tablexplore Documentation

Damien Farrell

Apr 15, 2023

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Tablexplore is an application for data analysis and plotting built in Python using the PySide2/Qt toolkit. It uses the pandas DataFrame class to store the table data. Pandas is an open source Python library providing high-performance data structures and data analysis tools.

This application is intended primarily for educational/scientific use and allows quick visualization of data with convenient plotting. 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. The focus is on data manipulation rather than data entry. Though basic cell editing and row/column changes are supported.

CHAPTER

TWO

INSTALLATION

For all operating systems with Python and pip installed:

```
pip install -e git+https://github.com/dmnfarrell/tablexplore.git#egg=tablexplore
```

2.1 Linux

There is also a [snap](https://snapcraft.io/tablexplore) available, which can be installed using:

```
snap install tablexplore
```

2.2 Windows

A Windows standalone binary can be downloaded [here](https://dmnfarrell.github.io/tablexplore/).

CHAPTER

THREE

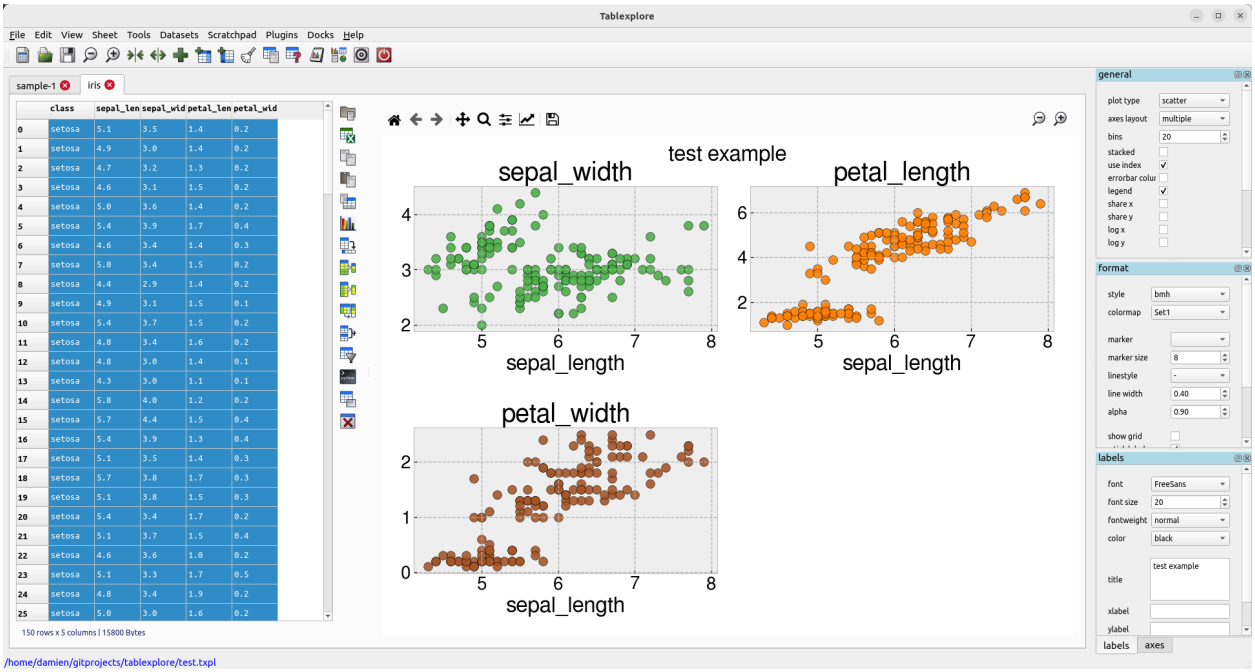
CURRENT FEATURES

- save and load projects
- import csv/hdf/from urls
- delete/add columns
- groupby-aggregate/pivot/transpose/melt operations
- merge tables
- show sub-tables
- plotting mostly works
- apply column functions, resample, transform, string methods and date/time conversion
- python interpreter

