CSC318 Final Report: Atlas

Avery Willets

University of Advancing Technology

CSC318: Software Engineering Principles

Professor Hinton

Aug 24, 2024

Table of Contents

Project Overview

Atlas is a Python-based GUI tool that accelerates the process of data preparation by automating the presentation of errors and error summary statistics for both numerical and categorical data within CSV/TSV datasets. Data professionals spend much of their time in data exploration and preparation, rather than performing analyses that drive business strategies. The goal is to accelerate data preparation by creating a guide/map for users to find errors in their data and subsequently resolve them without needing to use Pandas or Excel. This tool is geared towards data analysts, data engineers, data scientists, developers, and researchers who need quick summary statistics on errors within data files they plan to use.

Atlas has three error-finding functions: One for null values, one for values of erroneous datatypes, and one for numerical outliers. If a value falls into one of these categories, the cell which contains that value will be highlighted according to Atlas's color code: Red for null values, yellow for values of incorrect datatypes, and blue for numerical outliers. The column in which the cell is contained also gets highlighted in a fainter version of the same color to indicate that an error lies within it. These errors are also compiled into error summary statistics and visualized using a bar chart for quick understanding of errors.

Objectives

- Simplify data exploration with automated visual statistical error summaries.
- Accelerate data preparation by highlighting cells and columns with errors.
- Support both numerical and categorical columns.
- Export results for external use and further analysis.

System Design and UML Models

High-level architecture

Atlas is a desktop-based data analysis and validation tool built using PyQt5 for the GUI and pandas/numpy for backend data processing. It is split into two main layers:

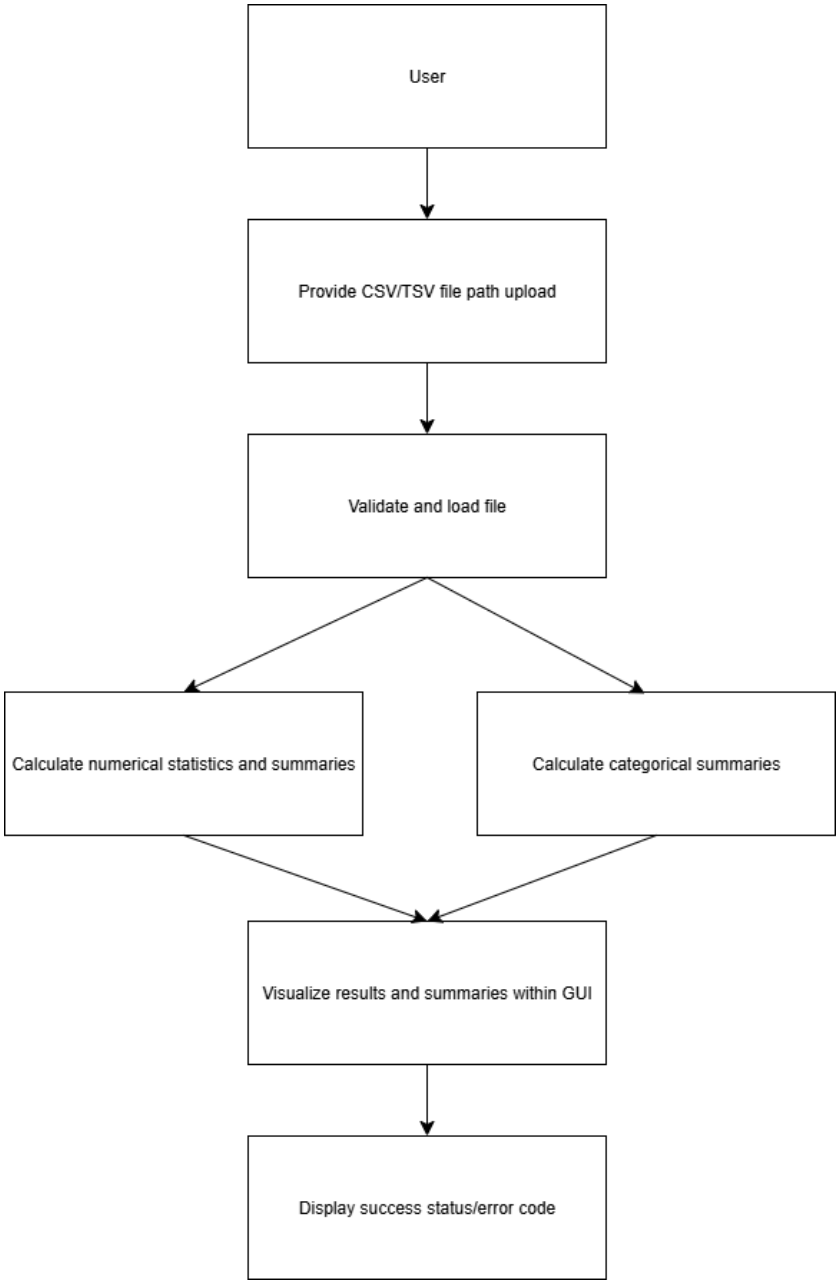
Presentation Layer: This layer handles all user interactivity and display elements. It is built with PyQt5 components such as QTableView, QPushButton, QTextEdit, and QLabel. The GUI is managed within the AtlasApp class, which coordinates file input/output, data visualization, and error summaries.

