CSCI 370 Final Report

Team Golden Rocks

Carla Ellefsen

Kira Hanson

Connor Sparks

Asa Sprow

Revised December 8th, 2023

CSCI 370, Fall 2023

Prof. Donna Bodeau

Table 1: Revision history

Revision	Date	Comments
New	09/03/23	Completed the initial draft of the requirements section
Rev – 2	09/15/23	Completed the “Team Profile” and “System Architecture” sections of the report
Rev – 3	10/13/23	Completed the “Software Test and Quality” section of the report
Rev – 4	11/10/23	Completed the “Project Ethical Considerations”, “Project Completion Status”, “Future Work”, and “Lessons Learned” sections of the report
Rev – 5	12/05/23	Completed the “Acknowledgements” sections of the report along with cleaning up the grammar and spelling
Rev – 6	12/08/23	Made final revisions to report based off of feedback from other groups

Table of Contents

I. Introduction

Computed Tomography (CT) scans are used extensively in the medical field to produce detailed, X-ray, cross-sectional images of hard and soft tissues inside the human body [1]. The properties that make CT scanning so indispensable in medicine—its ability to generate 3D views in a non-destructive manner and its high spatial and density resolution—also enhance its value in unrelated fields. Most notably for our project, in the geosciences for rock and sediment core analysis [2]. Rock and sediment cores can be extremely valuable for geologists and engineers as they present a relatively easy way to gain representative information about subsurface rock properties without having to move large amounts of rock or sediment. Rock core drilling is often a crucial step in many engineering applications including petroleum exploration (gauging whether the formation is favorable for oil or assessing an oil well's productivity)[3] and construction (determining rock quality, hardness, etc.) [4]. Sediment cores, often taken from lake and ocean beds, are particularly useful for researchers interested in past climatic conditions, as they can contain trapped gasses representative of the atmosphere at the time along with organic material such as pollen or other plant matter [5].

Although a wide variety of software tools have been developed to view and analyze medical CT scans, there are no effective tools specifically designed for the needs of geologists in analyzing the properties of rock and sediment core CT scans. Our clients, Dr. Zane Jobe and Dr. Ryan Venturelli in the Geology and Geological Engineering Department at the Colorado School of Mines, have recognized such a need through their own research. Dr. Zane Jobe is the director of the Chevron Center of Research Excellence (CoRE), an industry-academic partnership conducting world-class research on challenges facing the energy sector. CoRE uses core sampling among other techniques to analyze stratigraphic architecture and understand submarine environments [6]. Dr. Ryan Venturelli heads the Rates and Dates (RaD) Laboratory at Mines, a team of paleo-glaciologists and isotope geochemists studying the pre-observational record of glaciers [7]. In conjunction with the Subglacial Antarctic Scientific Access Team (SALSA), Dr. Venturelli collected sediment core data from lake beds in glacial regions to aid in understanding present climate changes through understanding past climate changes [8]. To aid them and other geologists in their research, Dr. Zane Jobe and Dr. Ryan Venturelli asked our team to build a flexible, extensible, and easy-to-use suite of software tools with Python to aid in the visualization and analysis of rock core CT scans.

II. Requirements

A. Functional Requirements

1. Project Structure

Our project took inspiration from a couple of past software tools as a jumping-off point. As part of her ongoing research, our client Dr. Venturelli has written specialized Python code to assist her in analyzing sediment core data taken from Lake Mercer with the SALSA team. Additionally, SedCT, a MATLAB-based application developed at Oregon State University, does provide an amount of functionality for analyzing core data [9]. However, the SedCT MATLAB code is in a single file with thousands of uncommented lines, making it hard to decipher and add to. As such, our clients wanted to begin migrating their software suite to Python in the pursuit of extensibility and maintainability. By building our software suite in Python we will be able to provide our clients with code that is well documented and structured, allowing them to easily use and add to the library. This will remove the need for individual researchers to write large amounts of code to analyze their core data. Following from the software they have already written, the technical requirements of our project are as follows:

- Usable within a Jupyter Notebook environment
- Minimal command line interaction necessary to run the software
- Ability to easily import core scans in the DICOM file format

2. Visualization Tools

The majority of rock core analysis is conducted visually. As such, our clients requested that our software contain methods that allow them to generate visualizations of the scanned core. The specific visualization features requested by our clients are listed below:

- Produce visualizations for both horizontal and vertical slices of a core
- Generate a 3D representation of the core that supports rotation and zooming in/out
- Create a brightness trace of a specific slice of a core
- Filter sections of the core by brightness

Out of these different requirements, the particular visualization that our clients are interested in is the brightness trace. This type of visualization consists of a picture depicting a slice of the core next to graphs displaying the average brightness and standard deviation for each layer of the code. An example of a brightness trace can be seen below in Figure 1.

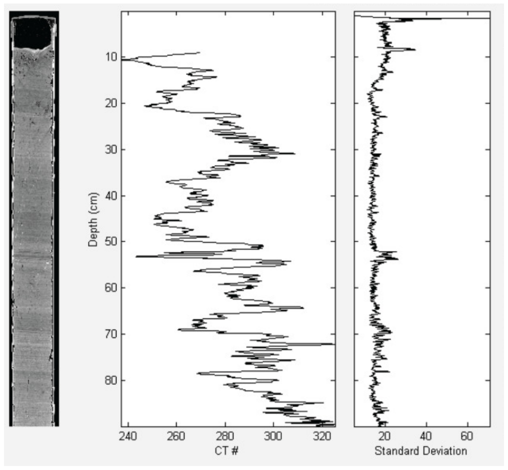


Figure 1: Brightness trace taken from SedCT

3. Mathematical Analysis

In addition to assisting in the visual analysis of rock cores, our clients also wanted our software to provide more quantitative data about the rock core scans that it processes. Specifically, our clients were interested in information regarding the density of different areas of the core. To help them access this information they requested that our software be able to perform the following operations:

- Compare the densities of different, specifiable core sections
- Calculate the mass-density of a core through the conversion of Hounsfield units (HU)
- Change the units of distance and mass the visualizations are displayed in

B. Non-Functional Requirements

1. Code Quality/Extensibility

One of the non-functional requirements specified by our client was that our code should be easily extensible, expandable, and maintainable by people who don't specialize in software development. Unlike the SedCT MATLAB code they were previously using, they asked that our code be well-structured and thoroughly documented so that any geologist hoping to analyze rock core CT scans can easily understand and use our software.

