

---

# **Python Data Viewer Application**

***Release 1.7.1***

**Rafael Arvelo**

**Mar 23, 2025**



## CONTENTS:

<b>1</b>	<b>Overview</b>	<b>1</b>
1.1	Supported Input Formats . . . . .	1
1.2	Supported Operating Systems . . . . .	1
<b>2</b>	<b>Setup Instructions</b>	<b>3</b>
2.1	Dependencies . . . . .	3
2.2	Application Setup / Installation . . . . .	3
<b>3</b>	<b>Starting the Application</b>	<b>5</b>
<b>4</b>	<b>User Manual</b>	<b>7</b>
4.1	Application Menus / Actions . . . . .	8
4.1.1	File Explorer . . . . .	9
4.1.2	Opening Folders . . . . .	10
4.1.3	Opening Tables . . . . .	10
4.1.4	Merging Tables . . . . .	11
4.2	Table Viewer Actions . . . . .	11
4.2.1	Search button . . . . .	12
4.2.1.1	Column Search . . . . .	12
4.2.1.2	Row Search . . . . .	13
4.2.2	Creating Plots . . . . .	13
4.2.2.1	Plotting String Data . . . . .	14
4.2.2.2	Plotting Multiple Columns . . . . .	15
4.2.2.3	Creating Custom Plots . . . . .	15
4.2.3	Side Menus . . . . .	17
4.2.3.1	Show / Hide Columns . . . . .	17
4.2.3.2	Saved Plots . . . . .	17
4.2.3.3	Color Rules . . . . .	18
4.2.3.4	Query / Filter . . . . .	19
4.2.4	Context Menus . . . . .	20
4.2.4.1	Table Viewer Context Menu . . . . .	20
4.3	Application Settings . . . . .	21
4.3.1	Table / Plot Settings . . . . .	21
4.3.2	Window Settings . . . . .	22
4.3.3	Restoring a previous session . . . . .	22
4.3.4	File Parser Settings . . . . .	23
<b>5</b>	<b>LICENSE</b>	<b>25</b>



## OVERVIEW

The **DataFrameViewer** application is a Qt Python application to view, edit, plot, and filter data from various file types.

The DataFrameViewer utilizes the `pandas` module along with the `Qt for Python` module to provide a familiar spreadsheet-like GUI for any type of data that can be stored in a `pandas DataFrame`.

The intention of this application is to provide a high-performance, cross-platform application to review and analyze data. The DataFrameViewer provides a faster and more optimized alternative for viewing and plotting data files in a table format as opposed to other applications such as Microsoft Excel or OpenOffice.

### 1.1 Supported Input Formats

Note: Input formats are automatically recognized based on the filename.

The Data Viewer currently supports the following input formats:

- CSV (comma-delimited, tab-delimited)
- TXT (plain-text files)
- JSON (Javascript Object Notation)
- PICKLE (Python Pickle Format)
- XLSX (Microsoft Excel or OpenOffice files)
- HDF5 (Hierarchical Data Format)

### 1.2 Supported Operating Systems

The following operating systems have been tested and confirmed to operate the application nominally:

- Windows 10
- MacOS Version 11.2 (Big Sur) using Apple M1
- Linux (CentOS, Ubuntu)

Other operating systems are untested but will likely function if they are supported by the Qt for Python version documented in `requirements.txt`



## SETUP INSTRUCTIONS

### 2.1 Dependencies

- pandas
- numpy
- PyQt5
- openpyxl
- matplotlib
- QDarkStyle

### 2.2 Application Setup / Installation

Note: If you are using an Anaconda installation, you can skip these setup steps and proceed directly to the *Starting the Application* section.

The recommended setup method is to use an isolated installation via the `virtualenv` module.

`virtualenv` installation on Windows:

```
virtualenv venv
source venv/Scripts/activate
pip install dataframeviewer
```

`virtualenv` installation on MacOS / Linux:

```
virtualenv venv
source venv/bin/activate
pip install dataframeviewer
```

Local installation (on any platform):

```
pip install dataframeviewer
```

Installation using a proxy in windows Powershell:

```
$PROXY("<your-proxy-url>"
$env:HTTP_PROXY="$PROXY"
$env:HTTPS_PROXY="$PROXY"
```

(continues on next page)

(continued from previous page)

```
pip config set global.trusted-host "pypi.org files.pythonhosted.org pypi.python.org"
pip install dataframeviewer
```



## STARTING THE APPLICATION

Run as a module

```
python -m dataframeviewer
```

Run with sample data

```
python -m dataframeviewer --example
```

Run with input file(s)

```
python -m dataframeviewer -f file1.csv file2.csv ...
```

To show the full command line option list

```
python -m dataframeviewer --help
```

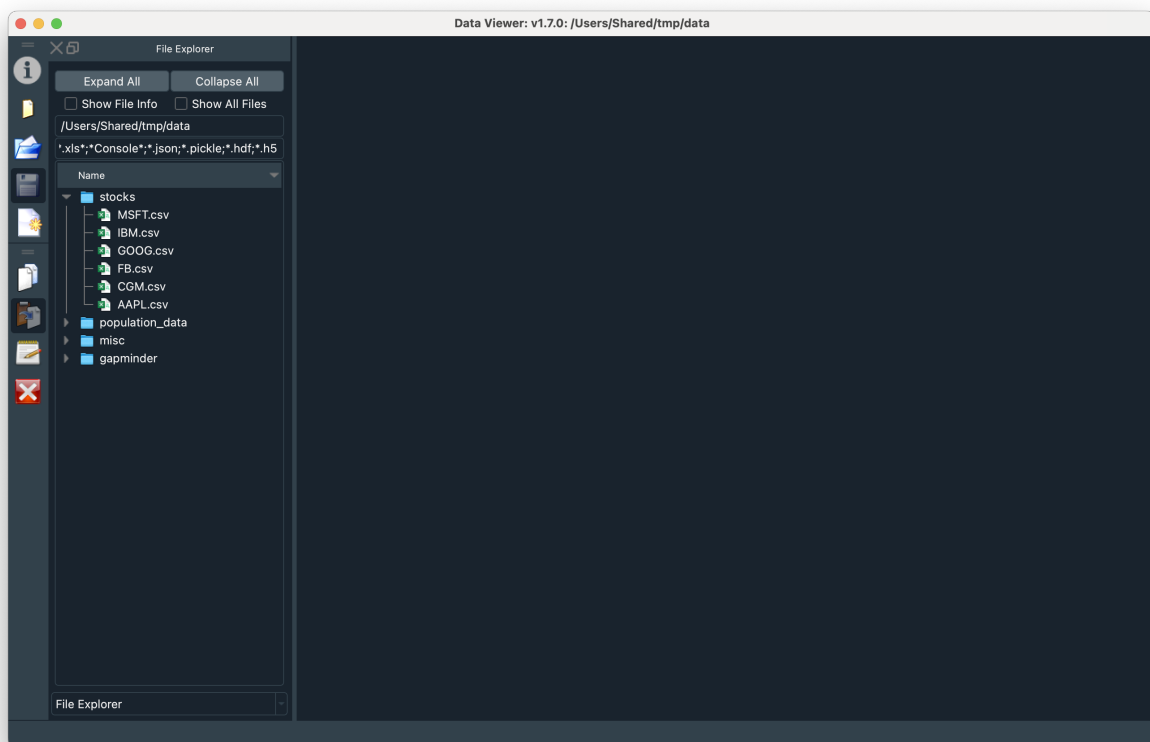
See the [User Manual](#) for application usage instructions.



