



Secure and Efficient Data Pipeline: Using Python APIs for Third-Party Data Integration

Pankaj Dureja

Pankaj.Dureja@gmail.com

ABSTRACT

In research and business analytics, there is an increasing requirement to integrate diverse data sources that can be extracted or loaded from third-party website. This paper describes a systematic method to provide secure data access and data integration by Python APIs. We explore powerful solutions for API credential management, request authentication and response data manipulation using the rich library support of Python that Obsidian provides to connectivity professionals. The suggested approach provides data protection and encryption, takes care of common issues such as user credential management, error handling or performance optimization.

In addition to the core implementation, this paper discusses potential extended use cases of the methodology, including real-time data analysis and business intelligence enhancement. We also evaluate the impact of secure data extraction on decision-making processes and strategic planning within organizations. By providing practical examples and best practices, this research aims to serve as a valuable resource for practitioners and researchers seeking to harness the power of Python APIs for secure and efficient data integration from third-party sources.

Keywords: Data Extraction, Python API, Third-party Websites, Credentials, Data Loading, Secure Access, Python Requests Library, JSON Data, API Authentication, Data Integration, Secure Data Management, Real-Time Data Analysis.

INTRODUCTION

In today's data-driven world, the ability to efficiently extract and load data from various sources is paramount for researchers and businesses alike. Third-party websites often contain valuable data that can enhance analytics, drive decision-making, and support strategic initiatives. However, accessing this data securely and efficiently presents several challenges, including the need for proper authentication, handling large datasets, and ensuring data integrity. Python, with its rich ecosystem of libraries and tools, offers a robust platform for interacting with APIs and managing these challenges effectively.

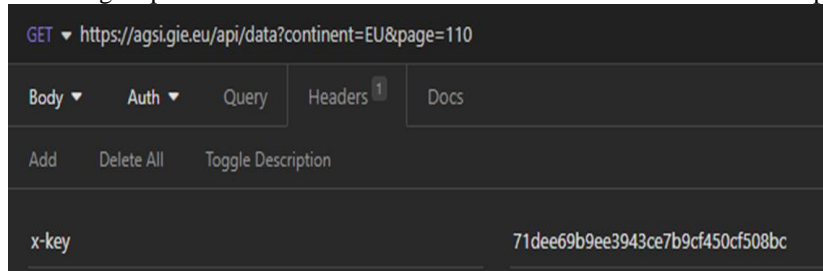
This paper focuses on the methodologies and best practices for extracting and loading data from third-party websites using Python APIs. We delve into the process of managing API credentials securely, making authenticated requests using the Python `requests` library, and handling data responses in various formats, particularly JSON. By addressing the common issues associated with API-based data extraction, such as credential security and error handling, we provide a comprehensive guide for researchers and practitioners. The implementation details, coupled with potential use cases and impacts, highlight the practical applications and benefits of using Python for secure data integration.

PROBLEM STATEMENT

In many sectors, comprehensive data analysis is a key to achieve as it can help informed conclusions be made. This includes a number of datasets and other sources; one example is Aggregated Gas Storage Inventory (AGSI) that gives granular information on gas storage across Europe. Energy market analysis, policy design and economic forecasting all rely on AGSI data. To extract this data is transparent in a secure and controlled manner which requires process.

It works by retrieving the AGSI data through its API from index Aggregators, but you need a permission to access and an authentication tool in order for that operation to work which is basically api key (x-key) Safe handling of this API key is a prerequisite to prevent unauthorized access and data breaches. In addition to, JSON-formatted data response needs heavy parsing and error-handling practices for proper handling of data.

Below screenshot show the get api to access the AGSI data in JSON format which needs to be processed.



API Output in the form of JSON

```

1  {
2      "last_page": 145,
3      "total": 30,
4      "dataset": "EU",
5      "gas_day": "2021-07-04",
6      "data": [
7          {
8              "name": "EU",
9              "code": "eu",
10             "url": "eu",
11             "updatedAt": "2021-04-18 10:47:32",
12             "gasDayStart": "2015-07-22",
13             "gasInStorage": "610.4131",
14             "consumption": "4035",
15             "consumptionFull": "15.13",
16             "injection": "3613.3",
17             "withdrawal": "99.9",
18             "netWithdrawal": "-3513.4",
19             "workingGasVolume": "1048.8641",
20             "injectionCapacity": "10096.32",
21             "withdrawalCapacity": "17215.32",
22             "contractedCapacity": "-",
23             "availableCapacity": "-",
24             "coveredCapacity": "-",
25             "status": "C",
26             "trend": "0.34",
27             "full": "58.2",
28             "info": []
29         },
30         {
31             "name": "EU",
32             "code": "eu",
33             "url": "eu",
34             "updatedAt": "2021-01-18 10:47:32",
35             "gasDayStart": "2015-07-21",
36             "gasInStorage": "606.8647",
37             "consumption": "4035",
38             "consumptionFull": "15.04",
39             "injection": "4010.29",
40             "withdrawal": "120.6",
41             "netWithdrawal": "-3889.7",