2. Low Barrier to Entry/Ease of Use

As touched upon in the previous section, another non-functional requirement for our project was that our code should be easy to use by geologists who are not comfortable writing code or using the command line. Our clients are hoping to reduce the barrier to entry as much as possible so more researchers start analyzing CT scans. As such, they asked that our software be easy to set up and usable with minimal programming knowledge

III. Risks

A. Technical Risks

Due to the lack of existing software that we needed to integrate with, our project came with minimal technical risks. The only real issues that our project had was the lack of standardization around how the CT scans of rock and sediment cores are organized and stored. Knowing that, we defined our project's technical risks to be as follows:

- There might not be any libraries available to process the file types we are given.
- Files might have different metadata associated with them, making it difficult to standardize our software.
- Core scans might not possess enough information for us to properly analyze them.

B. Skill Risks

The skill risks associated with this project mainly revolved around our team's lack of experience with the field of geology and working with CT scans. As such, we identified our main skill risks to be as listed below:

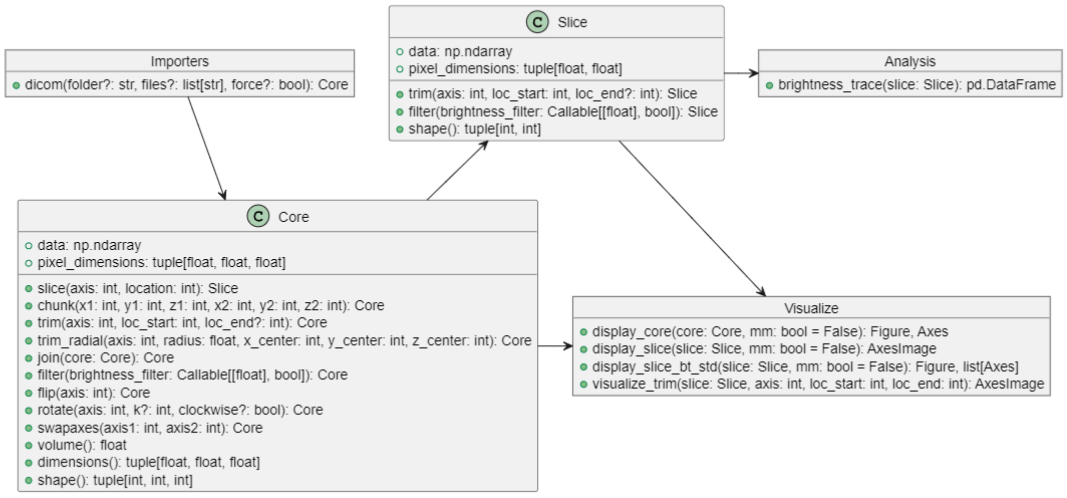
- None of our team members have worked with CT scans before
- General lack of geological knowledge among our group
- Only minor experience with Python visualization tools within our team

IV. Definition of Done

Working together with our client, we decided that our project would be considered complete once we had created a Python library capable of reading in DICOM files containing rock or sediment core data and producing useful visualizations of that data. The library should have an intuitive system that allows any user to easily load and display a core. Additionally, when a core is visualized, the visualization should use colors to represent its different density levels and it should display a graph that shows the average density of each layer of the core. Finally, the library should allow for parts of the cores that are not a specified density to be filtered out.

V. System Architecture

To fulfill the requirements we were given, we decided that the final deliverable for this project should be an open-source Python library, as this would make it easy for our clients to download and use our code with Jupyter Notebooks. To help us in creating this library, we constructed a Unified Modeling Language (UML) diagram (Figure 2) outlining the classes and methods that we created. As can be seen in the UML, we developed multiple classes and modules to assist in the analysis of CT scans. The diagram along with descriptions of these classes are shown below.



Importers
dicom(folder?: str, files?: list[str], force?: bool): Core
C Slice
data: np.ndarray
pixel_dimensions: tuple[float, float]
trim(axis: int, loc_start: int, loc_end?: int): Slice
filter(brightness_filter: Callable[[float], bool]): Slice
shape(): tuple[int, int]
Analysis
brightness_trace(slice: Slice): pd.DataFrame
C Core
data: np.ndarray
pixel_dimensions: tuple[float, float, float]
slice(axis: int, location: int): Slice
chunk(x1: int, y1: int, z1: int, x2: int, y2: int, z2: int): Core
trim(axis: int, loc_start: int, loc_end?: int): Core
trim_radial(axis: int, radius: float, x_center: int, y_center: int, z_center: int): Core
join(core: Core): Core
filter(brightness_filter: Callable[[float], bool]): Core
flip(axis: int): Core
rotate(axis: int, k?: int, clockwise?: bool): Core
swapaxes(axis1: int, axis2: int): Core
volume(): float
dimensions(): tuple[float, float, float]
shape(): tuple[int, int, int]
Visualize
display_core(core: Core, mm: bool = False): Figure, Axes
display_slice(slice: Slice, mm: bool = False): AxesImage
display_slice_bt_std(slice: Slice, mm: bool = False): Figure, list[Axes]
visualize_trim(slice: Slice, axis: int, loc_start: int, loc_end: int): AxesImage