## USER MANUAL

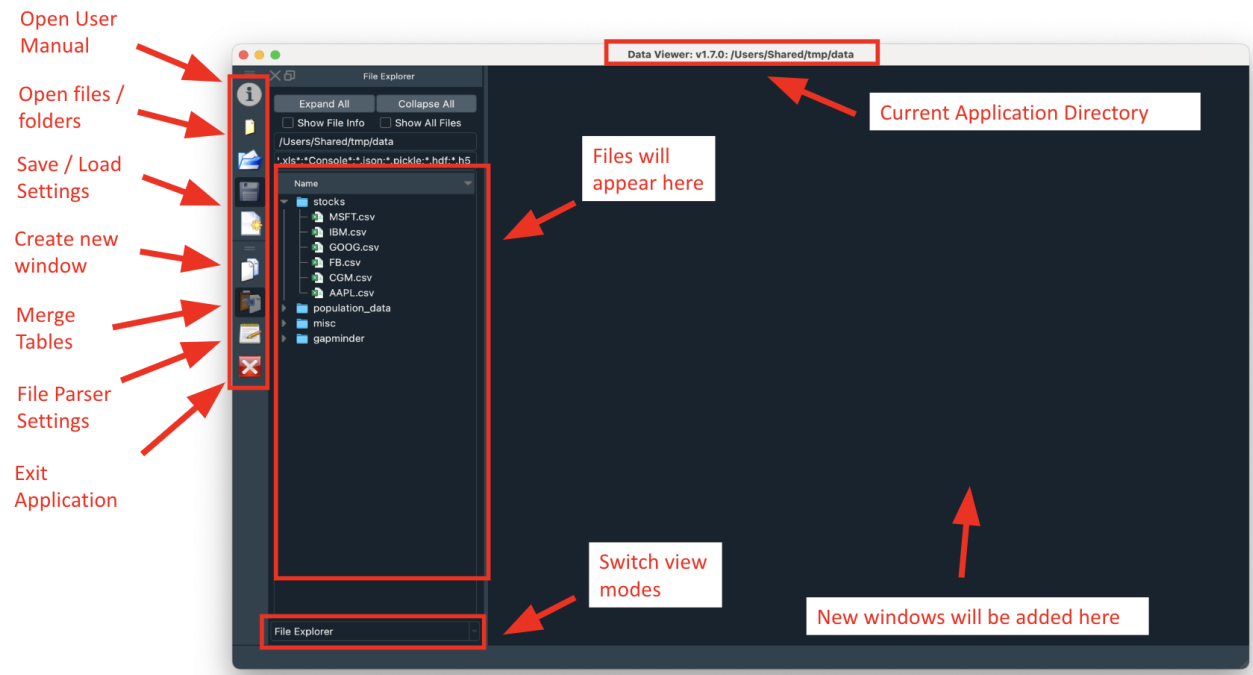
The application will open for the first time and show a single File Explorer window and a toolbar.

Note: the “look and feel” of the application will vary between operating systems













## 4.1 Application Menus / Actions

The image below highlights the buttons on the main user interface of the Data Viewer.



The toolbar icons are pinned to the left of the application window by default but can be dragged / dropped on any side of the application.

The table below describes the available actions from the main toolbar:





Icon	Action	Description
	Open User Manual	Open the User Manual PDF for the dataframeviewer application.
	Open File(s)	Start a File Dialog to select 1 or more file to open as tables.
	Open Folder	Open a folder to show in the File Explorer.
	Save Window Settings	Save the state of all tables and charts to an INI file.
	Load Window Settings	Restore all of the tables and charts from a selected INI file.
	Restore Previous Session	Restore all of the tables and charts from the previous session.
	New Window	Open a new window with the same current directory.
	Merge Tables	Open a dialog to join two tables on a common column.
	File Parser Settings	Open an editor to modify the backend parsers that read files.
	Exit Application	Close all windows and exit the application.

### 4.1.1 File Explorer






The application will open for the first time to the user's Home directory. Whenever new files are opened, the File Explorer will change to display the files in the parent directory.

The File Explorer will display a `QTreeView` based on a `QFileSystemModel` where the `rootPath()` is set to the current directory.

You can raise a context menu by right-clicking on any row of the File Explorer. The context menu options for a directory are show in the table below:

Icon	Action	Description
	Open Folder	Open the selected folder in the File Explorer window
	Open with System Editor	Open the selected folder using the default OS File Explorer.
	Copy path to clipboard	Copy the path of the folder to the OS Clipboard.
	Open Parent Folder	Open the parent folder using the default OS File Explorer.

The context menu options for data files are show in the table below:

Icon	Action	Description
	Open Table	Open a new <i>TableViewer</i> tab for the selected file
	Open Plot	Open a new blank tab where a Custom Chart can be created.
<varies>	Open in Python	Open a new Python Interpreter with the selected file as a DataFrame named "df".
	Open with Settings	Open a new <i>TableViewer</i> tab for the selected file with the selected settings
<varies>	Open with System Editor	Open the selected file using the default OS File Explorer.
	Copy path to clipboard	Copy the path of the file to the OS Clipboard.
	Open Parent Folder	Open the parent folder using the default OS File Explorer.

The File Explorer can also optionally be switched to a single list of all open tables by switching the combo box below the File Explorer to the Table List option. Selecting rows while in the Table List mode will switch the focus to the selected tab.

### 4.1.2 Opening Folders

Opening a directory with the **Open Folder** option only affects the **File Explorer** window and does not attempt to read all of the files in a directory.

Folders can be opened via the **Open Folder** action from the main toolbar or via **File > Open > Directory**.

### 4.1.3 Opening Tables

The *TableViewer* is the primary visualization tool provided by the *dataframeviewer* package.

