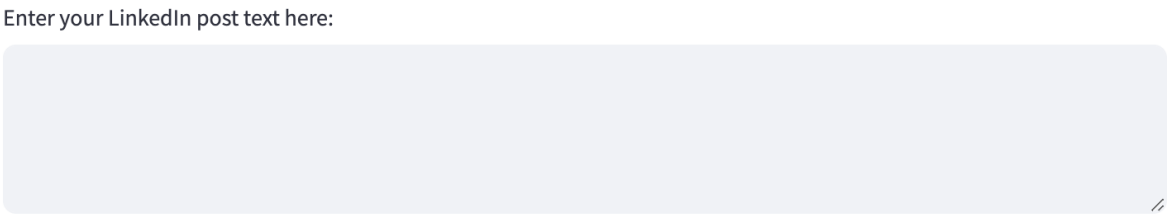
1. Key Features

Text Input Area: A resizable text box where users can enter a LinkedIn post.
Predict Button: Triggers the prediction process once text is entered.
Dynamic Output: Displays the user's input and the predicted category with styled feedback.
Model Information: Shows details about the underlying classification model.
Loading Feedback: A spinner indicates processing during prediction.

2. User Interaction

Text Input Area:

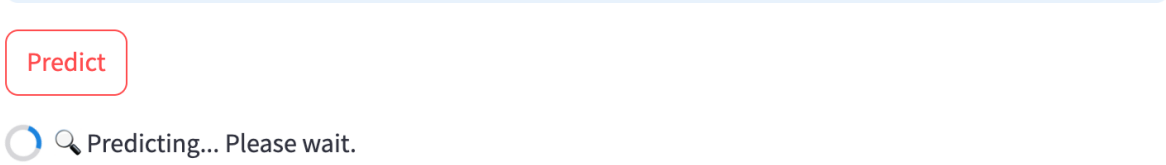
- Users enter a LinkedIn post in a text area labeled "Enter your LinkedIn post text here:".



Predict Button:

- A "Predict" button becomes active once text is entered (disabled if the input is empty).

- Clicking the button initiates the prediction process with a brief delay and spinner



3. Streamlit Components Used

st.text_area(): Provides a multi-line input box for user text.

st.button(): Triggers the prediction with a "Predict" button, conditionally enabled.

st.spinner(): Shows a loading animation during prediction processing.

st.write(): Displays the user input in the result section.

st.success(): Presents the predicted label in a green-styled box with an emoji.

st.info(): Displays the instructional tip in a blue-styled box.

4. Frontend-Backend Communication

Streamlit handles seamless communication between the frontend and backend:

- **Input Collection**: Text is captured via st.text_area() and passed to the backend.
- **Backend Processing**: The predict_single_text() function uses the loaded model and tokenizer to classify the input.
- **Output Display**: Results are rendered using st.write() and st.success(), with no manual JavaScript required.
- **Model Loading**: The load_model() function retrieves the pre-trained model and tokenizer from specified paths.

In the backend:

- The making_label_prediction() function processes the input and generates the prediction.
- A simulated delay (time.sleep(1)) mimics processing time for user experience.

5. How to Use

Enter Text:

- Type or paste a LinkedIn post into the text area provided.

Finalize Input:

- Press Ctrl + Enter to lock your text (optional, as per the tip).

Trigger Prediction:

- Click the "Predict" button to process your input.

View the Result:

- After a brief loading animation, the app displays your input and the predicted category.

Below is the documentation for the provided Streamlit application code, formatted consistently with the previous example you shared:

Visualization Page

1. Key Features

Visualization Selection: Radio buttons allow users to choose between "Word Cloud," "Top 10 Words," or "Post Length" visualizations.
Category Selection: A dropdown menu enables filtering by LinkedIn post categories, with an "All Categories" option for Post Length visualizations.
Dynamic Visualizations:

- Word Cloud: Displays a word cloud for the selected category.
- Top 10 Words: Shows a horizontal bar chart of the most frequent words.
- Post Length: Renders histograms of post lengths, either for a single category or all categories.

Custom Stop Words: Filters out common and quirky words (e.g., "https," "lnkd") for cleaner visualizations.

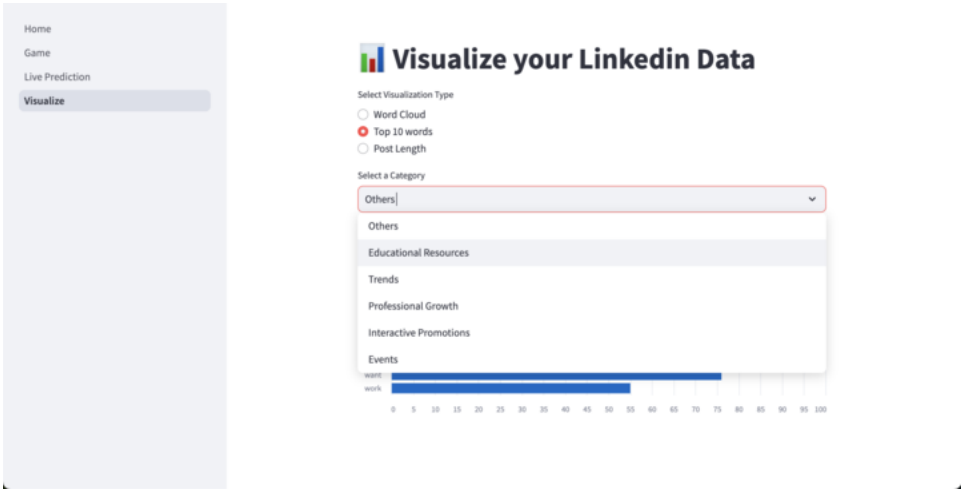
2. User Interaction

Visualization Selection:

- Users select a visualization type ("Word Cloud," "Top 10 Words," or "Post Length") using radio buttons.
- The interface updates dynamically based on the chosen visualization.

