Module 6 Challenge

Start Assignment

Due Dec 11 by 11:59pm **Points** 100 **Submitting** a text entry box or a website url

Background

You've been tasked to prepare some data for a recommendation system to help people find movie reviews and related movies. You will extract data from two different sources: The New York Times API and The Movie Database, then merge the data together. The text extracted from these APIs can later be used with natural language processing methods.

Files

Download the following files to help you get started:

[Module 6 Challenge files](https://static.bc-edx.com/ai/ail-v-1-0/m6/lms/starter/M6_Starter_Code.zip) (https://static.bc-edx.com/ai/ail-v-1-0/m6/lms/starter/M6_Starter_Code.zip)

Before You Begin

Before starting the Challenge, be sure to complete the following steps:

- Create a new repository for this project called `data-sourcing-challenge`. **Do not add this homework assignment to an existing repository**. When creating your new repository, under the "Add .gitignore" option, make sure you select "Python". This will prevent you from accidentally uploading your API keys in your `.env` file, exposing them to the world.
- Clone the new repository to your computer.
- Inside your local Git repository, add the starter files `retrieve_movie_data.ipynb` and `example.env` from your file downloads.
- Rename `example.env` to `.env` and add your API keys to the file.
- Ensure the `.env` file is not listed when you perform a `git status` check on the repo, before performing your `git add` action.
- Push these changes to GitHub or GitLab.

NOTE

The CSV file included in the `output` folder in your starter files is to help you identify how your final CSV file should be structured. Do not copy this file to your own repo. You will instead upload the CSV file you create as part of the Challenge.

Instructions

This Challenge has three parts, and must be completed in order:

- Part 1: Access the New York Times API.
- Part 2: Access The Movie Database API.
- Part 3: Merge and Clean the Data for Export.

The starter code includes importing the required dependencies and your API keys from your `.env` file, but you will need to ensure your API keys are added to that file.

Part 1: Access the New York Times API

1. The base URL is included in the starter code, along with the search string and query dates. Consult the **New York Times Article Search API documentation** (https://developer.nytimes.com/docs/articlesearch-product/1/overview) to help you build your `query_url` using these variables.

   If you accidentally delete these variables, they are:

```
# Set the base URL
url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?"

# Filter for movie reviews with "love" in the headline
# section_name should be "Movies"
# type_of_material should be "Review"
filter_query = 'section_name:"Movies" AND type_of_material:"Review" AND headline:"love"'

# Use a sort filter, sort by newest
sort = "newest"

# Select the following fields to return:
# headline, web_url, snippet, source, keywords, pub_date, byline, word_count
field_list = "headline,web_url,snippet,source,keywords,pub_date,byline,word_count"

# Search for reviews published between a begin and end date
begin_date = "20130101"
end_date = "20230531"
```

2. Create an empty list called `reviews_list` to store the reviews you retrieve from the API.
3. The Article Search API limits results to 10 per page, but we want to try to retrieve 200. To do this, create a `for` loop to loop through 20 pages (starting from page 0). Inside the loop, perform the following actions:
   - Extend the `query_url` created in Step 1 to include the `page` parameter.

    - Make a `GET` request to retrieve the page of results, and store the JSON data in a variable called `reviews`.
    - Add a 12-second interval between queries to stay within API query limits.

      Important: The New York Times limits requests to 500 per day and 5 per minute.

    - Write a try-except clause that performs the following actions:
        - `try`: loop through the `reviews["response"]["docs"]` and append each review to the list, then print out the query page number (i.e. the number of times the loop has executed).
        - `except`: Print the page number that had no results then break from the loop.

          Note: If your loop breaks at the `except` clause, it is possible you have tried to make a request that fell outside of the rate limit. You should be able to loop through all 20 pages with the provided query parameters.

4. Preview the first five results in JSON format using `json.dumps` with the argument `indent=4` to format the data.
5. Convert `reviews_list` to a Pandas DataFrame using `json_normalize()`
6. Extract the movie title from the `"headline.main"` column and save it to a new column `"title"`. To do this, you will use the Pandas `apply()` method and the following `lambda` function:

   ```
   lambda st: st[st.find("\u2018")+1:st.find("\u2019 Review")]
   ```

   This code takes the string in the cell and extracts the characters between the unicode quotation marks, as long as a space and the word "Review" follows the closing quotation mark.

7. Use the supplied `extract_keywords` function to convert the `"keywords"` column from a list of dictionaries to strings using the `apply()` method.
8. Create a list called `titles` from the `"title"` column using `to_list()`. These titles will be used in the query for The Movie Database.