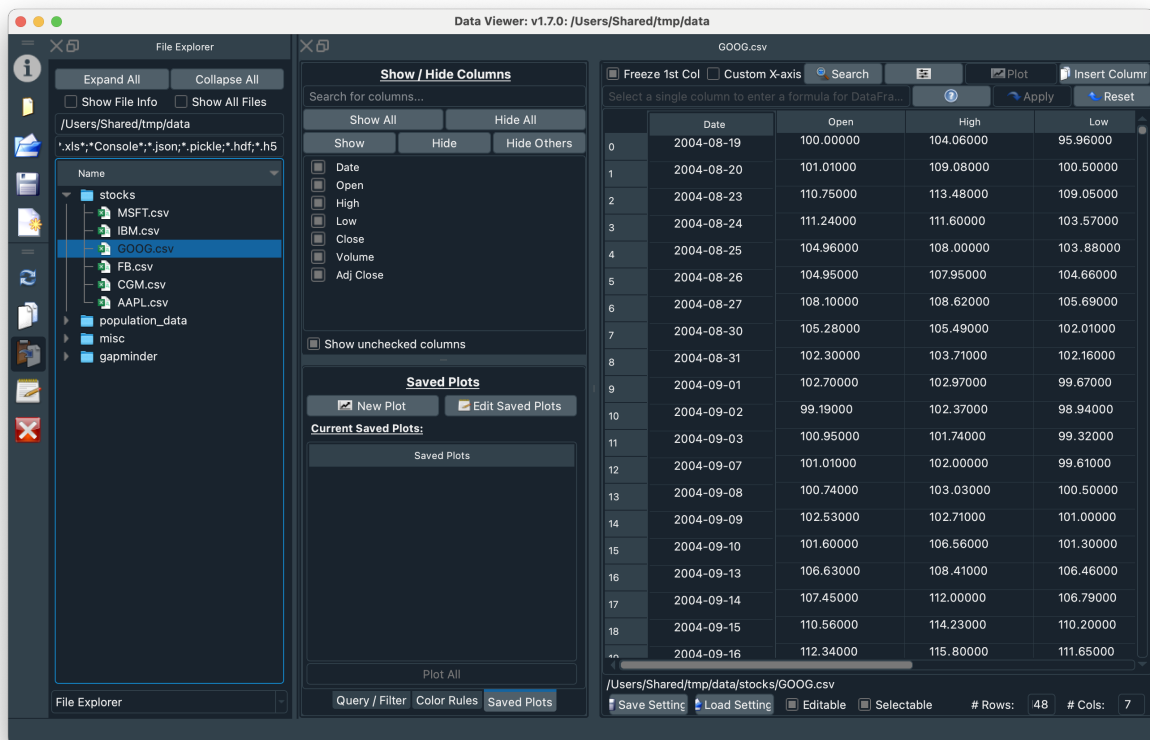
A *TableViewer* is a *QDockWidget* to display a *QTableView* that encapsulates a *pandas DataFrame* using PyQt5's *Model / View API*.

Tables can be opened by using any one of the following methods:

- Using the **File** button on the toolbar.
- Select **File > Open > Files** from the application Menu.
- Double-click the filename in the **File Explorer**.
- Drag and drop files into space next to the **File Explorer**.

Once opened, tables will open to the right of the **File Explorer** as shown in the screenshot below:

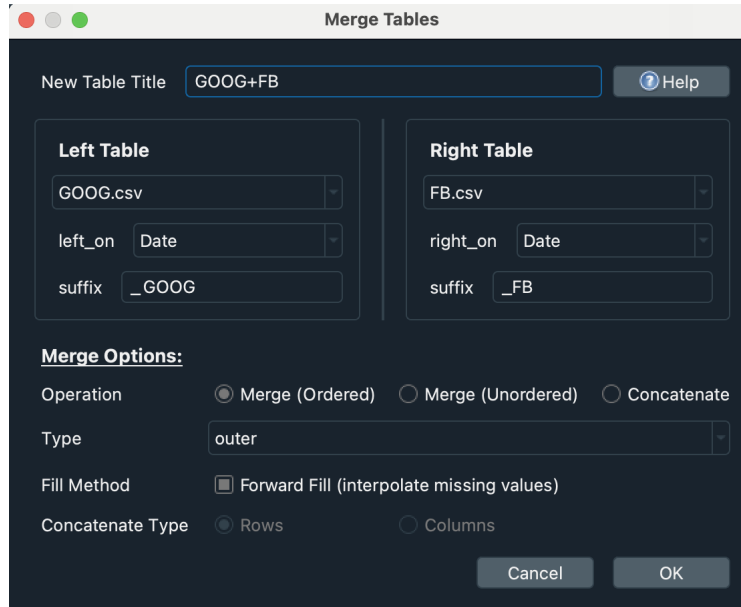
**Note:** By default, the first column will be duplicated and will remain frozen to the left while the ``Freeze 1st col`` checkbox is checked. Modifications made to the table happen in RAM only and do not affect files on the file system.



### 4.1.4 Merging Tables








In many cases it is desired to view data “side-by-side” or overlay data from separate data sets. In order to accomplish this, the application supports “merging” tables using the Merge Tables dialog.

Merging tables requires at least 2 open tables and uses the `pandas.merge` function to join the tables. The merge tables dialog is shown below.



## 4.2 Table Viewer Actions

The table below highlights the primary buttons / checkboxes to interact with a `TableViewer` window.

Icon	Action	Description
<Checkbox>	Freeze 1st column	Freeze the leftmost column when scrolling in table.
<Checkbox>	Custom X-Axis	Use the first selected column as the X-Axis for plots.
<Checkbox>	Editable	Allow modifications to the data displayed in the table.
<Checkbox>	Selectable	Allow column selection within the table (may slow rendering)
	Search Columns or Rows	Raise a Search Dialog to select desired columns or rows.
	Create a Plot	Creates a line chart using the selected columns.
	Create a Custom Plot	Opens a dialog to create a customized chart.
	Insert a new column	Insert a new column after the currently selected column.
	Save Table Settings	Save all table customized settings to a JSON file.
	Load Table Settings	Load previously saved table settings from a JSON file.
	Help Button	Click to get instructions or link to documentation.