Figure 2: UML diagram of our Python library

1. Importer — Handles the importing of DICOM and IBF files into a Juypter Notebook
    a. dicom() — Imports a scan saved in the DICOM file format and returns a Core object
2. Core — An abstraction of a scanned rock core with various methods to assist in analysis
    a. data — A 3D numpy array representing the scanned core
    b. pixel_dimensions — A list containing the dimensions for each pixel
    c. slice() — Returns a new Slice object representing a slice of the core
    d. chunk() — Returns a new Core object representing a subsection of the core scan
    e. trim() — Returns a new Core object with a portion of the scan data from a specified axis
    f. trim_radial() — Returns a new Core with portions of the scan data removed in a radial fashion
    g. join() — Returns a new Core with the combined data of two Core objects
    h. filter() — Returns a new Core with values filtered out by a provided lambda function
    i. flip() — Returns a new Core with the scan data flipped along a given axis
    j. rotate() — Returns a new Core with the scan data rotated about a given axis
    k. swapaxes() — Returns a new Core containing scan data where the given axes have been swapped
    l. volume() — Returns the approximate volume of the core
    m. dimensions() — Returns dimensions of the data array
    n. shape() — Returns the dimensions of the core
3. Slice — An abstraction of a rock core slice with various methods to assist in analysis
    a. data — A 2D numpy array representing the slice
    b. pixel_dimensions — A list containing the dimensions for each pixel
    c. trim() — Returns a new trimmed Slice
    d. filter() — Returns a new Slice with values filtered out by a provided lambda function
    e. shape() — Returns the dimensions of the slice
4. Analyze — Contains methods that can be used to perform quantitative analysis on a scanned core
    a. brightness_trace() — Returns a vector containing the average brightness and standard error for each layer of a slice
5. Visualize — Allows a user to easily visualize a core and any associated data
    a. display_core() — Creates a 3D visualization of a core
    b. display_slice() — Visualizes a slice of a core
    c. display_slice_bt_std() — Constructs a graph displaying a brightness trace of a Slice
    d. visualize_trim() — Creates a graph that visualizes where you are going to trim the Core

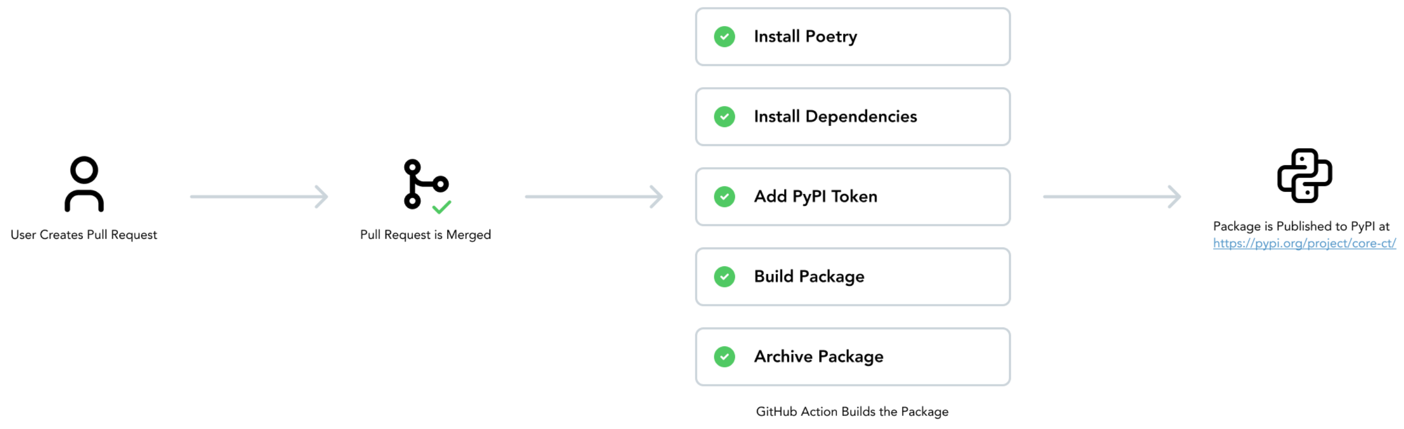
VI. Technical Design

As discussed in the introduction of this report, our clients are geologists, and as such, they come from a limited computer science background. With this in mind, our group wanted to create systems that would reduce the amount of technical work that our clients would need to perform once we handed off the project. To achieve this, we developed automated pipelines to handle publishing updates and generating documentation for our package.

A. Automatic PyPI Deployment

We created a GitHub Action responsible for automatically building and publishing our package. GitHub Actions are CI/CD tools used to automatically execute code when certain events happen—in this case, when a branch is merged into main.

In order to successfully publish our code to PyPI (Python Package Index), our GitHub Action runs through a couple of different steps (Figure 3). First, the action installs Poetry, which is the Python package manager we used while developing our library. Once Poetry is installed, the action then installs the dependencies that the project needs to be built into a package. With this complete, the action uses GitHub's built-in secrets manager to get access to the token it needs to publish the package to PyPI. Finally, the pipeline builds the package, saves a copy of that build to the repository, and publishes it to PyPI.



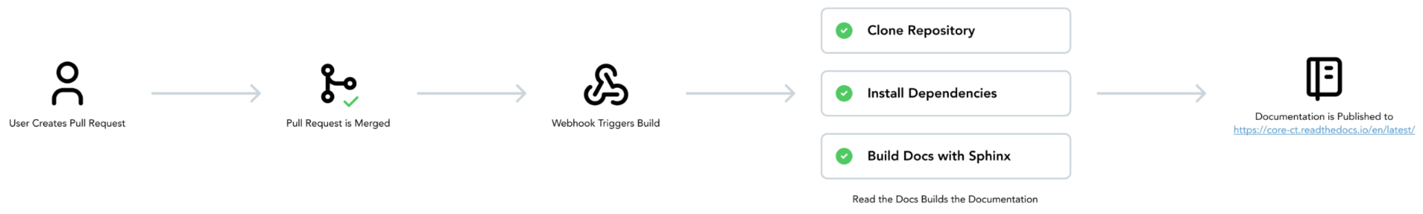
User Creates Pull Request
Pull Request is Merged
Install Poetry
Install Dependencies
Add PyPI Token
Build Package
Archive Package
GitHub Action Builds the Package
Package is Published to PyPI at
https://pypi.org/project/core-ct/

Figure 3: PyPI deployment pipeline

B. Building Documentation Automatically

Similar to the publishing of our Python package, we didn't want our clients to be responsible for maintaining a documentation website. To solve this problem, we once again created an automated process that automatically builds and deploys documentation to make sure information is always up-to-date and easily accessible.

