JAMIA Open, 2025, **8(1)**, ooae146
https://doi.org/10.1093/jamiaopen/ooae146
Application Notes

Application Notes

pyBioPortal: a Python package for simplifying cBioPortal data access in cancer research

Matteo Valerio, MscEng[1,*], Alessandro Inno, MD[1], Stefania Gori, MD[1]

[1]Medical Oncology, IRCCS Sacro Cuore Don Calabria Hospital, 37024 Negrar di Valpolicella, Verona, Italy

*Corresponding author: Matteo Valerio, MscEng, Medical Oncology, IRCCS Sacro Cuore Don Calabria Hospital, Via Don A. Sempreboni, 5, 37024 Negrar di Valpolicella, Verona, Italy (matteo.valerio@sacrocuore.it)

Abstract

Objectives: In recent years, the rise of big data and artificial intelligence has led to an increasing expansion of databases and web services in biomedical research. cBioPortal is one of the most widely used platforms for accessing cancer genomic and clinical data. The primary objective of this study was to develop a tool that simplifies programmatic interaction with cBioPortal's web service.

Materials and Methods: We developed the *pyBioPortal* Python package, which leverages the cBioPortal REST API to access genomic and clinical data. The retrieved data is returned as a Pandas DataFrame, a format widely used for data analysis in Python.

Results: *pyBioPortal* offers an efficient interface between the user and the cBioPortal database. The data is provided in formats conducive to further analysis and visualization, promoting workflows and improving reproducibility.

Discussion: The development of *pyBioPortal* addresses the challenge of accessing and processing large volumes of biomedical data. By simplifying the interaction with the cBioPortal API and providing data in Pandas DataFrame format, *pyBioPortal* allows users to focus more on the analytical aspects rather than data extraction.

Conclusion: This tool facilitates the retrieval of heterogeneous biological and clinical data in a standardized format, making it more accessible for analysis and enhancing the reproducibility of results in cancer informatics. Distributed as an open-source project, *pyBioPortal* is available to the broader bioinformatics community, promoting collaboration and advancing research in cancer genomics.

Lay Summary

The advent of big data and artificial intelligence has revolutionized cancer research, making large amounts of data available for analysis. However, accessing this data can be challenging, especially for researchers without specific programming skills or those who prefer to focus on data analysis and interpretation rather than on the technical aspects of data extraction. *pyBioPortal* is a newly developed Python tool designed to simplify the process of retrieving data from cBioPortal, a widely used platforms for accessing cancer genomic and clinical data. By making data easier to access and analyze, *pyBioPortal* enables researchers to focus on uncovering new insights that drive advancements in cancer research. The tool is freely available as an open-source project, promoting widespread use and collaboration within the bioinformatics community.

Key words: cBioPortal; cancer research; bioinformatics; Python.

Introduction

In recent years, with the advent of big data and artificial intelligence, there has been a growing proliferation of databases and web services in the field of biomedical research,[1] which have now become essential tools for sharing and accessing crucial information in cancer informatics. Data access is fundamental in cancer research, where the analysis of large amounts of genomic, transcriptomic, and clinical data can lead to new discoveries and therapeutic developments.cBioPortal[2–4] is one of the most widely used platforms for accessing such data, offering both an interactive web-based exploration interface and public Representational State Transfer (REST) Application Programming Interface (API)[5] for retrieving information from a wide range of cancer studies. However, directly using the downloadable data from cBioPortal without significant processing, or specifically its available API, could be challenging for researchers and university students without specific programming skills, as the data may be presented in complex, nested formats that require conversion into usable tables. Moreover, users who prefer to focus on data analysis and interpretation rather than on the technical aspects of data extraction and preparation may find these tasks prohibitive. To address this issue, a new Python package called *pyBioPortal* has been developed.

Objectives

The main objective that led to the development of *pyBioPortal* is to simplify programmatic interaction with cBioPortal's public API and provide the retrieved data in a standardized format, directly usable for analysis, with the aim of improving the reproducibility of results in cancer research.