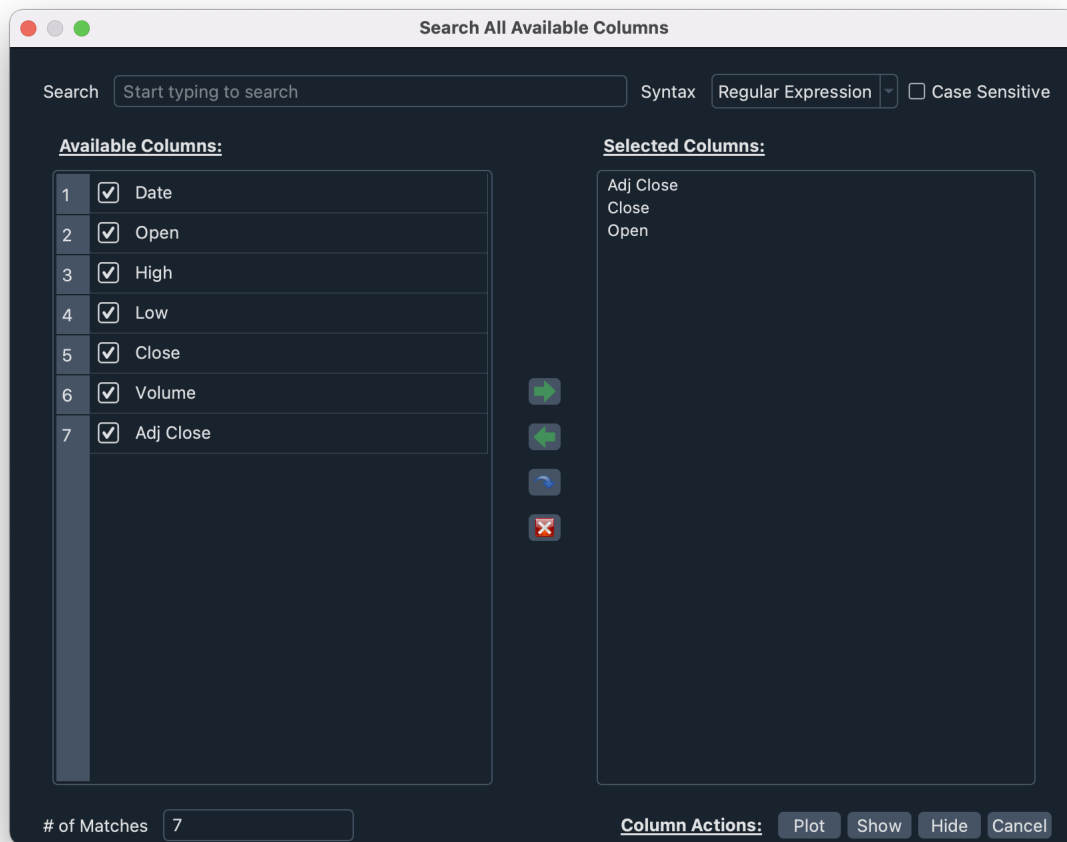
## 4.2.1 Search button

The Search button on the top of each table can be used to either search the columns of the table or search the values of a single column.

Row Search is only performed when a single column is selected.

### 4.2.1.1 Column Search

To search the available columns, click the Search button or press Ctrl+F while no columns are selected.



Columns selected from the Available Columns list are shown in the Selected Columns list.

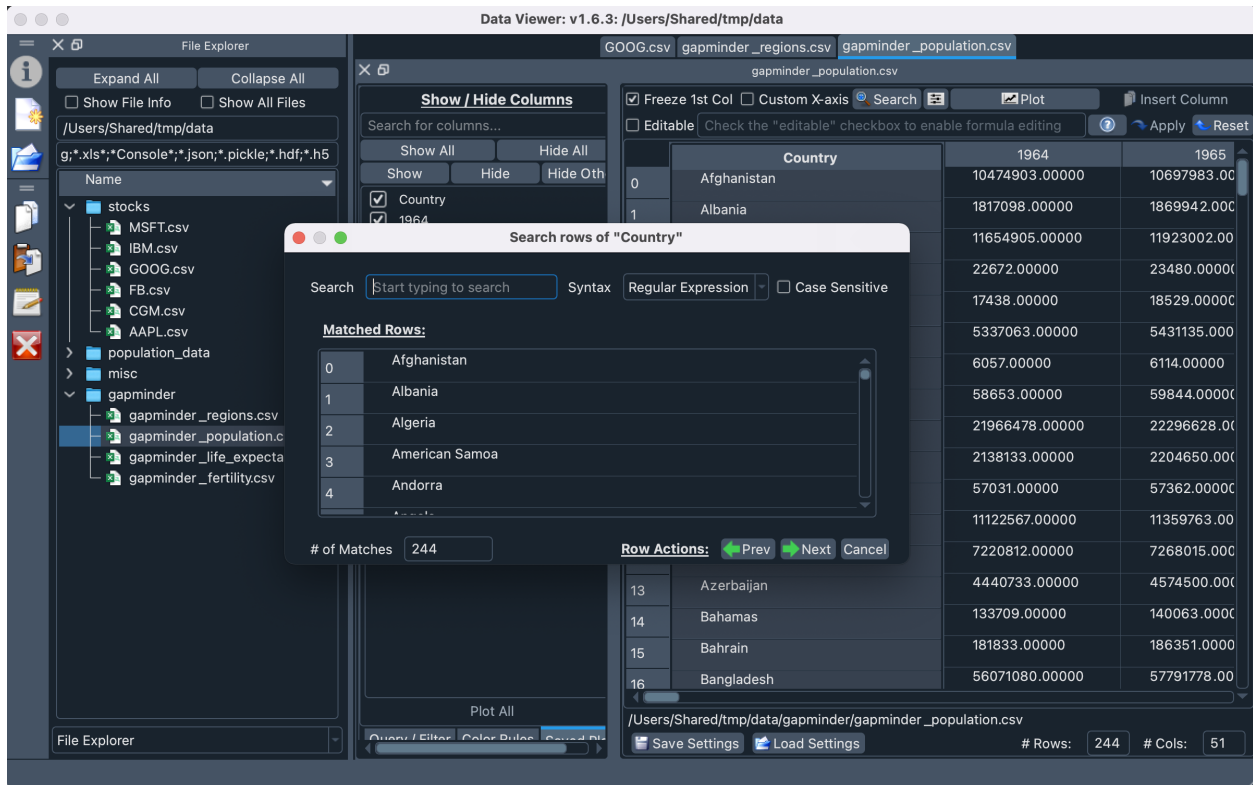
Once selected, any one of the following actions may be selected:

- **Plot** - Create a line chart with the selected columns and the default X-Axis.
- **Show** - Hide all other columns except the currently selected columns
- **Hide** - Show all other columns except the currently selected columns
- **Cancel** - Close the dialog



