Final Project

DATA 5420/6420

Name: Stephanie Liechty

The purpose of the final project is to produce an MVP that is a culmination of the skills you have learned in each of the previous units. This MVP should be a cohesive product in that it combines methods in some logical pipeline, it should NOT simply be a collection of methods implemented independently/separately with no clear end goal/state. You will be tasked with applying at least four methods from across the four units, which I've outlined below:

Unit 1

- Chatbots
- Basic Text Statistics
- NLP Pipeline (Preprocessing & Normalization)
- Compiling Corpora via APIs

Unit 2

- Bag of Words Models (TF-IDF and Count Vectorization)
- Document Classification
- Sentiment Analysis

Unit 3

- Document Summarization
- Topic Modeling
- Text Similarity

> - Information Retrieval (Search)
> - Recommendation Systems

- Document Clustering

> - KMeans
> - Affinity Prop
> - Wards Agglomerative Hierarchical

Unit 4

- Word Embeddings
- Pretrained Transformers
- Question-Answering Systems
- Speech-to-Text (hopefully)

You will of course need to perform some form of cleaning/text normalization and feature engineering (bag of words and/or word embeddings), but the way you go about that will be problem dependent -- on top of those two steps, you will need to incorporate at least two other model types as well that form some coherent end-stage MVP.

For example:

1) corpus of a news articles pulled from the Bing News API that is cleaned/normalized

2) uses word embeddings to feature engineer the text

3) performs sentiment analysis to score sentiment of all articles

4) articles are sortable by sentiment, and ranked based on their relevance to keywords/search queries (information retrieval)

The MVP is a NewsFeed showing a table of articles displayed in an interactive dashboard

As you are performing your analyses consider:

- What cleaning and normalization steps are necessary for my text, and which are not?

- What sort of feature engineering do I need to utilize, both in terms of using BoW or word embeddings, and in terms of document or word vectorization? Do I need to use different methods for different analysis types?
- What is the purpose of performing your selected methods and how do they meaningful build on one another?
- What are the practical applications of the models you developed?

What four (+) methods have you chosen and how do they fit together?

1. NLP Pipeline
2. Sentiment Analysis
3. Text Similarity
4. Question-Answering System

Description of how these methods will be meaningfully combined:

First, the NLP Pipeline cleans and gets the text ready, making sure everything's uniform for deeper digging. This tidy text is then the base for Sentiment Analysis, which figures out if the advice is positive or negative, and Text Similarity, which finds posts that are alike. This helps you easily find and compare different advice topics and feelings. The Question-Answering System takes this neat text and lets you ask specific questions, giving you personalized answers. All these steps come together in a dashboard that's easy to use. You can sift through, understand, and get involved with the world of parenting advice, showcasing how mixing different text mining methods can unlock useful insights from loads of messy data.

```
# General imports for data manipulation and numerical operations
import pandas as pd
import numpy as np

# NLP Pipeline imports
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import spacy
import re

# Sentiment Analysis imports
from transformers import pipeline
from collections import Counter

# Text Similarity imports
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import gensim

# Question-Answering System imports (using OpenAI)
!pip install openai==0.28
import openai
import ipywidgets as widgets
from IPython.display import display, clear_output

# Data Fetching for Reddit
!pip install praw
import praw
from datetime import datetime
from datetime import date

# Initialize NLTK (if you haven't already)
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('vader_lexicon')

# Setting up Secret Key usage
from google.colab import userdata

# Setting up OpenAI
openai.api_key = userdata.get('API')

# Setting up PRAW
reddit = praw.Reddit(client_id='cvFqWtnTSzLAF_sqtXLa_g',
                     client_secret= userdata.get('Reddit'),
                     user_agent='reddit_app/v1')
```

```
Requirement already satisfied: openai==0.28 in /usr/local/lib/python3.10/dist-packages (0.28.0)
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (4.66.2)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (3.9.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (2024.2
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (4.0.3)
Requirement already satisfied: praw in /usr/local/lib/python3.10/dist-packages (7.7.1)
Requirement already satisfied: prawcore<3,>=2.1 in /usr/local/lib/python3.10/dist-packages (from praw) (2.4.0)
Requirement already satisfied: update-checker>=0.18 in /usr/local/lib/python3.10/dist-packages (from praw) (0.18.0)
Requirement already satisfied: websocket-client>=0.54.0 in /usr/local/lib/python3.10/dist-packages (from praw) (1.7.0)
Requirement already satisfied: requests<3.0,>=2.6.0 in /usr/local/lib/python3.10/dist-packages (from prawcore<3,>=2.1->praw) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.6.0->prawcore<
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.6.0->prawcore<3,>=2.1->pra
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.6.0->prawcore<3,>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0,>=2.6.0->prawcore<3,>=2.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Method 1: NLP Pipeline

Implementing a Natural Language Processing pipeline to preprocess and normalize text data will involve steps such as tokenization, removing stopwords, lemmatization, and removing special characters.