CHAPTER

FOUR

SCREENSHOTS



CHAPTER

FIVE

VIDEOS

- Introduction
- Summarizing Data
- Plotting
- Table Filtering
- Joining tables

CHAPTER

SIX

USING THE PROGRAM

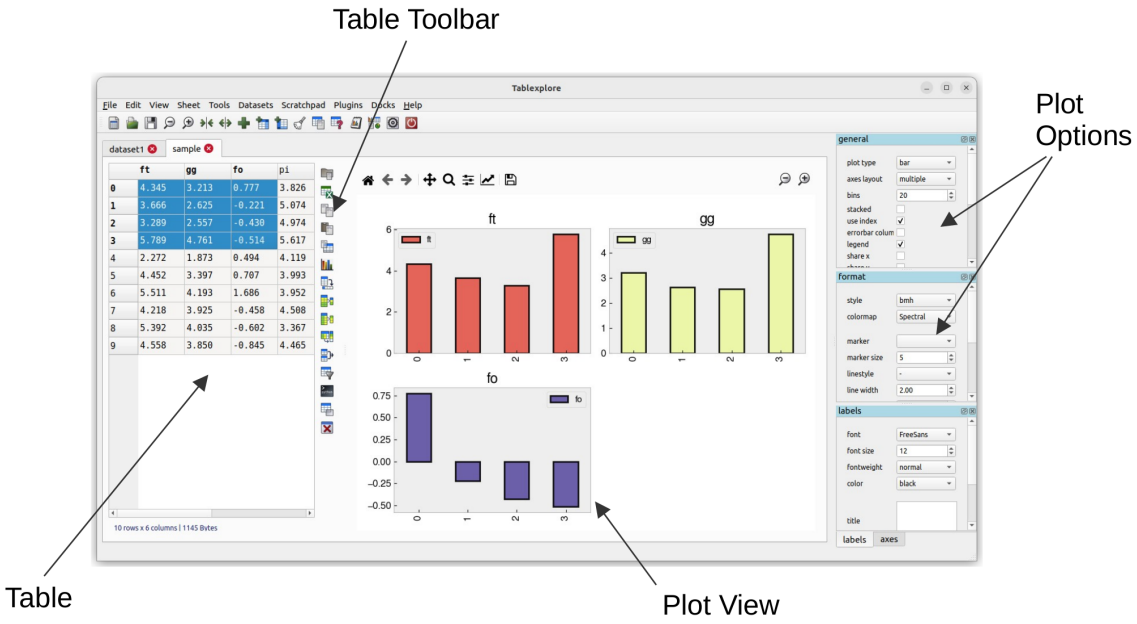
This page details some of the tasks available in tablexplore. For a general introduction also see the screencast at https://youtu.be/Ss0QIFywt74. Most of this functionality is available when you just use the table widget as well as the tablexplore application. Installing in windows or with a snap in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

6.1 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 rows/columns at once. So although you can edit cells it is not especially meant for data entry. You can use a spreadsheet for that and then import the data. 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 basic 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.

6.2 Interface layout

The table is laid out with headers for row and columns. Much functionality can be accessed from the tools menu but also by 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 row headers can use indexes (arbitrary labels). You can set any column as an index. This has extra functionality when it comes to plotting. Duplicate column name indexes are allowed though should be avoided if you want predictable behaviour. Every table has a vertical toolbar on the right and a plot view associated with it. You can make the plot hidden if needed.



6.3 Command Line

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

Show help:

```
tablexplore -h
```

Open a project file:

```
tablexplore -p <project file>
```

Open a csv file and try to import it:

```
tablexplore -i <csv file>
```

Open an excel file and try to import it:

```
tablexplore -x <excel file>
```

6.4 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. The table at the bottom of the import dialog contains the column data types (dtype) of each column. You can set this manually if you want to make sure a column is imported correctly. For example, numbers with leading zeroes will have this removed by default as the column is set to int. manually setting dtype to object avoids this.

