pandastable Documentation

Damien Farrell

Sep 19, 2022

Contents

Contents:

CHAPTER 1

Introduction

The pandastable library provides a table widget for Tkinter with plotting and data manipulation functionality. It uses the pandas DataFrame class to store table data. Pandas is an open source Python library providing high-performance data structures and data analysis tools. Tkinter is the standard GUI toolkit for python. It is intended for the following uses:

- for python/tkinter GUI developers who want to include a table in their application that can store and process large amounts of data
- for non-programmers who are not familiar with Python or the pandas API and want to use the included DataExplore application to manipulate/view their data
- it may also be useful for data analysts and programmers who want to get an initial interactive look at their tabular data without coding

CHAPTER 2

Current Features

- add, remove rows and columns
- spreadsheet-like drag, shift-click, ctrl-click selection
- edit individual cells
- sort by column, rename columns
- reorder columns dynamically by mouse drags
- set some basic formatting such as font, text size and column width
- save the DataFrame to supported pandas formats
- import/export of supported text files
- rendering of very large tables is only memory limited
- interactive plots with matplotlib, mostly using the pandas plot functions
- basic table manipulations like aggregate and pivot
- filter table using built in dataframe functionality
- graphical way to perform split-apply-combine operations

CHAPTER 3

The DataExplore application

Installing the package creates a command *dataexplore* in your path. Just run this to open the program. This is a standalone application for data manipulation and plotting meant for education and basic data analysis. More details are in the 'Using dataexplore' section. Also see the home page for this application at http://dmnfarrell.github.io/pandastable/

CHAPTER 4

Links

http://openresearchsoftware.metajnl.com/articles/10.5334/jors.94/

http://dmnfarrell.github.io/pandastable/

https://youtu.be/Ss0QIFywt74

CHAPTER 5

Citation

If you use this software in your work please cite the following article:

Farrell, D 2016 DataExplore: An Application for General Data Analysis in Research and Education. Journal of Open Research Software, 4: e9, DOI: http://dx.doi.org/10.5334/jors.94

CHAPTER 6

Installation

6.1 For Dataexplore

For **Windows users** there is an MSI installer for the DataExplore application. This is recommended for anyone using windows not using the library directly as a widget.

On linux snaps are highly recommended:

```
snap install dataexplore
```

6.2 pandastable library

On all operating systems installations of Python should include the pip tool. If not use your distributions package manager to install pip first. Then a simple call as follows should install all dependencies:

```
pip install pandastable
```

This might not work well in some cases because matplotlib has library dependencies that users might find confusing. Though it should work ok on recent versions of Ubuntu. Advice for each OS is given below.

Dependencies

- numpy
- pandas
- matplotlib
- numexpr

Optional dependencies

- statsmodels
- seaborn (requires scipy)

6.3 Linux

For the python linbrary using easy_install or pip should work well but for matplotlib might require more packages such as python headers for compiling the extension. You need the tk8.6-dev package to provide the tkagg backend.

Otherwise, to use the package manager in Ubuntu/Debian based distributions you can issue the command:

```
sudo apt install python-matplotlib
```

You should install pandas with pip as it will provide the most recent version. This will likely be done automatically anyway:

For python 3 installs

You need to use the command pip3 instead if python 2 is also on your system, like in Ubuntu. When installing packages with apt you likely need to specify python 3. e.g. python3-numpy instead of python-numpy.

For python 2.7 ONLY

You will also need the future package. Run *pip install future* to install them. Python 2.6 has NOT been tested and probably won't work.

6.4 Windows

It is much easier to install matplotlib in windows using the binary installer rather than using pip. You can download this [here](http://matplotlib.org/downloads.html). Pick the appropriate file for your python version e.g. 'matplotlib-1.4.3.win32-py3.4.exe' for python 3.4.

pandas should install ok with the pip installer. In windows pip.exe is located under C:Python34Scripts. The future package is needed for python 2.7.

Note that the Python pydata stack can also be installed at once using miniconda, http://conda.pydata.org/miniconda.html. This includes a version of Python itself.

6.5 Mac OSX

There are multiple packaged installers for scientific Python, the best of which is probably anaconda. Miniconda is a smaller version if you don't want all the packages. To use it download and run the Mac OS X installer from http://conda.pydata.org/miniconda.html. The installer will automatically configure your system to use the Anaconda Python. You can then use pip to install the package.

If using macports:

```
sudo port install py34-pip
sudo pip-3.4 install matplotlib numpy pandas numexpr
```

Using the source distribution file

You can download the latest tar.gz file [here](https://github.com/dmnfarrell/pandastable/releases/latest/) and do the following:

```
tar -xzvf pandastable.version.tar.gz
cd pandastable
sudo python3 setup.py install
```

Note that you still need to have installed the dependencies as above.

CHAPTER 7

Using DataExplore

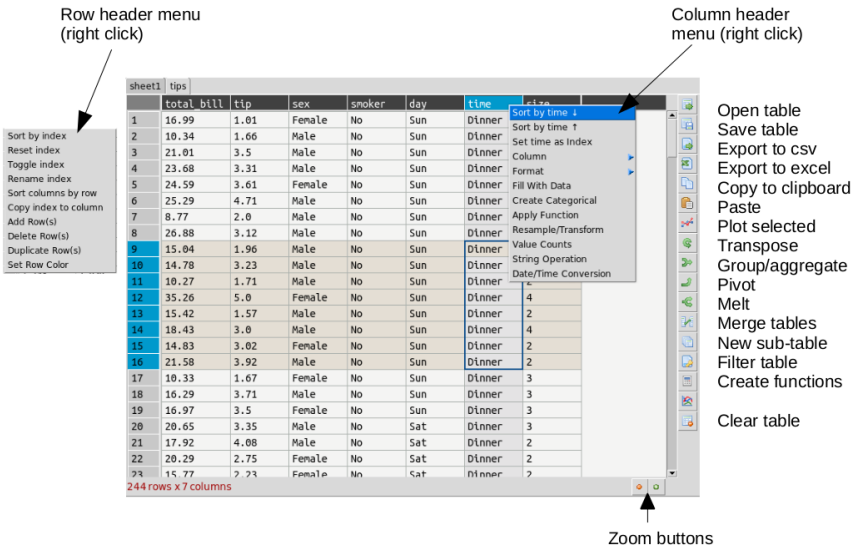
This page details some of the tasks available in dataexplore. For a general introduction also see the screencast at https://youtu.be/Ss0QIFywt74. Most of this functionality is available when you just use the table widget as well as the dataexplore application. Installing in windows or with a snap in linux should provide a menu item to launch the app. Otherwise use the command line, detailed below.

7.1 Purpose of the program

This program is for analyzing tabular data but is not meant to be a spreadsheet. Data is treated in a row/column centric fashion and a lot of the analysis is done in bulk on entire columns at once. So although you can edit cells it is not really meant for data entry. You can use a spreadsheet for that. Cell formulas are not possible for instance. You can however delete rows, columns and clear blocks of cells. New columns can be created through the use of functions. The primary goal is to let users explore their tables interactively without any prior programming knowledge and make interesting plots as they do this. One advantage is the ability to load and work with relatively large tables as compared to spreadsheets. So several million rows should not be a problem and is limited only by your computer memory.

7.2 Table layout

The table is laid out with headers for row and columns. Much functionality can be accessed from the tools menu but also but right clicking on the row and column headers. You can resize columns by dragging in the header. Rows cannot be resized independently (zoom in to enlarge). Unlike spreadsheets column and ro headers can use indexes. You can set any column as an index. This has extra functionality when it comes to plotting.



7.3 Command Line

Launching dataexplore from the command line allows you to provide several options using unix type '-' switches.

Show help:

```
dataexplore -h
```

Open a project file:

```
dataexplore -p <project file>
```

Open a dataframe stored as a messagepack file:

```
dataexplore -f <msgpack file>
```