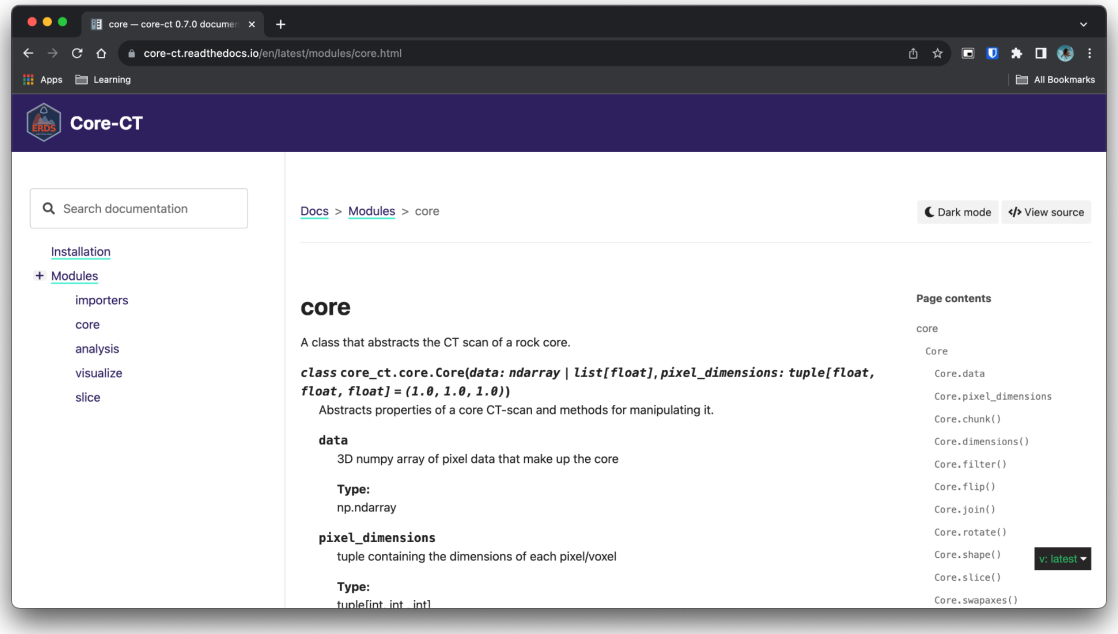
To create this pipeline we ended up using a couple of different technologies (Figure 4). In particular we used Read the Docs, Sphinx, and webhooks. Read the Docs is a website that allows users to freely build and host documentation for their Python projects. To build the documentation for these websites, Read the Docs uses Sphinx, which is a Python package that can build a properly formatted documentation website with the user only needing to configure a couple settings. Our group ended up configuring Sphinx to use an extension called autodoc as it allowed for Sphinx to automatically create documentation for our codebase using the docstrings on the modules and methods we wrote. The final part of getting this all to work was getting Read the Docs to automatically build and deploy the site, for which we used webhooks. Essentially, we gave Read the Docs a URL to watch, and whenever a commit was pushed GitHub would send an alert to this URL, telling Read the Docs that there was a new commit. Read the Docs would then make sure the commit was pushed to our main branch, and if it was, it would automatically rebuild the documentation so it contained any changes that were made to the codebase.



User Creates Pull Request
Pull Request is Merged
Webhook Triggers Build
Clone Repository
Install Dependencies
Build Docs with Sphinx
Read the Docs Builds the Documentation
Documentation is Published to
https://core-ct.readthedocs.io/en/latest/

Figure 4: Documentation deployment pipeline

This site can be accessed at https://core-ct.readthedocs.io/en/latest/ and a screenshot of our documentation website is shown below (Figure 5).



core

A class that abstracts the CT scan of a rock core.

class **core_ct.core.Core**(*data: ndarray | list[float]*, *pixel_dimensions: tuple[float, float, float] = (1.0, 1.0, 1.0)*)

Abstracts properties of a core CT-scan and methods for manipulating it.

data

3D numpy array of pixel data that make up the core

Type:
np.ndarray

pixel_dimensions

tuple containing the dimensions of each pixel/voxel

Type:
tuple[int, int, int]

Figure 5: Screenshot of our documentation website

VII. Software Test and Quality

As defined in previous sections of this report, the goal of this project was to build an open-source and publicly accessible Python library. Due to its open-source nature, additional contributions will likely be made to our codebase even after field session has concluded. With this in mind, we wanted our project to have systems in place that would ensure a consistent level of readability and quality throughout our code.

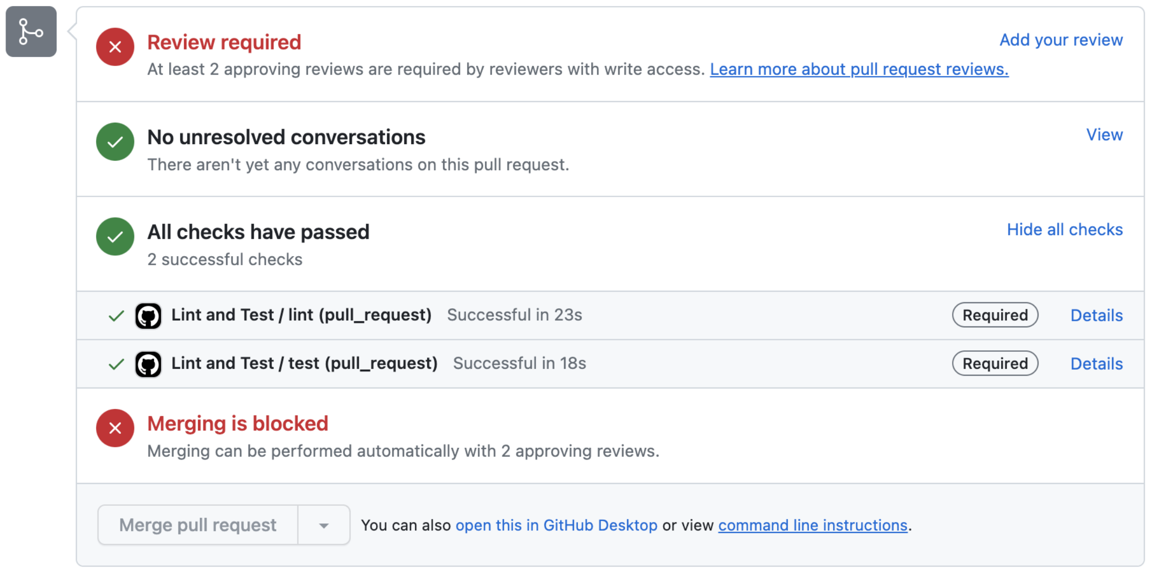
To bring this vision to fruition, we focused our efforts on two main areas: functionality and readability. In regards to project functionality, we created a testing framework for our code, and in regard to readability, we implemented both linting tools and code reviews to ensure that the code we produced was well-written and easily readable.