Received: September 19, 2024; **Revised:** November 25, 2024; **Editorial Decision:** December 2, 2024; **Accepted:** December 4, 2024

Materials and methods

Implementation and architecture

Python 3[6] was chosen as the programming language for *pyBioPortal*, given its growing popularity in bioinformatics and compatibility with all major operating systems. The version of Python used for development is 3.9, which ensures stability and support in the more recent versions of Python, as version 2 has not been supported since 2020.

The package uses the REST API web services provided by cBioPortal to retrieve the data of interest, leveraging external libraries such as Requests[7] and Pandas[8] for handling Hypertext Transfer Protocol (HTTP) requests[9] and processing the retrieved data. The acquired and appropriately processed data are returned to the user as a Pandas DataFrame object, a widely used format for data analysis in Python, as shown in Figure 1.

The structure of the package is organized into modules, each of which is dedicated to a specific type of data, as defined by the cBioPortal API described in the related documentation (https://www.cbioportal.org/api/swagger-ui/index.html) and shown in Figure 1.

Each module in the package consists of various functions that provide simplified access to the resources, handling the interaction with the web service transparently for the user, without the need to know how the HTTP request to be sent to the server should be structured.

pyBioPortal manages the entire set of endpoints provided by the cBioPortal REST API, supporting HTTP requests with both GET and POST methods. The functions within each module are named according to a convention outlined in the API documentation, which identifies the type of request sent to the server: functions using the GET method are prefixed with "get," while those using the POST method are prefixed with "fetch."

Database query and data retrieval

To specify and request which data you want to retrieve from the cBioPortal database, it is necessary to define a query having parameters whose type and meaning are described in the online documentation provided by cBioPortal. For this purpose, by properly setting these parameters and passing them as arguments to the relevant *pyBioPortal* functions, based on the type of data of interest, the following operations are performed transparently for the user: composing the HTTP request to send to the server; parsing the responses obtained from the web API provided in JSON format (JavaScript Object Notation)[10]; processing the results and returning a Pandas DataFrame containing the desired data.

To deliver data directly in a tabular format as a Pandas DataFrame, the package leverages private auxiliary functions, like *flatten_dict_columns* and *flatten_dict_list_columns*, to automatically convert nested dictionaries and lists of dictionaries into flat DataFrame columns. This internal processing is entirely transparent to the user, allowing for immediate access to the data in a usable, analysis-ready format without requiring additional data manipulation. This feature exemplifies the package's approach to simplifying data extraction from complex JSON formats (for technical details, full function implementations are available in the GitHub repository).

In addition to the standard parameters required by the API, some functions accept additional arguments that allow for more processed results. For example, it is possible to obtain a DataFrame with data organized in either "wide" or "long" format. In the "wide" format, variables associated with the same record (ie, associated with the same patient or sample) are arranged across multiple columns, which can be useful for visualizations or specific types of statistical analyses (an illustrated example of this transformation is provided further in the text). Conversely, in the "long" format, these variables are spread over multiple rows, allowing for easier application of certain data manipulation techniques and analyses that require data to be in a "tall" structure.

For a detailed overview of how to use and correctly set the parameters for each function, consult the documentation created for the package, available online.

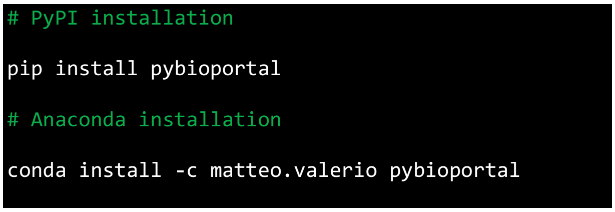
Runtime execution errors are handled by displaying the error description returned natively by the web service in the Python console, along with custom error messages if incorrect arguments are passed to the functions in the various modules.

Results

Installation

pyBioPortal has been published on the Python Package Index (PyPI) (https://pypi.org/project/pybioportal/) and on Anaconda.org (https://anaconda.org/matteo.valerio/pybioportal), the 2 most popular platforms used for distributing Python packages.