```
# Specify the subreddit names for parenting advice
subreddit_names = ['beyondthebump', 'Mom', 'NewParents',
                   'Parenting', 'AskParents']
```

```
# Create empty lists to store post attributes and comments
post_attributes = []
comment_attributes = []

# Iterate through each subreddit
for subreddit_name in subreddit_names:
    subreddit = reddit.subreddit(subreddit_name) # set subreddit
    posts = subreddit.top(time_filter='month', limit=20)  # set post parameters from top posts from the past month, up to 20 posts

    # Iterate through the top posts in the subreddit
    for post in posts:
        # Append post attributes to the list
        post_attributes.append({
            'Post_ID': post.id,
            'Post_Title': post.title,
            'Post_Content': post.selftext or 'No Content',  # Ensure no null values in Post_Content
            'Post_URL': post.url,
            'Post_Date': datetime.utcfromtimestamp(post.created_utc).strftime('%Y-%m-%d'),
            'Provider': subreddit_name
        })

        # Fetch the top comments for the current post
        post.comments.replace_more(limit=0)
        top_comments = post.comments[:1]

        # Iterate through the top comments and append attributes to the list
        for comment in top_comments:
            comment_attributes.append({
                'Post_ID': post.id,  # Add Post_ID to link comments back to the posts
                'Post_Content': post.selftext or 'No Content',
                'Comment_Content': comment.body if comment.body else 'No Content',  # Ensure no null values in Comment_Content
                'Comment_Score': comment.score,
                'Comment_Date': datetime.utcfromtimestamp(comment.created_utc).strftime('%Y-%m-%d'),
                'Provider': subreddit_name
            })

# Create DataFrames for comments
df_comments = pd.DataFrame(comment_attributes)

# Adjust 'Provider' column for DataFrame
df_comments['Provider'] = 'r/' + df_comments['Provider']
```

```
WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
See https://praw.readthedocs.io/en/latest/getting_started/multiple_instances.html#discord-bots-and-asynchronous-environments for more

WARNING:praw:It appears that you are using PRAW in an asynchronous environment.
It is strongly recommended to use Async PRAW: https://asyncpraw.readthedocs.io.
```

```
df_comments.head()  # display dataframe
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11	I mean, you did it. You hit the partner	1623	2024-03-24	r/beyondthebump