Open a csv file and try to import it:

```
dataexplore -i <csv file>
```

Launch a basic test table with no plot frame

```
dataexplore -t
```

7.4 Import text files

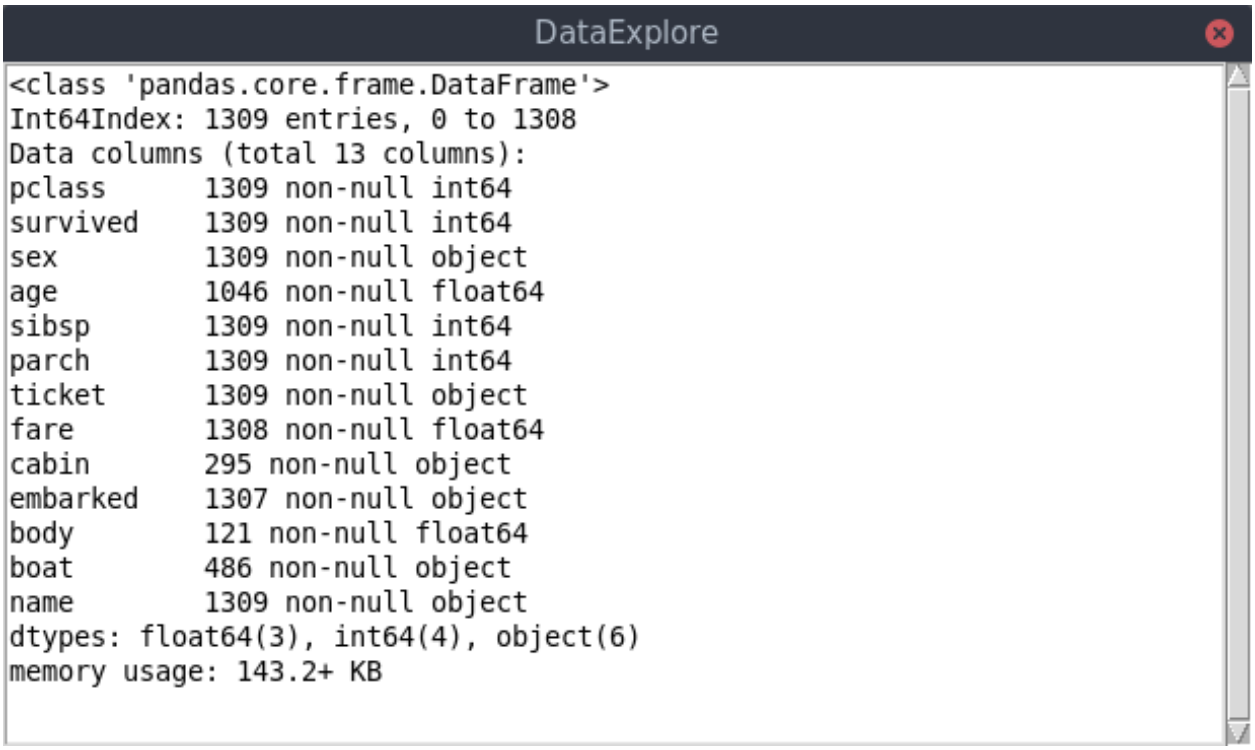
Import of csv and general plain text formats is done from the file menu, toolbar or by right-clicking anywhere in the table and using the context menu. The dialog has most basic options such as delimiter, header selection, comment symbol, rows to skip etc. When you change the import option you can update the preview to see if the new table will look correct. You then press import. Note that it is generally a good idea to remove empty lines and bad data if you can before importing.

7.5 Saving data

Dataexplore projects (multiple groups of sheets with the plot view for each) are saved in **messagepack** format and have the .dexpl file extension. Tables can also be saved on their own as messagepack or pickle files and then opened directly in Python. Using the messagepack format is more efficient than csv as it takes up less space and loads faster. Though quite reliable and efficient, it is not recommended that you use these formats for long term backup, *always keep a copy your raw data* if it is important. Exporting to csv is also possible and saving individual tables to excel.

7.6 Getting table info

The status bar at the bottom left shows the size of the table in rows and columns at all times. For a more detailed summary use Tools->Table info. This brings up a window showing the type of each column and memory usage. 'object' columns are those with text/mixed data and float and int must be numbers only.



7.7 Cleaning data

Pandas supports a variety of options for data 'cleaning' or dealing with missing data. The most basic are available from DataExplore from the main menu.

- Drop rows/columns with missing (empty) data
- Fill missing data with a symbol
- Forward or backfill with neighbouring row values
- Interpolate missing data (filling in the points between)
- Drop duplicates

7.8 String operations

Accessed by right clicking on the column header menu. String operations can be carried out on any column as long as they are object data types and not pure numbers.

The following string methods are supported:

- split, with separator symbol - will create multiple new columns
- strip, remove whitespace
- lower/upper case conversion
- title, convert to TitleCase
- swap case
- get length of string
- concat, concatenate strings in first two cols with given separator
- slice, slice string by start/end indexes
- replace

7.9 Summarizing and grouping data

For overall table statistics you can use the tools->describe table command. For individual columns you can get value counts by right clicking on the header.

The primary way to summarize data is to use the aggregate dialog. It is accessed on the right toolbar. Tables can be grouped and aggregated on multiple columns to create new summary tables. The results will be placed in the sub table below the main one and can then be copied to new sheets. Normally you would group by category columns (rather then a continuous variable like decimal numbers). The dialog has a list of columns to group by and another list box for column(s) to aggregate these groups using one or more functions. See the animated example (click to enlarge):

It is often easiest to test the selections out until you get the required result.

7.10 Merging two tables

Merging tables is done in dataexplore by first putting your second table in the sub-table below. You can do that by pasting it from another sheet or making an empty sub-table and importing. Once this is done you open the merge dialog in the toolbar. You select which columns in each table to merge on (at least one columns should be shared between each). The apply and the result is opened in the dialog to preview. You can copy this to a new sheet.

7.11 Pivoting tables

Pivot tables is an operation some people might be familiar with from excel. A pivot might best be described as way of summarizing data by 'unstacking' the grouped data into new columns. It is a more specialized version of the aggregation method above. A comprehensive explanation is given here: https://www.dataquest.io/blog/pandas-pivot-table/

7.12 Transpose tables

A transpose is rotating the table on its axes so the rows become columns and vice versa. This can be useful for plotting purposes when you want to treat the row data as series. This is illustrated in the animation below where the same table is plotted first with the years as series and then with 'col1' and 'col2' as series and years as data points. Your row index will become the new columns when you transpose, so you should make sure the **correct index is set** beforehand. If you make a mistake you can undo or transpose again to reverse. Note: transposing extremely large tables might be slow.

7.13 Filtering tables

Filtering tables is done using either a string query and/or one or more pre-defined filters defined with widgets.

7.13.1 Query with widgets

Pressing the filtering button will bring up the dialog below the table. Manual predefined filters can be added by pressing the + button. These are used alone or in conjunction with the string query as shown below. The filters are joined together using the first menu item using either 'AND', 'OR' or 'NOT' boolean logic. When filtered results are found the found rows are highlighted. You can also limit the table to show the filtered set which can be treated as usual (i.e. plots made etc). Closing the query box restores the full table. If you want to keep the filtered table you can copy and paste in another sheet.

7.13.2 String query

String based query are made up fairly intuitive expressions. The one caveat is that column names cannot contain spaces to be used in an expression. It is best in these cases to convert column names (i.e. replace spaces with an underscore '_'). You may also use Python/pandas style expressions to perform filters, useful with string based queries.

Examples:

```
x>4 and y<3 #filter by values of columns x and y
x.str.contains("abc") #find only values of column x containing substring #abc
x.str.len()>3 #find only rows where length of strings in x is greater than 3
```

7.14 Applying functions

Unlike a spreadsheet there are no cell based formulas. Rather functions are applied to columns over all rows, creating a new column. New columns can be created in several ways through computations on other columns. The column header menu provides some of these like resample/transform a column or the apply function dialog. Another more general way to add functions is to use the calculation button on the toolbar. This brings up a dialog below the table where you can type function as text expressions.

The same as for filtering, a string is entered like a formula and if it can be parsed a new column is created. For example entering 'x = a + b' will create a new column x that is the sum of a and b.

Examples:

```
x = a+b  #sum a and b
x = a*a  #a squared
x = sin(a)
x = sqrt(a+b)/log(c)
```

Supported functions in expressions: sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, log, log10, exp

7.15 Converting column names

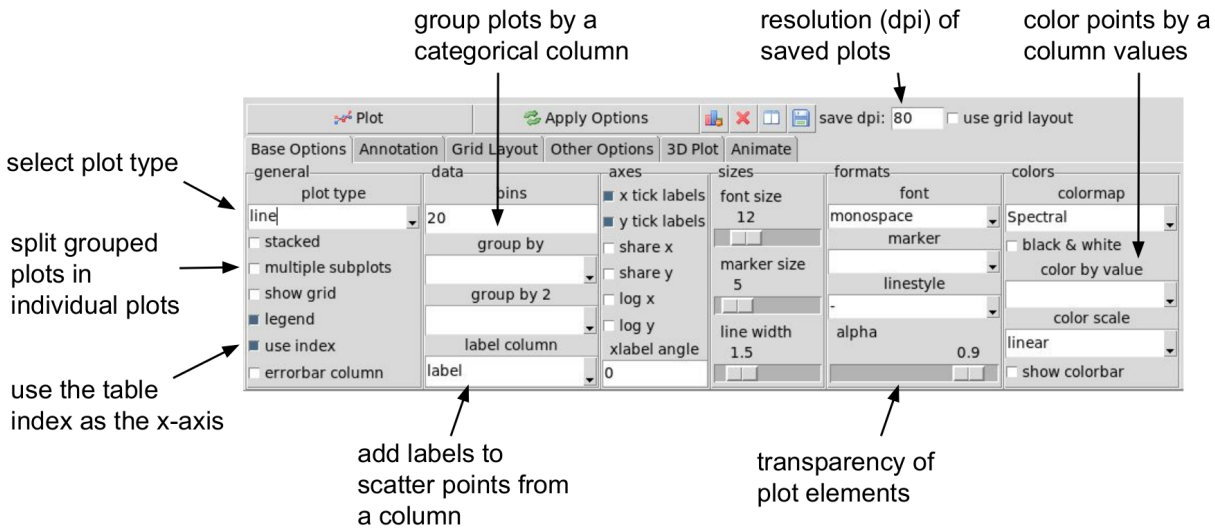
It may sometimes be necessary to re-format column names, for example to remove unwanted characters. If you have dozens or more columns this would be time consuming, so there is a function in dataexplore to do this in one step. Accessed from Tools->Convert column names, this dialog allows you to replace characters e.g. replace spaces with '_' symbol. You can also convert cases.