The installation of the package is straightforward and follows the standard procedure for both platforms with the following commands:

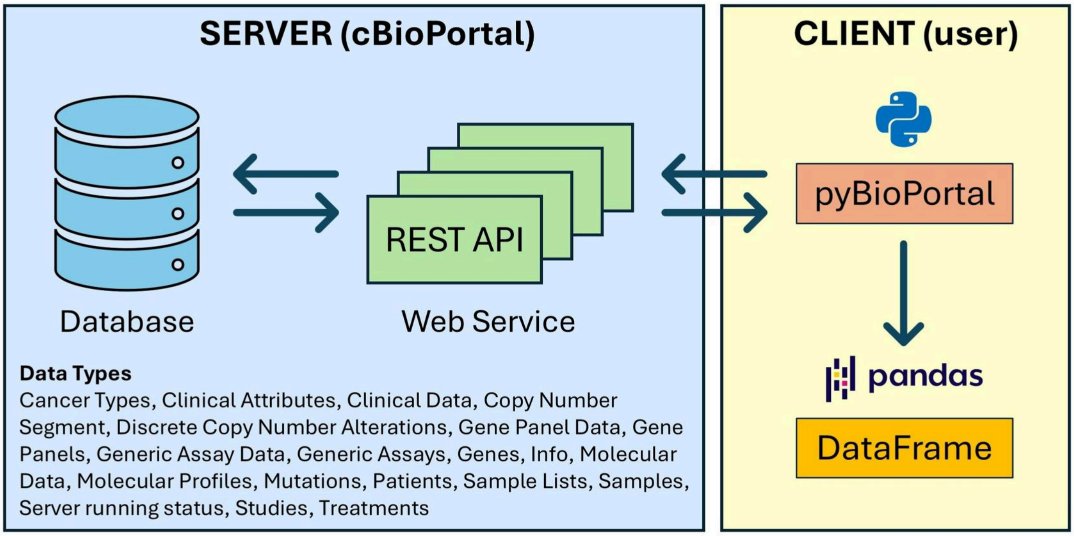


```
# PyPI installation

pip install pybioportal

# Anaconda installation

conda install -c matteo.valerio pybioportal
```

Example of usage

pyBioPortal implements a simple and efficient interface between the user and the cBioPortal database. To illustrate its usage, an example is provided for retrieving clinical data on patient survival from a study available in cBioPortal. The study considered is related to The Cancer Genome Atlas Serous Ovarian Cancer project,[11] identified by the code "ov_tcga_pub."

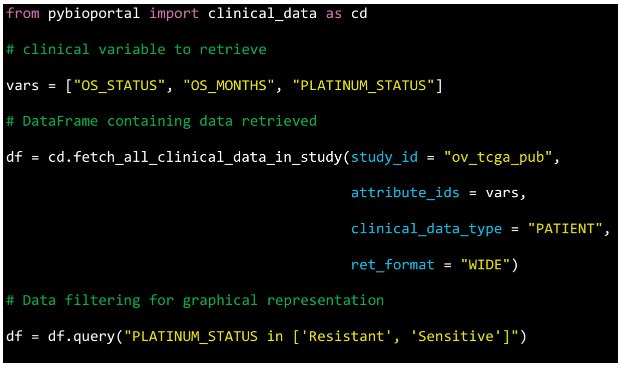
First, import the module that handles the specific type of data, in this case, "clinical_data." Next, identify the appropriate function from the module for the data to be retrieved, in this case, "fetch_all_clinical_data_in_study." Then, specify the variables to be acquired, according to the cBioPortal documentation. In this example, the selected variables are "os_status" (overall survival status), "os_months" (overall survival time in months), and "platinum_status" (the patient's response to platinum-based treatment). Finally, based on the specifications of the parameters to be passed to



Data Types
Cancer Types, Clinical Attributes, Clinical Data, Copy Number Segment, Discrete Copy Number Alterations, Gene Panel Data, Gene Panels, Generic Assay Data, Generic Assays, Genes, Info, Molecular Data, Molecular Profiles, Mutations, Patients, Sample Lists, Samples, Server running status, Studies, Treatments

