Quality Assurance of Data Imported from the Genomic Data Commons (GDC) into the Xena Browser

Gabriel Jaimes

UCSC Genomics Institute, Xena Cancer Browser.

Abstract

The Xena Browser is a web-based tool that enables researchers to visualize and analyze cancer functional genomics data by integrating genomic, clinical, and survival data from different sources, including the Genomic Data Commons (GDC). To ensure the integrity of the data imported from the GDC via the Xena GDC Extract Transform Load (ETL) pipeline, automated quality assurance testing code has been developed here to validate this data. This testing code utilizes orthogonal data retrieval and comparison methods, independently verifying genomic, clinical, and survival data against data retrieved from the GDC. Multiple methods are implemented to accomplish this, such as rounding optimizations for gene expression data, handling multiple data values for single samples, addressing inconsistencies in the clinical and mutation files, and creating a logger file to document all inconsistencies. The testing code has been rigorously tested using flawed data and files to ensure its effectiveness in identifying errors. This automated approach not only saves time in QAing this data but also ensures the quality of important cancer functional genomics data for research purposes.

1 Introduction

The Xena browser [2] is a tool that allows researchers to visualize and analyze cancer functional genomics data. By integrating genomic, clinical, and survival data, Xena enables researchers to dive deep into tumor biology. Xena allows for the visualization and analysis of many different types of data from various sources, including the Genomic Data Commons (GDC) [8]. The GDC has generated and collected many valuable datasets from several projects, including data from The Cancer Genome Atlas (TCGA) [7], the Human Cancer Models

Initiative (HCMI) [3], and the Clinical Proteomic Tumor Analysis Consortium (CPTAC) [4]. To give researchers access to the open-access data in the GDC through the Xena browser, the Xena GDC Extract Transform Load (ETL) pipeline was created. It extracts data from the GDC, transforms the data files into Xena matrix files, a format that the Xena browser can ingest, and loads that data into the Xena browser. It ingests three main types of data: genomic data, clinical data, and survival data. The genomic data includes copy number, mutation, gene expression, DNA methylation, protein, and miRNA expression data.

Data in the GDC can be mapped to different entities, including the case (patient), the sample, and the aliquot. As data in Xena is always mapped to the sample, this requires remapping many data values, especially for the clinical data, which can be mapped to any of these entities. Additionally, the clinical data is prone to errors because the clinical data contains deeply nested dictionaries with information on treatments, diagnoses, pathology details, annotations, and more. These nested dictionaries need to be unpacked, resulting in a list with n entries, where each entry corresponds to one of the n treatments, diagnoses, or other data values. For consistency, data from different fields within each nested dictionary—for, say, a treatment—must be aligned at the same index across all lists.

This project aims to create automated testing code to ensure that the data generated by the Xena GDC ETL pipeline is of high quality. The ETL code and the code written in this project must have distinct data retrieval paths to help ensure that the data is verified through multiple angles. This code was built upon very preliminary code generated by a high school student over the summer quarter.

2 Methods

2.1 Ensuring Distinct Data Retrieval Paths Between ETL Code and QA Code

Building upon the preliminary code, I generated a new workflow for the genomic data testing code, with some slight modifications to the comparison method. This was done for two different reasons. The first is that the preliminary code did not use an orthogonal way to retrieve and test the validity of the data, which is important. The second is that this new workflow would first check that the proper samples were retrieved before downloading the corresponding files to save some time in case the samples were not equal.

First, the code parses the project name and Xena matrix path name from the command line. Then a pandas [6] data frame is created to represent the Xena matrix. Then, using the project name, all the samples associated with that project are retrieved using the GDC API from the /cases endpoint. Then another API request is made to get all the samples that have the necessary data (for example, miRNA data). Then another request is made to get all of the files that have the necessary data, filtering by the samples retrieved from

the previous request as well as filters to only get the wanted file types. At this point, we have retrieved all the samples from the GDC that contain the data that we want to compare, using a method independent of that of the Xena GDC ETL pipeline. A comparison is then made between the retrieved samples and the samples within the Xena matrix. If the samples do not match, then the testing code quits and outputs the difference in samples. Otherwise, the files are downloaded from the GDC using the /files endpoint. Then the comparison is done between the data retrieved from the API and the data in the Xena matrix, cell by cell. These same methods were used to implement all of the genomic data test code, with the comparison method varying from test to test.