Data Handling/Logic Layer: This layer is responsible for reading and parsing CSV/TSV files into Pandas Dataframes, validating data for nulls/erroneous datatypes/statistical outliers, and generating error summaries with visuals using matplotlib charts.

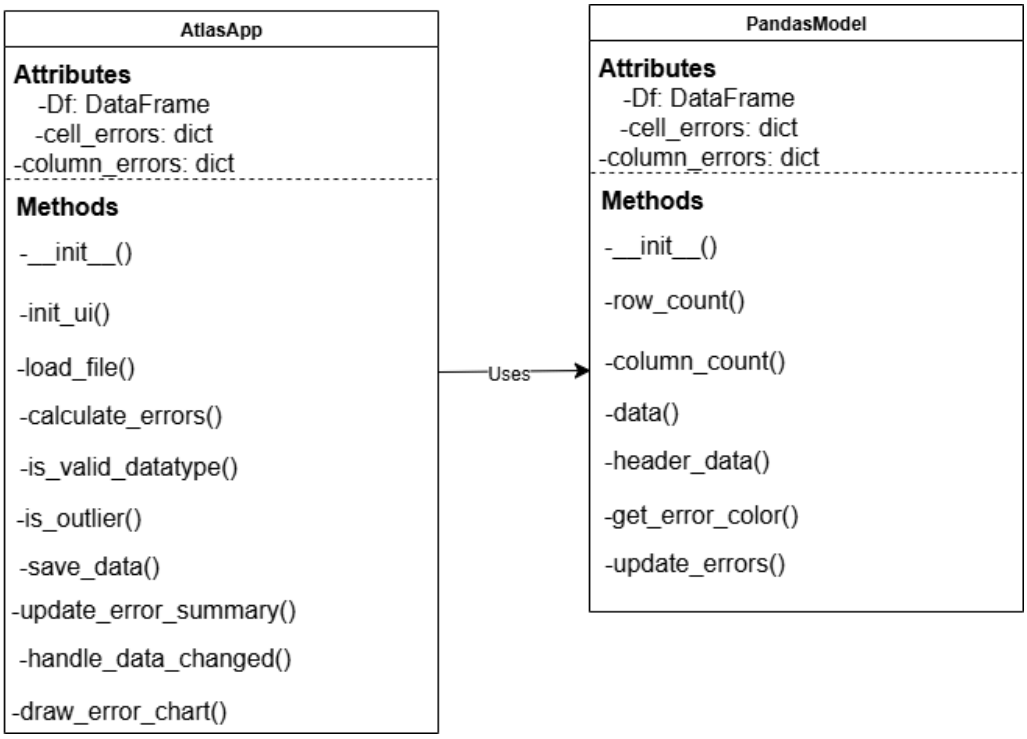
Validation logic is encapsulated within the AtlasApp class, while presentation of the tabular data to the user is held within a custom model class I created called PandasModel, which extends PyQt5's class QAbstractTableModel.

Using matplotlib, a small embedded chart displays a bar graph representing the distribution of data errors. The summary statistics and error summaries are displayed in a read-only text box in the bottom right corner of the GUI.

Use Case Diagram



UML Class Diagram



Design Principles and Patterns

1. **Separation of Concerns:** Atlas separates responsibilities of classes clearly:
    a. The UI is handled in the AtlasApp class, which holds the interface layout and event triggers controlling the PandasModel class.
    b. Data rendering is handled within the PandasModel class, which determines how the data is presented to the user depending on any calculated errors from the AtlasApp class.
    c. Processing logic is self-contained within specific methods in the AtlasApp class.
2. **Model-View-Controller (MVC)**: While not strictly an MVC model, Atlas possesses several qualities of one:
    a. ***Model:*** The PandasModel class holds the logic for how data is presented to the user, as well as stores what kind of errors are within the dataset and where.

   b. ***View:*** I used imported PyQt5 widgets for actually presenting the data, using widgets including QTableView and QTextEdit.
   c. ***Controller:*** The AtlasApp class is what holds the controller logic and the methods managing interactions and application flow.