A. Testing

As mentioned above, we built a testing framework into our project that verifies all our code is working properly at any given time. The backbone of this framework is unit tests. For every function we wrote a corresponding set of unit tests was developed. This ensured that we didn't introduce bugs or regressions as we expanded upon our project.

Inside our repository, all of these tests were stored within the tests directory, and inside this directory, each library module has its own suite of tests. Each module test is stored in a file prefixed with "test_" and suffixed with the name of the module being tested. Each suite of tests was responsible for running testing all the methods inside a module. As such, anytime a new method was created, a corresponding unit test needed to be created within that module's testing file. These tests could then be run using the pytest Python library.

In addition to this, we implemented a GitHub Action that ran all of the tests whenever a pull request was made to the main branch. In order for said pull request to get merged, all of the unit tests needed to pass. This ensured that no broken code was merged into production. This behavior is shown below in Figure 6.


Review required
At least 2 approving reviews are required by reviewers with write access. Learn more about pull request reviews.
Add your review

No unresolved conversations
There aren't yet any conversations on this pull request.
View

All checks have passed
2 successful checks
Hide all checks

✓ Lint and Test / lint (pull_request) Successful in 23s Required Details

✓ Lint and Test / test (pull_request) Successful in 18s Required Details

Merging is blocked
Merging can be performed automatically with 2 approving reviews.

Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

Figure 6: Screenshot of the GitHub Action verification on a pull request

B. Linting

To enforce code readability throughout our project we used linting which is the practice of statically analyzing a codebase to find programmatic and syntactic errors. To add this functionality into our project, our group chose to use `ruff` (a Python linter built in Rust) to ensure stylistic consistency within our code. Made as a replacement for older Python linters, Ruff has the ability to mimic the linting behavior of other linters through flags specified in the project's `pyproject.toml` file.

Through the use of these flags, our group enabled the following linting behaviors in our project: `pycodestyle`, `pyflakes`, and `pydocstring`. The first two of these, `pycodestyle` and `pyflakes`, were used to ensure basic code cleanliness and functionality—that is, checking things such as our code indentation and spacing. The addition of the `pydocstring` flag was to make sure that we documented all of our modules, classes, and methods through proper Python comments.

Finally, to confirm that all these linting rules were followed, we set up a GitHub Action to run a linting check on the codebase everytime a pull request to the main branch was made (Figure 6). If the linting check failed, the pull request would not be allowed to merge until all errors were resolved. This ensured that all of our production code was properly styled. To help developers check that their code meets the linting standards expected by Ruff, our repository contained a pre-commit hook that can prevent code from being committed if it does not pass the linting check; thus ensuring that no developers were surprised by a failing GitHub Action when they created a pull request.

C. Reviewing

The final pillar of our quality assurance framework was code reviews. Following traditional Agile procedures, we required that any branch getting merged into our production branch got approval from at least two other members of our team. This ensured that any code getting pushed to the main branch has been manually reviewed for errors and is agreed upon by our group to meet the high standard that we have set for ourselves.

Importantly, these code reviews were enforced through GitHub branch rules. Not only were two approvals from other team members required before merging, but we also had rules set up to make sure that no code got pushed directly to the master branch and that all comments on a pull request were addressed before merging.

VIII. Project Ethical Considerations

For our project, we identified two major categories of ethical considerations: data integrity and environmental impact. Because our library will primarily be used in a research setting, we are particularly concerned with ensuring the highest quality of data analysis and acknowledging the larger-scale ethical implications of the research we are contributing to.

Accurately analyzing and displaying data is paramount to our project. The researchers who will be using our program to load and manipulate CT scans expect our program to correctly display a core, the brightness trace, and compute the standard deviation of the brightness. Any errors in our program endanger the credibility of those who use it, and researchers run the risk of having their papers rejected from publication if the error is caught. However, if a paper using erroneous data generated by our program is accepted for publication, inaccurate information may be propagated as other researchers cite that paper and build upon inaccurate information presented in the literature. Therefore, we have an obligation to ensure to the best of our ability that we are accurately portraying and analyzing the CT scans to protect academic integrity and contribute in a positive manner to geoscientific research. To help prevent any possible errors, we adhered to a strict software quality and testing plan as outlined above. This dedication to software quality helps ensure any bugs are caught before they are deployed.