For the survival data, we also need to ensure an orthogonal testing method. There are two pieces of information needed for the survival data to complete a Kaplan-Meier analysis: the date at which an event occurred to a patient, i.e., death or dropping out of the study, and the date at which it occurred. There was no orthogonal way to obtain the date at which an event occurred, so the same /analysis endpoint was used as the ETL code. For the event, the previous code was able to use an orthogonal method by detecting the event through the /cases endpoint using the field demographic.vital_status. However, when running the previous code, there were inconsistencies in the event retrieved from the QA and ETL code. This was because some of the vital_status fields would have the value "not reported" or "unknown." This could be due to various reasons, such as a participant did not follow up with the study or by the end of the study the event, in this case death, did not occur. In a Kaplan-Meier analysis, these cases are right-censored, meaning they appear in the visualization as a tick mark, thus contributing useful data up to the last moment of observation. To account for these cases in the QA code, if any value besides "Alive" or "Dead" is encountered, the code will query the /survival/analysis endpoint to retrieve the correct state.

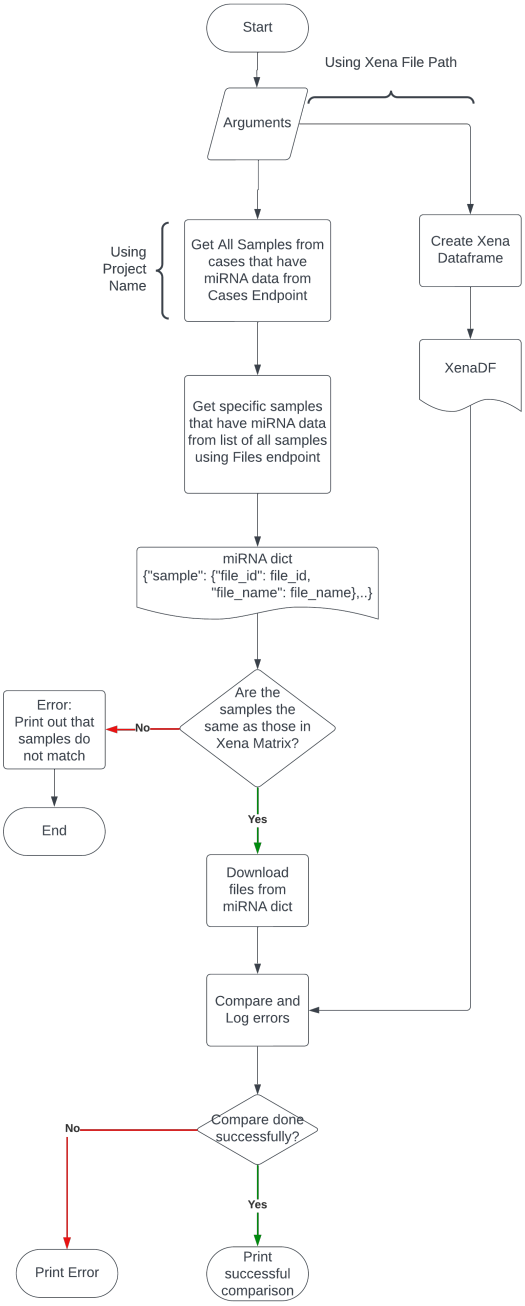

Fig. 1 A flowchart of the new workflow

For the clinical data, we similarly needed to ensure an orthogonal testing method. Initially, the testing code for the clinical data used the same /cases endpoint as the ETL code. We instead chose the /files endpoint to gather the clinical data for the testing code. To ensure that all of the available fields were retrieved from the /files endpoint that were needed for the clinical data, the /files/_mapping endpoint was used. This endpoint specifies all of the available fields present in the /files endpoint. Since each case has its own clinical data that is shared amongst all of its samples, we only gather fields that begin with "cases" since all clinical data is originally retrieved from the /cases endpoint. A predefined list of fields, as well as summary and follow-up data, is dropped from the API call as these fields are not needed by the Xena Browser.

After further inspection of the API response, it was found that some cases had 0 samples associated with them, although it was known from other API endpoints that there were samples in that case. These issues arise from how the /files endpoint is structured. The /files endpoint only returns samples that have a file associated with them. That means from the perspective of the /files endpoint, if a sample does not have a file associated with it, the /files endpoint does not have data on that sample. Additionally, the endpoint was less developed by the GDC than the /project or /cases endpoint and occasionally had missing values, such as cases.project.disease_type. While it was not ideal to use the /project or /cases endpoints for these issues since these are the same endpoints used by the ETL code, we decided to prioritize ensuring that all values underwent testing. Separate API calls were made to retrieve data from these endpoints, and the resulting data was mapped to the case ID and merged with the clinical data retrieved from the /files endpoint.

