1. Assistant Information:

- **Assistant Name**: "Puerto Rico Travel Assistant"
- **Assistant Instructions**: Behavior:The guide should be served through an application where the main interface with the user is presented as a chatbot that interacts with a prospective traveler and suggests visiting spots in the Island of Puerto Rico based on the interests of the user. It should be able to provide information on several Puerto Rico landmarks, answer questions about events, and help the user compile a list of points of interest to visit. The chatbot should always maintain a friendly and informative tone while remaining focused on helping the user compile a list of points of interest to visit. The end result of a successful interaction with an interested user should be a list of landmarks of interest for the user. The end result of a successful interaction with a non-interested user should be the termination of the conversation in a graceful mode.

2. Data Format:

- **Data Files**: I created a dictionary with each data frame. Then chunked the data frame by rows. After this It was embedded with ADA-2 and saved as JSON. Later it was stored in FAISS as the Vector DB. (e.g., CSV, JSON, etc.)
    - **Landmarks Data**: JSON format
    - **Municipalities Data**: JSON format
    - **Fiestas Data**: JSON format
- If possible, could you share the names of the columns in your data?
    - **Landmarks Data**: Name, Description, Latitude, Longitude, Municipality, Category
    - **Municipalities Data**: Name, Description, Latitude, Longitude, Region
    - **Fiestas Data**: Date, Municipality, Name

3. Vector Store Setup:

- **What kind of files** I uploaded JSON files

4. Region and Category Details:

- Landmark has a category column, but I would like the model to retrieve this information from description in the dictionaries.
- Could you clarify how the regions and categories are linked to the municipalities, landmarks, or festivals? The regions are directly associated with the municipalities because that df had a column for region. The other 2 data frames will need to associate the municipality in their own data frame (dictionary) to be able to associate with the region assigned to that municipality in the municipalities_dict.

5. Output Expectations:

- What type of responses do you expect from the assistant? For example:
    - A list of destinations (landmarks and fiestas (if the dates coincide with the travel dates and the correct region) based on user interests and region.

If we need to feed again the regions and categories key words, these are the expected categories for each:

region_mapping = {

"North": ["Arecibo", "Barceloneta", "Camuy", "Dorado", "Florida", "Hatillo", "Manatí", "Toa Alta", "Vega Alta", "Vega Baja"],

"Metropolitan Area": ["Bayamón", "Carolina", "Cataño", "Guaynabo", "San Juan", "Toa Baja", "Trujillo Alto"],

"East": ["Caguas", "Canóvanas", "Ceiba", "Culebra", "Fajardo", "Gurabo", "Humacao", "Juncos", "Las Piedras", "Loíza", "Luquillo", "Maunabo", "Naguabo", "Río Grande", "San Lorenzo", "Yabucoa", "Vieques"],

"Central": ["Adjuntas", "Aguas Buenas", "Aibonito", "Barranquitas", "Cayey", "Ciales", "Cidra", "Comerío", "Corozal", "Jayuya", "Morovis", "Naranjito", "Orocovis", "Utuado"],

"South": ["Arroyo", "Caja de Muertos", "Coamo", "Guayama", "Guayanilla", "Juana Díaz", "Patillas", "Peñuelas", "Ponce", "Salinas", "Santa Isabel", "Villalba"],

"West": ["Aguada", "Aguadilla", "Añasco", "Cabo Rojo", "Guánica", "Hormigueros", "Isabela", "Lajas", "Lares", "Las Marías", "Maricao", "Mayagüez", "Moca", "Mona", "Quebradillas", "Rincón", "Sabana Grande", "San Germán", "San Sebastián", "Yauco"]

}

categories_keywords = {

"Beach": ["beach", "coast", "shore", "sea", "ocean"],

"Historical": ["fort", "castle", "ancient", "historic", "ruins"],

"Cultural": ["museum", "art", "theater", "cultural", "festival"],

"Mountains and rivers": ["mountain", "peak", "summit", "range", "cliff", "rivers", "river"]