### 4.2.1.2 Row Search

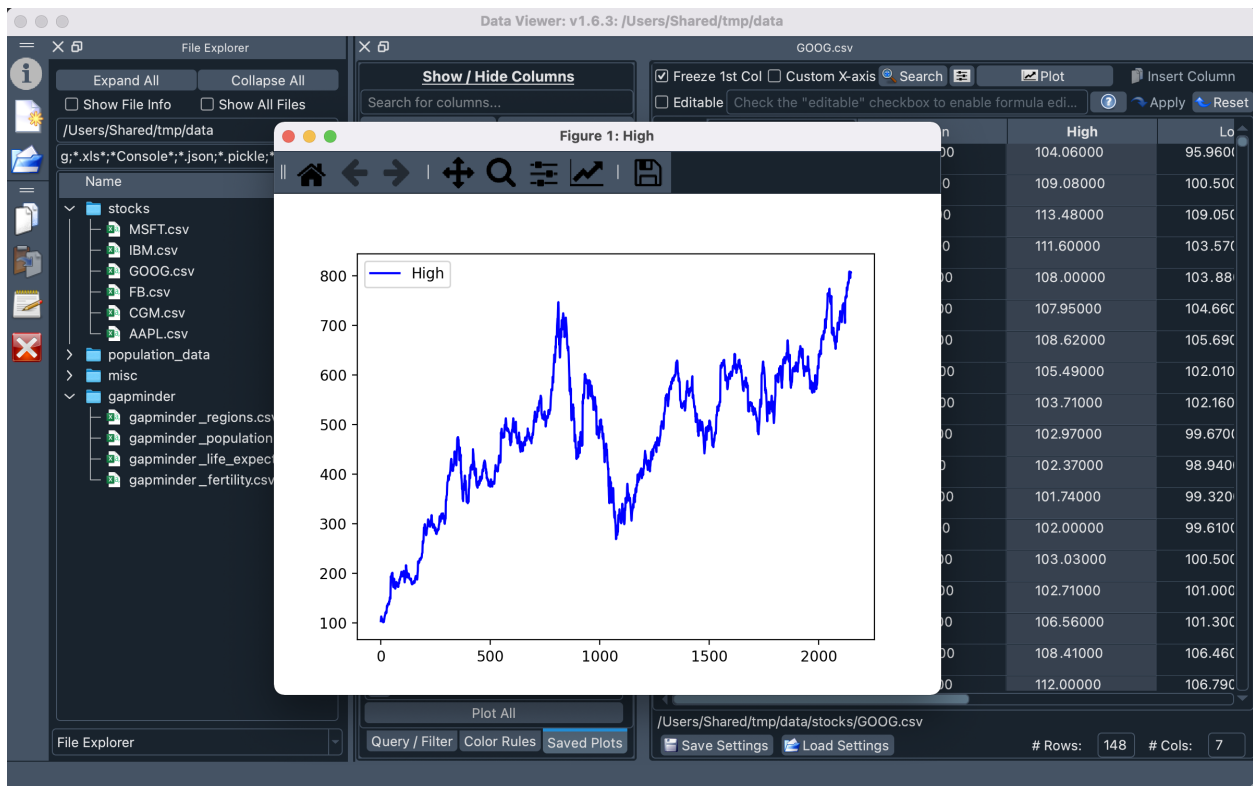
To search the rows for a specific column, click the Search button or press Ctrl+F while a single column is selected.



### 4.2.2 Creating Plots

Plots can be created from tables via the context menu or by clicking the Plot button at the top of a table after selecting one or more column

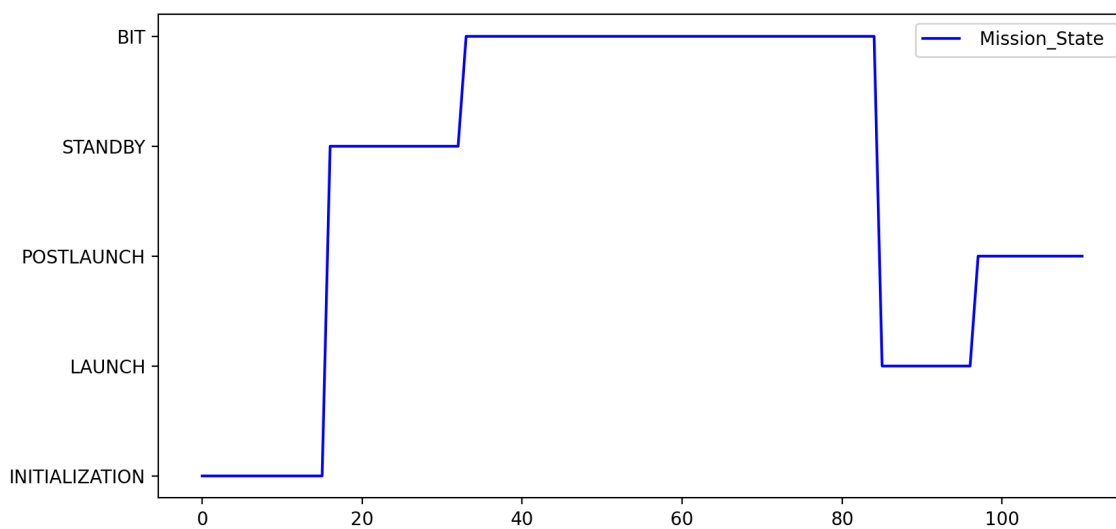
Note: By default, the X-axis will be the row number for each value in the selected column(s). To use a specific column as the x-axis, check the ``Custom X-Axis`` checkbox and then the first column that is selected will be used as the X-Axis in the chart.



By default plots are interactive. See the [Matplotlib Documentation](#) for details on the navigation toolbar.

#### 4.2.2.1 Plotting String Data

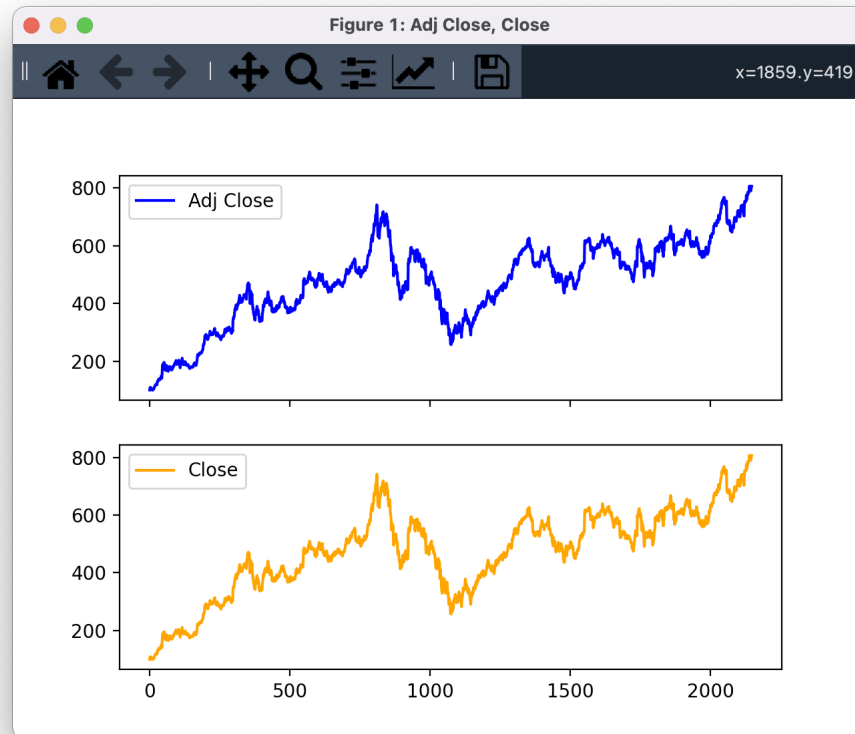
Any columns containing data that is non-numeric (i.e. not convertible to floating point) will be automatically enumerated and assigned labels for plotting similar to the example below.