6.5 Import multiple files

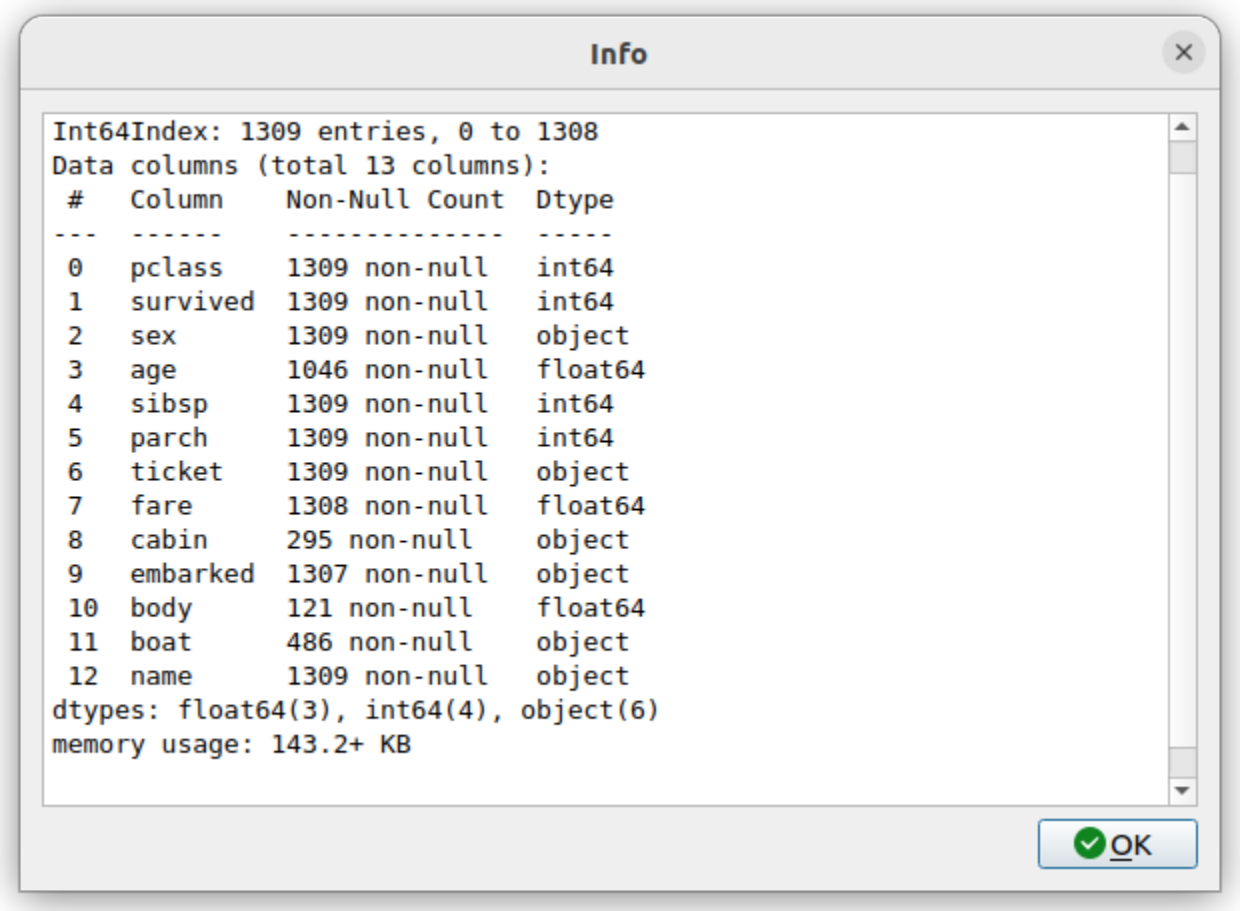
You can bulk import files using the File->Batch Import option. This allows you to choose a selection of files or to choose a folder and all the csv files inside will be recursively selected. If the files represent the same data structure they can be joined together into one table.

6.6 Saving your work

tablexplore projects (multiple groups of sheets with the plot and settings for each) are saved in **pickle** format and have the .txpl file extension. The program will remember table selections also. Note: it is not recommended that you use this project format for long term backup as it is subject to change with new versions. You should *always keep a copy your raw data* if it is important. Exporting to csv is also possible and saving individual tables to excel files.

6.7 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.