** Please add keywords to the categories (if needed to better help the model)

Code to create the assistant:

```
pip install --upgrade openai

from openai import OpenAI

# Initialize the OpenAI client with your API key
client = OpenAI(api_key="your api key")

# Create the Assistant with file_search tool enabled
assistant = client.beta.assistants.create(
    name="Puerto Rico Travel Assistant",
    instructions="You are a friendly assistant helping users plan their trip to Puerto Rico. Ask them about their travel dates, interests, and preferred region, and suggest relevant landmarks and festivals based on their responses.",
    model="gpt-3.5-turbo",  # Using the appropriate model
    tools=[{"type": "file_search"}],  # Enabling file_search to query the vector store
)



# Create the vector store
client = OpenAI(api_key="your api key")

# Path to the files containing your embeddings
file_paths = [
    r"C:\Users\sombe\landmarks_embeddings.json",
    r"C:\Users\sombe\fiestas_embeddings.json",
```

```
    r"C:\Users\sombe\municipalities_embeddings.json"
]

# Open the files in binary read mode
file_streams = [open(path, "rb") for path in file_paths]

# Upload and poll the file batch to ensure it's done
file_batch = client.beta.vector_stores.file_batches.upload_and_poll(
    vector_store_id=vector_store.id, files=file_streams
)

# Check the status and file counts after uploading
print(file_batch.status)
print(file_batch.file_counts)


*if the files do not upload. If they do, this step is unnecessary:
print(file_batch)


# Check for 'files' or 'errors' attribute in the file_batch
print(file_batch.__dict__)  # Print all attributes of the object

# Test uploading files one by one to see which one causes the issue
for file in file_paths:
    try:
```

```
        file_stream = open(file, "rb")
        file_batch = client.beta.vector_stores.file_batches.upload_and_poll(
            vector_store_id=vector_store.id, files=[file_stream]
        )
        print(f"Uploaded {file} successfully.")
    except Exception as e:
        print(f"Failed to upload {file}: {e}")



assistant = client.beta.assistants.update(
    assistant_id=assistant.id,  # Your assistant's ID
    tool_resources={"file_search": {"vector_store_ids": [vector_store.id]}}
)
```

User query variable and functions to clean the answer:

```
user_query = "the query" #edit the query to test the assistant



# Assuming you've already created an assistant
# First, create a thread to start the conversation
thread = client.beta.threads.create(
    messages=[
        {"role": "user", "content": user_query}
    ]
```

```
)



# Create the run and wait for it to complete

run = client.beta.threads.runs.create_and_poll(thread_id=thread.id,
assistant_id=assistant.id)


# Retrieve the messages from the thread

messages = client.beta.threads.messages.list(thread_id=thread.id, run_id=run.id)


# Access the first message in the data list (if you only need the first one)

first_message = messages.data[0]


# Access the content of the message

message_content = first_message.content[0].text


# Print the content of the first message

print(message_content)




# Extract the citations from the annotations

citations = []

for annotation in message_content.annotations:

    file_id = annotation.file_citation.file_id  # Get the file ID from the citation

    citations.append(f"Cited from file ID: {file_id}")
```

```
# Print the citations
print("Citations found:", citations)

# You can also print the text if needed
print("Response text:", message_content.value)
```

This last code need to be tested and maybe fixed:

```
import re

def extract_and_clean_citations_from_annotations(message_content):
    # Extract the citations from the annotations
    citations = []
    for annotation in message_content.annotations:
        file_id = annotation.file_citation.file_id  # Get the file ID from the citation
        citations.append(f"Cited from file ID: {file_id}")

    # Regular expression to find the file IDs and citation markers
    citation_pattern = r'file-[\w\d]+'  # Regular expression to match file IDs

    # Clean the response text (remove the file IDs but keep the citation markers)
    cleaned_text = re.sub(citation_pattern, '', message_content.value)

    return citations, cleaned_text
```