2.2 Genomic Data Testing Code

Several changes were made to the code to optimize it and increase its robustness. The original testing code would repeatedly download files for testing, even if the test already found that the samples from the GDC did not match the samples found in the Xena matrix. I implemented code to check this before downloading, saving considerable time. Additionally, the original gene expression testing code required the sample order to be the same between the GDC and the Xena matrix files generated by the GDC ETL code, which was frequently not the case. I fixed this using a dictionary that maps each sample to its corresponding file. For the segmented copy number data and mutation data, a different method was needed since there are multiple lines of data per sample, and in the case of CPTAC-3, if these samples are then mapped back to the case, this results in multiple samples having the identical case identifier despite the information coming from different files which are mapped to the sample. For these data types, it is not possible to iterate sample by sample since all the samples are mapped to the case. Instead, the data generated from the GDC and the Xena matrix are sorted and compared as a whole since it is not possible to compare them on a lower level of granularity due to the aforementioned problem. I optimized the code for the gene expression data so it

took less time to run a test. This was done by implementing a different rounding method, vectorizing the function when rounding each column from the retrieved data files, and comparing an entire column of data at once instead of each value separately. Finally, I changed the code so that when a test was rerun, it would not re-download data. A new method was implemented to check the MD5sums of existing files and verify that they match the MD5sums of the files retrieved from the GDC. The code then identifies the files that either do not exist in the or files that have been updated since they have been downloaded and only downloads a subset of all the files retrieved from the GDC, resulting in a faster runtime.

Much work was required to ensure that float numbers were properly compared between the QA and ETL testing codes due to rounding errors inherent in floating point numbers. The previous method of rounding float values used a function that rounded floats to 3 significant digits. However, this function was unreliable because after rounding the number, there could sometimes be imprecision in the rounding, leading to inexact values being returned.

For example:

```
>>> roundsf(0.0012315, 3)
    0.0012300000000000002
```

To keep precision in the code when comparing data from the GDC, I instead utilized NumPy's format_float_scientific function [5] using a precision of 8 places after the decimal point. This function avoids the aforementioned issue by performing string comparisons on numbers.

Another challenge I overcame was handling multiple values mapped to a single sample. In the preliminary code, when multiple values were associated with a single sample, only one value was taken. While having multiple values mapped to a single sample is rare, it does occur in several data types, including miRNA, Copy Number Segment, and Gene Level Copy Number data. To address this challenge, I implemented a function to accurately average values across multiple files associated with the same sample. The code is designed to process and aggregate data from these files, taking into account the presence of NaN (undefined) values. Simply summing the values and dividing by the number of files would be inaccurate when NaN values are present. To account for NaN values, the function keeps track of valid (non-NaN) values for each cell across the files. NaN values are temporarily replaced with the value 0 to avoid skewing the sum during aggregation. After summing, each value is divided by the count of non-NaN values, ensuring that the average is correctly calculated, even in the presence of missing data. If the count of non-NaN values for a cell is 0, the final value for that cell is set to NaN to maintain the integrity of the data by accurately representing missing information. This method ensures that the averaging process accurately reflects the available data, without being distorted by NaN values.

Lastly, I overcame the challenge of empty mutation files. Empty mutation files mean that though the sample underwent sequencing, and thus should be reported as not having a mutation, the sample instead was marked as having

never undergone sequencing and having no data. To ensure empty mutation files were correctly annotated, I marked these samples with a "no mutation" representation, which consists of having a genomic start position and end position as -1. This annotation will never render in the Xena visualization but will cause the browser to know sequencing was done. A further complication is that some samples had multiple mutation files, with some being empty and some normal files containing data. To account for samples having both empty and normal files, the no mutation representation of the sample was dropped only if there was at least one mutation file with valid mutation data. Otherwise, the no mutation representation of the sample was kept.

2.3 Clinical Data Testing Code

One of the first challenges I overcame was that the API calls for previous testing of the clinical data exceeded the length allowed by the GDC. Hence, I split the clinical testing data requests across two different API responses. Initially, data was merged by mapping all data to its submitter ID to merge data from the same case. This worked successfully on a case level of granularity. However, within each case dictionary, the data relating to diagnoses and treatments still needed to be properly merged. For example, treatment data for the same treatment was dispersed across different dictionaries, leading to fragmentation of the information. To merge this data back together, the treatment ID and diagnosis ID were added to the API call. This ensures accurate mapping and merging of all data related to specific treatments and diagnoses.