7.16 Resampling columns

Resampling is a way to average data over specific windows or periods. It is a possible way to smooth out noisy data for example or get an average trend. You can resample columns from the column header menu. In the example below this is used to smooth out the sawtooth shaped C02 data. The larger the window the more averaging will take place.

7.17 Plot options

The plot frame has an options dialog underneath with multiple tabs grouped by functionality. Most default formatting options such as the type of plot, whether to show a legend etc. are in the first tab. The dialogs may look a bit cluttered for some users but the idea is to have all available options quickly accessible rather than hidden in menus. If you use the program regularly you will be familiar with where things are. Some of the less obvious options are explained in the image below.



The following plot types are currently supported:

- line
- scatter
- bar
- barh
- pie
- histogram
- box plot
- dot plot
- heatmap
- area
- hexbin
- contour
- scatter matrix
- venn

Other tabs contain options for grid layouts, text annotation such as titles and text boxes, and access to the plot animation settings.

7.18 Plotting grouped data

Rather than grouping the table directly it is also possible to plot data grouped. This requires you select the appropriate columns including the one to be grouped by and select the grouping column in the 'groupby' menu in the plot options. Plots can be grouped by 2 columns at once.

7.19 Plotting in a grid

The gif animation below shows how to use the grid layout tool to generate subplots by clicking and dragging in the grid to select the area for your next plot. Note that subplots will be overwritten if you select the same cell as one currently occupied but if you drag over this cell the region will be plotted over. The tool assumes the user will know how to avoid overlaps. So it's best to have a good idea of how to layout the plots beforehand, or just use trial and error. You can remove subplots from the drop down menu, listed according to their positions.

Grid layout includes other modes 'split data and 'multiview'. Split data lets you pick a grid size and splits up the rows into chunks, plotting each separately. The multiview mode allows you to auto-generate different kinds of plot in the grid for the same data every time you plot. This could be useful for quickly previewing regions of data repeatedly without having to set the plot type each time. This will overwrite whatever plot you currently have displayed. The feature is also illustrated in the gif above.

7.20 Animated plots

Plots can be animated and save as video files using the plot animation options tab. This would mostly be useful for time series based line plots but any kinds of plots can be animated. Formatting can be changed or column selections

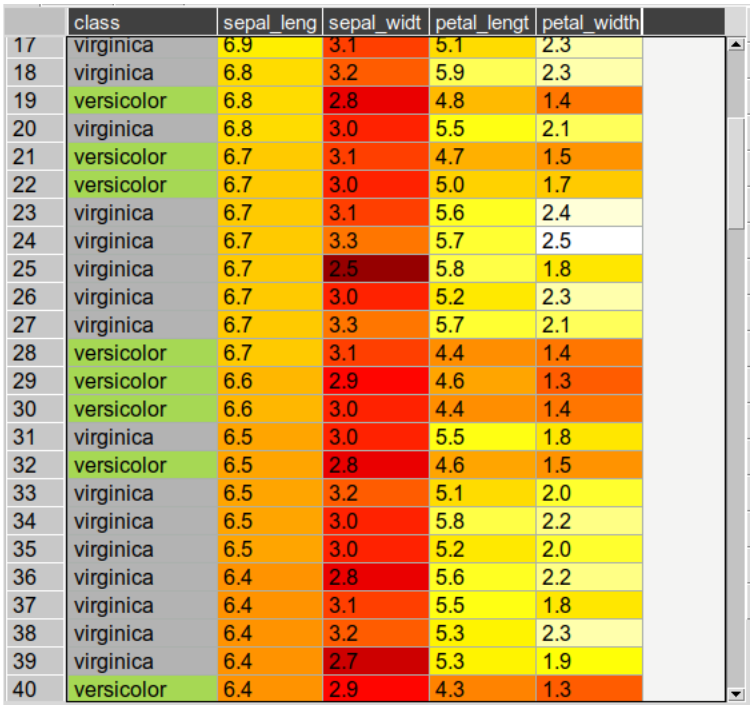
altered as the plot is updating, leading to some odd plot displays.

see http://dmnfarrell.github.io/dataexplore/2018/05/15/animation

7.21 Table Coloring

Column colors can be set by right clicking in the column header and choosing 'set color'. A color picker will open. The formatting is saved with the project file. You can clear the formatting from the table popup menu.

You can set row and cell colors in several ways. Firstly, if right clicking on the row header or inside the table the 'set color' option lets you color the selected rows/columns with a single color. You can also set colors for the entire table/column according to the cell values. This is done by choosing 'color by value' from the column header and will allow you to select a color map. String values will be mapped to categorical integers and then to colors. See below:



	class	sepal_leng	sepal_widt	petal_lengt	petal_width
17	virginica	6.9	3.1	5.1	2.3
18	virginica	6.8	3.2	5.9	2.3
19	versicolor	6.8	2.8	4.8	1.4
20	virginica	6.8	3.0	5.5	2.1
21	versicolor	6.7	3.1	4.7	1.5
22	versicolor	6.7	3.0	5.0	1.7
23	virginica	6.7	3.1	5.6	2.4
24	virginica	6.7	3.3	5.7	2.5
25	virginica	6.7	2.5	5.8	1.8
26	virginica	6.7	3.0	5.2	2.3
27	virginica	6.7	3.3	5.7	2.1
28	versicolor	6.7	3.1	4.4	1.4
29	versicolor	6.6	2.9	4.6	1.3
30	versicolor	6.6	3.0	4.4	1.4
31	virginica	6.5	3.0	5.5	1.8
32	versicolor	6.5	2.8	4.6	1.5
33	virginica	6.5	3.2	5.1	2.0
34	virginica	6.5	3.0	5.8	2.2
35	virginica	6.5	3.0	5.2	2.0
36	virginica	6.4	2.8	5.6	2.2
37	virginica	6.4	3.1	5.5	1.8
38	virginica	6.4	3.2	5.3	2.3
39	virginica	6.4	2.7	5.3	1.9
40	versicolor	6.4	2.9	4.3	1.3

For very large tables, adding colors for all cells will increases the file size of saved projects.

7.22 Setting preferences

Preferences for table formatting can be set from the edit->preferences menu item. This uses a text configuration file stored in ~/.dataexplore/default.conf. The preferences dialog is used to apply the settings to the current table and/or save them to this file. This file can be edited manually in a text editor if you wish. Any new tables will use these settings. The file looks like this:

```
[base]
align = w
cellwidth = 80
floatprecision = 2
font = Arial
fontsize = 12
```

(continues on next page)

(continued from previous page)

```
linewidth = 1
rowheight = 22

[colors]
cellbackgr = #F4F4F3
grid_color = #ABB1AD
rowselectedcolor = #E4DED4
textcolor = black
```

7.23 Batch processing

A plugin provides the ability to batch import and/or plot multiple files at once. This is generally designed for many similarly formatted files that you wish to clean or plot in bulk without loading each individually. You can also use this to join many files into one table. Access this tool from Plugins->Batch Process.

7.24 Other examples

Other guides are available as blog posts:

- http://dmnfarrell.github.io/dataexplore/titanic-example
- http://dmnfarrell.github.io/dataexplore/grouped-plots
- http://dmnfarrell.github.io/dataexplore/sea-ice-example

CHAPTER 8

Code Examples

This section is for python programmers you want to use the table widget in their own programs.

8.1 Basics

Create a parent frame and then add the table to it:

```
from tkinter import *
from pandastable import Table
#assuming parent is the frame in which you want to place the table
pt = Table(parent)
pt.show()
```

Update the table:

```
#alter the DataFrame in some way, then update
pt.redraw()
```

Import a csv file:

```
pt.importCSV('test.csv')
```

A class for launching a basic table in a frame:

```
from tkinter import *
from pandastable import Table, TableModel, config

class TestApp(Frame):
        """Basic test frame for the table"""
        def __init__(self, parent=None):
            self.parent = parent
            Frame.__init__(self)
            self.main = self.master
```

(continues on next page)

(continued from previous page)

```
            self.main.geometry('600x400+200+100')
            self.main.title('Table app')
            f = Frame(self.main)
            f.pack(fill=BOTH,expand=1)
            df = TableModel.getSampleData()
            self.table = pt = Table(f, dataframe=df,
                                    showtoolbar=True, showstatusbar=True)
                        pt.show()
                        #set some options
                        options = {'colheadercolor':'green','floatprecision': 5}
                        config.apply_options(options, pt)
            pt.show()
            return

app = TestApp()
#launch the app
app.mainloop()
```