3. **Single Responsibility Principle (SRP):** Each class within Atlas has a distinct responsibility:
   a. AtlasApp manages the overall application and GUI interaction logic.
   b. PandasModel manages how the data is displayed and handles highlighting of errors.
4. **Observer Pattern:** Atlas utilizes PyQt's dataChanged signal to trigger UI updates in the AtlasApp class. This is a use case of an Observer pattern, which allows components to respond automatically to changes in the model.

Development Process

Development Methodology (Agile/UP)

When creating Atlas, I primarily used the Agile methodology. As a SIP project, I was able to work on Atlas over the course of several sprints, each of which with dedicated goals for the functionality of the project. The project started as a CLI tool that took simulated data, then took CSV files, then I implemented a GUI, and finally added the error summary visualizations alongside the error modeling within the GUI table.

During each sprint, I kept myself accountable for my work through weekly standups. This allowed me to stay focused on short-term goals without requiring perfection from day 1. This Agile methodology also allowed for minor changes to Atlas without the need for restructuring/refactoring everything.

Planning Tools Used

I primarily used Trello as my planning and progress reporting tool. I am most familiar with Trello through Production Studios, and the simple card system meant I didn't need to expend much energy when moving through different phases of the plan. It also meant that I got the satisfaction of literally watching my progress in real-time and understanding the work I was doing.

Version Control Strategy

My primary version control tool for Atlas was actually Canvas, where I would track submissions of my project as I would upload updated files for my assignments on a near weekly basis. I also attempted to use GitHub for version control, but I found that Canvas was easier to revisit and check progress against.

Development Timeline

1. **Sprint 1: Initial Idea and Planning Phase**
    a. During the first sprint, I created the initial plan for Atlas, created initial documentation and diagrams for the prototype, and created my progress tracker in Trello.
2. **Sprint 2: Filtering functions in Google Colab**
    a. In this sprint, I created the initial filtering functions that would be used to find errors and tested their viability using the Google Colab IDE.
3. **Sprint 3: Migration to VSC**
    a. In this sprint, I migrated my code to the Visual Studio Code ID, tested their viability via the CLI, and began planning out how to connect my filtering functions to a UI system.

4. **Sprint 4: UI Design with Tkinter**
    a. During this sprint, I created a simple initial UI system using Tkinter. I ran into many problems when trying to visualize the data properly according to my plans, and spent a lot of time debugging and coming up with custom functions to make it work.
5. **Sprint 5: File I/O**
    a. During this sprint, I created my I/O functions for Atlas which allowed users to upload their own CSV/TSV files to the program, rather than using simulated data or files using a hardcoded path. I also created the download function which exports the data file to the same directory Atlas is located in.
6. Sprint 6: UI Migration to PyQt5
    a. After much research, I found that PyQt5 had many built-in widgets and functions that would be much easier to use than the custom functions I was trying to create for Tkinter. The migration process was incredibly tedious, and I had to refactor Atlas's code several times to make the UI work as intended. By the end, I was much more satisfied with the PyQt5 UI than the Tkinter UI.
7. **Sprint 7: Visualization Dashboard**
    a. In this sprint, I created a small visualization dashboard for summary statistics and error summaries on the side of the main window. While not part of my initial plan, it was recommended to me by a professor and I was incredibly grateful for the feedback. The dashboard was very simple to make, and made Atlas feel much more polished as an exploration and preparation tool.

Testing Methodologies

Types of Testing (manual, functional, edge cases)

- **Functional Testing:** Manual tests were used to verify that each primary feature worked as intended. These included:
    - Loading valid and invalid CSV/TSV files
    - Correct display of data in the table
    - Proper error detection (nulls, wrong datatypes, outliers)
    - Generation of error summaries
    - Saving/exporting of cleaned data
- **Edge Case Testing:** Atlas was tested against various edge cases to ensure robustness in its functionality:
    - Datasets with missing headers
    - Empty files and header-only files
    - Non-numeric data in numeric columns
    - Columns with mixed data types
    - Files with special characters and unusual encodings
    - Large files with 1,000,000+ rows