Next steps: Generate code with df_comments View recommended plots

Part 1 (Lowercase, Special Character & Whitespace Removal) - NLP Pipeline

In the first part, we are converting the text to lowercase, removing special characters, and removing whitespaces. Doing this will make the text data more amenable for analysis.

```
def clean_text(text):
    # Convert text to lowercase
    text = text.lower()

    # Remove special characters and digits
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)

    # Remove extra spaces, tabs, and new lines
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Apply the cleaning function to both the Post_Content and Comment_Content columns
df_comments['Cleaned_Post_Content'] = df_comments['Post_Content'].apply(clean_text)
df_comments['Cleaned_Comment_Content'] = df_comments['Comment_Content'].apply(clean_text)

# Display the first few rows of the DataFrame to verify the changes
df_comments.head()
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11 after bub had his bottle...	I mean, you did it. You hit the partner teamwo...	1623	2024-03-24	r/beyondthebump
2	1brvfep	I was angry at you today. You asked me to	This is.. rough. In your writing I can	796	2024-03-30	r/beyondthebump

Next steps: Generate code with `df_comments` View recommended plots

Part 2 (Tokenization & Stopword Removal) - NLP Pipeline

In the second part, stopwords are going to be removed. Tokenization will split the string text into individual words, which helps with stopword removal. This is a necessary task because it will reduce dataset size, bring more focus on meaningful words, and improve the performance of our NLP models.

```
def remove_stopwords(text):
    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Get the list of English stopwords
    stop_words = set(stopwords.words('english'))

    # Remove stopwords from the tokens
    filtered_tokens = [word for word in tokens if word not in stop_words]

    # Rejoin the filtered tokens back into a string
    filtered_text = ' '.join(filtered_tokens)

    return filtered_text

df_comments['Cleaned_Post_Content'] = df_comments['Cleaned_Post_Content'].apply(remove_stopwords)
df_comments['Cleaned_Comment_Content'] = df_comments['Cleaned_Comment_Content'].apply(remove_stopwords)

df_comments.head()
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11 after bub had his bottle...	I mean, you did it. You hit the partner teamwo...	1623	2024-03-24	r/beyondthebump
2	1brvfep	I was angry at you today. You asked me to	This is.. rough. In your writing I can	796	2024-03-30	r/beyondthebump

Next steps: Generate code with `df_comments` View recommended plots

Part 3 (Lemmatization) - NLP Pipeline

In the third part, we are reducing words to their root form (lemmatization). This will provide more precise model performance because their are not different versions of the same word.

```
# Load the English language model in spaCy
nlp = spacy.load('en_core_web_sm')

def lemmatize_text(text):
    # Process the text using spaCy
    doc = nlp(text)

    # Extract the lemma for each token and join back into a string
    lemmatized_text = ' '.join([token.lemma_ for token in doc])

    return lemmatized_text

# Assuming 'df' is your DataFrame and 'Cleaned_Text' is the column you wish to lemmatize
# Apply the lemmatization function to the Cleaned_Text column
df_comments['Cleaned_Post_Content'] = df_comments['Cleaned_Post_Content'].apply(lemmatize_text)
df_comments['Cleaned_Comment_Content'] = df_comments['Cleaned_Comment_Content'].apply(lemmatize_text)

# Display the first few rows to verify the changes
df_comments.head()
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11 after bub had his bottle...	I mean, you did it. You hit the partner teamwo...	1623	2024-03-24	r/beyondthebump
2	1brvfep	I was angry at you today. You asked me to	This is.. rough. In your writing I can	796	2024-03-30	r/beyondthebump

Next steps: Generate code with `df_comments` View recommended plots

Part 4 (Vectorization) - NLP Pipeline

Vectorization is a crucial step that transforms text data into a numerical format, making it understandable and processable my machine learning algorithms.

```
# Initialize the TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(
    max_df=0.85,  # discard words appearing in more than 85% of the documents
    min_df=2,     # discard words appearing in less than 2 documents
    ngram_range=(1, 2),  # consider unigrams and bigrams
)


# Concatenate post and comment content into a new column
df_comments['Combined_Text'] = df_comments['Cleaned_Post_Content'].str.cat(df_comments['Cleaned_Comment_Content'], sep=' ')

# Proceed with vectorization on the 'Combined_Text' column
tfidf_matrix = tfidf_vectorizer.fit_transform(df_comments['Combined_Text'])

# The resulting 'tfidf_matrix' is a sparse matrix representation of the TF-IDF values.
# You can convert it to a DataFrame for better readability (optional):
feature_names = tfidf_vectorizer.get_feature_names_out()
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=feature_names)

tfidf_df.head()
```

	abandon	abandon user	ability	able	absolute	absolutely	accept	acceptable	accident	a
0	0.0	0.0	0.0	0.0	0.0	0.08483	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.00000	0.0	0.0	0.0	

5 rows × 1838 columns

Method 2: Sentiment Analysis

Assessing the emotional tone of the parenting advice comments will help identify whether posts express positive, negative, or neutral sentiments towrds parenting issues and advice.

