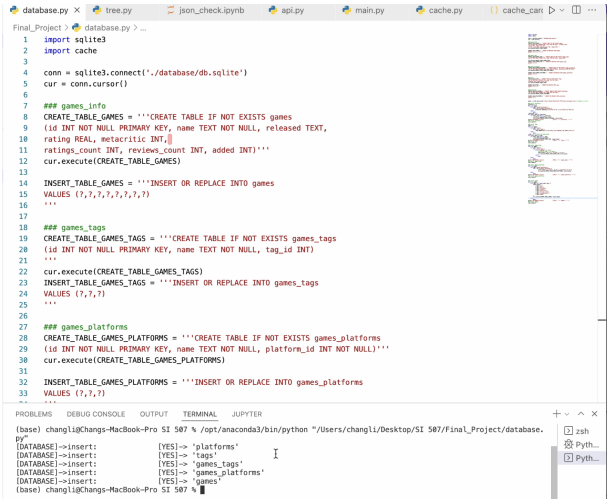
Project Code

- https://github.com/maguaaa/Card-Game-Recommendation
- **README:** This program uses API provided by RAWG. The API key can be acquired after signing up in https://rawg.io/. Additionaly, you need to find your session id in https://store.steampowered.com/. Then, you need to create a file named secrets.py and type api_key = 'your_api_key' and cookie = 'your_session_id' into the file to run the program. **Plotly chart** and **command lines** are used to interface with users.
- Python packages: json, requests, bs4, pandas, datetime, plotly, sqlite

Data Sources

Web API you haven't used before that requires API key or HTTP Basic authorization ✣ (4pts)

1. **Origin** : Documentation `GET https://api.rawg.io/api/games?genres=card`
2. **Format**: json
3. **Data accessing and caching**: Use API key and cache data as json files. I build a database to cache all the data into 5 tables, "games", "games_tags", "games_platforms", "tags" and "platforms".

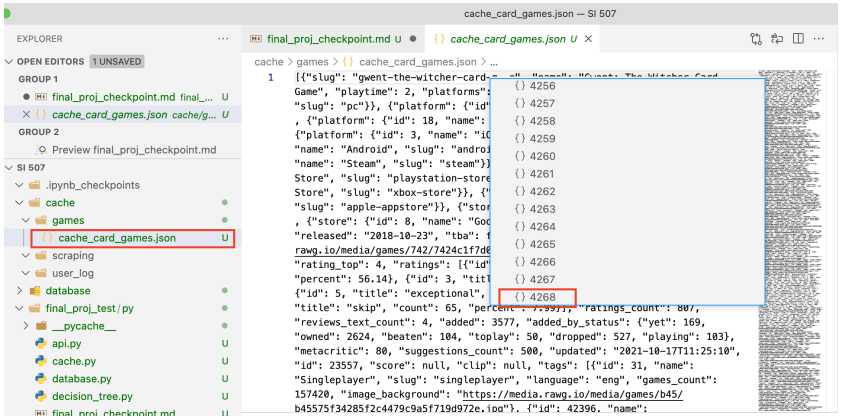


4. **Summary of data**

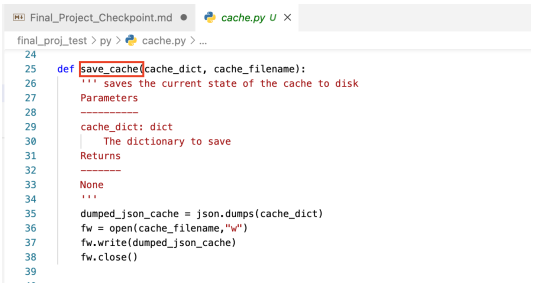
- 4268 records retrieved
- description of data
    - important fields:
        - "id": int, id of each game
        - "name": string, name of each game
        - "slug": string, lower case string connected by '-' between words

    - "rating": float, rating of each game
    - "meatacritic": int, rating by Metacritic
    - "ratings_counts": int, the amount of ratings
    - "reviews_counts": int, the amount of reviews
    - "released": string, the date each game released at, (time format: %Y-%m-%d)
    - "parent_platforms": string, the parent platform of each game