#### 4.2.2.2 Plotting Multiple Columns

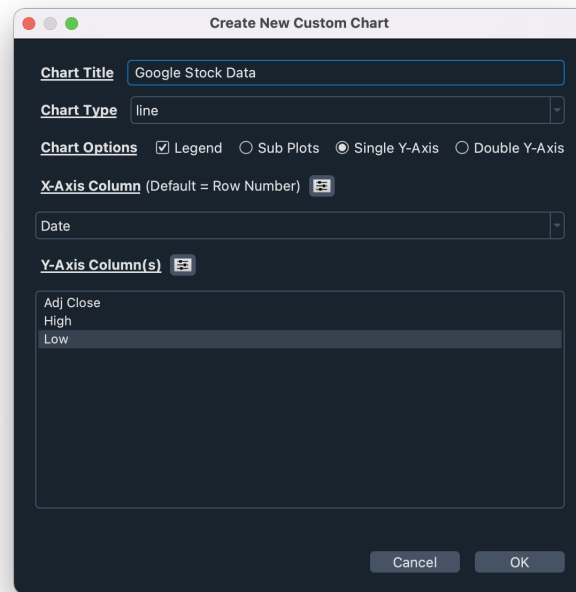
Charts can be created for any number of columns. To create a plot with multiple columns, simply select as many desired columns as you would like and click *Plot*.

By default, plots will share the same linked X-Axis. If different behavior is desired, use the [Creating Custom Plots](#) instructions.

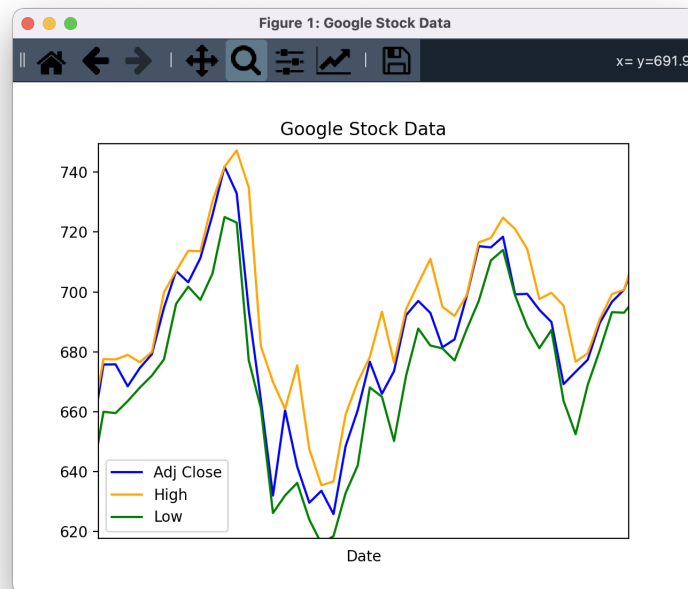


#### 4.2.2.3 Creating Custom Plots

Line plots are created by default, but a custom plot menu dialog can be raised by clicking the Custom Plot button at the top of the table. The image below shows the custom plot menu.



The image below shows an example custom line chart with multiple Y-axes:



The following chart types are also supported in addition to line charts using [matplotlib.pyplot](#):

Plot Type	Description
Line plot	Plotting data points connected by straight lines.
Scatter plot	Visualizing individual data points as markers on a graph.
Bar plot	Representing data with vertical rectangular bars.
Horizontal Bar Plot	Representing data with horizontal rectangular bars.
Histogram	Displaying the distribution of numerical data through bins.
Pie chart	Illustrating proportions of a whole by dividing a circle into sectors.
Box plot	Summarizing the distribution of a dataset using quartiles.
Violin (kde) plot	Combining aspects of box plot and kernel density estimation.
Filled area plot	Highlighting the area between data points and a reference axis.

### 4.2.3 Side Menus

The left-hand side of every table has the following 4 menus that can be used to interact with the table viewer:

Custom Menu	Description
Show / Hide Columns	Allows users to show or hide specific columns in the table.
Saved Plots	Provides easy access to saved plots for visualization.
Color Rules	Manages color rules (conditional formatting) for cells in the table.
Query / Filter	Enables users to filter rows in the table based on a custom <a href="#">query</a> .

The left-hand panel is split by a [QSplitter](#) that can be dynamically resized to any desired width.

#### 4.2.3.1 Show / Hide Columns

The Show / Hide Columns list can be used to quickly search all available columns in the table.

Clicking on one or more rows in the list will select and scroll to the column(s) in the table view. Once a column selection is made you can show or hide the desired columns in the table.

#### 4.2.3.2 Saved Plots






Each time a plot is created from the table a new row is added to the Saved Plots list.

Clicking on a row in the Saved Plots list will re-open the plot with the same settings.

The settings for the saved plots can be modified by clicking the Edit Saved Plots button and new plots can be created by clicking the New Plot button.

Details on creating new plots can be found in the [Creating Custom Plots](#) section.

A context menu for each Saved Plots table row can be activated by right-clicking on any saved plot. The table below shows the available actions from the Saved Plots Context Menu:

Icon	Action	Description
	Open Plot	Opens a popup window dialog of the Saved Plot.
	Open Plot in New Window	Opens a new tab of the Saved Plot within the application window.
	Edit Saved Plot	Open a dialog to edit plot settings.
	Copy Saved Plot	Creates a new row in the table with duplicate plot settings.
	Delete Saved Plot	Delete the selected Saved Plot Row.

#### 4.2.3.3 Color Rules

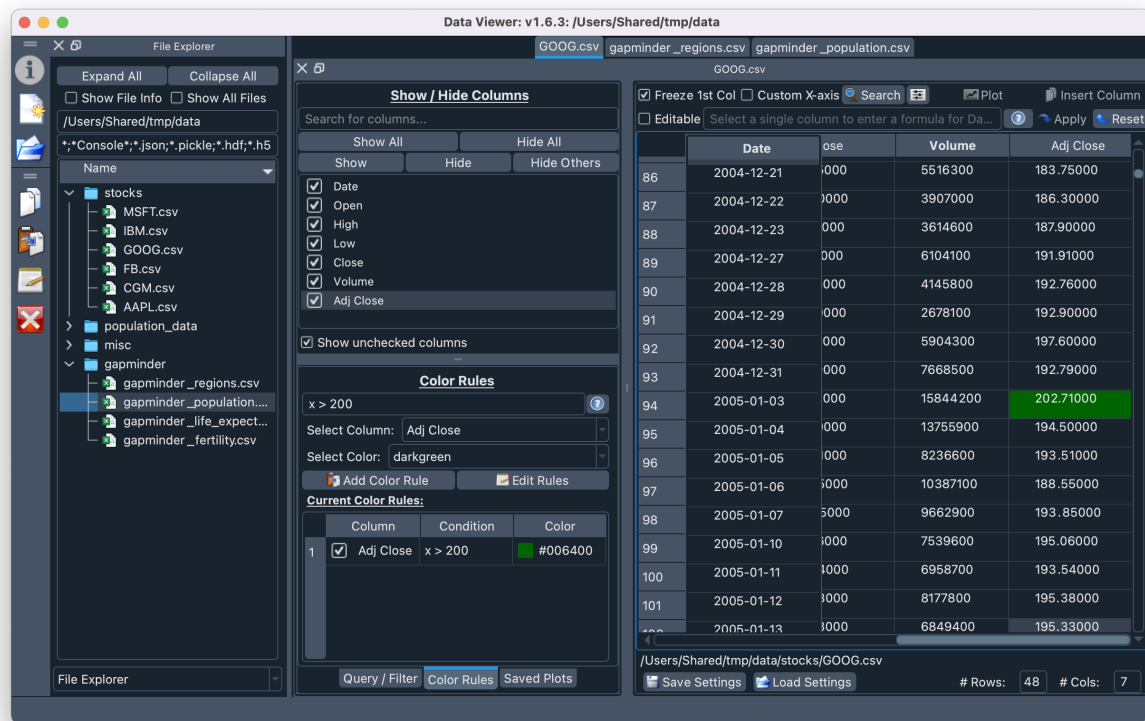
The background color of each cell can be controlled by adding one or more rules via the Color Rules menu. This functions similar to “Conditional Formatting” in a Microsoft Excel Spreadsheet.