8.2 Sub-class the Table

Override the right click popup menu:

```
class MyTable(Table):
    """Custom table class inherits from Table. You can then override required methods"
↪""
    def __init__(self, parent=None, **kwargs):
        Table.__init__(self, parent, **kwargs)
        return

          def handle_left_click(self, event):
        """Example - override left click"""

        Table.handle_left_click(self, event)
  #do custom code here
        return

    def popupMenu(self, event, rows=None, cols=None, outside=None):
        """Custom right click menu"""

        popupmenu = Menu(self, tearoff = 0)
        def popupFocusOut(event):
            popupmenu.unpost()
                                # add commands here
    # self.app is a reference to the parent app
        popupmenu.add_command(label='do stuff', command=self.app.stuff)
        popupmenu.bind("<FocusOut>", popupFocusOut)
        popupmenu.focus_set()
        popupmenu.post(event.x_root, event.y_root)
        return popupmenu
```

8.3 Table methods

You can use the Table class methods to directly access data and perform many more functions. Check the API for all the methods. Some examples are given here:

```
#add 10 empty columns
table.autoAddColumns(10)
#resize the columns to fit the data better
table.autoResizeColumns()
#clear the current formatting
table.clearFormatting()
#reduce column witdths proportionally
table.contractColumns()
#get selected column
table.getSelectedColumn()
#sort by column index 0
table.sortTable(0)
#enlarge all table elements
table.zoomIn()
#set row colors
table.setRowColors(rows=range(2,100,2), clr='lightblue', cols='all')
#delete selected rows
table.setSelectedRows([[4,6,8,10]])
#delete current row
table.deleteRow()
#set current row
table.setSelectedRow(10)
#insert below current row
table.insertRow()
```

8.4 Accessing and modifying data directly

The tables use a pandas DataFrame object for storing the underlying data. If you are not familiar with pandas you should learn the basics if you need to access or manipulate the table data. See http://pandas.pydata.org/pandas-docs/stable/10min.html

Each table has an object called model with has the dataframe inside it. The dataframe is referred to as df. So to access the data on a table you can use:

```
df = table.model.df
```

Examples of simple dataframe operations. Remember when you update the dataframe you will need to call table.redraw() to see the changes reflected:

```
df.drop(0) #delete column with this index
df.T #transpose the DataFrame
df.drop(columns=['x'])
```

8.5 Set table attributes

You can set table attributes directly such as these examples:

```
            table.textcolor = 'blue'
            table.cellbackgr = 'white'
            table.boxoutlinecolor = 'black'
            #set header colors
            self.table.rowheader.bgcolor = 'orange'
            self.table.colheader.bgcolor = 'lightgreen'
            self.table.colheader.textcolor = 'black'
            #make editable or not
table.editable = False
```

8.6 Set Preferences

Preferences are normally loaded from a configuration file that can be edited manually or via the menu. You can also programmatically set these preferences using the config module:

```
#load from a config file if you need to (done by default when tables are created)
options = config.load_options()
#options is a dict that you can set yourself
options = {'floatprecision': 2}
config.apply_options(options, table)
```

You can set the following configuration values:

```
{'align': 'w',
 'cellbackgr': '#F4F4F3',
 'cellwidth': 80,
 'floatprecision': 2,
 'thousandseparator': '',
 'font': 'Arial',
 'fontsize': 12,
 'fontstyle': '',
 'grid_color': '#ABB1AD',
 'linewidth': 1,
 'rowheight': 22,
 'rowselectedcolor': '#E4DED4',
 'textcolor': 'black'}
```

8.7 Table Coloring

You can set column colors by setting the key in the columncolors dict to a valid hex color code. Then just redraw:

```
table.columncolors['mycol'] = '#dcf1fc' #color a specific column
table.redraw()
```

You can set row and cell colors in several ways. `table.rowcolors` is a pandas dataframe that mirrors the current table and stores a color for each cell. It only adds columns as needed. You can update this manually but it is easiest to use the following methods:

```
table.setRowColors(rows, color) #using row numbers
table.setColorByMask(column, mask, color) #using a pre-defined mask
table.redraw()
```

To color by column values:

```
table.multiplecollist = [cols] #set the selected columns
table.setColorbyValue()
table.redraw()
```

To clear formatting:

```
table.clearFormatting()
table.redraw()
```

Note: You should generally use a simple integer index for the table when using colors as there may be inconsistencies otherwise.

8.8 Writing DataExplore Plugins

Plugins are for adding custom functionality that is not present in the main application. They are implemented by sub-classing the Plugin class in the plugin module. This is a python script that can generally contain any code you wish. Usually the idea will be to implement a dialog that the user interacts with. But this could also be a single method that runs on the current table or all sheets at once.

8.8.1 Implementing a plugin

Plugins should inherit from the Plugin class. Though this is not strictly necessary for the plugin to function.

```
from pandastable.plugin import Plugin
```

You can simply copy the example plugin to get started. All plugins need to have a *main()* method which is called by the application to launch them. By default this method contains the *_doFrame()* method which constructs a main frame as part of the current table frame. Usually you override main() and call _doFrame then add your own custom code with your widgets.

_doFrame method has the following lines which are always needed unless it is a non GUI plugin:

```
self.table = self.parent.getCurrentTable() #get the current table
#add the plugin frame to the table parent
self.mainwin = Frame(self.table.parentframe)
#pluginrow is 6 to make the frame appear below other widgets
self.mainwin.grid(row=pluginrow,column=0,columnspan=2,sticky='news')
```

You can also override the quit() and about() methods.

8.8.2 Non-table based plugins

Plugins that don't rely on using the table directly do not need to use the above method and can have essentially anything in them as long as there is a main() method present. The Batch File Rename plugin is an example. This is a standalone utility launched in a separate toplevel window.

see https://github.com/dmnfarrell/pandastable/blob/master/pandastable/plugins/rename.py

8.9 Freezing the app

Dataexplore is available as an exe with msi installer for Windows. This was created using the cx_freeze package. For anyone wishing to freeze their tkinter app some details are given here. This is a rather hit and miss process as it seems

to depend on your installed version of Python. Even when the msi/exe builds you need to check for runtime issues on another copy of windows to make sure it's working. Steps:

- Use a recent version of python (>=3.6 recommended) installed as normal and then using pip to install the dependencies that you normally need to run the app.
- The freeze script is found in the main pandastable folder, freeze.py. You can adopt it for your own app.
- Run the script using *python freeze.py bdist_msi*
- The resulting msi is placed in the dist folder. This is a 32 bit binary but should run fine on windows 10.

You can probably use Anaconda to do the same thing but we have not tested this.

CHAPTER 9

pandastable

9.1 pandastable package

9.1.1 Submodules

9.1.2 pandastable.annotation module

9.1.3 pandastable.app module

9.1.4 pandastable.core module

Implements the core pandastable classes. Created Jan 2014 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.core.**ChildToolBar**(*parent=None*, *parentapp=None*)

Bases: *pandastable.core.ToolBar*

Smaller toolbar for child table

class pandastable.core.**Table**(*parent=None*, *model=None*, *dataframe=None*, *width=None*, *height=None*, *rows=20*, *cols=5*, *showtoolbar=False*, *showstatusbar=False*, *editable=True*, *enable_menus=True*, ***kwargs*)

Bases: tkinter.Canvas

A tkinter class for providing table functionality.

Parameters

- **parent** – parent Frame
- **model** – a TableModel with some data
- **dataframe** – a pandas DataFrame
- **width** – width of frame
- **height** – height of frame
- **rows** – number of rows if creating empty table
- **cols** – number of columns if creating empty table
- **showtoolbar** – whether to show the toolbar, default False
- **showstatusbar** – whether to show the statusbar

addColumn(*newname=None*)
Add a new column

addRows(*num=None*)
Add new rows

adjustColumnWidths(*limit=30*)
Optimally adjust col widths to accomodate the longest entry in each column - usually only called on first redraw. :param limit: max number of columns to resize

aggregate()
Show aggregate dialog

applyColumnFunction(*evt=None*)
Apply column wise functions, applies a calculation per row and ceates a new column.

applyStringMethod()
Apply string operation to column(s)

applyTransformFunction(*evt=None*)
Apply resampling and transform functions on a single column.

autoAddColumns(*numcols=None*)
Automatically add x number of cols

autoResizeColumns()
Automatically set nice column widths and draw

checkDataEntry(*event=None*)
do validation checks on data entry in a widget

cleanData()
Deal with missing data

clearData(*evt=None*)
Delete cells from gui event

clearFormatting()

clearSelected()
Clear selections

clearTable()
Make an empty table

close(*evt=None*)

closeChildTable()
Close the child table

colorColumns(*cols=None*, *color='gray'*)
Color visible columns

colorRows()
Color individual cells in column(s). Requires that the rowcolors dataframe has been set. This needs to be updated if the index is reset

contractColumns(*factor=10*)
Reduce column widths

convertColumnNames(*s='_'*)
Convert col names so we can use numexpr

convertDates()
Convert single or multiple columns into datetime

convertNumeric()
Convert cols to numeric if possible

copy(*rows*, *cols=None*)
Copy cell contents from clipboard - overwrites table.