Example Test Cases and Outputs

Feature Tested	Input Type	Expected Result	Outcome
Load Valid CSV	Properly formatted CSV file	Data displayed without errors	Pass
Load Empty File	CSV with no content	Error message detailing empty file as invalid	Pass
Null Detection	CSV file with missing values	Nulls highlighted in red, columns with nulls highlighted in	Pass

		faint red	
Wrong Datatype	Numerical columns with letters in certain rows	Type errors highlighted in yellow, columns with values of wrong datatype highlighted in faint yellow	Pass
Outlier Detection	Column with extreme numerical values	Values with z-score above 3 highlighted in blue, columns with outliers highlighted in faint blue	Pass
Save Data	After analysis, user input on save data button	File saved as updated_data.csv	Pass
Mixed Data types	CSV with columns with even split between two data types	Type errors flagged, other columns load as normal	Pass
Missing headers	CSV file without headers	Load prevented, validation error message detailing error of missing headers	Pass

Tools and Libraries (if any):

- PyQt5 GUI for user interaction testing.
- Excel for inspecting input/output files.
- Print statements for debugging during development and identifying specific errors.
- Manual test dataset creation in Excel for quick datasets and Python for lengthy ones.

Bug Tracking and Resolution Process

Bug tracking was performed though error logs in the CLI and output messages within the GUI. As part of error validation, I created small message boxes that pop up when the user

attempts to perform an invalid action via the GUI, such as uploading a file without headers. I also maintained a constant watch of the CLI to ensure that if the program wouldn't load for some reason, I could add a layer of error validation to ensure that Atlas wouldn't crash for the end user.

Challenges Faced and Lessons Learned

Technical Hurdles

1. **PyQt5 vs Tkinter:** While I struggled to get Tkinter to load and visualize the data properly, I ran into few actual errors in the process. When I migrated to Py!t5, I found that the widgets provided me much of the functionality I was missing in Tkinter, but they were all difficult to use and I struggled to get everything to interact properly. After much struggling and debugging, I was finally able to get everything to connect and present the data properly.
2. **Slow Runtime:** Another issue I ran into initially was a very slow runtime. I had three separate error-finding functions, each of which I applied to every cell one at a time. This meant that for even small datasets of only 1,000 rows, runtime could be upwards of 60 seconds. I had to rectify this by instantiating a single controller function that would iterate over the cells and check for each kind of error, which reduced both the runtime and computing requirements of the program.

Project Management Challenges

1. **Time Management on Tight Schedule:** During several sprints, I was also working on major assignments for other classes. I still had to do my best to complete the work for the given sprint on time, or risk future sprints falling behind schedule as well. This meant that Trello became my best friend over the

course of the project, as I eventually restructured it with the weeks and even days when I was going to work on Atlas.

2. **Working in an Echo Chamber:** As I was the sole developer for Atlas, there were several times where I thought I had a good idea for an improvement or modification that ended up leading nowhere and costing me time. I believe that if I had been able to bounce ideas off of another developer who was as invested in the project as I was, I could have narrowed my focus and made sure to work on core functionality. I found that the check-ins with professors often provided a good place for me to test out ideas, which were often rejected in favor of maintaining scope (for which I am grateful).

Key Takeaways

1. **Keep Yourself Accountable:** As the sole developer of Atlas, there were times when I fell behind during sprints or got busy with other assignments and had no one checking in with me on Atlas. This often led to increased stress, particularly during weekends when I felt like I was scrambling to catch up. Trello became rather integral to keeping myself accountable, as it gave me a place to actually create and complete short-term goals and maintain a good timeline so I wouldn't feel so stressed.
2. **Maintain Scope:** I had a lot of ideas of where to take Atlas, including but not limited to testing other kinds of errors, creating a find-and-replace function for errors, and allowing slicer toggles to only show certain kinds of errors at a time. While these are all useful functions, they were not part of my core functionality. When I brought up these ideas to my professors, they often told me it would be

good for me to include them as a stretch goal after I finished my MVP, but to focus on core functionality during SIP. I am very grateful for this lesson on scope, as I barely finished the project on time as is. I would have had to sacrifice a lot of validation functionality to add those features, which would have made Atlas weaker overall in my opinion.

3. **Don't Reinvent the Wheel:** While I was initially working on the UI using Tkinter, I had to make many custom functions to present the data and its errors in the way I wanted. I wondered why I had to make so much of it myself, and upon further research I found that PyQt5 had much stronger support for things I wanted in the form of widgets. While refactoring my code took a while, I feel that Atlas is stronger and will work more consistently than if I had continued to struggle with making my own custom functions.

