DATA515 Final Project Report

Author: Weiqing Ivy Lin

Project Name: Wikipedia API wrapper

Code Repo: https://github.com/IvyLinMS/wikipedia_api

Background

Last quarter I had a class called "DATA 512 A Au 21: Human-Centered Data Science", we did a lot of projects using Wikipedia data, then I found the Wikipedia API is not very convenient to use for our research. Since we learned a lot of Software Design in Data512, so I decided to create a python package to make it easy to use class for accessing Wikipedia page view API by providing unified API for page view aggregated data with set only once common parameters such as User-Agent and hide the detail of HTTP request construction, also provide top result per country on monthly level API which currently the Wikipedia API only support daily level.

Project Motivation

Wikipedia provide a set of RESTful APIs to get Wikipedia's page view data. Users of these APIs can provide different Wikipedia projects they are interested in and specify date range as well as different filters such as access method, agent type etc. but there are some issues with the APIs.

- Most of the parameters are in string format, such as the access method or agent type, which is error prone as typo or mis capitalization could happen
- Some of the API could set hourly, daily, monthly granularity, but the date format specified required to pass in hour information even user didn't specify to get the data at hourly granularity level
- Aggregated page view API has two version, one for legacy data from 12/01/2007 to 07/01/2015, while another API provide data after 07/01/2015, if user need to get data

from after 12/01/2007 to a date after 07/01/2015, user will have to explicit call the two API separately and provide different set of parameters for access method or agent type
- Top viewed article per country API only supports daily granularity without monthly level support

User profile.

- Researchers
- Students
- Corporate Dev/Data Engineers

The users are someone who are using Wikipedia data to doing some research.

Design considerations

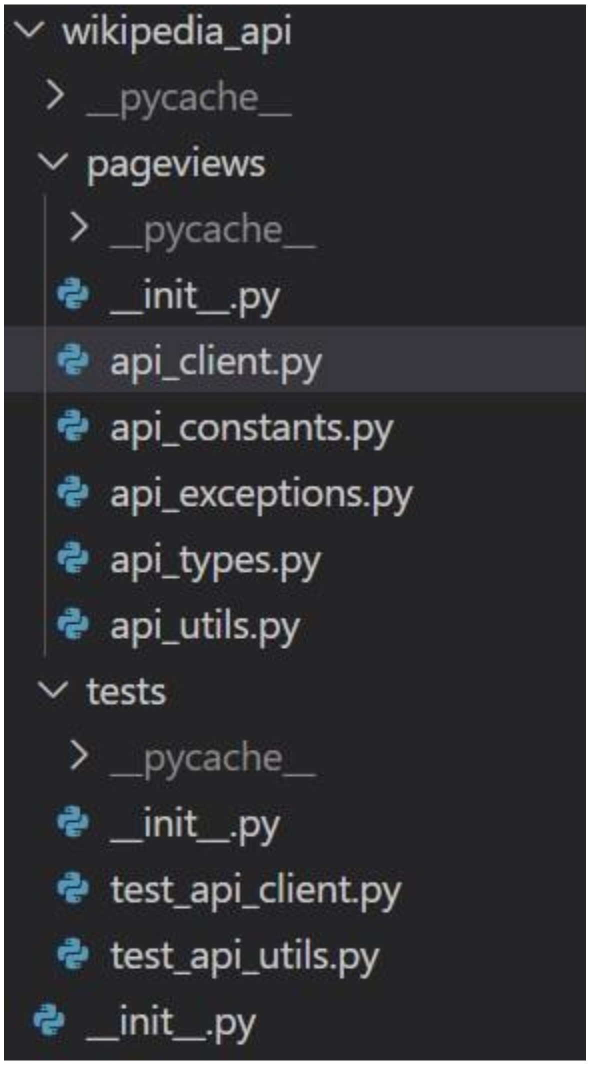
- Encapsulate the detail calling Restful API and parsing of json data, with returned panda data frame that available for further processing
- Initialize once Wikipedia project and API headers without providing for each API call
- Strong typed access method, agent type, granularity enumeration to avoid accident error
- Strong typed request data structure for easy future extension or change if the underlying API changed to have more parameters
- Flexible data range format support
- Parameter validation with detail error result