copyColumn()
Copy a column

copyIndex()
Copy index to a column

copyTable(*event=None*)
Copy from the clipboard

corrMatrix()
Correlation matrix

createCategorical()
Get a categorical column from selected

createChildTable(*df*, *title=None*, *index=False*, *out=False*)
Add the child table

crosstab()
Cross tabulation

deleteCells(*rows*, *cols*, *answer=None*)
Clear the cell contents

deleteColumn(*ask=True*)
Delete currently selected column(s)

deleteRow(*ask=False*)
Delete a selected row

describe()
Create table summary

doBindings()
Bind keys and mouse clicks, this can be overriden

doCombine()
Do combine/merge operation

doExport(*filename=None*)
Do a simple export of the cell contents to csv

drawCellEntry(*row*, *col*, *text=None*)
When the user single/double clicks on a text/number cell, bring up entry window and allow edits.

drawGrid(*startrow*, *endrow*)
Draw the table grid lines

drawHighlighted()
Color an arbitrary selection of cells. Set the 'highlighted' attribute which is a masked dataframe of the table.

drawMultipleCells()
Draw an outline box for multiple cell selection

drawMultipleCols()
Draw multiple column selections

drawMultipleRows(*rowlist*)
Draw more than one row selection

drawRect(*row*, *col*, *color=None*, *tag=None*, *delete=1*)
Cell is colored

drawRowHeader()
User has clicked to select a cell

drawSelectedCol(*col=None*, *delete=1*, *color=None*, *tag='colrect'*)
Draw a highlight rect for the current column selection

drawSelectedRect(*row*, *col*, *color=None*, *fillcolor=None*)
User has clicked to select a cell

drawSelectedRow()
Draw a highlight rect for the currently selected rows

drawText(*row*, *col*, *celltxt*, *align=None*, *single_line=True*)
Draw the text inside a cell area

duplicateRows()
Make copy of rows

evalBar(*evt=None*)
Use pd.eval to apply a function colwise or preset funcs.

evalFunction(*evt=None*)
Apply a function to create new columns

expandColumns(*factor=10*)
Reduce column widths

fillAcross(*collist*, *rowlist*)
Fill across a row, or multiple rows

fillColumn()
Fill a column with a data range

fillDown(*rowlist*, *collist*)
Fill down a column, or multiple columns

findDuplicates()
Find duplicate rows

findText(*evt=None*)
Simple text search in whole table

flattenIndex()
Flatten multiindex

functionsBar(*evt=None*)
Apply python functions from a pre-defined set, this is for stuff that can't be done with eval strings

getCanvasPos(*row*, *col*)
Get the cell x-y coords as a fraction of canvas size

getCellCoords(*row*, *col*)
Get x-y coordinates to drawing a cell in a given row/col

getColPosition(*x*)
Get column position at coord

getFonts()

getGeometry(*frame*)
Get frame geometry

getPlotData()
Plot data from selection

getRowPosition(*y*)
Set row position

getRowsFromIndex(*idx=None*)
Get row positions from index values

getRowsFromMask(*mask*)

getScale()

getSelectedColumn()
Get currently selected column

getSelectedDataFrame()
Return a sub-dataframe of the selected cells. Will try to convert object types to float so that plotting works.

getSelectedRow()
Get currently selected row

getSelectedRowData()
Return a sub-dataframe of the selected rows

getSelectionValues()
Get values for current multiple cell selection

getVisibleCols(*x1*, *x2*)
Get the visible column range

getVisibleRegion()
Get visible region of canvas

getVisibleRows(*y1*, *y2*)
Get the visible row range

get_col_clicked(*event*)
Get column where event on the canvas occurs

get_memory()
memory usage of current table

get_row_clicked(*event*)
Get row where event on canvas occurs

gotonextCell()
Move highlighted cell to next cell in row or a new col

gotonextRow()
Programmatically set next row - eg. for button events

gotoprevRow()
Programmatically set previous row - eg. for button events

groupby(*colindex*)
Group by

handleCellEntry(*row*, *col*)
Callback for cell entry

handleEntryMenu(**args*)
Callback for option menu in categorical columns entry

handle_arrow_keys(*event*)
Handle arrow keys press

handle_double_click(*event*)
Do double click stuff. Selected row/cols will already have been set with single click binding

handle_left_click(*event*)
Respond to a single press

handle_left_ctrl_click(*event*)
Handle ctrl clicks for multiple row selections

handle_left_release(*event*)
Handle left mouse button release event

handle_left_shift_click(*event*)
Handle shift click, for selecting multiple rows

handle_mouse_drag(*event*)
Handle mouse moved with button held down, multiple selections

handle_right_click(*event*)
respond to a right click

hidePlot()
Hide plot frame

hideRowHeader()
Hide the row header, must have run show() first

importCSV(*filename=None*, *dialog=False*, ***kwargs*)
Import from csv file

importHDF(*filename=None*, *dialog=False*, ***kwargs*)

insertRow()
Insert a new row

isInsideTable(*x*, *y*)
Returns true if x-y coord is inside table bounds

load(*filename=None*)
load from a file

loadExcel(*filename=None*)
Load excel file

loadPrefs(*prefs=None*)
Load preferences from defaults

melt()
Melt table

merge(*table*)
Merge with another table.

mouse_wheel(*event*)
Handle mouse wheel scroll for windows

moveColumns(*names=None*, *pos='start'*)
Move column(s) to start/end, used for large tables

movetoSelection(*row=None*, *col=0*, *idx=None*, *offset=0*)
Move to a specific row/col, updating table

new()
Clears all the data and makes a new table

paste(*event=None*)
Paste a new table from the clipboard

pivot()
Pivot table

placeColumn(*col1*, *col2*)
Move col1 next to col2, useful for placing a new column made from the first one next to it so user can see it easily

plot3D()

plotSelected()
Plot the selected data in the associated plotviewer

popupMenu(*event*, *rows=None*, *cols=None*, *outside=None*)
Add left and right click behaviour for canvas, should not have to override this function, it will take its values from defined dicts in constructor

query(*evt=None*)
Do query

queryBar(*evt=None*)
Query/filtering dialog

recalculateFunctions(*omit=None*)
Re evaluate any columns that were derived from functions and dependent on other columns (except self derived?)

redraw(*event=None*, *callback=None*)
Redraw table

redrawCell(*row=None*, *col=None*, *recname=None*, *colname=None*)
Redraw a specific cell only

redrawVisible(*event=None*, *callback=None*)
Redraw the visible portion of the canvas. This is the core redraw method. Refreshes all table elements. Called by redraw() method as shorthand.

Parameters

- **event** – tkinter event to trigger method, default None
- **callback** – function to be called after redraw, default None

remove()
Close table frame

renameIndex()
Rename the row index

resample()
Table time series resampling dialog. Should set a datetime index first.

resetColors()

resetIndex(*ask=True*, *drop=False*)
Reset index and redraw row header

resizeColumn(*col*, *width*)
Resize a column by dragging

resized(*event*)
Check if size changed when event triggered to avoid unnecessary redraws

save()
Save current file

saveAs(*filename=None*)
Save dataframe to file

selectAll(*evt=None*)
Select all rows and cells

selectNone()
Deselect current, called when table is redrawn with completely new cols and rows e.g. after model is updated.

setAlignment(*colnames=None*)
Set column alignments, overrides global value

setColPositions()
Determine current column grid positions

setColorByMask(*col*, *mask*, *clr*)
Color individual cells in a column using a mask.

setColorbyValue()
Set row colors in a column by values

setColumnColors(*cols=None*, *clr=None*)
Set a column color and store it

setColumnType()
Change the column dtype

setFont()
Set font tuple

setPrecision(*x*, *p*)
Set precision of a float value

setRowColors(*rows=None*, *clr=None*, *cols=None*)
Set rows color from menu. :param rows: row numbers to be colored :param clr: color in hex :param cols: column numbers, can also use 'all'

setRowHeight(*h*)
Set the row height

setSelectedCells(*startrow*, *endrow*, *startcol*, *endcol*)
Set a block of cells selected

setSelectedCol(*col*)
Set currently selected column

setSelectedRow(*row=None*)
Set currently selected row and reset multiple row list

setSelectedRows(*rows*)

setTheme(*name='light'*)
Set theme

setWrap()
Toogle column header wrap

set_defaults()
Set default settings

set_rowcolors_index()

set_xviews(**args*)
Set the xview of table and col header

set_yviews(**args*)
Set the xview of table and row header

setcellbackgr()

setgrid_color()

setindex()
Set indexes

setrowselectedcolor()
Set selected row color

show(*callback=None*)
Adds column header and scrollbars and combines them with the current table adding all to the master frame provided in constructor. Table is then redrawn.

showAll()
Re-show unfiltered

showIndex()
Show the row index

showInfo()
Show dataframe info

showPlot()

showPlotViewer(*parent=None*)
Create plot frame

showPreferences()
Preferences dialog

showRowHeader()
Show the row header if hidden, must have run show() first

show_progress_window(*message=None*)
Show progress bar window for loading of data

showasText()
Get table as formatted text - for printing

sortColumnIndex()
Sort the column header by the current rows values

sortTable(*columnIndex=None*, *ascending=1*, *index=False*)
Sort rows based on currently selected columns

statsViewer()
Show model fitting dialog

storeCurrent()
Store current version of the table before a major change is made

tableChanged()
Callback to be used when dataframe changes so that other widgets and data can be updated

tableFromSelection()
Create a new table from the selected cells

transform()
Apply element-wise transform

transpose()
Transpose table

undo(*event=None*)
Undo last major table change

updateFunctions()
Remove functions if a column has been deleted

updateModel(*model=None*)
Should call this method when a new table model is loaded. Recreates widgets and redraws the table.

updateWidgets()
Update some dialogs when table changed

update_rowcolors()
Update row colors if present so that it syncs with current dataframe.

valueCounts()
Value counts for column(s)

values_to_colors(*x*, *cmap='jet'*, *alpha=1*)
Convert columnn values to colors

zoomIn()
Zoom in, increases font and row heights.

zoomOut()
Zoom out, decreases font and row heights.

class pandastable.core.**ToolBar**(*parent=None*, *parentapp=None*)
Bases: tkinter.ttk.Frame

Uses the parent instance to provide the functions

class pandastable.core.**statusBar**(*parent=None*, *parentapp=None*)
Bases: tkinter.ttk.Frame

Status bar class

update()
Update status bar

9.1.5 pandastable.data module

Module implementing the Data class that manages data for it's associated PandasTable.

