BeyondChats-Assessment

This project is a Python program designed to fetch data from an API, identify sources in the responses, make them as citations, and present the results in a user-friendly UI using Streamlit.

Task Description:-

- **Fetch Data:** Retrieve data from the specified API.
- **Identify Sources:** Check each response to see if it contains any of the specified sources.
- **List Sources:** List the sources from which the response was formed. Return an empty array if no sources are found.
- **Return Citations:** Provide the citations for all objects from the API.
- **UI Development:** Display results in a user-friendly table with Streamlit.

UI Description:-

Run the Streamlit app using the commands below. The app displays a UI to enter the number of pages you want to fetch. Click on "Fetch" to process the request and display citations in a table format.

Modules Used:-

- **Re:-** The re module in Python provides support for regular expressions. Regular expressions are a powerful tool for pattern matching and manipulation of strings. They allow you to search for patterns within text, extract specific parts of text, and perform substitutions or modifications. In this project, the re module is used to match and identify specific patterns within the API responses, helping to determine if certain sources are present.
- **Pandas:-** Pandas is a popular Python library for data manipulation and analysis. It provides data structures like DataFrame, which allows for easy handling and manipulation of structured data. pandas is widely used for tasks such as data cleaning, transformation, analysis, and visualization. In this project, pandas is used to store and organize the citations extracted from the API responses, making it easier to display them in a tabular format.
- **Requests:-** The requests module is a simple yet powerful HTTP library for making HTTP requests in Python. It provides an easy-to-use interface for sending HTTP requests and handling responses. With requests, you can perform various types of HTTP requests such as GET, POST, PUT, DELETE, etc., and handle response data efficiently. In this project, requests is used to fetch data from the specified API URL.

- **Streamlit:-** Streamlit is an open-source Python library that allows you to create interactive web applications for Machine Learning and Data Science projects with minimal effort. It provides a simple and intuitive API for building interactive UI components, allowing you to quickly prototype and deploy data-driven applications. In this project, streamlit is used to create a user-friendly interface for fetching data from the API, identifying sources, and displaying the citations in a table format.
- **Python 3.x:-** Python 3.x refers to the latest major version of the Python programming language. Python 3 introduced several improvements and new features compared to Python 2, including better Unicode support, cleaner syntax, and various library enhancements. It is recommended to use Python 3.x for new projects as Python 2 has reached its end of life. In this project, Python 3.x is the required version for running the code.

Code Explanation :-

#Import required Libraries

```
import re
import requests
import pandas as pd
```

#Declare constant URL

```
url = "https://devapi.beyondchats.com/api/get_message_with_sources"
```

#Function to fetch the data from URL

```
def fetch_messages_with_sources(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()
        total_pages = data['data']['last_page']
        return data, total_pages
    except requests.exceptions.HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")
    except Exception as err:
        print(f"Other error occurred: {err}")


data, total_pages = fetch_messages_with_sources(url)
print("Total Pages :", total_pages)
print(data)
```

Output:-

Total Pages : 13
{'status': 'success', 'status_code': 200, 'message': 'Sample Sources fethced successfully!', 'dat
a': {'current_page': 1, 'data': [{'id': 1, 'response': 'Yes, we offer online delivery services throu
gh major platforms like Swiggy and Zomato. You can also order directly from our
website!', 'source': [{'id': '71', 'context': 'Order online Thank you for y
our trust in us! We are available on all major platforms: [Order online Ord
er directly from our website](https://orders.brikoven.com), [Order from Swi
ggy](https://www.swiggy.com/direct/brand/7389?source=swiggy-direct&subSourc
e=generic), [Order from zomato](https://www.zomato.com/bangalore/delivery?c
hain=18224650)', 'link': ''}, {'id': '8', 'context': 'Breakfast Reservation
s\r For Breakfast, we recommend making reservations in advance. \r For walk
-ins, we only seat parties on a first come, first served basis. \r Your res
ervation is confirmed…….