6.8 Cleaning data

Pandas supports a variety of options for data 'cleaning' or dealing with missing data. The most basic are available in tablexplore 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

6.9 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
- concatenate strings in multiple columns
- slice, slice string by start/end indexes
- replace

6.10 Date/Time conversion

Also by right clicking on a column you can convert it to *datetime* which is the internal format used to store dates and is useful for plotting time series. Normally the program can infer the dates but you can select the format.

6.11 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.

6.12 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/ The example below shows the titanic data being pivoted to show average ages per sex by pclass.

6.13 Merging two tables

Merging tables is done in tablexplore 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.

6.14 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. 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.

6.15 Filtering tables

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

6.15.1 Filter 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.

6.15.2 String filter

String based queries are made up fairly intuitive expressions in Python syntax. 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 of string filters:

```
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
```

Example of usage:

6.16 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.

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

6.17 Converting column names

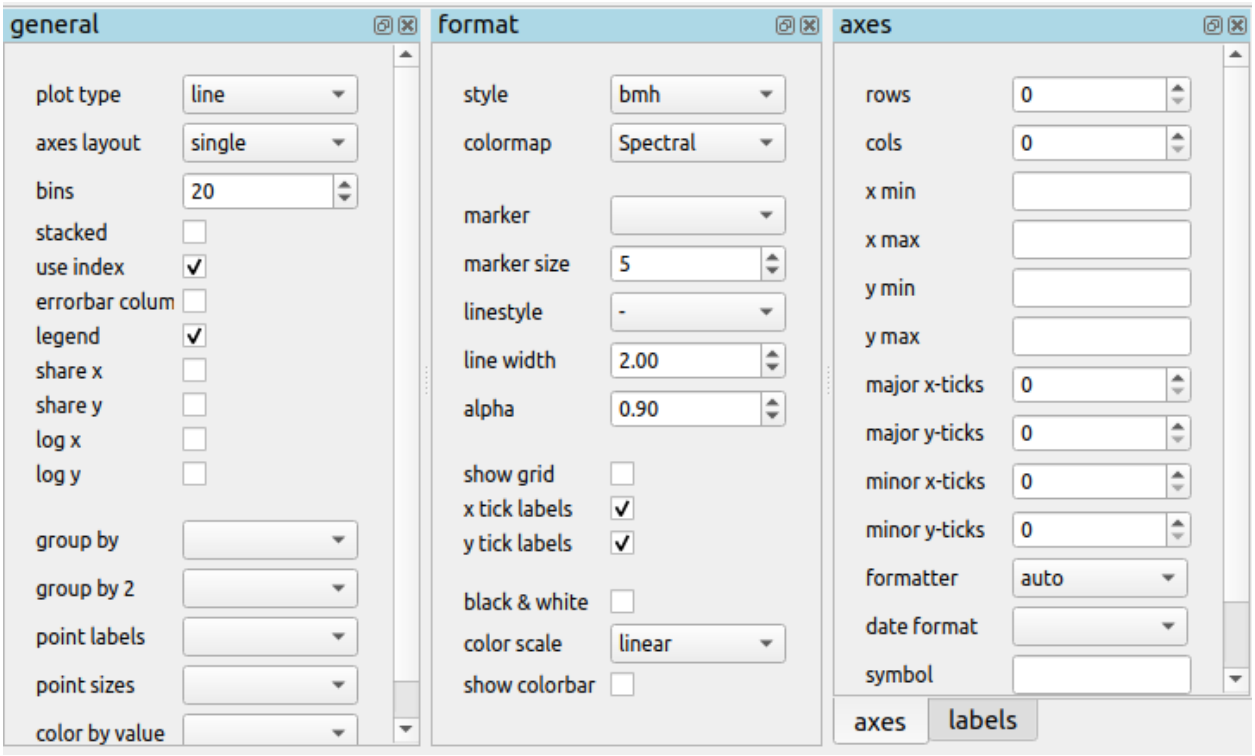
It may sometimes be necessary to re-format column names, for example to remove unwanted characters. If you have dozens or more columns this would be time consuming, so there is a function in tablexplore to do this in one step. Accessed from Tools->Convert column names, this dialog allows you to replace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

6.18 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 we create a date column and then use resampling to smooth out the data in another column using a window of 7.