Created Jan 2014 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.data.**TableModel**(*dataframe=None*, *rows=20*, *columns=5*)
Bases: object

A data model for the Table class that uses pandas

Parameters

- **dataframe** – pandas dataframe
- **rows** – number of rows if empty table
- **columns** – number of columns if empty table

addColumn(*colname=None*, *dtype=None*, *data=None*)
Add a column

autoAddRows(*num*)
Add n rows to end of dataframe. Will create rows with index starting from highest previous row count

copyIndex()
Copy index to a column

deleteCells(*rows*, *cols*)

deleteColumn(*colindex*)
delete a column

deleteColumns(*cols=None*)
Remove all cols or list provided

deleteRow(*row*, *unique=True*)
Delete a row

deleteRows(*rowlist=None*, *unique=True*)
Delete multiple or all rows

filterby()

getColumnCount()
Returns the number of columns in the data model

getColumnName(*columnIndex*)
Returns the name of the given column by columnIndex

getColumnType(*columnIndex*)
Get the column type

classmethod getIrisData()
Get iris dataset

getRecordAtRow(*rowindex*)
Get the entire record at the specifed row

getRowCount()
Returns the number of rows in the table model.

classmethod getSampleData(*rows=400*, *cols=5*, *n=2*)
Generate sample data :param rows: no. of rows :param cols: columns :param n: length of column names

classmethod getStackedData()
Get a dataframe to pivot test

getValueAt(*row*, *col*)
Returns the cell value at location specified by columnIndex and rowIndex.

getlongestEntry(*colindex*, *n=500*)
Get the longest string in the column for determining width. Just uses the first n rows for speed

groupby(*cols*)
Group by cols

initialiseFields()
Create meta data fields

insertRow(*row*)
Inserts a row at the required index by append/concat

keywords = {'colors': 'colors'}

load(*filename*, *filetype=None*)
Load file, if no filetype given assume it's pickle format

moveColumn(*oldindex*, *newindex*)
Changes the order of columns

query()

resetIndex(*drop=False*)
Reset index behaviour

save(*filename*)
Save dataframe

setValueAt(*value*, *row*, *col*, *df=None*)
Change dataframe according to row/col numbers. You can also pass an arbitrary dataframe here.

setindex(*colindex*)
Index setting behaviour

setup(*dataframe*, *rows=20*, *columns=5*)
Create table model

transpose()
Transpose dataframe

9.1.6 pandastable.dialogs module

Dialog classes. Created Oct 2014 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.dialogs.**AggregateDialog**(*parent=None*, *df=None*)
Bases: *pandastable.dialogs.BaseDialog*

Provides a frame for split-apply-combine operations

apply()
Apply operation

copyResult()

createWidgets(*m*)
Create a set of grp-agg-func options together

help()
Help button code

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

class pandastable.dialogs.**AutoScrollbar**(*master=None*, ***kw*)
Bases: tkinter.ttk.Scrollbar

A scrollbar that hides itself if it's not needed. only works if you use the grid geometry manager.

pack(***kw*)

place(***kw*)

set(*lo*, *hi*)
Set the fractional values of the slider position (upper and lower ends as value between 0 and 1).

class pandastable.dialogs.**BaseDialog**(*parent=None*, *df=None*, *title='*)
Bases: tkinter.ttk.Frame

Generic dialog - inherit from this and customise the createWidgets and apply methods.

apply()
Code to run when Apply is pressed

buttonsFrame()

createWidgets(*m*)
Override this

help()
Help button code

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

class pandastable.dialogs.**BaseTable**(*parent=None*, *width=280*, *height=190*, *rows=2*, *cols=2*, ***kwargs*)

Bases: tkinter.Canvas

Basic table class based on tk canvas. inherit from this to add your own functionality

doBindings()

drawGrid()

drawMultipleCells(*rows*, *cols*)
Draw more than one row selection

drawSelectedRect(*row*, *col*, *color='#c2c2d6'*, *pad=4*, *tags=''*)
User has clicked to select area

getCellCoords(*row*, *col*)
Get x-y coordinates to drawing a cell in a given row/col

get_col_clicked(*event*)
Get column where event on the canvas occurs

get_row_clicked(*event*)
Get row where event on canvas occurs

handle_left_click(*event*)
Respond to a single press

handle_mouse_drag(*event*)
Handle mouse moved with button held down, multiple selections

redraw()

update()
Enter event loop until all pending events have been processed by Tcl.

class pandastable.dialogs.**CombineDialog**(*parent=None*, *df1=None*, *df2=None*)

Bases: tkinter.ttk.Frame

Provides a frame for setting up merge/combine operations

apply()
Apply operation

getResult(*df*)
Show result of merge and let user choose to replace current table

help()

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

replaceTable()
replace parent table

class pandastable.dialogs.**CrosstabDialog**(*parent=None*, *df=None*, *title=''*)

Bases: *pandastable.dialogs.BaseDialog*

apply()
Apply crosstab

createWidgets(*m*)
Create a set of grp-agg-func options together

help()
Help button code

class pandastable.dialogs.**EasyListbox**(*parent*, *width*, *height*, *yscrollcommand*, *listItemSelected*)

Bases: tkinter.Listbox

Customised list box to replace useless default one

clear()
Deletes all items from the list box.

getIndex(*item*)
Returns the index of item if it's in the list box.

getSelectedIndex()
Returns the index of the selected item or -1 if no item is selected.

getSelectedItem()
Returns the selected item or the empty string if no item is selected.

setSelectedIndex(*index*)
Selects the item at the index if it's in the range.

triggerListItemSelected(*event*)
Strategy method to respond to an item selection in the list box. Runs the client's listItemSelected method with the selected index.

class pandastable.dialogs.**FilterBar**(*parent*, *parentframe*, *cols*)

Bases: tkinter.ttk.Frame

Class providing filter widgets

booleanops = ['AND', 'OR', 'NOT']

close()
Destroy and remove from parent

getFilter()
Get filter values for this instance

operators = ['contains', 'excludes', 'equals', 'not equals', '>', '<', 'is empty', 'no

update(*cols*)
Enter event loop until all pending events have been processed by Tcl.

class pandastable.dialogs.**FindReplaceDialog**(*table*)

Bases: tkinter.ttk.Frame

Find/replace dialog.

clear()

close()

find()
Do string search. Creates a masked dataframe for results and then stores each cell coordinate in a list.

findAll()
Highlight all found cells

findNext()
Show next cell of search results

replace()
Replace all instances of search text

setup()

updated(*name=''*, *index=''*, *mode=''*)
Widgets changed so run search again

class pandastable.dialogs.**ImportDialog**(*parent=None*, *filename=None*)
Bases: tkinter.ttk.Frame

Provides a frame for figure canvas and MPL settings

doImport()
Do the import

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

showText(*encoding='utf-8'*)
Show text contents

update()
Reload previews

class pandastable.dialogs.**MultipleValDialog**(*parent*, *title=None*, *initialvalues=None*, *labels=None*, *types=None*, *tooltips=None*, *width=14*, ***kwargs*)
Bases: tkinter.simpledialog.Dialog

Simple dialog to get multiple values

apply()
process the data

This method is called automatically to process the data, *after* the dialog is destroyed. By default, it does nothing.

body(*master*)
create dialog body.

return widget that should have initial focus. This method should be overridden, and is called by the __init__ method.

getResults(*null=None*)
Return a dict of options/values

class pandastable.dialogs.**Progress**(*parent*, *side='left'*)
Bases: object

threaded progress bar for tkinter gui

pb_clear()
stops the progress bar

pb_complete()
stops the progress bar and fills it

pb_start()
starts the progress bar

pb_stop()
stops the progress bar

class pandastable.dialogs.**ProgressDialog**
Bases: tkinter.Toplevel

class pandastable.dialogs.**QueryDialog**(*table*)
Bases: tkinter.ttk.Frame

Use string query to filter. Will not work with spaces in column names, so these would need to be converted first.

addFilter()
Add a filter using widgets

applyFilter(*df*, *mask=None*)
Apply the widget based filters, returns a boolean mask

close()

colorResult()
Color filtered rows in main table

query(*evt=None*)
Do query

setup()

update()
Enter event loop until all pending events have been processed by Tcl.