A Color Rule can be added by entering a Python statement where “x” will be replaced with cell values. If the statement evaluates to True, then the color will be applied.

##### Color Rule Examples:

```
x == 1
x % 2 == 0
0 < x < 5
math.floor(abs(x)) % 2 == 0
"PASS" in str(x)
```

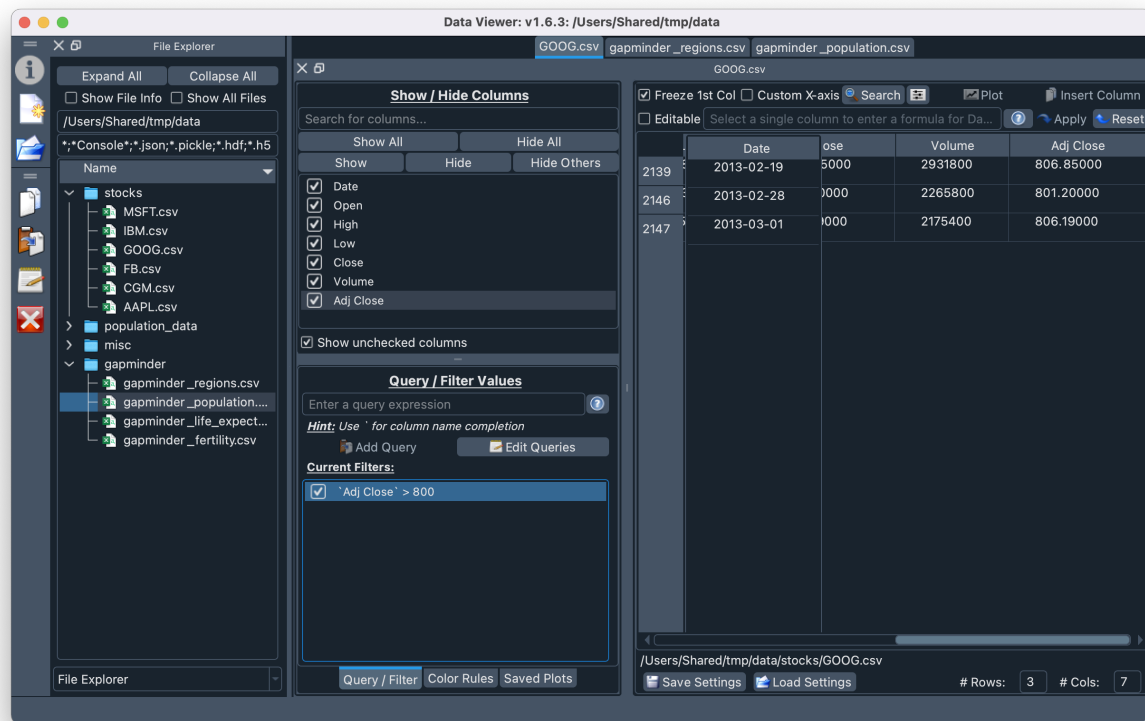
The image below shows how color rules will appear in the table:



#### 4.2.3.4 Query / Filter

The Query / Filter menu allows you to enter a string that will be provided to the Table Viewers internal DataFrame via the `query` method.

The image below show an example of filtering rows in the table:



## 4.2.4 Context Menus

### 4.2.4.1 Table Viewer Context Menu

You can open a context menu for the table by right-clicking anywhere on the table viewer. The actions in the context menu will operate on any columns that you have selected.

The following actions are supported from the `TableViewer` context menu:



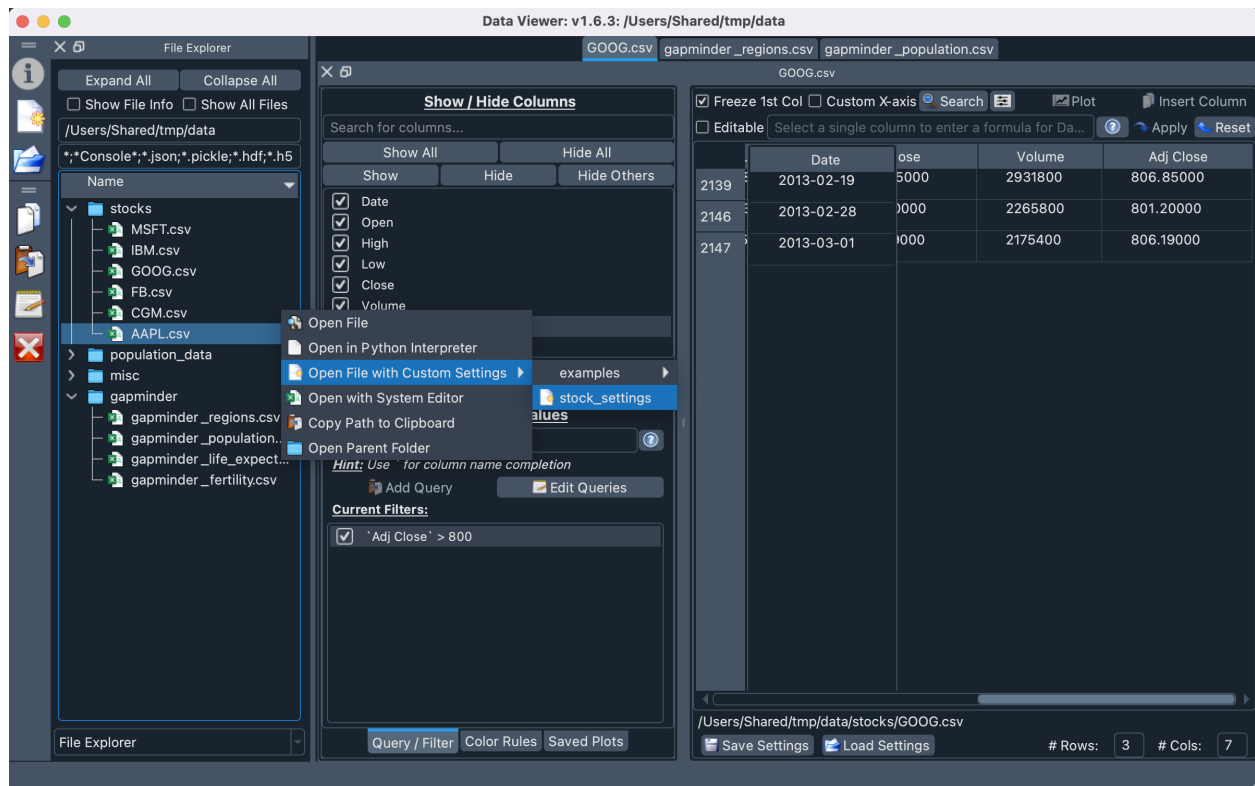
Context Menu Action	Description
Apply Formula	Menu to apply a formula to the selected column.
Insert Column	Action to insert a new column into the table.
Rename Column	Action to rename an existing column.
Copy Column(s)	Action to copy selected column(s).
Delete Column(s)	Action to delete selected column(s).
Sort Column	Action to sort the table by a selected column.
Get Statistics	Action to compute and display statistics of the data.
Hide Column(s)	Action to hide selected column(s).
Hide Other Column(s)	Action to hide all columns except selected one(s).
Unhide All Column(s)	Action to unhide all hidden columns.
Autofit Column(s)	Action to adjust the width of selected column(s).
Edit Color Rules	Action to edit color formatting rules.
Load Settings	Action to load table settings from a file.
Save Settings	Action to save table settings to a file.
Export Table to CSV	Action to export the table data to a CSV file.
Export Table to Other Format	Menu to export the table data to other file formats based on the <a href="#">File Parser Settings</a>

## 4.3 Application Settings

### 4.3.1 Table / Plot Settings

Settings can be saved to JSON (.json) files directly from the application so they can be loaded and applied to multiple tables.

Settings can be applied to tables by right-clicking a file in the File Explorer window and selecting an available filter from the **Open with Settings** Menu.



## 4.3.2 Window Settings

At any time the user can select the **Save Window Settings** action from the main toolbar to save all of the open tables and charts to the a new .ini configuration file. The **Load Window Settings** action from the main toolbar can be used to restore all of the saved tables and charts.

A key difference between window settings and table settings apart from the file format is that Table Settings are not specific to a file and are intended to be applied generically to any table based on column names, whereas the window settings restore specific tables and chart viewer windows for specific files and also attempt to restore the size and position of all windows.

## 4.3.3 Restoring a previous session

The Main Application can be closed at any time by clicking the exit icon or by using **File > Exit** from the Main Application Menu. Once closed, the application will save any currently open tables and charts to the local configuration file. When reopening the application, a previous session can be restored from the config file.

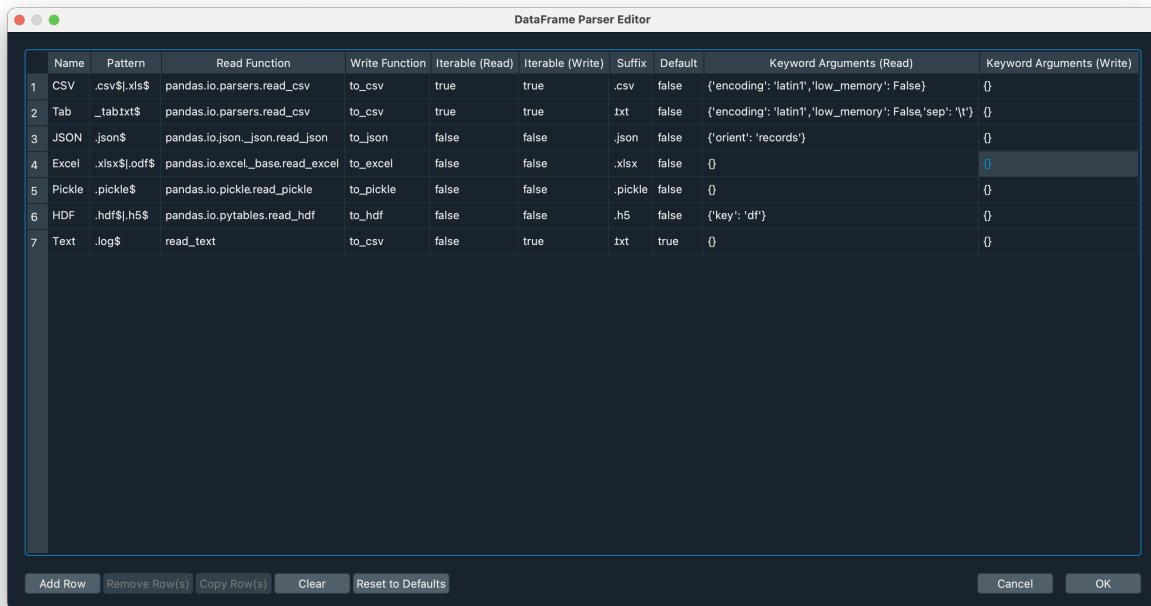
**Note:** The config file for the application can be found in the user's home directory as a file named `~/dataframeviewer/dataframeviewer_settings.ini`

### 4.3.4 File Parser Settings

The backend used to parse files into a Pandas `DataFrame` can be configured using the **File Parser Settings** menu. The table below describes the required and optional fields to add a new parser to the application.

Key	Example Values	Description
name	CSV, JSON	Unique Name for the parser
pattern	.csv\$ .json\$	Regular expression to match filename when opening files
read_func	pandas.read_csv, das.read_json	Name of a function to read a file into a DataFrame
write_func	'to_csv', 'to_json'	Name of pandas.DataFrame function for writing to a file
suffix	.csv, .json	Suffix to add to table names when writing to a file
default	True, False	True if parser should be used for unknown file types
read_iterable	True, False	True if read_func accepts a <code>chunks</code> argument for progress during read
write_iterable	True, False	True if write_func accepts a <code>chunks</code> argument for progress during write
read_kwargs	{ 'sep', 't' }	Optional Keyword arguments to pass to read_func
write_kwargs	{ 'encoding', 'utf-8' }	Optional Keyword arguments to pass to write_func

The image below shows the **DataFrame Parser Editor**. Rows can be modified in the table to adjust the parser settings. Once saved, the parser settings are written to the *Application Config File*





**LICENSE****MIT License**

Copyright (c) 2024 Rafael Arvelo

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.