Project Blueprint: "The One With The AI"

End-to-End NLP Project with Friends TV Show Dataset

0. Project Overview

Goal: Build an interactive Web Application that allows users to explore the "Friends" TV show universe using Natural Language Processing (NLP). **Core Features:**

1. **Semantic Search:** Users can search for quotes by *meaning* (e.g., "I'm hungry" finds "Joey doesn't share food!"), not just by keyword.
2. **Personality Matcher:** Users react to a scenario (image/text), and the AI predicts which character (Monica, Joey, Chandler, etc.) matches their vibe based on their response.

1. Data Requirements & Sources

A. Text Data (The Script)

You need a structured dataset that separates **Dialogue** from **Speakers**.

- **Recommended Source:** Friends TV Show Script (Kaggle) or similar.
- **Required Columns:** `Season` , `Episode` , `Scene` , `Speaker` , `Text` .
- **Volume:** Approx. 10 seasons, ~230 episodes. This is small enough to fit entirely in RAM.

B. Image Data (For the App UI)

You will need two types of images:

1. **Character Avatars:** 6 images (one for each main friend) to show when a match is found.
   - *Source:* Manual download from Google Images (save to `assets/characters/`).
2. **Quiz Scenarios:** 3-5 generic images to trigger user responses.
   - *Examples:* A messy room (for Monica trigger), A large pizza (for Joey trigger), A breakup scene (for Ross trigger).
   - *Source:* Unsplash or Pexels (Royalty-free).

2. Hardware & Environment

- **Device:** MacBook Pro (M4 Pro).
- **Capability:** This machine is overkill (in a good way). Training the Word2Vec model on this dataset will take < 1 minute.
- **Key Advantage:** You can run `Streamlit` locally with instant reload times, making development very fast.

3. Model Architecture

You will use a **Hybrid Approach**: Training your own model for domain specificity, but keeping a pre-trained model as a backup if vocabulary is too small.

The Models

1. **Primary Model:** `Gensim Word2Vec` (Trained from scratch on Friends scripts).
   - *Why:* To capture show-specific context (e.g., "Pivot", "Break", "Unagi").
2. **Architecture:**
   - **Type:** CBOW (Continuous Bag of Words) or Skip-gram.
   - *Recommendation:* Use **Skip-gram** (`sg=1` in Gensim) because it works better with smaller datasets.
   - **Dimensions:** 100-300 dimensions.

4. Pre-processing Strategy (The "Secret Sauce")

NLP models are only as good as the cleaning.

Steps:

1. **Regex Cleaning:** * Remove scene directions (e.g., `[Scene: Central Perk]`, `(Jumping on the bed)`).
   - Remove speakers' names if they appear in the text line.
2. **Tokenization:** Split sentences into words.
3. **Normalization:**
   - Convert to lowercase.
   - **Stopwords:** *Be careful here.* For "Personality Matching", words like "not", "very", "hate", "love" are crucial. **Do not remove all stopwords.** Only remove "noise" words (the, a, an, etc.).
   - **Lemmatization:** Convert "running" -> "run". Use `spacy` or `nltk` .

5. Training The Model

Library: `gensim` **Process:**

1. Feed the cleaned list of sentences (list of lists of words) into `Word2Vec` .
2. **Hyperparameters:**
   - `min_count=2` : Ignore words that appear less than twice.
   - `window=5` : Look 5 words behind and 5 words ahead for context.
   - `epochs=30` : Since the data is small, train for more epochs.

6. Feature Implementation Logic

Feature A: Semantic Search

- **Database Creation:**
  1. Take every dialogue line in the script.
  2. Convert it to a **Sentence Vector** by averaging the Word2Vec vectors of its words.
  3. Store: `[{ "text": "Joey doesn't share food", "vector": [0.12, -0.4, ...] }, ...]`
- **Search Logic:**
  1. User types: "I want a sandwich".
  2. Convert query to vector.
  3. Calculate **Cosine Similarity** between Query Vector and all Database Vectors.
  4. Return top 5 matches.

Feature B: Personality Matcher (The "Classifier")

- **Profile Generation (Offline):**
  1. Filter all lines spoken by **Monica**.
  2. Clean and vectorize them.
  3. Calculate the **Centroid** (Average) of ALL Monica's lines. This is the "Monica Vector".
  4. Repeat for all 6 friends.
- **Matching Logic (Real-time):**
  1. User sees image of a messy room.
  2. User inputs: "It makes me anxious, I need to clean it."
  3. Convert input to vector.
  4. Compare input vector against "Monica Vector", "Joey Vector", etc.
  5. Winner = Highest Similarity Score.

7. Frontend Design (Streamlit)

Layout:

- **Sidebar:** Navigation ("Search Quotes", "Personality Quiz", "About").
- **Page 1: Search**
  - Big Search Bar.
  - Results displayed as "Cards" with the Character Name and the Season/Episode.
- **Page 2: Quiz**
  - `st.image()` to show the scenario.
  - `st.text_input()` for user reaction.
  - `st.button("Analyze Personality")`.

- **Result:** Show the Character's face and a progress bar showing similarity % (e.g., "70% Monica, 20% Ross").

8. Deployment Strategy

1. **Version Control:** Push code to GitHub.
2. **Hosting:** Connect GitHub repo to **Streamlit Cloud** (Free).
3. **Blog:** Write a Medium article explaining the "Why" and "How".

9. Next Steps Checklist

- [ ] Download Dataset from Kaggle.
- [ ] Create folder structure (`data/` , `src/` , `assets/`).
- [ ] Write `preprocessing.py` to clean the scripts.