class pandastable.dialogs.**SimpleEditor**(*parent=None*, *width=100*, *height=40*, *font=None*)
Bases: tkinter.ttk.Frame

Simple text editor

onClear()
Clear text

onFind()

onSave()
Save text

class pandastable.dialogs.**ToolTip**(*widget*)
Bases: object

Tooltip class for tkinter widgets

classmethod createToolTip(*widget*, *text*)
Create a tooltip for a widget

hidetip(*event=None*)
Hide tooltip

showtip(*text*, *event=None*)
Display text in tooltip window

class pandastable.dialogs.**VerticalScrolledFrame**(*parent*, *height=None*, *width=None*, **args*, ***kw*)
Bases: tkinter.ttk.Frame

A pure Tkinter scrollable frame see http://tkinter.unpythonic.net/wiki/VerticalScrolledFrame. Use the 'interior' attribute to place widgets inside the scrollable frame.

pandastable.dialogs.**addButton**(*frame*, *name*, *callback*, *img=None*, *tooltip=None*, *side='top'*, *compound=None*, *width=None*, *padding=1*)
Add a button with image, toolip to a tkinter frame

pandastable.dialogs.**addListBox**(*parent*, *values=[]*, *width=10*, *height=6*, *label=''*)
Add an EasyListBox

pandastable.dialogs.**applyStyle**(*w*)
Apply style to individual widget to prevent widget color issues on linux

pandastable.dialogs.**dialogFromOptions**(*parent*, *opts*, *groups=None*, *callback=None*, *sticky='news'*, *layout='horizontal'*)
Auto create tk vars and widgets for corresponding options and and return the enclosing frame

pandastable.dialogs.**getBestGeometry**(*win*, *width=None*)
Calculate optimal geometry from screen size or given width

pandastable.dialogs.**getDictfromTkVars**(*opts*, *tkvars*, *widgets*)

pandastable.dialogs.**getListBoxSelection**(*w*)

pandastable.dialogs.**getParentGeometry**(*parent*)

pandastable.dialogs.**pickColor**(*parent*, *oldcolor*)

pandastable.dialogs.**setGeometry**(*win*, *width=None*)
Set window geometry to center of screen

pandastable.dialogs.**setWidgetStyles**(*widgets*)
set styles of list of widgets

9.1.7 pandastable.handlers module

Module for plot viewer event classes.

Created Jan 2016 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.handlers.**DragHandler**(*parent*, *figure=None*)
Bases: object

A simple class to handle picking and dragging

button_press_event(*event*)

connect()
Connect events

disconnect()
disconnect all the stored connection ids

drawSelectionRect()
Draw a selection box

key_press_event(*event*)
Handle key press

on_pick_event(*event*)
Store which text object was picked and were the pick event occurs.

on_release_event(*event*)
Update and store text/annotation position

9.1.8 pandastable.headers module

Implements the pandastable headers classes. Created Jan 2014 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.headers.**ColumnHeader**(*parent=None*, *table=None*, *bg='gray25'*)
Bases: tkinter.Canvas

Class that takes it's size and rendering from a parent table and column names from the table model.

drawRect(*col*, *tag=None*, *color=None*, *outline=None*, *delete=1*)
User has clicked to select a col

draw_resize_symbol(*col*)
Draw a symbol to show that col can be resized when mouse here

handle_double_click(*event*)
Double click sorts by this column.

handle_left_click(*event*)
Does cell selection when left mouse button is clicked

handle_left_ctrl_click(*event*)
Handle ctrl clicks - for multiple column selections

handle_left_release(*event*)
When mouse released implement resize or col move

handle_left_shift_click(*event*)
Handle shift click, for selecting multiple cols

handle_mouse_drag(*event*)
Handle column drag, will be either to move cols or resize

handle_mouse_move(*event*)
Handle mouse moved in header, if near divider draw resize symbol

handle_right_click(*event*)
respond to a right click

handle_right_release(*event*)

leave(*event*)
Mouse left canvas event

popupMenu(*event*)
Add left and right click behaviour for column header

redraw(*align='w'*)
Redraw column header

renameColumn()
Rename column

setDefaults()

within(*val*, *l*, *d*)
Utility funtion to see if val is within d of any items in the list l

class pandastable.headers.**IndexHeader**(*parent=None*, *table=None*, *width=40*, *height=25*, *bg='gray50'*)
Bases: tkinter.Canvas

Class that displays the row index headings.

handle_left_click(*event*)
Handle mouse left mouse click

redraw(*align='w'*)
Redraw row index header

class pandastable.headers.**RowHeader**(*parent=None*, *table=None*, *width=50*, *bg='gray75'*)
Bases: tkinter.Canvas

Class that displays the row headings (or DataFrame index). Takes it's size and rendering from the parent table. This also handles row/record selection as opposed to cell selection

clearSelected()
Clear selected rows

drawRect(*row=None*, *tag=None*, *color=None*, *outline=None*, *delete=1*)
Draw a rect representing row selection

drawSelectedRows(*rows=None*)
Draw selected rows, accepts a list or integer

handle_left_click(*event*)
Handle left click

handle_left_ctrl_click(*event*)
Handle ctrl clicks - for multiple row selections

handle_left_release(*event*)

handle_left_shift_click(*event*)
Handle shift click

handle_mouse_drag(*event*)
Handle mouse moved with button held down, multiple selections

handle_right_click(*event*)
respond to a right click

popupMenu(*event*, *rows=None*, *cols=None*, *outside=None*)
Add left and right click behaviour for canvas, should not have to override this function, it will take its values from defined dicts in constructor

redraw(*align='w'*, *showkeys=False*)
Redraw row header

setWidth(*w*)
Set width

toggleIndex()
Toggle index display

pandastable.headers.**createSubMenu**(*parent*, *label*, *commands*)

9.1.9 pandastable.images module

Images stored as PhotoImage objects, for buttons and logos. Created Oct 2008 Copyright (C) Damien Farrell

pandastable.images.**accept**()

pandastable.images.**add**()

pandastable.images.**add_col**()

pandastable.images.**add_row**()

pandastable.images.**aggregate**()

pandastable.images.**calculate**()

pandastable.images.**color_swatch**()

pandastable.images.**contract_col**()

pandastable.images.**copy**()

pandastable.images.**cross**()

pandastable.images.**del_col**()

pandastable.images.**del_row**()

pandastable.images.**delete**()

pandastable.images.**end**()

pandastable.images.**excel**()

pandastable.images.**expand_col**()

pandastable.images.**filtering**()

pandastable.images.**fit**()

pandastable.images.**font**()

pandastable.images.**function**()

pandastable.images.**importcsv**()

pandastable.images.**melt**()

pandastable.images.**merge**()

pandastable.images.**new_proj**()

pandastable.images.**next**()

pandastable.images.**open_proj**()

pandastable.images.**paste**()

pandastable.images.**pivot**()

pandastable.images.**plot**()

pandastable.images.**plot_clear**()

pandastable.images.**plot_prefs**()

pandastable.images.**prefs**()

pandastable.images.**prev**()

pandastable.images.**refresh**()

pandastable.images.**save**()

pandastable.images.**save_proj**()

pandastable.images.**search**()

pandastable.images.**start**()

pandastable.images.**table_delete**()

pandastable.images.**table_multiple**()

pandastable.images.**tableapp_logo**()

pandastable.images.**tilehorizontal**()

pandastable.images.**tilevertical**()

pandastable.images.**transpose**()

pandastable.images.**zoom_in**()

pandastable.images.**zoom_out**()

9.1.10 pandastable.plotting module

Module for pandastable plotting classes .

Created Jan 2014 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.plotting.**AnimateOptions**(*parent=None*)

Bases: *pandastable.plotting.TkOptions*

Class for live update/animation of plots.

addWidgets()

Custom dialogs for manually adding annotation items like text

getWriter()

showDialog(*parent*, *layout='horizontal'*)

Create dialog widgets

start()
start animation using a thread

stop()
Stop animation loop

stream()
Stream data into table and plot - not implemented yet

update()
do live updating

updateCurrent(*writer=None*)
Iterate over current table and update plot

class pandastable.plotting.**AnnotationOptions**(*parent=None*)
Bases: *pandastable.plotting.TkOptions*

This class also provides custom tools for adding items to the plot

addArrow(*kwds=None*, *key=None*)
Add line/arrow

addObject()
Add an annotation object

addTextBox(*kwds=None*, *key=None*)
Add a text annotation and store it using key

addWidgets()
Custom dialogs for manually adding annotation items like text

clear()
Clear annotations

redraw()
Redraw all stored annotations in the right places after a plot update

showDialog(*parent*, *layout='horizontal'*)
Override because we need to add custom widgets

class pandastable.plotting.**ExtraOptions**(*parent=None*)
Bases: *pandastable.plotting.TkOptions*

Class for additional formatting options like styles

addWidgets()
Custom dialogs for manually adding annotation items like text

apply()

reset()

showDialog(*parent*, *layout='horizontal'*)
Create dialog widgets

class pandastable.plotting.**MPL3DOptions**(*parent=None*)
Bases: *pandastable.plotting.MPLBaseOptions*