Category Selection:

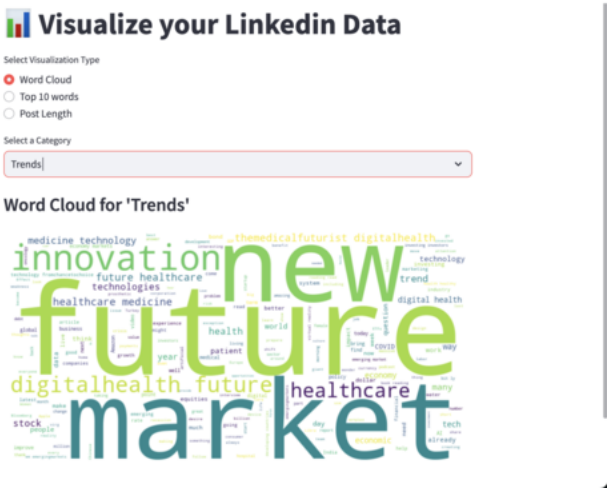
- A dropdown menu (st.selectbox) allows users to pick a category from the available labels in the dataset.



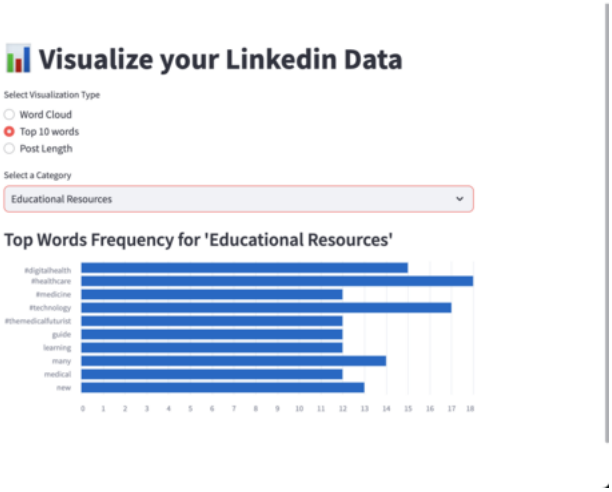
- For "Post Length," an additional "All Categories" option triggers comparative histograms across all categories.

Result Display:

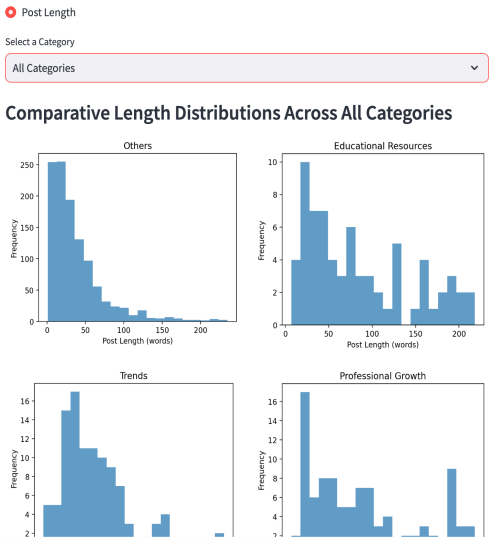
- Word Cloud: A graphical representation of word frequency is displayed using `matplotlib` and WordCloud.



- Top 10 Words: A horizontal bar chart shows the 10 most common words, filtered for stop words and punctuation.



- Post Length: The bar charts display the distribution of post lengths. Users can filter by category or view all.



3. Streamlit Components Used

st.radio(): Allows users to select the visualization type.
st.selectbox(): Provides a dropdown for category selection.
st.subheader(): Displays the title of the selected visualization and category.
st.pyplot(): Renders matplotlib figures (e.g., word clouds, histograms).
st.bar_chart(): Displays the Top 10 Words frequency as a horizontal bar chart.
st.columns(): Organizes multiple histograms side-by-side for "All Categories" in Post Length mode.

4. Frontend-Backend Communication

Streamlit handles the communication between the frontend and backend seamlessly:

Input Collection:

- Visualization type is captured via st.radio().
- Category selection is collected using st.selectbox().

Backend Processing:

- Data is loaded using load_data() from utils.data_util.
- Visualization logic filters data by category (data[data['label'] == selected_category]) and processes it using pandas, WordCloud` or Counter.
- Figures are generated with matplotlib or directly with Streamlit's st.bar_chart.

Output Display: Results are rendered in real-time using st.pyplot() or st.bar_chart(), with no manual JavaScript required.

In the Backend:

- **Word Cloud**: Text is concatenated, filtered with custom stop words, and passed to WordCloud.generate().
- **Top 10 Words**: Words are tokenized, cleaned (stop words, punctuation removed), and counted with Counter.
- **Post Length**: Post lengths are calculated using `apply(lambda x: len(str(x).split()))`, and histograms are plotted.

5. How to Use

Select Visualization Type:

- Choose "Word Cloud," "Top 10 Words," or "Post Length" from the radio buttons.

Choose a Category:

- Pick a category from the dropdown menu. For "Post Length," optionally select "All Categories."

View the Result:

- The app dynamically generates and displays the visualization:

Interact and Explore:

- Adjust the visualization type or category to explore different perspectives of the LinkedIn data.