Figure 1. Schematic representation of the data flow from cBioPortal to the user through *pyBioPortal*.

the function, as reported in the *pyBioPortal* documentation, it can be executed as shown in the code below:



```
from pybioportal import clinical_data as cd

# clinical variable to retrieve

vars = ["OS_STATUS", "OS_MONTHS", "PLATINUM_STATUS"]

# DataFrame containing data retrieved

df = cd.fetch_all_clinical_data_in_study(study_id = "ov_tcga_pub",

                                         attribute_ids = vars,

                                         clinical_data_type = "PATIENT",

                                         ret_format = "WIDE")

# Data filtering for graphical representation

df = df.query("PLATINUM_STATUS in ['Resistant', 'Sensitive']")
```

The Pandas DataFrame returned by the function is shown in Figure 2.

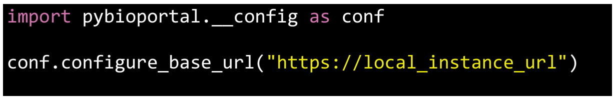
In this example, an additional argument, "ret_format," is passed to the function "fetch_all_clinical_data_in_study," specifying that the DataFrame should be returned in "wide" format. This way, each row of the DataFrame corresponds to a patient, making it very convenient to use, for example, to plot Kaplan-Meier survival curves using the Python package Lifelines,[12] obtaining the graph shown in Figure 3.

For further details on how to use the functions of the various *pyBioPortal* modules, refer to the online documentation published on Read the Docs (https://pybioportal.readthedocs.io), which includes various application examples organized in Jupyter Notebooks.[13]

Configuration for local instances of cBioPortal

The *pyBioPortal* package is designed to interact with the public API of cBioPortal, with the default Uniform Resource Locator (URL) set to https://www.cbioportal.org/api. However, the package also supports the use of local instances of cBioPortal, which can be implemented by institutions or companies that wish to maintain their data on private servers.

To use a local instance instead of the public one, API base URL can be configured using the "configure_base_url" function from the *pyBioPortal* configuration module ("__config"). This allows HTTP requests to be sent to a specific cBioPortal instance, installed locally or on a private server, rather than to the public instance. The following code example shows how to set a custom base URL:



```
import pybioportal.__config as conf

conf.configure_base_url("https://local_instance_url")
```

This feature is particularly useful for users working with sensitive data or who require complete control over their analysis environment, avoiding dependence on the public cBioPortal infrastructure.