User Manual

Introduction

Atlas is a Python-based GUI tool that accelerates the process of data preparation by automating the presentation of errors and error summary statistics for both numerical and categorical data within CSV/TSV datasets. The goal is to accelerate data preparation by creating a guide/map for users to find errors in their data and subsequently resolve them without needing to use Pandas or Excel. This tool is geared towards data analysts, data engineers, data scientists, developers, and researchers who need quick summary statistics on errors within data files they plan to use.

Installation Instructions

- **Required dependencies**
    - Python 3.7+
    - Pandas library
    - Numpy library
    - Matplotlib library
    - PyQt5 library
- **Installation commands**
    - To install all required Python dependencies for running Atlas, run the following command in the terminal (for path sensitivity) or in the command prompt (for more control over path location):
        - Pip install pandas numpy matplotlib PyQt5
    - It is recommended to use a virtual environment to avoid dependency conflicts.

Launching the App

- **How to run the application**
    a. Navigate to the project directory in the terminal.
    b. Run the Python file with the following command:
        - Python Atlas_UpdatedSummaryStats.py
    c. Atlas should open in a desktop GUI window.

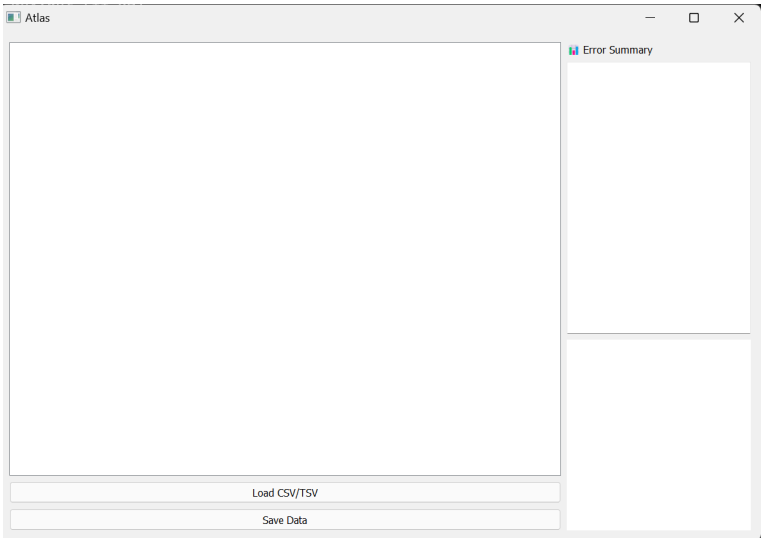
Using the Application

- Step-by-step usage guide
    1. **Launch the Application**

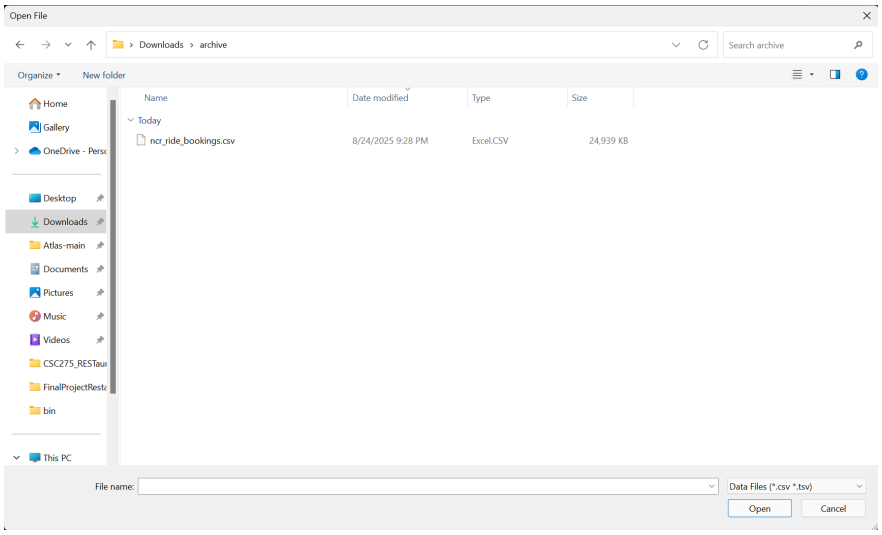
a. Run the script using the provided command or via an IDE. A window titled “Atlas” will appear.