The second ethical consideration that is especially pertinent to our project concerns the research that we are contributing to. Although as software developers our work does not have any immediate physical impact on the environment, we are affecting it through the type of research we are participating in. Our client Dr. Jobe is extensively involved with Chevron, a major oil and gas company, and rock core drilling is often done as a step in petroleum exploration. The negative impacts of oil extraction and our continued reliance on fossil fuels on the planet are well understood. The industry causes various forms of pollution, including water pollution through oil spills and hydraulic fracturing [10], noise pollution from seismic surveys done for offshore petroleum exploration [11], and air pollution from the burning of fossil fuels. Because our client works with Chevron, we must reflect upon our contributions to the oil and gas industry and recognize that we are working in partnership with a particularly unsustainable industry. However, it is also worth noting that our tool can also be used in various ways, and some of those may also be beneficial to the planet. Our other client, Dr. Ryan Venturelli, has used sediment core data gathered from subglacial Antarctic lakes to better understand climate change in the past. Understanding how the polar regions have changed in the past may provide valuable insight into the current climate change crisis [8]. The long short is that our tool can be used in a variety of different settings, and we acknowledge that we are contributing to whatever impacts come of the research we have helped enable. On our project's scale, it is unlikely that we will be the cause of any specific harm or benefit, but we still have a part to play.

IX. Results

When our group initially began this project, our goal was to create an open-source Python library that would provide geologists with a set of tools specifically tailored to assist them in the analysis of rock core CT scans. While the final state of our project isn't a perfect reflection of our initial vision, we can unequivocally say that we achieved our initial goal. As outlined in the functional requirements portion of this document, our software had three sets of requirements that it had to meet: project structure, visualization tools, and mathematical analysis. So, to show that we meet all the initial goals of our project, the rest of the section will explore our group's success in meeting each set of requirements.

A. Project Structure

In regards to the structure of our project, our clients had three main requests: it needed to be usable within a Jupyter Notebook environment, it should require minor command line interaction, and they should be easily able to import core scans in the DICOM file format. Ultimately, we were able to meet each of these requirements.

To make our project usable inside of a Jupyter Notebook, the final deliverable of our project ended up being an open-source Python library published to the Python Package Index or PyPI available at https://pypi.org/project/core-ct/. To use our library, all one has to do is install the package using the command `pip install core-ct`, and import the modules that they want to use into their Jupyter Notebook. Since the installation of our library is entirely handled by the `pip` package manager, it means that any installation of the package involves minor command-line interaction as requested. Though users do still have to interact with the command line to run `pip` and install the package, this amount of command line usage is standard for using Python and doesn't ask our clients to perform any actions that they are unfamiliar with. Finally, to meet the last requirement of making it easy for our clients to import DICOM files, we built out a module called `importers` inside of our Python library that allows for a rock core scan to be loaded into a Python environment via a single function call, `importers.dicom`, and passing it the location of the folder that contains the scan.

B. Visualization Tools

Through our library's `core`, `analysis`, and `visualize` modules we were able to implement functionality for three out of the four requirements that our clients originally presented us with: produce visualizations for slices of the core, create a brightness trace for a slice of the core, and filter out sections of the core by brightness. The only requirement we didn't meet was adding the ability to visualize a core in 3D, as after talking with our client we decided to focus our efforts on geology specific functionality (such as creating a brightness trace).

To allow our clients to visualize a slice of the core, we created a method in our visualize module called `display_slice` (Figure 7). This method takes a Slice object as input and outputs a matplotlib figure that shows the slice. Since slices can be across any of the core's three axes, it can create the horizontal and vertical visualizations our clients asked for. Additionally, we created a `display_core` method that takes a Core object as input and simultaneously shows slices across the core's x, y, and z axes (Figure 8).

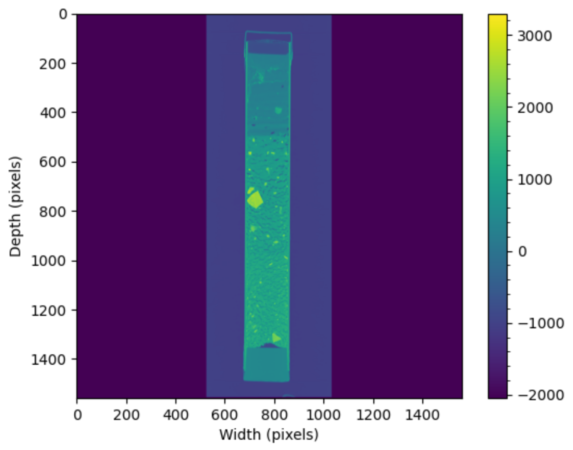


Figure 7: The display_slice result

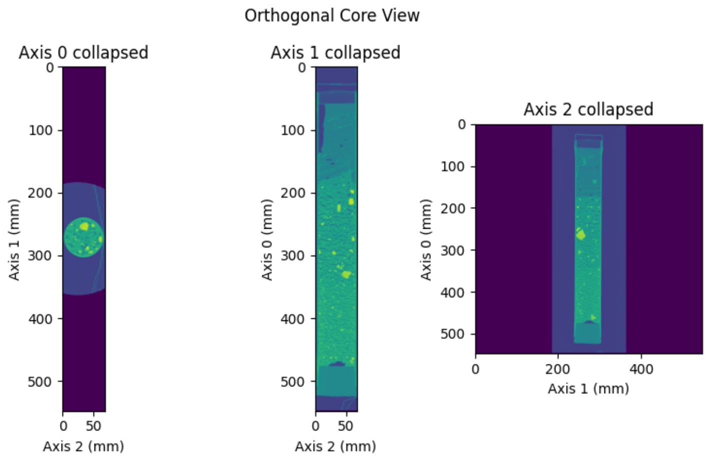


Figure 8: The display_core result

To assist in creating a brightness trace for the core we wrote methods in both the analysis and visualize modules. If a user only wants to visualize a brightness trace, shown below in Figure 9, a Slice object can be passed to the `display_slice_bt_std` method on the visualize class. However, if a user wants to get the data associated with the brightness trace, the `analysis` module contains a method called `brightness_trace` for that purpose. When passed a `Slice` object, `analysis.brightness_trace` returns a Pandas dataframe containing two columns: the first one contains the mean brightness across the rows of the slice and the second one contains the standard deviation of each row.