Detailed Design

Wikipedia Page View API client wrapper:

- Save common initialization parameter e.g., project or API header to avoid passing for each API call
- use strong typed parameter to avoid error-prone
- strong typed parameter for access or agent or granularity type
- do basic validation on user input such as date time format or invalid filter type or date range, and provided two extended functionalities
    a) extend to combine both legacy and current page view API result for aggregated page view API result
    b) extend the top viewed article per country API to support get result on whole month

Project Structure

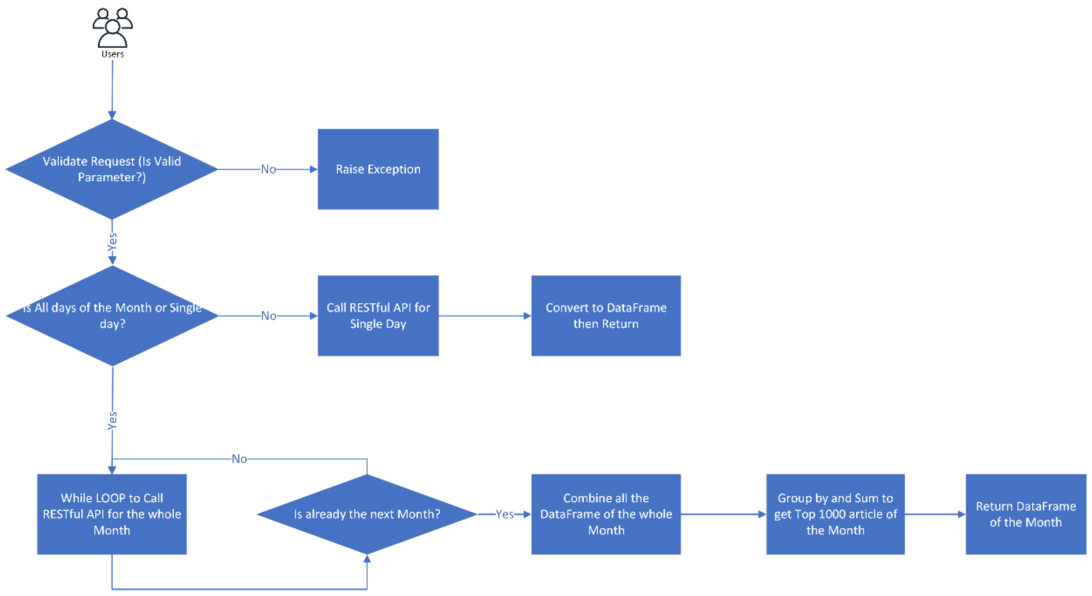


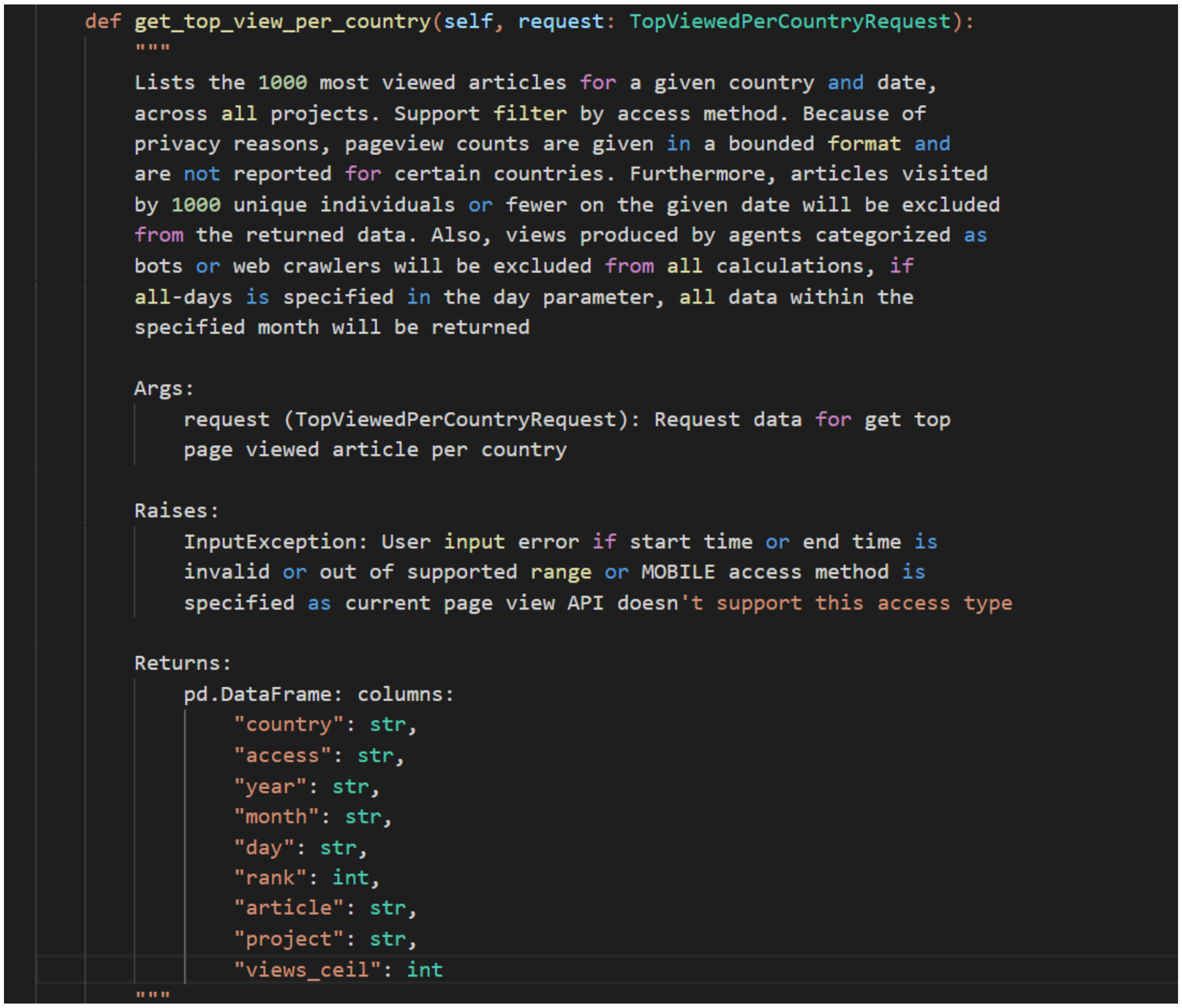
```
wikipedia_api
  __pycache__
  pageviews
    __pycache__
    __init__.py
    api_client.py
    api_constants.py
    api_exceptions.py
    api_types.py
    api_utils.py
  tests
    __pycache__
    __init__.py
    test_api_client.py
    test_api_utils.py
  __init__.py
```

Project Method Example

Based on what we have learned for Software Design, first we need to consider Object Oriental Design concept: Encapsulate the detail calling Restful API. Second, we should write a deep method instead of a shallow method, which means the interface should be simple, and the users can easy make the call without knowing the detailed logic inside.

- Using one method **get_top_view_per_country** as an example, the diagram for the detailed logic, but the interface is as simple as sending request data (TopViewedPerCountryRequest)

- 
Users
Validate Request (Is Valid Parameter?)
No
Raise Exception
Yes
Is All days of the Month or Single day?
No
Call RESTful API for Single Day