```

SOLUTION IMPLEMENTED

To address the above problem, a python program was implemented, and the following libraries and methodologies were used.

Libraries:

1. Requests: For making HTTP requests.
2. Pandas Data Frame: For data manipulation and analysis.
3. SQL Alchemy: For database interactions.

Methodology:

1. Credential Management: Using environment variables and secure vaults to store credentials.
2. API Interaction: Utilizing the `requests` library to make authenticated requests.

3. Data Extraction: Parsing the response data using JSON or XML parsers.
4. Data Loading: Storing the extracted data into databases or data frames for analysis.
5. Error Handling: Implementing robust error handling to manage failed requests and retries.

Python Implementation:

```
def LoadGasStorage(INPUT):
    try:
        RAW_TABLE_NAME = INPUT['raw_table']
        REGION_SCOPE = INPUT['region_scope']
        MS_ENGINE = INPUT['engine']

        # if COMPLETE:
        #     START_FETCH_DATE = pd.to_datetime(START_DATE).strftime('%Y-%m-%d')
        #     END_FETCH_DATE = pd.to_datetime(date.today()).strftime('%Y-%m-%d')
        # else:
        #     START_FETCH_DATE = (pd.to_datetime(date.today()) - timedelta(7)).strftime('%Y-%m-%d')
        #     END_FETCH_DATE = pd.to_datetime(date.today()).strftime('%Y-%m-%d')

        l_sqllist = []
        l_sqllist.append("START TRANSACTION")
        l_sqllist.append("DELETE FROM %s WHERE UPPER(region) = '%s' " %(RAW_TABLE_NAME, REGION_SCOPE))
        l_sqllist.append("COMMIT")
        logging.info(l_sqllist)
        execMemSQLStatement(MS_ENGINE, l_sqllist)

        gas_storage_url = INPUT['url']
        logging.info(gas_storage_url)

        DATA_SCOPE_RESPONSE = {}
        DATA_SCOPE_RESPONSE = requests.request("GET", url=gas_storage_url, headers=INPUT['header'])
        STATUS_CODE = DATA_SCOPE_RESPONSE.status_code
        logging.info(STATUS_CODE)
        data_json = DATA_SCOPE_RESPONSE.text
        df = pd.read_json(StringIO(data_json))
        last_page = int(df['last_page'][0])

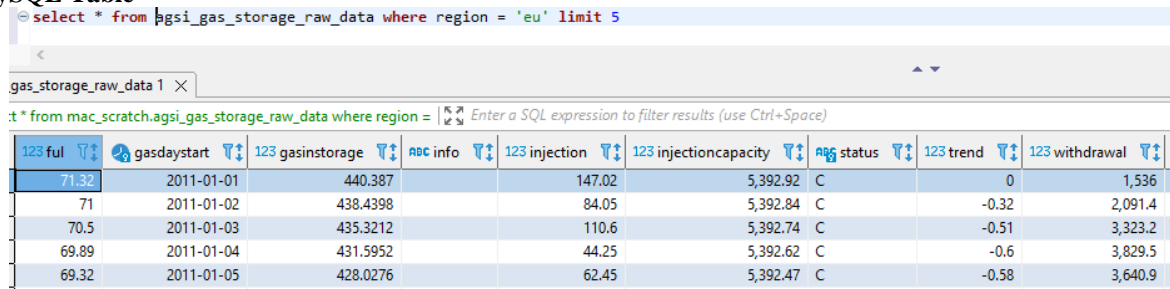
    for ps in range(1, last_page+1):
        l_sqllist = []
        gas_storage_ps_url = gas_storage_url + '&page=' + str(ps)
        logging.info(gas_storage_ps_url)
        DATA_SCOPE_PS_RESPONSE = requests.request("GET", url=gas_storage_ps_url, headers=INPUT['header'])
        STATUS_CODE_PS = DATA_SCOPE_PS_RESPONSE.status_code
        logging.info(STATUS_CODE_PS)
        data_json_ps = DATA_SCOPE_PS_RESPONSE.text
        if data_json_ps:
            df_ps = pd.read_json(StringIO(data_json_ps))
            #logging.info(df['last_page'][0])
            #logging.info(df['data'][0]['info'])
            #df.columns = map(RenameDFColumn, df.columns)
            logging.info(len(df_ps.index))
            for r in range(0, len(df_ps.index)):
                ful = df_ps['data'][r]['full']
                gasdaystart = df_ps['data'][r]['gasDayStart']
                gasinstorage = df_ps['data'][r]['gasInStorage']
                info = ''
                injection = df_ps['data'][r]['injection']
                injectioncapacity = df_ps['data'][r]['injectionCapacity']
                status = df_ps['data'][r]['status']
                trend = df_ps['data'][r]['trend']
                withdrawal = df_ps['data'][r]['withdrawal']
                withdrawalcapacity = df_ps['data'][r]['withdrawalCapacity']
                workinggasvolume = df_ps['data'][r]['workingGasVolume']
                region = REGION_SCOPE
                code = df_ps['data'][r]['code']
                name = df_ps['data'][r]['name']
                url = df_ps['data'][r]['url']
                consumption = df_ps['data'][r]['consumption']
                consumptionfull = df_ps['data'][r]['consumptionFull']
                row_values = (ful, gasdaystart, gasinstorage, info, injection, injectioncapacity, status, trend, withdrawal, w
                l_sqllist.append('INSERT INTO '+ RAW_TABLE_NAME + ' (ful, gasdaystart, gasinstorage, info, injection, injec
                +str(row_values))

                l_sqllist.append("COMMIT")
                # logging.info(l_sqllist)
                execMemSQLStatement(MS_ENGINE, l_sqllist)

    except Exception as e:
        logging.info(" agsi gas storage scope: %s raw table: %s data job failed.", REGION_SCOPE, RAW_TABLE_NAME)
        raise
```