- evidence of caching
    - retrieved data



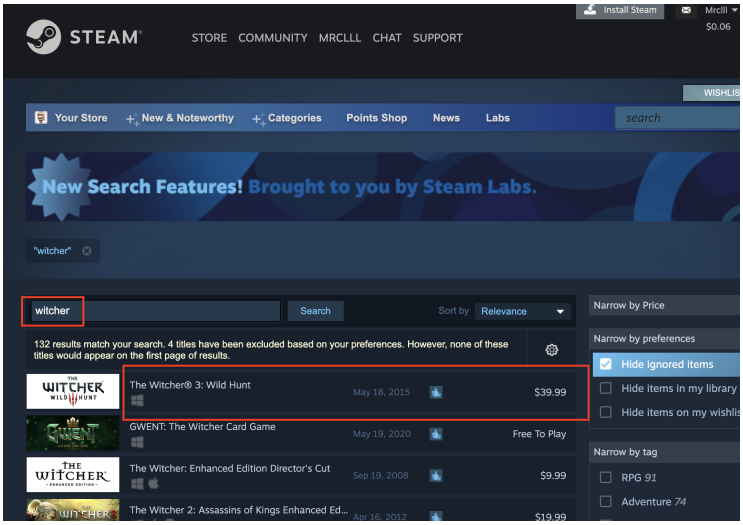
    - function of saving cache



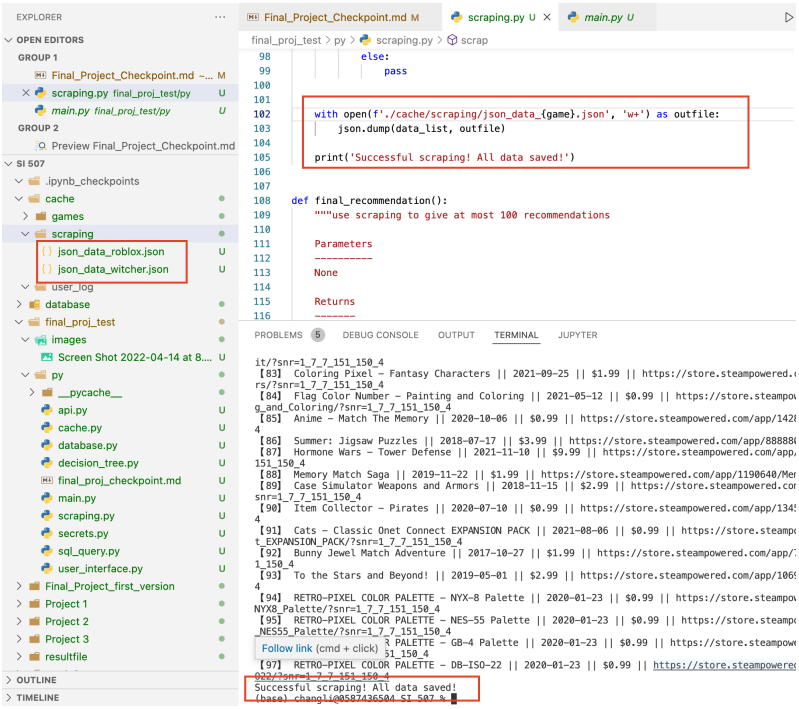
    - function of requesting api and caching data

Crawling [and scraping] multiple pages in a site you haven't used before ✣ (8 pts)

1. **Origin** : url `https://store.steampowered.com/search/results`
2. **Format** : html

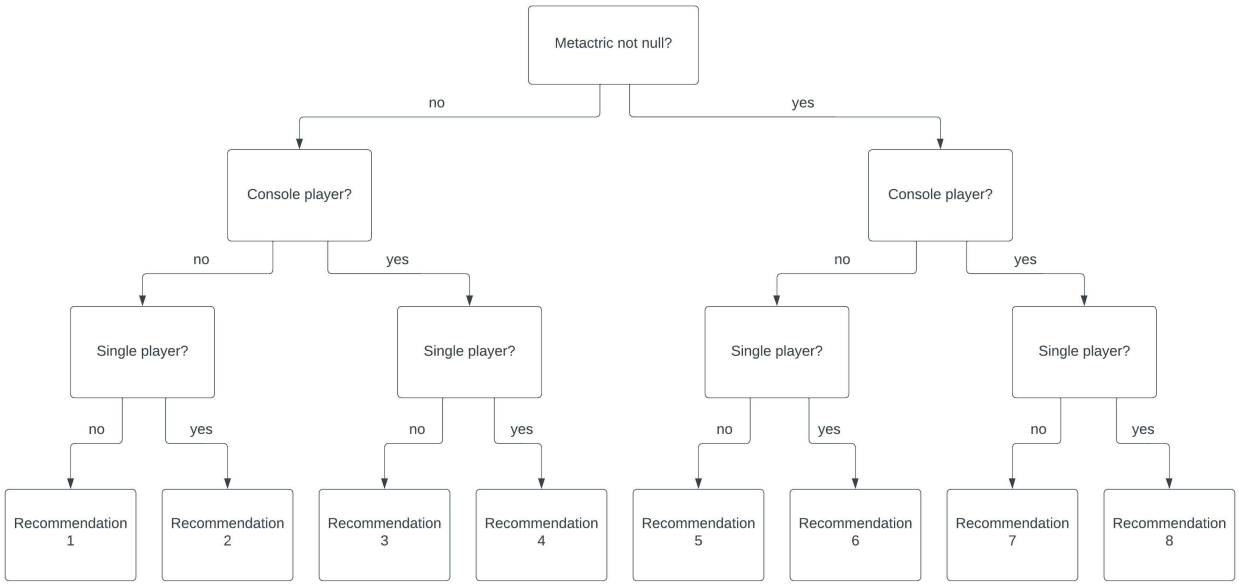
3. **Data accessing and caching** : Use BeautifulSoup to scrape and crawl data, and then cache data to json files. 