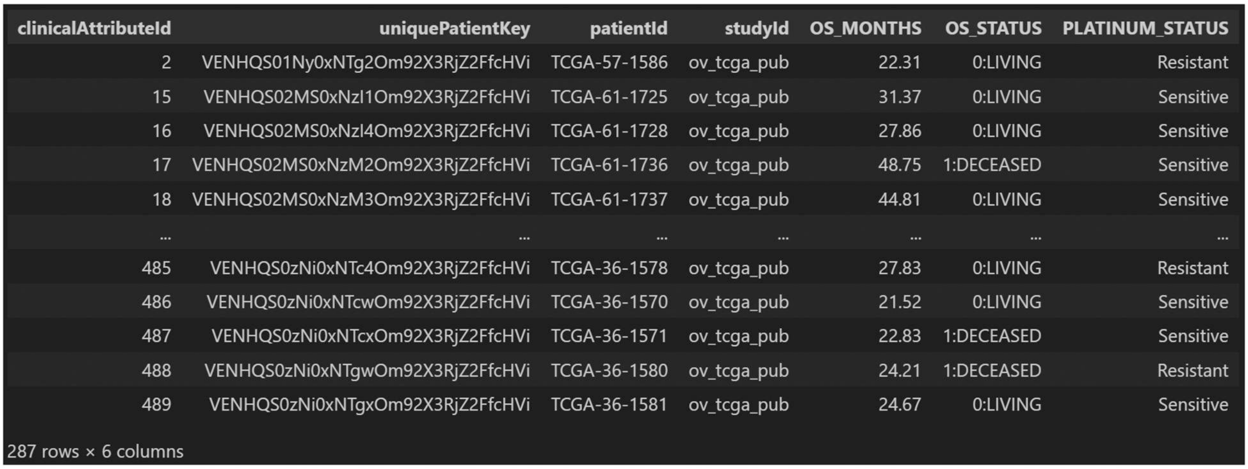
Discussion

The *pyBioPortal* package was developed with the main goal of simplifying interaction with the public cBioPortal API, making biological and clinical data more accessible to researchers or university students who are not familiar with advanced programming.

This tool significantly reduces the complexity of the data acquisition process, allowing users to focus on analysis rather than data extraction techniques. As demonstrated in the results, data are acquired with just a few lines of Python code, a simplified approach that enhances efficiency and contributes to improving the reproducibility of analyses, a key objective in bioinformatics research.

From the official documentation of cBioPortal (https://docs.cbioportal.org/web-api-and-clients/), there are 2 main ways to access the API using Python: the Bravado package, which allows direct interaction with the API, and the cbio_py package, a wrapper for the API.

Compared to these approaches, *pyBioPortal* provides advantages in terms of usability and functionality. While Bravado allows direct and flexible interaction with the API, it



clinicalAttributeId	uniquePatientKey	patientId	studyId	OS_MONTHS	OS_STATUS	PLATINUM_STATUS
2	VENHQS01Ny0xNTg2Om92X3RjZ2FfcHVi	TCGA-57-1586	ov_tcga_pub	22.31	0:LIVING	Resistant
15	VENHQS02MS0xNzI1Om92X3RjZ2FfcHVi	TCGA-61-1725	ov_tcga_pub	31.37	0:LIVING	Sensitive
16	VENHQS02MS0xNzI4Om92X3RjZ2FfcHVi	TCGA-61-1728	ov_tcga_pub	27.86	0:LIVING	Sensitive
17	VENHQS02MS0xNzM2Om92X3RjZ2FfcHVi	TCGA-61-1736	ov_tcga_pub	48.75	1:DECEASED	Sensitive
18	VENHQS02MS0xNzM3Om92X3RjZ2FfcHVi	TCGA-61-1737	ov_tcga_pub	44.81	0:LIVING	Sensitive
...	...	...	...	...	...	...
485	VENHQS0zNi0xNTc4Om92X3RjZ2FfcHVi	TCGA-36-1578	ov_tcga_pub	27.83	0:LIVING	Resistant
486	VENHQS0zNi0xNTcwOm92X3RjZ2FfcHVi	TCGA-36-1570	ov_tcga_pub	21.52	0:LIVING	Sensitive
487	VENHQS0zNi0xNTcxOm92X3RjZ2FfcHVi	TCGA-36-1571	ov_tcga_pub	22.83	1:DECEASED	Sensitive
488	VENHQS0zNi0xNTgwOm92X3RjZ2FfcHVi	TCGA-36-1580	ov_tcga_pub	24.21	1:DECEASED	Resistant
489	VENHQS0zNi0xNTgxOm92X3RjZ2FfcHVi	TCGA-36-1581	ov_tcga_pub	24.67	0:LIVING	Sensitive

287 rows × 6 columns

Figure 2. Pandas DataFrame returned by the function execution.