Convert to DataFrame then Return
Yes
No
While LOOP to Call RESTful API for the whole Month
Is already the next Month?
Yes
Combine all the DataFrame of the whole Month
Group by and Sum to get Top 1000 article of the Month
Return DataFrame of the Month

- 

```
    def get_top_view_per_country(self, request: TopViewedPerCountryRequest):
        """
        Lists the 1000 most viewed articles for a given country and date,
        across all projects. Support filter by access method. Because of
        privacy reasons, pageview counts are given in a bounded format and
        are not reported for certain countries. Furthermore, articles visited
        by 1000 unique individuals or fewer on the given date will be excluded
        from the returned data. Also, views produced by agents categorized as
        bots or web crawlers will be excluded from all calculations, if
        all-days is specified in the day parameter, all data within the
        specified month will be returned

        Args:
            request (TopViewedPerCountryRequest): Request data for get top
            page viewed article per country

        Raises:
            InputException: User input error if start time or end time is
            invalid or out of supported range or MOBILE access method is
            specified as current page view API doesn't support this access type

        Returns:
            pd.DataFrame: columns:
                "country": str,
                "access": str,
                "year": str,
                "month": str,
                "day": str,
                "rank": int,
                "article": str,
                "project": str,
                "views_ceil": int
        """
```