The screenshot below displays data that has been loaded into a database using a Python program. It illustrates each record organized by date, with JSON values parsed into distinct columns and rows.

MySQL Table



```
select * from agsi_gas_storage_raw_data where region = 'eu' limit 5
```

123 ful	gasdaystart	123 gasinstorage	abc info	123 injection	123 injectioncapacity	ags status	123 trend	123 withdrawal
71.32	2011-01-01	440.387		147.02	5,392.92	C	0	1,536
71	2011-01-02	438.4398		84.05	5,392.84	C	-0.32	2,091.4
70.5	2011-01-03	435.3212		110.6	5,392.74	C	-0.51	3,323.2
69.89	2011-01-04	431.5952		44.25	5,392.62	C	-0.6	3,829.5
69.32	2011-01-05	428.0276		62.45	5,392.47	C	-0.58	3,640.9

POTENTIAL EXTENDED USE CASES

1. Ingesting data feeds for near real-time analytics
2. Machine Learning to train ML models using the extracted data
3. Automating the data extraction process for repetitive updates.
4. Business Intelligence - Business intelligence platforms are amplified with external data sources.

IMPACT

Integrating secured data extraction and loading helps increase the performance and accuracy of your projects directed by a certain set of Data points. Enterprises can draw upon the capabilities of external data sources yet maintain control over their own in a way that enforces good governance and preserves the trust, accuracy, quality and provenance of sensitive information for informed decision-making & policies.

SCOPE

The scope of this methodology can be extended to various domains such as finance, healthcare, marketing, and social media analytics. Future work can focus on enhancing security measures, optimizing performance, and expanding the range of supported APIs and data formats.

CONCLUSION

This paper presents a comprehensive approach to extracting and loading data from third-party websites using Python APIs and credentials. By following best practices in credential management, API interaction, and data handling, researchers and practitioners can ensure secure, efficient, and reliable data extraction processes.

REFERENCES

- [1]. Working with JSON Data, Chapter 16, <https://automatetheboringstuff.com/2e/chapter16/>
- [2]. Python for Data Analysis by Wes McKinney, O'Reilly Media, 2017, pp. 220-245.
- [3]. Aggregated Gas Storage Inventory, Available at <https://agsi.gie.eu/>
- [4]. Python Requests Documentation, Available at <https://requests.readthedocs.io/en/latest/>
- [5]. Pandas Documentation, Available at <https://pandas.pydata.org/pandas-docs/stable/>
- [6]. SQL Alchemy Documentation, Available at <https://docs.sqlalchemy.org/>