6.19 Plot options

The plot options are a series of docked widgets on the right side of the plot, grouped by functionality. The docks can be dragged to the other sides of the application window or closed. Re-opening is done from the dock menu. Most default formatting options such as the type of plot, whether to show a legend etc. are in the 'general' tab. If you use the program regularly you will be familiar with where things are.



The following plot types are currently supported:

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

6.20 Plotting grouped data

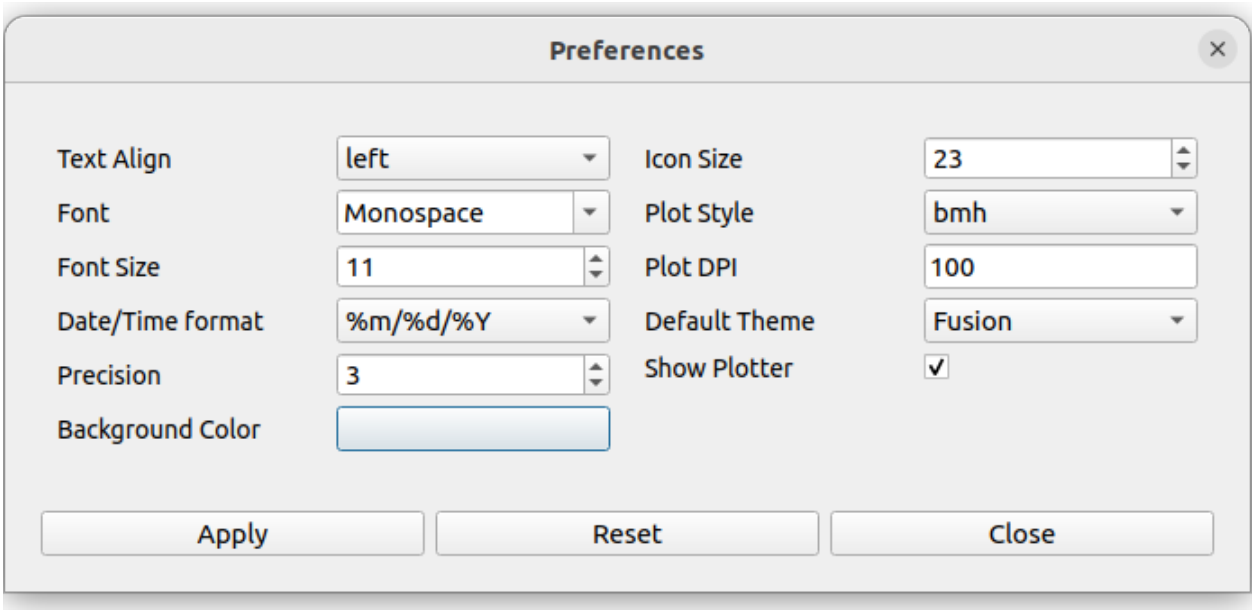
Rather than grouping the table directly and then plotting, it is also possible to plot data grouped. This requires you select the appropriate columns including the one to be grouped by and select the grouping column in the 'groupby' menu in the plot options. Plots can be grouped by 1-2 columns at once.

6.21 Scratchpad

The scratchpad is used to store plots as you go along, that can be viewed and saved later. It's also used for tables in text. It can be useful for resizing plots before saving for example. The plots stored here are saved with your project so can be retrieved at any time.

6.22 Setting preferences

Application settings are set from the Edit->Preferences menu. The image below shows the settings which are mostly self explanatory. If settings get corrupted or you want to restore defaults use the 'reset' button.



6.23 The terminal

For those familiar with Python and pandas a basic terminal is included, accessible from the toolbar. This will appear below the table. You can then run any Python command via the intepreter. The current table data is initially assigned to the *df* variable and the table can be accessed from the *table* variable. For example to add a column called 'new' you would do the following:

```
df['new'] = 3
table.refresh()
```

Any pandas operation can be performed. You need to call `table.refresh()` to update the table after making changes to the underlying dataframe.