2. **Load the Dataset**
    a. Click the “Load CSV/TSV" button. A file dialog will open. Choose a .csv or .tsv file from your system.
3. **View the Data**
    a. Once loaded, the dataset appears in a table format. Each cell displays the original content of the dataset.
4. **Check for Errors**
    a. Atlas automatically scans for:
        i. Null Values (highlighted in red)
        ii. Wrong Datatypes (highlighted in yellow)
        iii. Outliers (highlighted in blue)
    b. A summary and visualization of detected errors will appear on the right-hand side dashboard.
5. **Save the Analyzed Changes**
    a. Click the “Save Data” button to export the current dataset (including any changes once that functionality is added) as ‘updated_data.csv’ in the same project directory as Atlas.

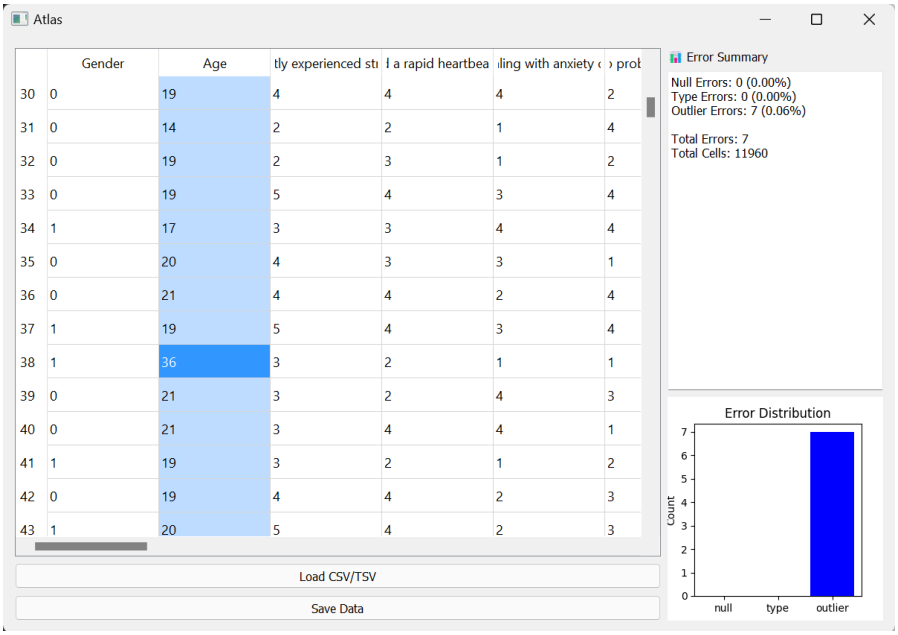
- **Screenshot Recommendations**
    - *Figure 1: Atlas Main Interface*



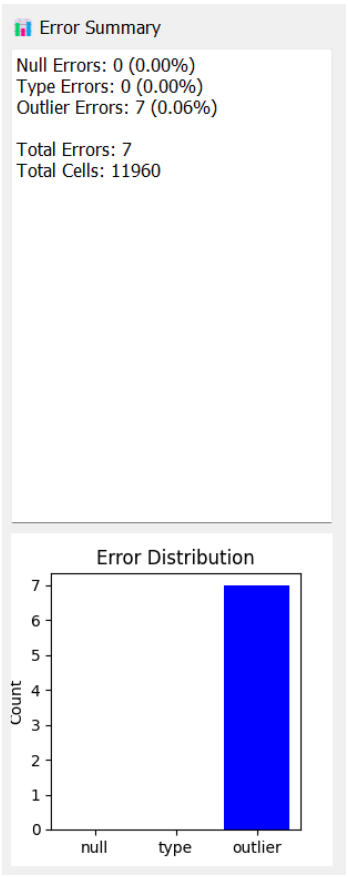
- *Figure 2: Selecting Data File*



- *Figure 3: Data Table with error highlights*



- *Figure 4: Error Summary Chart*



Supported Formats and Limitations

- **File formats**
    - CSV (Comma-Separated Values)
    - TSV (Tab-Separated Values)
    - Files must include headers/column names in the first row to load effectively,
- **Known issues**
    - **Performance:** Large files (over 100k rows) may cause the UI to freeze during error calculation. Files with over 1,000,000 rows may cause Atlas to run indefinitely or crash.
    - **File Saving:** Saved output is hardcoded as updated_data.csv and cannot be renamed using Atlas's interface.
    - **Editing:** While the current version does not support in-table editing, future versions will.
    - **Error Detection:** While the current version of Atlas only supports the three error types, future versions will include more robust error detection.
    - **No Automated Updates:** If users update a file while it is being used by Atlas, they must re-load the app to view the changes.