#Function to fetch the data from all the pages using URL and stores data into separate df

```
def fetch_and_transform_data(num_pages, base_url):
    all_rows = []

    for page in range(1, num_pages + 1):
        url = f"{base_url}?page={page}"
        try:
            response = requests.get(url)
            response.raise_for_status()
            data = response.json()

            for item in data['data']['data']:
                message_id = item['id']
                response_text = item['response']
                for source in item['source']:
                    source_id = source['id']
                    source_context = source['context']
                    source_link = source.get('link', None)
                    all_rows.append([message_id, response_text, source_id, source_context,
source_link])

        except requests.exceptions.HTTPError as http_err:
            print(f"HTTP error occurred: {http_err}")
        except Exception as err:
            print(f"Other error occurred: {err}")

    df = pd.DataFrame(all_rows, columns=['id', 'response', 'source_id', 'source_context',
'source_link'])
    return df
```

 # Calling the function and store required attributes into df

```
num_pages = total_pages
df = fetch_and_transform_data(num_pages, url)
```

#Identify whether the response for each response-sources pair came from any of the sources

```
url_pattern = re.compile(r'https?://[^\s)]+')
def find_first_url(text):
    match = url_pattern.search(text)
    return match.group(0) if match else None
for index, row in df.iterrows():
    if df.iloc[index]['source_link'] in [None, ""]:
        found_url = find_first_url(row['source_context'])
        if found_url:
            df.at[index, 'source_link'] = found_url
```

 # Get the citations and list down them

```
citations = []

for index, row in df.iterrows():
    if row['source_link'] is not None and row['source_link'] != "":
        citation = {
            "id": str(row['source_id']),
            "link": row['source_link']
        }
        citations.append(citation)
citations
```

Output:-

```
[{'id': '71', 'link': 'https://orders.brikoven.com'},
 {'id': '8', 'link': 'https://www.brikoven.com/reservations'},
 {'id': '73', 'link': 'https://orders.brikoven.com'},
 {'id': '57',
  'link': 'https://www.swiggy.com/direct/brand/7389?source=swiggy-direct&subSource=gen
eric'},
 {'id': '11', 'link': 'https://www.brikoven.com/'},
 {'id': '9', 'link': 'https://www.brikoven.com/stores-direction-info-copy'},
 {'id': '2', 'link': 'https://www.brikoven.com/deli'}……..
```

The provided code snippet starts by importing necessary libraries including re, requests, and pandas. These libraries are essential for handling regular expressions, making HTTP requests , and managing data in tabular format, respectively.

Next, a constant URL is declared representing the endpoint of the API from which data will b e fetched. This URL remains constant throughout the execution of the script.

A function named fetch_messages_with_sources is defined to fetch data from the specified URL. This function makes a GET request to the URL and retrieves the JSON response. Additio nally, it extracts the total number of pages available from the API.

Following the function definition, the fetch_messages_with_sources function is invoked to r etrieve data from the API. The fetched data and the total number of pages are stored in resp ective variables.

Another function called fetch_and_transform_data is defined to fetch and transform data fr om all pages of the API. This function iterates over each page, makes a GET request to the p age URL, and extracts relevant data like message ID, response text, source ID, source contex t, and source link. The extracted data is then stored in a list of lists.

The fetch_and_transform_data function is then called with the total number of pages and th e base URL to retrieve and transform data from all pages. The result is stored in a pandas Da taFrame (df) for further processing.

A function named find_first_url is defined using regular expressions to search for the first U RL in a given text. This function is utilized to extract URLs from the source context.

The DataFrame (df) is processed to extract citations. Iterating over each row in the DataFra me, the code checks if the source_link column is empty. If it is empty, the find_first_url func tion is used to search for a URL in the source_context column. If a URL is found, it populates the source_link column with the found URL.

After populating the source_link column, the code iterates over each row in the DataFrame again to extract citations. For each row where source_link is not empty, a citation dictionary containing the source ID and the source link is created. These citation dictionaries are appen ded to a list (citations).

Finally, the list of citations is printed as output.