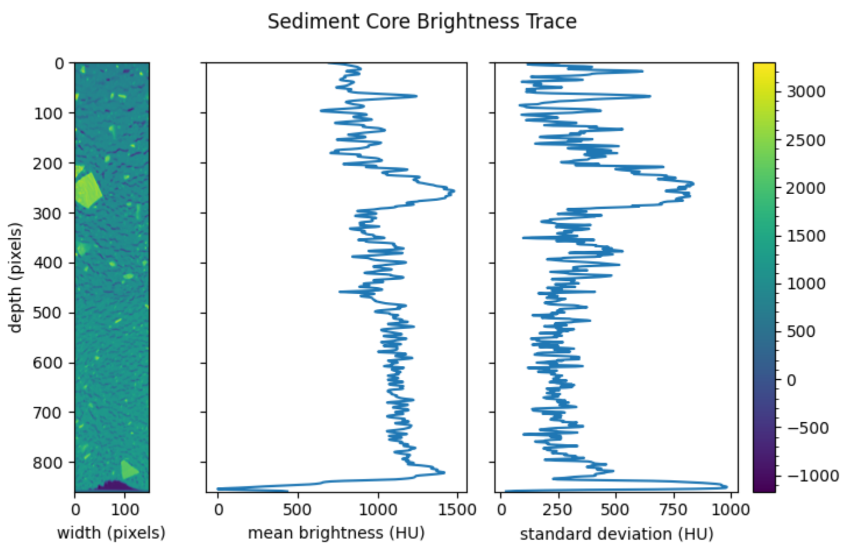


Figure 9: The brightness_trace result

Finally, to allow our client to remove values from a brightness trace, we created a `filter` method in the `Core` class that can be passed a boolean lambda function and then returns a new Core object with nullified values where the previous brightness values didn't pass the lambda function (Figure 10).

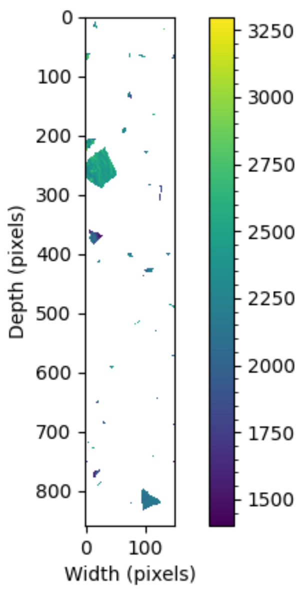


Figure 10: The filter result

C. Mathematical Analysis

In regards to mathematical analysis, we were ultimately able to meet two of the three requirements our clients had: compare the densities of different core sections and specify the units of distance/mass the visualizations are displayed in. The final requirement, which involved calculating the mass-density of a core through the conversion of Hounsfield units (HU), we weren't able to meet since there currently aren't mathematical methods available for performing this calculation.

To compare the densities of different core sections our clients can make use of a couple of different methods across our various modules. The `filter` method (Figure 10) on the `Core` class allows our clients to remove values from a core that they don't want to analyze, making it easier to compare relevant parts of cores to each other. Additionally, the `trim` and `chunk` methods can be used on a `Core` object to get subsections of the core so users of our library can more easily compare sections of cores to each other. Finally, to fulfill the second requirement, each method in the visualization module can take parameters to change the units that the visualizations are displayed in.

X. Future Work

While we were able to meet the initial set of requirements given to us by our client, there are still a handful of features that could be implemented within our library. As such, if this project is to be resumed in the future, that work could take the form of any of the features listed below. By adding these features to our project we would enable our clients to use our software on additional cores, more intuitively visualize important core data, and perform more complex analysis.

- Add support for other file formats such as IBF and TIFF
- Add interactivity to visualization methods
    - Users can use sliders on figures to dynamically trim sections of the core
    - Users can use sliders on figures to dynamically filter out ranges of density values
- Create methods for performing algorithmic auto-segmentation of features within a rock core
- Create methods for rendering the core in 3D

Compared to the current set of features built into our project, it is important to note that implementing the features listed above would present our group with some new challenges. For example, the IBF file format isn't open-source, meaning that we would need to work with the company who made the IBF file format in order for our program to successfully parse it. Other features such as performing auto-segmentation or rendering the core in 3D could prove potentially difficult as they would require us to work with new tools such as Meta's "Segment Anything Model" or PyVista. However, even though these features do pose potential challenges, we are confident that if we continued work on this project we could successfully implement most or all of these features.

XI. Lessons Learned

A. Setting Standards Early

We waited until about halfway through the semester to implement linting and automated tests. This resulted in us having to go back through our codebase and spend a bunch of time working on the same code twice. When we started linting we ended up having to go back through the entire codebase to make sure it was up to our standards. This could have easily been avoided if we were linting from the start. We also ended up with low test coverage in a couple of places since some code was merged before we started writing unit tests, which meant we had to go back and do them later. If we were to do this in the future, we would get our software quality framework up earlier to avoid these issues.

B. Using Correct Data Types

As we got deeper into our project, we quickly learned that our initial choice of data type for some of our variables was less than ideal. This was particularly apparent in our Core and Slice classes where we stored pixel dimensions as a Python list. In Python, lists are passed by reference, so if a user made edits to that list it would persist across instances of the Core throughout the code. Since our library was written to run sequentially in a Jupyter Notebook, it meant that if a user accidentally changed a core's pixel area, the results of any other cells using that Core object would now be changed, potentially leading to incorrect results. We ultimately fixed this by storing pixel dimensions as a tuple instead because tuples are immutable. While this is just a single example, problems like these taught us to think more critically about how we want to structure our data, and in the future we will make sure to consider inherent properties of data structures—i.e. mutability, runtime, and space requirements—when deciding on the architecture for our code.

C. Avoiding Premature Optimization

When we initially designed the Core class, we wrote the transformation functions (trim, chunk, rotate, etc.) to be in place i.e. it would edit the object you called the function on. We did this because we thought we could save memory and improve performance (avoid reallocating and copying the entire data matrix) by doing these operations in place instead of returning a new Core with transformed data. This effort ended up being futile because it created a worse experience for users. In particular, doing transformations in place was a headache if you tried to use the library in a Notebook. Rerunning a cell that transformed a Core would perform that transformation again, so you could very easily end up accidentally trimming a core too many times, rotating it differently, flipping it multiple times, etc.. We have since changed all transformation functions to return completely new Core objects, which is a much more reliable and error resistant solution.