Comparison

This Wikipedia API wrapper is to make an easy-to-use class for accessing Wikipedia page view API by providing unified API for page view aggregated data with set only once common parameters such as User-Agent and hide the detail of HTTP request construction, also provide top result per country on monthly level API which currently the Wikipedia API only support daily level.

Comparing to my Data512 Wikipedia project, this Wikipedia API wrapper is much more convenient for use. I wrote some code which is using both this Wikipedia API wrapper and the Wikipedia page view API.

Reference to:

https://github.com/IvyLinMS/wikipedia_api/blob/main/examples/wikipedia_api_examples.ipynb

The new interface is very clean with one simple call, which we encapsulate all the detail calling Restful API and parsing of json data, with returned panda data frame with a deep method.

```python
# original way to call Wikipedia API
import requests
import pandas as pd

endpoint_legacy = 'https://wikimedia.org/api/rest_v1/metrics/legacy/pagecounts/aggregate/{project}/{access-site}/{granularity}/{start}/{end}'
endpoint_pageviews = 'https://wikimedia.org/api/rest_v1/metrics/pageviews/aggregate/{project}/{access}/{agent}/{granularity}/{start}/{end}'

headers = {
    'User-Agent': 'https://github.com/IvyLinMS',
    'From': 'ivylin@uw.edu'
}

pagecounts_mobile_site_params = {
    "project" : "en.wikipedia.org",
     "access-site" : "mobile-site",
     "granularity" : "monthly",
     "start" : "2007120100",
     # for end use 1st day of month following final month of data
     "end" : "2015070100"
}

pageviews_mobile_app_params = {
    "project" : "en.wikipedia.org",
    "access" : "mobile-app",
    "agent" : "user",
    "granularity" : "monthly",
    "start" : "2015070100",
    # for end use 1st day of month following final month of data
    "end" : '2021090100'
}

# Method to call API and dump result into json file
def api_call(endpoint,parameters):
    call = requests.get(endpoint.format(**parameters), headers=headers)
    response = call.json()
    return response

legacy_data = api_call(endpoint_legacy, pagecounts_mobile_site_params)
legacy_df =  pd.DataFrame.from_dict(legacy_data["items"])
pageview_data = api_call(endpoint_pageviews, pageviews_mobile_app_params)
pageview_df= pd.DataFrame.from_dict(pageview_data["items"])
df =  pd.concat([legacy_df, pageview_df], ignore_index=True)
```

```python
# sample use the Wikipedia API Library
from wikipedia_api.pageviews.api_client import WikipediaPageViewApiClient
from wikipedia_api.pageviews.api_types import (
    APIHeader,
    AccessMethod,
    AgentType,
    AggregatePageViewRequest,
    Granularity,
)

project = "en.wikipedia"
api_header = APIHeader("'https://github.com/IvyLinMS',", "ivylin@uw.edu")
client = WikipediaPageViewApiClient(project, api_header)
request = AggregatePageViewRequest(
    access=AccessMethod.MOBILE_APP,
    agent=AgentType.USER,
    granularity=Granularity.MONTHLY,
    start_time="20071201",
    end_time="20210901"
)
df = client.get_aggregated_pageviews(request)
```

Learnings

For this project, I spent a lot of time for setting up continuous integration on this project, there're couple of learning here:

1. At first, I tried to setup Travis CI, but it never works https://app.travis-ci.com/github/IvyLinMS/wikipedia_api/branches
2. As an alternative solution, I switched to use GitHub Actions, it saved a lot of time for my manual testing.

Lesson learned I should setup the continuous integration at the beginning of my project, it will be easier at the early stage of the project lifecycle. Since the cost of CI will be much less than I setup in the middle of the project, it will also save me a lot of cost for debugging and testing!