```
# Using BERT for Sentiment Analysis
sentiment_model = pipeline("sentiment-analysis", model='cardiffnlp/twitter-roberta-base-sentiment-latest')
```

```
Some weights of the model checkpoint at cardiffnlp/twitter-roberta-base-sentiment-latest were not used when initializing RobertaForSeque
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model trained on another task or wi
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the checkpoint of a model that you expect to be exa
```

```
def truncate_sequence(sequence, max_length):
    if len(sequence) > max_length:
        sequence = sequence[:max_length]
    return sequence # Anything that is after the max length position is cut off

# Apply function to Sentence column & apply sentiment model
df_comments['Comment_Sentiment'] = df_comments['Cleaned_Comment_Content'].apply(lambda x: sentiment_model(truncate_sequence(x, 512))[0]['lab
```

```
# Print the sentiment for each article
df_comments.head()
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11 after bub had his bottle...	I mean, you did it. You hit the partner teamwo...	1623	2024-03-24	r/beyondthebump
2	1brvfep	I was angry at you today. You asked me to go p...	This is.. rough. In your writing I can feel th...	796	2024-03-30	r/beyondthebump
3	1bnfst8	Mine is that part of the reason newborns cry i...	That colic is a lazy diagnosis and synonymous ...	1249	2024-03-25	r/beyondthebump
4	1bqtnfg	It's a frequent topic in this sub that healthc...	Yup. Postpartum is the worst time because we g...	703	2024-03-29	r/beyondthebump

Next steps: Generate code with df_comments View recommended plots

Method 3: Text Similarity

Text Similarity measures the degree of similarity between pairs of text documents. This method will help identify parenting advice posts that are closely related or similar in content, allowing users to explore similar discussions or solutions to common parenting challenges.

```
doc_sim = cosine_similarity(tfidf_matrix)                                                     # compute document similarity by examining the c
doc_sim_df = pd.DataFrame(doc_sim)                                                  # take doc_sim, convert to dataframe
# pull up a heading of the dataframe
doc_sim_df.head()
```

	0	1	2	3	4	5	6	7	8
0	1.000000	0.100629	0.079585	0.060444	0.080766	0.008945	0.132982	0.076545	0.075344
1	0.100629	1.000000	0.063205	0.112776	0.055801	0.000000	0.071298	0.060771	0.074459
2	0.079585	0.063205	1.000000	0.055986	0.105060	0.000000	0.035473	0.105133	0.161393
3	0.060444	0.112776	0.055986	1.000000	0.041319	0.000000	0.054639	0.089344	0.068930
4	0.080766	0.055801	0.105060	0.041319	1.000000	0.014587	0.068611	0.298174	0.138665