XII. Acknowledgments

We would like to thank our wonderful clients, Dr. Zane Jobe (left) and Dr. Ryan Venturelli (middle) for their support and guidance throughout this project. Our success would not have been possible without their willingness to share their geologic expertise with us. We would also like to express our gratitude to our advisor Donna Bodeau (right) for helping us navigate field session through her valuable inputs to our project.

XIII. Team Profile

A. Carla Ellefsen

Carla Ellefsen is a senior studying Computer Science with a data science focus. She has been involved in a variety of research including Multi-Instance Learning (MIL) for image segmentation, intelligent traffic prediction, and protein structure-function prediction using neural networks. She currently works as the lead teaching assistant for discrete mathematics. Outside of school she is passionate about social justice, art, and the natural world.

B. Kira Hanson

Kira Hanson is a junior studying Computer Science. She has been working for four years at Lockheed Martin as an intern doing development work. From that she has learned HTML, CSS, Javascript, and UI5 development. In her free time she enjoys reading, creative writing, programming, and flying hot air balloons.

C. Connor Sparks

Connor Sparks is currently a senior studying to get his B.S. in Computer Science. Primarily experienced in front-end development, Connor has built out web applications in Angular, Vue, and React while also dabbling with other technologies such as Go, C++, and TensorFlow. Outside of school he likes to rock climb and is also a part of the Mines' swing dancing team.

D. Asa Sprow

Asa Sprow is a junior in Computer Science. He's been working at Sherpa 6 Inc. as a software engineering intern since May 2023, continuing into the school year. In his time there he has learned full-stack web development (Angular and Go), database design (SQL), and PDF parsing (Python) while working in a SCRUM team. In his free time he likes developing games, reading, and computers. He's recently put together a home server using secondhand parts and has been enjoying learning about server management/hosting.

XIV. References

[1] “Computed Tomography (CT) Scan.” Accessed: Nov. 10, 2023. [Online]. Available: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/computed-tomography-ct-scan

[2] G. van Kaick and S. Delorme, “Computed tomography in various fields outside medicine,” *Eur Radiol*, vol. 15 Suppl 4, pp. D74-81, Nov. 2005, doi: 10.1007/s10406-005-0138-1.

[3] “What Is Oil Well Coring?,” Sciencing. Accessed: Dec. 05, 2023. [Online]. Available: https://sciencing.com/info-7872027-oil-well-coring.html

[4] “Rock Coring | Geotechnical Investigations,” Earth Engineering Incorporated. Accessed: Dec. 05, 2023. [Online]. Available: https://earthengineering.com/geotechnical-investigations/rock-coring/

[5] “Ice and Sediment Cores | Ice Stories: Dispatches From Polar Scientists.” Accessed: Dec. 05, 2023. [Online]. Available: http://icestories.exploratorium.edu/dispatches/big-ideas/ice-and-sediment-cores/index.html

[6] “CoRE Home,” Chevron Center of Research Excellence. Accessed: Dec. 08, 2023. [Online]. Available: https://core.mines.edu/

[7] “Rates and Dates Laboratory Home,” Ryan A. Venturelli. Accessed: Dec. 08, 2023. [Online]. Available: https://www.mines.edu/rad/

[8] J. C. Priscu, J. Kalin, J. Winans, T. Campbell, M. R. Siegfried, M. Skidmore, J. E. Dore, A. Leventer, D. M. Harwood, D. Duling, R. Zook, J. Burnett, D. Gibson, E. Krula, A. Mironov, J. McManis, G. Roberts, B. E. Rosenheim, B. C. Christner, K. Kasic, H. A. Fricker, W. B. Lyons, J. Barker, M. Bowling, B. Collins, C. Davis, A. Gagnon, C. Gardner, C. Gustafson, O.-S. Kim, W. Li, A. Michaud, M. O. Patterson, M. Tranter, R. Venturelli, T. Vick-Majors, and C. Elsworth, “Scientific access into Mercer Subglacial Lake: scientific objectives, drilling operations and initial observations,” *Annals of Glaciology*, vol. 62, no. 85-86, pp. 340–352, 2021.

[9] B. T. Reilly, J. S. Stoner, and J. Wiest, “SedCT: MATLAB^{TM} tools for standardized and quantitative processing of sediment core computed tomography (CT) data collected using a medical CT scanner,” *Geochemistry, Geophysics, Geosystems*, vol. 18, no. 8, pp. 3231–3240, Aug. 2017, doi: 10.1002/2017GC006884.

[10] “Oil and the environment - U.S. Energy Information Administration (EIA).” Accessed: Nov. 10, 2023. [Online]. Available: https://www.eia.gov/energyexplained/oil-and-petroleum-products/oil-and-the-environment.php

[11] G. Prideaux and M. Prideaux, “Environmental impact assessment guidelines for offshore petroleum exploration seismic surveys,” *Impact Assessment and Project Appraisal*, vol. 34, no. 1, pp. 33–43, Jan. 2016, doi: 10.1080/14615517.2015.1096038.

[12]“What is DICOM Image Format & Why is It Important in Radiology?,” Intelerad. Accessed: Nov. 10, 2023. [Online]. Available: https://www.intelerad.com/en/2023/02/23/handling-dicom-medical-imaging-data/

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Computed Tomography (CT)	*A diagnostic imaging technique that uses X-rays and computer technology to produce highly detailed, cross sectional images of the inner structures of a person or object [1].*
Digital Imaging and Communications in Medicine (DICOM)	*The international standard file format for storing and processing medical imaging information [12]. Many CT machines output DICOM files; thus our core data is in the DICOM format.*
CI/CD	*Continuous integration and/or continuous deployment. This term is used to describe the process in which changes to a codebase are pushed to production frequently and automatically.*
PyPI	*The Python Package Index. This is where Python packages are stored and installed from.*