After the data is mapped correctly, it must be systematically unpacked so that fields related to various treatments and other categories are organized into lists where each entry across different fields corresponds to the same treatment, pathology detail, annotation, etc. I introduced a new recursive function to accomplish this complicated task. In the case that these fields must be unpacked into a list where each index in a list corresponds to a specific, say, treatment, these treatments would appear as a list of dictionaries in the response returned by the GDC API [1], where each dictionary represents a unique treatment. The code starts by identifying fields that are lists of dictionaries and recursively steps into them to unpack their contents. It will first loop over all the keys across each of the dictionaries in the list and compile a comprehensive list of keys across all the dictionaries. Then the code loops through all the dictionaries to retrieve each value associated with the keys in the comprehensive list and appends it to a new list. If a key does not have an associated value in a dictionary, an empty string is appended to the list to indicate that the specified entity does not have that data recorded.

The last challenge is that the Xena Browser is designed to integrate genomic and clinical data from the same sample. However, many samples in a project have only clinical data, or they have genomic data that is controlled access, or they only have slide data but no sequencing data. These samples do not have genomic data, so they are, by definition, not included in the genomic data files. These samples do need to be removed from the clinical and survival data. This

is done by checking all the samples to see if they have transcriptome, proteome, or methylation files associated with them; if they do, then those samples are kept since genomic data is associated with them.

3 Results

This project culminated in the creation of testing code for various different data types exported from the GDC to the Xena Browser. These include clinical testing code, survival testing code, protein testing code, miRNA data testing code which has support for both miRNA and miRNA isoform data, gene level copy number variation (CNV) testing code which has support for 4 different CNV calling workflows: ABSOLUTE, ASCAT2, ASCAT3, and AscatNGS, segmented copy number variation testing code which has support for masked segmented CNV DNAcopy data, allele-specific segmented CNV data for both ASCAT2 and ASCAT3, and unmasked segmented CNV DNAcopy data, gene expression testing code which has support for star_counts, star_tpm, star_fpkm, and star_fpkm-uq data, methylation testing code which has support for the platforms Illumina methylation epic, Illumina human methylation 27, Illumina human methylation 450, and Illumina methylation epic v2, somatic mutation testing code which has support for 2 different experimental strategies whole exome sequencing and targeted sequencing. All of the testing code utilizes a row or column comparison method to validate that the data is correct after it is run through the ETL pipeline. This saves a considerable amount of time compared to the previous test code, which compared each value cell by cell. This is especially important as many data types have an immense amount of data, especially the methylation data. All of the testing code was thoroughly tested against files that were known to not pass. For instance, I included erroneous data within the Xena matrix to test if the code would properly catch the errors. This was to ensure that the QA code functioned properly both when the Xena matrix was successfully extracted and transformed in the ETL pipeline, and when it encountered a failure.

An example usage statement is

```
>>> python3 script.py -t protein -p TCGA-ACC
```

This command tests the protein data type for the project TCGA-ACC. The script will search for the protein Xena matrix in the parent directory from where script.py is located. It will then run the testing code for the protein data and output its process to a logger file. This will include a check for existing files and how many files are up to date based on the md5sum of the existing file and the md5sum retrieved from the GDC API. It will then test sample by sample and output the result to the logger file. If any samples do not pass the testing, then the logger file will contain a list of these failed samples for easy access. The -p parameter can contain one or more projects for the tests to run on, and the -t parameter can contain 0 or more tests that will be run on the specified projects. In the case of no -t argument(s) being passed to the script, such as

```
>>> python3 script.py -p TCGA-ACC
```

The script will search the raw data directory for this project, list all the data types present, and run all of those tests on the listed projects, in this case, TCGA-ACC.