4. **Summary of data** : The amount of available data records depends on the keyword created by user choice, it may be over 3,000, or it can be less than 100. So to make it controllable and readeable, I set *a upper limit of **100** records* for each scraping from Steam store. To be specific, after the user answers a set of questions, the program will return a keyword. Suppose the keyword is 'witcher', which is from the card game "Gwent: The Witcher Card Game". The caching process is shown below.

- description of data
  - important fields:
    - "name": name of each game
    - "price": price of each game, or "Free to play", or showing discount information
    - "released": string, the date each game released at, (time format: %Y-%m-%d)
    - "url": the url of each game
- evidence of caching 

Data Structures

- **README:** I use Binary tree to design a questions tree, each left child represents a "no" answer, and each right child represents a "yes" answer. The first question is "Do you care about Metacritic?", the second one is "Are you a big fan of game consoles such as PlayStation, Xbox and Nintendo?" and the last question is "Do you prefer to be a single player?". Hence, $2^3=8$, there are 8 paths in the binary question tree. The function returns a dictionary where keys are like "000", "001" etc., and values are the paired SQL query.
- Decision Tree Diagram



- JSON file: tree.json
- Python file that constructs tree: decision_tree.py
- Python file that loads tree: load_json.py
- Screenshot of tree

load_json.py 1, U ● {} tree.json U ●

cache > tree > {} tree.json > {} leftNode > content


```
{
    "node": 0,
    "content": "\nselect distinct t1.*\nfrom",
    "leftNode": {
        "node": 1,
        "content": "\n(select *\n    from games\nwhere metacritic is null\n)t1",
        "leftNode": {
            "node": 3,
            "content": "\nwhere t1.id not in\n    (select distinct id\n        from games_platforms\n    where platform_id in\n        (select id
            "leftNode": {
                "node": 7,
                "content": "\nand t1.id not in\n    (select distinct id\n        from games_tags \n    where tag_id in \n        (select id\n
                "leftNode": null,
                "rightNode": null
            },
            "rightNode": {
                "node": 8,
                "content": "\nand t1.id in\n    (select distinct id\n        from games_tags \n    where tag_id in \n        (select id\n
                "leftNode": null,
                "rightNode": null
            }
        },
        "rightNode": {
            "node": 4,
            "content": "\nwhere t1.id in\n    (select distinct id\n        from games_platforms\n    where platform_id in\n        (select id\n
            "leftNode": {
                "node": 9,
                "content": "\nand t1.id not in\n    (select distinct id\n        from games_tags \n    where tag_id in \n        (select id\n
                "leftNode": null,
                "rightNode": null
            },
            "rightNode": {
                "node": 10,
                "content": "\nand t1.id in\n    (select distinct id\n        from games_tags \n    where tag_id in \n        (select id\n
                "leftNode": null,
                "rightNode": null
            }
        }
    },
    "rightNode": {
        "node": 2,
        "content": "\n(select *\n    from games\nwhere metacritic is not null\n)t1",
        "leftNode": {
            "node": 5,
```


Interaction and Presentation Options

- Instructions: There are three questions:
    1. "Do you care about Metacritic?"

2. "Are you a big fan of game consoles such as PlayStation, Xbox and Nintendo?"
3. "Do you prefer to be a single player?" All users need to do is to answer yes or no for each of them. And the program will give a recommendation table with at most 5 card games based on their choices. This table will be visualized by **Plotly**. Obviously, there are only 8 possible recommdations in total. And the program will find the keyword of the game with the max reviews, ratings and added counts to search relevant results from Steam, for the purpose of giving recommedations beyond card game meanwhile conforming to the preference of a user. The 100-piece keyword relevant recommendations will be shown in command line. You can quit the game by typing "no" when every round ends.

Demo Video Link

- https://www.youtube.com/watch?v=D5ynoPMBMjo