Part 2: Access The Movie Database API

Consult the **Search & Query for Details documentation (https://developer.themoviedb.org/docs/search-and-query-for-details)** to build your query URLs. You will be making both types of requests to extract all of the details you need:

- The search query is used to find the movie ID from the search by title. Most of this query is included in your starter code, as follows, but you will need to include the movie title in the query.

  ```
  # Prepare The Movie Database query
  url = "https://api.themoviedb.org/3/search/movie?query="
  tmdb_key_string = "&api_key=" + tmdb_api_key
  ```

- The movie query is made once you have the movie ID.

You will use the `titles` list created in Part 1 to perform your queries with The Movie Database.

1. Create an empty list called `tmdb_movies_list` to store the results from your API requests. This will contain a list of dictionaries.
2. Create a variable called `request_counter` and initialize it with the value of `1`. This counter should do the following:
    - Increment by one every time you iterate through the `titles` list.
    - Use `time.sleep(1)` when it reaches a multiple of 50.
    - Print a message to indicate that the application is sleeping.
3. Loop through the `titles` list created from the movie reviews DataFrame, and perform the following actions:
    - Perform the actions outlined in Step 2.
    - Perform a `GET` request that sends the title to The Movie Database search and retrieves the JSON results.
    - Use a `try` clause that performs the following actions:
        - Collect the movie ID from the first result.
        - Make a `GET` request using the movie query (starting with `https://api.themoviedb.org/3/movie/`) and movie ID to retrieve the full movie details in JSON format.
        - Extract the genre names from the results into a list called `genres`.
        - Extract the `spoken_languages`' English name from the results into a list called `spoken_languages`.
        - Extract the `production_countries`' name from the results into a list called `production_countries`.
        - Create a dictionary with the following results: `title`, `original_title`, `budget`, `original_language`, `homepage`, `overview`, `popularity`, `runtime`, `revenue`, `release_date`, `vote_average`, `vote_count`, as well as the `genres`, `spoken_languages`, and `production_countries` lists you just created.
        - Append this dictionary to `tmdb_movies_list`.
        - Print out the name of the movie and a message to indicate that the title was found.
    - Use the except clause to print out a statement if a movie is not found.
4. Preview the first five results in JSON format using `json.dumps` with the argument `indent=4` to format the data.
5. Convert the results to a DataFrame called `tmdb_df` with `pd.DataFrame()`. You don't need to use `json_normalize()` this time because we don't have nested objects.

Part 3: Merge and Clean the Data for Export

Now that you have collected the data from both APIs, you need to merge the two DataFrames and clean the data, then export it for future use.

1. Merge the New York Times reviews and TMDB DataFrames on the `title` column.
2. The `genres`, `spoken_languages`, and `production_countries` columns were saved as lists, but we want the columns to be strings without the list characters (`[`, `]`, and `'`). To fix these columns, perform the following actions:
    - Create a list of the columns that need fixing called `columns_to_fix`.
    - Create a list of characters to remove called `characters_to_remove`.
    - Loop through `columns_to_fix` and do the following:
        - Use `astype()` to convert the column to a string.
        - Loop through the `characters_to_remove` and use the Pandas `str.replace()` method to remove the character from the string.
    - Print the head of the updated DataFrame to confirm the list characters were removed.
3. Delete any duplicate rows and reset the index.
4. Export data to a CSV file without the DataFrame's index.

Hints and Considerations

- Consider what you've learned so far. This Challenge builds on your Python and Pandas lessons, and you may want to review those activities to recall how to perform an action.
- If you're struggling with how to start, consider writing out the steps of the process using pseudocode.
- Remember to debug along the way. If you're unsure whether a section of code is running properly, write some print statements to print out variables or notes to yourself to help you locate the problem. Some common pitfalls that can lead to bugs and errors include:
    - Your environment variables are not set up properly.
    - Access is denied due to API key not being properly sent to the API.
    - The query string is not constructed properly.
    - Data within a JSON object is not properly referenced. Try printing the JSON object using `json.dumps()` and `indent=4` to check the structure.
    - Before creating a loop, ensure you can perform the actions you want to perform on a single item.
- Always commit your work and back it up with pushes to GitHub or GitLab. You don't want to lose hours of hard work! Also make sure that your repo has a detailed `README.md` file.