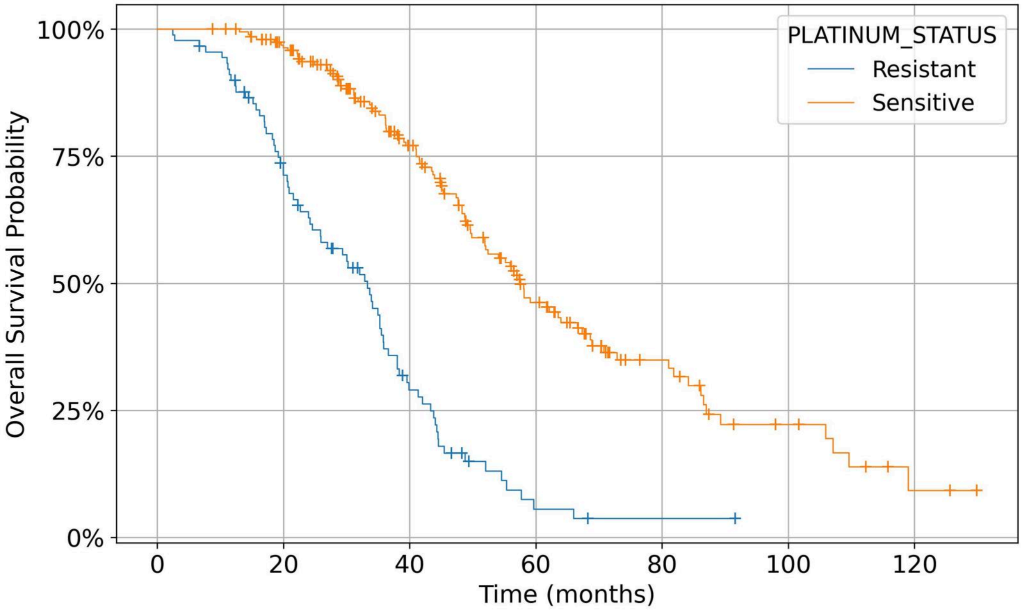


Figure 3. Kaplan-Meier survival curves obtained using Lifelines package with the clinical data retrieved from cBioPortal.

requires familiarity with API specifications. The retrieved data are structured as complex objects in the native API format, often requiring significant post-processing before data analysis. The cbio_py package simplifies this process by returning the retrieved data as a list of dictionaries, as well as providing it in the native format. However, as reported in the documentation, cbio_py is limited to GET endpoints and does not handle treatment-related data. Additionally, it still requires post-processing to prepare data for analysis. In contrast, *pyBioPortal* handles the entire set of cBioPortal API endpoints, including both GET and POST methods, and provides data directly in the user-friendly Pandas DataFrame format. This allows flexible and comprehensive access to various types of clinical and genomic data, facilitating data analysis.

The ability to easily configure queries through function arguments and receive data in Pandas DataFrame format provides a significant advantage to users, who can leverage useful functions applicable to DataFrames for describing and presenting data.

Currently, *pyBioPortal* uses the Pandas library to convert data obtained from cBioPortal into DataFrames, providing a familiar and versatile format for data analysis in Python. However, we recognize that processing very large datasets may present memory and performance limitations. Additionally, the retrieval time of large datasets from the cBioPortal API could be lengthy, slowing down the execution of code that depends on it. In the future, implementing techniques such as data streaming or data chunking could improve the handling of large datasets. These approaches would allow loading and processing data in smaller portions, reducing memory usage and increasing the efficiency of the package on extensive datasets. To optimize both memory usage and API response times, users are encouraged to specify targeted queries to avoid loading unnecessary data for their analysis.

The use of the Pandas DataFrame format, popular within the scientific community, enables integration of this information into custom Python scripts and pipelines for further processing and analysis, as well as leveraging Jupyter Notebooks to support workflows in scientific computing.

Given that cBioPortal is adopted by numerous institutions and companies with local installations, *pyBioPortal* includes functionality to configure the server URL for API interactions. This allows users to send requests to private servers, keeping data internally and adapting to the needs of managing sensitive data that require complete control over the analysis environment.

In the future, it is planned to implement new supporting features that will enhance and improve usage, such as user authorization support for local cBioPortal instances and the development of built-in analysis functions. Moreover, future API updates will be considered and included in subsequent versions of the package.

Conclusion

In this paper, we presented *pyBioPortal*, a Python package useful for retrieving data from cBioPortal through its REST web services, which provides quick and easy access to heterogeneous biological and clinical data.