A logger file is generated each time the script is run named, `test_<year><month><day>-<hour><minute><second>.log`. This includes how many files are retrieved from the GDC, how many files currently exist in the testing directory, how many of those files are up to date according to the md5sum of the files in the directory and those retrieved from the GDC, how many files need to be downloaded, the testing of each sample and if it passed or not, and finally the final result of the test. If the test did not pass, then a list is compiled of all the samples that failed testing to collect all failed samples in a convenient place for further analysis. An example of this is

```
>>> python script.py -p TCGA-ACC -t protein
2024-09-04 21:42:33 - 46 files found from the GDC for protein
    data for TCGA-ACC
2024-09-04 21:42:33 - 0 files found at gdcFiles/TCGA-ACC/protein
2024-09-04 21:42:33 - 46 files needed to download
2024-09-04 21:42:33 - Downloading from GDC:
                      <Download Status>
2024-09-04 21:42:43 - Testing in progress ...
2024-09-04 21:42:43 - [1/46] Sample: TCGA-OR-A5LL-01A - Passed
2024-09-04 21:42:43 - [2/46] Sample: TCGA-OR-A5K8-01A - Passed
2024-09-04 21:42:43 - [3/46] Sample: TCGA-OR-A5LN-01A - Passed
2024-09-04 21:42:43 - [4/46] Sample: TCGA-PA-A5YG-01A - Passed
.
.
.
2024-09-04 21:42:43 - [43/46] Sample: TCGA-OR-A5KU-01A - Passed
2024-09-04 21:42:43 - [44/46] Sample: TCGA-OR-A5JW-01A - Passed
2024-09-04 21:42:43 - [45/46] Sample: TCGA-OR-A5JZ-01A - Passed
2024-09-04 21:42:43 - [46/46] Sample: TCGA-OR-A5J9-01A - Passed
2024-09-04 21:42:43 - [protein] test passed for [TCGA-ACC].
2024-09-04 21:42:43 - protein data for TCGA-ACC has PASSED.
```

4 Discussion

Upwards of 150 data types have been run and passed with the Xena beta release of newly loaded GDC data. This newly automated testing method saves a significant amount of time and ensures a very high data quality. This is a considerable improvement both over the previous testing code and also previous manual efforts to assure data quality. I also made several changes to the code to make it easy to run a whole project with one command and log any inconsistencies between the data retrieved from the GDC and the data

generated by the ETL pipeline. This testing code allows Xena to streamline the process of validating data produced by the ETL pipeline. This, in turn, enables useful cancer research data retrieved from the GDC to be in the hands of more people in a more timely manner for research and analysis. The code is available at https://github.com/ucscXena/Xena-GDC-auto-test/tree/main.

Acknowledgments

I would like to extend my deepest gratitude to my mentors at the UCSC Genomics Institute, Cally Lin, Jingchun Zhu, and Mary Goldman, for their extremely valuable insight and guidance throughout the development of this project. I am fortunate to have had the opportunity to work with and learn from such dedicated and accomplished individuals. I would also like to thank Cally for her close mentorship throughout this project and her assistance in developing the testing script that allows all the testing code to be run conveniently. I would also like to thank the CZI's Essential Open Source Software for Science Diversity and Inclusion Grants for the funding (CZI EOSS-DI-0000000026) to make this project and mentorship opportunity possible.

References

[1] Getting started - GDC Docs. (n.d.). https://docs.gdc.cancer.gov/API/Users_Guide/Getting_Started/

[2] Goldman, M.J., Craft, B., Hastie, M. et al. Visualizing and interpreting cancer genomics data via the Xena platform. Nat Biotechnol 38, 675–678 (2020). https://doi.org/10.1038/s41587-020-0546-8

[3] Human Cancer Models Initiative (HCMI). Human Cancer Models Initiative - NCI. (n.d.). https://www.cancer.gov/ccg/research/functional-genomics/hcmi

[4] Li, Y., Dou, Y., Da Veiga Leprevost, F., Geffen, Y., Calinawan, A. P., Aguet, F., Akiyama, Y., Anand, S., Birger, C., Cao, S., Chaudhary, R., Chilappagari, P., Cieslik, M., Colaprico, A., Zhou, D. C., Day, C., Domagalski, M. J., Esai Selvan, M., Fenyö, D., Foltz, S. M., ... Clinical Proteomic Tumor Analysis Consortium (2023). Proteogenomic data and resources for pan-cancer analysis. Cancer cell, 41(8), 1397–1406. https://doi.org/10.1016/j.ccell.2023.06.009

[5] Numpy.format_float_scientific. numpy.format_float_scientific - NumPy v2.1 Manual. (n.d.). https://numpy.org/doc/stable/reference/generated/numpy.format_float_scientific.html

[6] pandas documentation — pandas 2.2.2 documentation. (n.d.). https://pandas.pydata.org/docs/index.html

[7] The Cancer Genome Atlas Research Network., Weinstein, J., Collisson, E. et al. The Cancer Genome Atlas Pan-Cancer analysis project. Nat Genet 45, 1113–1120 (2013). https://doi.org/10.1038/ng.2764

[8] Zhang, Z., Hernandez, K., Savage, J. et al. Uniform genomic data analysis in the NCI Genomic Data Commons. Nat Commun 12, 1226 (2021). https://doi.org/10.1038/s41467-021-21254-9