Requirements

Part 1: Access the New York Times API (35 points)

- `query_url` is correctly constructed (2 points).
- An empty list `reviews_list` is created (1 point).
- A `for` loop is created to loop through 20 times (3 points).
- The `query_url` is extended to include a `page` (1 point).
- A `GET` request is made to retrieve results and the JSON data is stored in a variable called `reviews` (4 points).
- A 12-second interval is used between queries (2 points).
- A `try-except` clause is used (2 points).
- Inside the `try` clause, there is a loop to loop through the `reviews["response"]["docs"]` list (3 points).
- The reviews results are correctly appended to `reviews_list` (2 points).
- The query page number is printed (1 point).
- The `except` clause prints out the page number that had no results, then breaks from the loop (2 points).
- `json.dumps` with the argument `indent=4` is used to preview the first five results (2 points).
- `reviews_list` is converted to a Pandas DataFrame using `json_normalize()` (2 points).
- The title is extracted from the `"headline.main"` column and is saved in a new column `"title"` (3 points).
- The `"keywords"` column is correctly converted to string data using the supplied `extract_keywords` function (3 points).
- A list called `titles` is created from the `"title"` column using `to_list()` (2 points).

Part 2: Access The Movie Database API (40 points)

Preparation (4 points):

- An empty list called `tmdb_movies_list` is created (1 point).
- A variable called `request_counter` is created and assigned the value of `1` (1 point).
- A `for` loop is created to loop through the `titles` list (2 points).

Inside the `titles` `for` loop (12 points):

- `request_counter` is incremented by `1` (1 point).
- `time.sleep(1)` when `request_counter` reaches a multiple of 50 (3 points).
- A `GET` request that sends the title to The Movie Database search is performed, and the JSON results are retrieved (4 points).
- A `try-except` clause is used (3 points).
- The `except` clause prints out a statement if a movie is not found (1 point).

Inside the `try` clause (20 points):

- The movie ID is collected from the first result and saved as a variable (2 points).
- A `GET` request is made using the movie query URL and movie ID to retrieve the full movie details in JSON format (4 points).

- The genre names are extracted from the results into a list called `genres` (2 points).
- The `spoken_languages`' English names are extracted from the results into a list called `spoken_languages` (2 points).
- The `production_countries`' names are extracted from the results into a list called `production_countries` (2 points).
- A dictionary is created with the specified 15 fields (4 points).
- The results dictionary is appended to the `tmdb_movies_list` list (3 points).
- A message is printed with the name of the movie to indicate that the title was found. (1 point)

Actions after the results are collected (4 points):

- The first five results are previewed using `json.dumps` with the argument `indent=4` (2 points).
- The results are converted to a DataFrame called `tmdb_df` with `pd.DataFrame()` (2 points).

Part 3: Merge and Clean the Data for Export (25 points)

- The New York Times reviews and TMDB DataFrames are merged on the title column (4 points).
- A list called `columns_to_fix` is created to store the names of the `genres`, `spoken_languages`, and `production_countries` columns (2 points).
- A list is created called `characters_to_remove` containing `[`, `]`, and `'` (2 points).
- A `for` loop is created to loop through `columns_to_fix` (2 points).
- The columns to fix are converted to the string data type (2 points).
- `characters_to_remove` is looped through to remove the characters from the string using the Pandas `str.replace()` method (4 points).
- The head of the updated DataFrame is displayed to confirm the list characters were removed (2 points).
- The `byline.person` column is dropped (2 points).
- Duplicate rows are deleted (1 point).
- The DataFrame index is reset (1 point).
- The DataFrame is exported to a CSV file without the index (3 points).

Grading

This challenge will be evaluated against the requirements and assigned a grade according to the following table:

Grade	Points
A (+/-)	90+
B (+/-)	80–89

Grade	Points
C (+/-)	70–79
D (+/-)	60–69
F (+/-)	< 60

Submission

To submit your Challenge assignment, click Submit, and then provide the URL of your GitHub repository for grading.

NOTE

> You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Next, and move on to the next module.

Comments are disabled for graded submissions in Bootcamp Spot. If you have questions about your feedback, please notify your instructional staff or your Student Success Manager. If you would like to resubmit your work for an additional review, you can use the Resubmit Assignment button to upload new links. You may resubmit up to three times for a total of four submissions.

IMPORTANT

> **It is your responsibility to include a note in the README section of your repo specifying code source and its location within your repo**. This applies if you have worked with a peer on an assignment, used code that you did not author or create, source code from a forum such as Stack Overflow, or received code outside curriculum content from support staff, such as an Instructor, TA, Tutor, or Learning Assistant. This will provide visibility to grading staff of your circumstance in order to avoid flagging your work as plagiarized.
>
> If you are struggling with a Challenge or any aspect of the curriculum, please remember that there are student support services available to you:
>
> 1. Office hours facilitated by your TA(s)

2. **Tutoring Guidelines** (https://docs.google.com/document/d/1hTldEfWhX21B_Vz9ZentkPeziu4pPfnwiZbwQB27E90/edit?usp=sharing) (schedule a session in the "Tutor Sessions" section of Bootcampspot)

3. Ask the class Slack channel/get peer support

4. AskBCS Learning Assistants