The aim is to provide a significant contribution to the bioinformatics community by offering a tool that facilitates programmatic access to cBioPortal data and enhances the reproducibility of the analyses in cancer research. For this reason, *pyBioPortal* has been distributed on GitHub and released as an open-source project under the Berkeley Software Distribution version 3 (BSD-3) license to maximize accessibility, collaboration, and usage.

Author contributions

Matteo Valerio designed, developed, and tested the *pyBioPortal* Python package module and example code, wrote the original draft, reviewed, and edited the manuscript. Alessandro Inno and Stefania Gori reviewed the manuscript and provided critical revisions. All authors have read and agreed to the published version of the manuscript.

Funding

This research received no external funding.

Conflicts of interest

The authors declare no conflicts of interest.

Data availability

pyBioPortal is freely available on PyPI at https://pypi.org/project/pybioportal/ and on Anaconda.org at https://anaconda.org/matteo.valerio/pybioportal. Documentation and examples are available online at https://pybioportal.readthedocs.io. *pyBioPortal* source code is publicly accessible at the following GitHub repository: https://github.com/Matteo-Valerio/pyBioPortal.

References

1. Luo J, Wu M, Gopukumar D, et al. Big data application in biomedical research and health care: a literature review. *Biomed Inform Insights*. 2016;8:1-10. https://doi.org/10.4137/BII.S31559
2. Cerami E, Gao J, Dogrusoz U, et al. The cBio cancer genomics portal: an open platform for exploring multidimensional cancer genomics data. *Cancer Discov*. 2012;2:401-404. https://doi.org/10.1158/2159-8290.CD-12-0095
3. Gao J, Aksoy BA, Dogrusoz U, et al. Integrative analysis of complex cancer genomics and clinical profiles using the cBioPortal. *Sci Signal*. 2013;6:pl1. https://doi.org/10.1126/scisignal.2004088
4. de Bruijn I, Kundra R, Mastrogiacomo B, et al.; AACR Project GENIE BPC Core Team, AACR Project GENIE Consortium. Analysis and visualization of longitudinal genomic and clinical data from the AACR project GENIE biopharma collaborative in cBioPortal. *Cancer Res*. 2023;83:3861-3867. https://doi.org/10.1158/0008-5472.CAN-23-0816
5. Fielding RT. Representational state transfer (REST). In: *Fielding RT. Architectural Styles and the Design of Network-Based Software Architectures [Dissertation]*. University of California, Irvine; 2000:76-106. https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
6. Van Rossum G, Python Development Team. Python Language Reference Release 3.11.3. Lulu Press, Incorporated; 2023. ISBN: 9781312573949.
7. Requests: HTTP for Humans^{TM}–Requests 2.32.3 documentation. Accessed August 26, 2024. https://requests.readthedocs.io/en/latest/
8. McKinney W. Data structures for statistical computing in Python. In: *Proceedings of the 9th Python in Science Conference*. 2010;56-61. https://doi.org/10.25080/Majora-92bf1922-00a
9. Fielding RT, Reschke J. Hypertext Transfer Protocol (HTTP/1.1): semantics and content. Internet Engineering Task Force. 2014. Accessed August 27, 2024. https://www.rfc-editor.org/rfc/pdfrfc/rfc7231.txt.pdf
10. Bray T. The JavaScript object notation (JSON) data interchange format. Internet Engineering Task Force. 2017. Accessed August 27, 2024. https://www.rfc-editor.org/rfc/pdfrfc/rfc8259.txt.pdf
11. Cancer Genome Atlas Research Network. Integrated genomic analyses of ovarian carcinoma. *Nature*. 2011;474:609-615. https://doi.org/10.1038/nature10166
12. Davidson-Pilon C. Lifelines: survival analysis in python. *JOSS*. 2019;4:1317. https://doi.org/10.21105/joss.01317
13. Kluyver T, Ragan-Kelley B, Pérez F, et al. Jupyter Notebooks—a publishing format for reproducible computational workflows. In: *Proceedings of the 20th International Conference on Electronic Publishing*. IOS Press; 2016:87-90.

JAMIA Open, 2025, 8, 1–5
https://doi.org/10.1093/jamiaopen/ooae146
Application Notes