Class to provide 3D matplotlib options

applyOptions()
Set the plot kwd arguments from the tk variables

defaultfont = 'monospace'

kinds = ['scatter', 'bar', 'contour', 'wireframe', 'surface']

class pandastable.plotting.**MPLBaseOptions**(*parent=None*)
Bases: *pandastable.plotting.TkOptions*

Class to provide a dialog for matplotlib options and returning the selected prefs

applyOptions()
Set the plot kwd arguments from the tk variables

defaultfont = 'monospace'

kinds = ['line', 'scatter', 'bar', 'barh', 'pie', 'histogram', 'boxplot', 'violinplot'

legendlocs = ['best', 'upper right', 'upper left', 'lower left', 'lower right', 'right

update(*df*)
Update data widget(s) when dataframe changes

class pandastable.plotting.**PlotLayoutGrid**(*parent=None*, *width=280*, *height=205*, *rows=2*, *cols=2*, ***kwargs*)
Bases: *pandastable.dialogs.BaseTable*

handle_left_click(*event*)
Respond to a single press

class pandastable.plotting.**PlotLayoutOptions**(*parent=None*)
Bases: *pandastable.plotting.TkOptions*

resetGrid(*event=None*)
update grid and redraw

setmultiviews(*event=None*)

showDialog(*parent*, *layout='horizontal'*)
Override because we need to add custom bits

updateAxesList()
Update axes list

updateFromGrid()

class pandastable.plotting.**PlotViewer**(*table*, *parent=None*, *showoptions=True*)
Bases: tkinter.ttk.Frame

Provides a frame for figure canvas and MPL settings.

Parameters

- **table** – parent table, required
- **parent** – parent tkinter frame
- **layout** – 'horizontal' or 'vertical'

addWidgets()
Add option widgets to control panel

applyPlotoptions()
Apply the current plotter/options

autoscale(*axis='y'*)
Set all subplots to limits of largest range

bar3D(*data*, *ax*, *kwds*)
3D bar plot

checkColumnNames(*cols*)
Check length of column names

clear()
Clear plot

close()
Close the window

contourData(*data*)
Get data for contour plot

dotplot(*df*, *ax*, *kwds*)
Dot plot

getView()

getcmap(*name*)

heatmap(*df*, *ax*, *kwds*)
Plot heatmap

meshData(*x*, *y*, *z*)
Prepare 1D data for plotting in the form (x,y)->Z

plot2D(*redraw=True*)
Plot method for current data. Relies on pandas plot functionality if possible. There is some temporary code here to make sure only the valid plot options are passed for each plot kind.

plot3D(*redraw=True*)
3D plot

plotCurrent(*redraw=True*)
Plot the current data

plotMultiViews(*plot_types=['bar', 'scatter']*)
Plot multiple views of the same data in a grid

plotSplitData()
Splits selected data up into multiple plots in a grid

removeSubplot()
Remove a specific axis from the grid layout

replot(*data=None*)
Re-plot using current parent table selection. Args: data: set current dataframe, otherwise use current table selection

savePlot(*filename=None*)
Save the current plot

scatter(*df*, *ax*, *alpha=0.8*, *marker='o'*, *color=None*, ***kwds*)
A custom scatter plot rather than the pandas one. By default this plots the first column selected versus the others

scatter3D(*data*, *ax*, *kwds*)
3D scatter plot

setAxisLabels(*ax*, *kwds*)
Set a plots axis labels

setFigureOptions(*axs*, *kwds*)
Set axis wide options such as ylabels, title

setGlobalOption(*name=''*, **args*)
Set global value from widgets

setOption(*option*, *value*)

setSubplotTitle()
Set a subplot title if using grid layout

setupGUI()
Add GUI elements

showWarning(*text='plot error'*, *ax=None*)
Show warning message in the plot window

toggle_options()
Show/hide plot options

updateData()
Update data widgets

updatePlot()
Update the current plot with new options

updateStyle()

updateWidgets()
Set global widgets from values

venn(*data*, *ax*, *colormap=None*, *alpha=0.8*)
Plot venn diagram, requires matplotlb-venn

violinplot(*df*, *ax*, *kwds*)
violin plot

zoom(*zoomin=True*)
Zoom in/out to plot by changing size of elements

class pandastable.plotting.**TkOptions**(*parent=None*)
Bases: object

Class to generate tkinter widget dialog for dict of options

apply()

applyOptions()
Set the plot kwd arguments from the tk variables

increment(*key*, *inc*)
Increase the value of a widget

setWidgetStyles()

showDialog(*parent*, *layout='horizontal'*)
Auto create tk vars, widgets for corresponding options and and return the frame

updateFromDict(*kwds=None*)
Update all widget tk vars using plot kwds dict

pandastable.plotting.**addFigure**(*parent*, *figure=None*, *resize_callback=None*)
Create a tk figure and canvas in the parent frame

pandastable.plotting.**get_defaults**(*name*)

9.1.11 pandastable.plugin module

Implements the dataexplore plugin class. Created Oct 2015 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.plugin.**Plugin**(*parent=None*)
Bases: object

Base Plugin class, should be inherited by any plugin

capabilities = []

main(*parent*)

menuentry = ''

quit(*evt=None*)

requires = []

pandastable.plugin.**describe_class**(*obj*)
Describe the class object passed as argument, including its methods

pandastable.plugin.**describe_func**(*obj*, *method=False*)
Describe the function object passed as argument. If this is a method object, the second argument will be passed as True

pandastable.plugin.**find_plugins**()

pandastable.plugin.**get_plugins_classes**(*capability*)
Returns classes of available plugins

pandastable.plugin.**get_plugins_instances**(*capability*)
Returns instances of available plugins

pandastable.plugin.**init_plugin_system**(*folders*)

pandastable.plugin.**load_plugins**(*plugins*)

pandastable.plugin.**parsefolder**(*folder*)
Parse for all .py files in plugins folder or zip archive

9.1.12 pandastable.config module

Implements a configuration class for pandastable Created Oct 2015 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

pandastable.config.**apply_options**(*options*, *table*)
Apply options to a table

pandastable.config.**check_options**(*opts*)
Check for missing default options in dict. Meant to handle incomplete config files

pandastable.config.**create_config_parser_from_dict**(*data=None*, *sections=odict_keys(['base', 'colors'])*, ***kwargs*)
Helper method to create a ConfigParser from a dict of the form shown in baseoptions

pandastable.config.**get_options**(*cp*)
Makes sure boolean opts are parsed

pandastable.config.**load_options**()

pandastable.config.**parse_config**(*conffile=None*)
Parse a configparser file

class pandastable.config.**preferencesDialog**(*parent*, *options*, *table=None*)
Bases: tkinter.ttk.Frame

Preferences dialog from config parser options

apply()
Apply options to current table

createWidgets()
create widgets

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

save()
Save from current dialog settings

updateFromOptions(*options*)
Update all widget tk vars using dict

pandastable.config.**print_options**(*options*)
Print option key/value pairs

pandastable.config.**update_config**(*options*)

pandastable.config.**write_config**(*conffile='default.conf'*, *defaults={}*)
Write a default config file

pandastable.config.**write_default_config**()
Write a default config to users .config folder. Used to add global settings.

9.1.13 pandastable.stats module

Module for stats and fitting classes.

Created June 2015 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

class pandastable.stats.**StatsViewer**(*table*, *parent=None*)
Bases: tkinter.ttk.Frame

Provides a frame for model viewing interaction

doFit()
Do model fit on selected subset of rows. Will only use the currently selected rows for fitting.

getModel(*formula*, *data*, *est='ols'*)
Select model to use

guessFormula()
Suggest a start formula

plotLogit(*fit*, *indvar*, *ax*, ***kwds*)
Plot Logit results

plotPrediction(*fit*, *ax*)
Plot predicted vs. test

plotRegression(*fit*, *indvar*, *ax*, ***kwds*)
Plot custom statsmodels fit result for linear regression

quit()
Quit the Tcl interpreter. All widgets will be destroyed.

setupGUI()
Add GUI elements

showPlot()
Do plots

summary()
Fit summary

updateData()
Update data widgets

9.1.14 pandastable.tests module

9.1.15 pandastable.util module

Implements the utility methods for pandastable classes. Created August 2015 Copyright (C) Damien Farrell

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

pandastable.util.**adjustColorMap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=100*)
Adjust colormap to avoid using white in plots

pandastable.util.**checkDict**(*d*)
Check a dict recursively for non serializable types

pandastable.util.**checkOS**()
Check the OS we are in

pandastable.util.**check_multiindex**(*index*)
Check if index is a multiindex

pandastable.util.**colorScale**(*hex_color*, *brightness_offset=1*)
Takes a hex color and produces a lighter or darker variant. :returns: new color in hex format

pandastable.util.**getAttributes**(*obj*)
Get non hidden and built-in type object attributes that can be persisted

pandastable.util.**getFonts**()
Get the current list of system fonts

pandastable.util.**getTextLength**(*text*, *w*, *font=None*)
Get correct canvas text size (chars) that will fit in a given canvas width

pandastable.util.**setAttributes**(*obj*, *data*)
Set attributes from a dict. Used for restoring settings in tables

9.1.16 Module contents

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z