5 rows × 100 columns

```
# saving all the unique movie titles to a list
advice_list = df_comments['Comment_Content'].values
advice_list
```

```
(https://old.reddit.com/r/ModCoord/comments/14ahqjo/mods_will_be_removed_one_way_or_another_spez/) [replace moderators]
(https://www.reddit.com/r/ModSupport/comments/14a5lz5/mod_code_of_conduct_rule_4_2_and_subs_taken/jo9wdol/) if they remain private.
Reddit has abandoned the users, the moderators, and countless people who support an ecosystem built on Reddit itself. \n\nPlease read
[Call to action - renewed protests starting on July 1st]
```

```python
def advice_recommender(advice_title, advice=advice_list, doc_sims=doc_sim_df):

    advice_idx = np.where(advice == advice_title)[0][0] # find advice indx of the advice you want

    advice_similarities = doc_sims.iloc[advice_idx].values # produce cosine similarity values

    similar_advice_idxs = np.argsort(advice_similarities)[1:6] # output top 5 advice in descending order that are similar to chosen advice

    similar_advice = advice[similar_advice_idxs] # produce advice titles

    print("Based on your interest in \"{}\", I'd recommend checking out:".format(advice_title))
    for advice in similar_advice:
        print("- {}".format(advice))
```

```python
advice_recommender('If you really really feel something is wrong, call someone.') # input advice
```

```
Based on your interest in "If you really really feel something is wrong, call someone.", I'd recommend checking out:
- So hard to loose it! No advice but I fell your pain. I never lost my baby weight 2nd time around
- Unrelated but holy cow! That baby has so much hair! Mine came out bald but with sideburns 😂
- I’d say that’s a strong positive.
- Sounds like your mom has an awesome sense of humour.
- I would be blasting them so hard on their FB right now.


How rude!
```

```python
def get_top_recommended_advice(similarity_matrix, df):
    # Initialize an empty list to store the top recommended advice
    top_recommendations = []

    # Iterate over each row in the similarity matrix
    for idx, similarities in enumerate(similarity_matrix):
        # We skip the first index since it's the similarity with itself, which should be the highest
        sorted_indices = np.argsort(-similarities)[1:]
        # Find the index of the top recommended advice that is not the same as the advice itself
        top_recommendation_idx = sorted_indices[0]
        # Retrieve the top recommended advice using the index
        top_recommendations.append(df.iloc[top_recommendation_idx]['Comment_Content'])

    return top_recommendations

# Assuming doc_sim_df.values gives us the numpy array of the similarity matrix
# We now call the function and assign the result to a new column in df_comments
df_comments['Top_Recommended_Advice'] = get_top_recommended_advice(doc_sim_df.values, df_comments)
```

```python
# Remove rows where any of the columns have a null value
df_comments = df_comments.dropna()

# If you want to remove rows where specific columns have null values, you can specify the subset
# For example, if you only want to remove rows where 'Comment_Content' is null:
df_comments = df_comments.dropna(subset=['Comment_Content'])



# Save the DataFrame to a CSV file
df_comments.to_csv('tm_dataset.csv', index=False)

df_comments.head()
```

	Post_ID	Post_Content	Comment_Content	Comment_Score	Comment_Date	Provider
0	1c2aazi	Just a scenario i thought was sad but also a l...	Sorry I laughed out loud at the title alone. B...	721	2024-04-12	r/beyondthebump
1	1bm36ak	Went to bed around 11 after bub had his bottle...	I mean, you did it. You hit the partner teamwo...	1623	2024-03-24	r/beyondthebump
2	1brvfep	I was angry at you today. You asked me to go p...	This is.. rough. In your writing I can feel th...	796	2024-03-30	r/beyondthebump
		Mine is that	because we g...			

Next Generate code with df_comments View recommended plots

Method 4: Question-Answering System

A question-answering system enables users to pose questions in natural language and recieve relevant answers extracted from parenting advice posts.

```
# Function to handle question asking and log management
def ask_question(question, chat_log):
    if len(chat_log) == 0:
        chat_log.append({"role": "system", "content": "Welcome! You can ask any question you like about parenting."})

    # Trim the chat log if it exceeds 12 entries to manage size and relevance
    if len(chat_log) >= 12:
        chat_log = chat_log[-12:]

    # Append the user's question to the chat log
    chat_log.append({"role": "user", "content": question})

    # Assuming you've set up your API key correctly
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=chat_log,
        max_tokens=150
    )

    # Extracting the generated response
    answer = response.choices[0].message['content']

    # Append the assistant's response to the chat log
    chat_log.append({"role": "assistant", "content": answer})

    return answer, chat_log
```