Working example is shown here:

6.24 Plugins

Plugins can be added by anyone (see code examples on how to do this). Currently there are are only a few useful built-in plugins. New ones will be added below. To add a third party plugin (just a .py file), place it in the plugin folder under <home dir>/.config/tablexplore. For security, you shouldn't just download and run any .py file without trusting it first.

6.24.1 Colormap tool

This allows you to add your own colormaps for plotting. The screen grab below shows you. You can generate random colors, then edit them. When done choose the type of colormap and then save. Pick a name and this is stored and added to the list of of colormaps in the plot options. You have to restart the program to see it. (Colormaps are kept under .config/tablexplore/cmaps.pkl which can be deleted if you want to clear them.)

6.24.2 Seaborn plugin

Seaborn is a statistical plotting package for Python. This plugin lets you use it as an alternative to the regular plotting tools. Note that you need to have installed tablexplore using pip for this to work and it is not currently part of the standalone windows application or the snap. The plugin has a set of drop down menus mostly for selecting which column in your table you want to be plotted in which dimension. These won't all be intuitive unless you have used seaborn.

It is assumed that your data is in 'long form' or 'tidy' format.

Typical usage is shown below:

CHAPTER

SEVEN

CODE EXAMPLES

This section is for Python programmers.

7.1 Basics

If you want to use the table widget in another GUI program:

```
python
from PySide2 import QtCore
from PySide2.QtWidgets import *
from PySide2.QtGui import *
import pandas as pd
from tablexplore import data, core, plotting, interpreter

class TestApp(QMainWindow):
    def __init__(self, project_file=None, csv_file=None):

        QMainWindow.__init__(self)
        self.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.setWindowTitle("Example")
        self.setGeometry(QtCore.QRect(200, 200, 800, 600))
        self.main = QWidget()
        self.setCentralWidget(self.main)
        layout = QVBoxLayout(self.main)
        df = data.getSampleData()
        t = core.DataFrameWidget(self.main,dataframe=df)
        layout.addWidget(t)
        #show a Python interpreter
        t.showInterpreter()
        return

if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    aw = TestApp()
    aw.show()
    app.exec_()
```

7.2 Writing a plugin

This is quite straightforward if you are familiar with PyQt5/Pyside2. Built in plugins are kept in the plugin folder where the program is installed. You can look at these to get an idea how the plugins are written. When you make your own plugin, just add the .py file to the plugin folder under <home dir>/.config/tablexplore. It will be loaded when the program starts and added to the menu. You can add any code into the script, usually designed to execute using the table or plotter. GUI based plugins will be added as docked widgets to the application.

Here is an example plugin:

```
class ExamplePlugin(Plugin):
    """Template plugin for TableExplore"""

    #uncomment capabilities list to appear in menu
                capabilities = ['gui']
    requires = ['']
    menuentry = 'Example Plugin'
    name = 'Example Plugin'

    def __init__(self, parent=None, table=None):
        """Customise this and/or doFrame for your widgets"""

        if parent==None:
            return
        self.parent = parent
        self.table = table
        self.createWidgets()
        return
```

CHAPTER

EIGHT

TABLEXPLORE

8.1 tablexplore package

8.1.1 Submodules

8.1.2 tablexplore.app module

TablExplore app Created November 2020 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 tablexplore.app.**Application**(*project_file=None*, *csv_file=None*, *excel_file=None*)

Bases: QMainWindow

about()

addDockWidgets()

Add plot dialogs to dock

addRecentFile(*fname*)

Add file to recent if not present

addSheet(*name=None*, *df=None*, *meta=None*)

Add a new sheet

applySettings()

Apply settings to GUI when changed

changeColumnWidths(*factor=1.1*)

checkSettings()

Check for missing settings

clearSheets(*ask=True*)

Clear all sheets

closeEvent(*event*)
Close event

concatSheets()
Combine sheets into one table

copy()

createMenu()
Main menu

createToolBar()
Create main toolbar

discoverPlugins()
Discover available plugins

do_saveProject(*filename*, *progress_callback=None*)
Does the actual saving. Save sheets inculding table dataframes and meta data as dict to compressed pickle.

duplicateSheet()
Make a copy of a sheet

exportAs()
Export as

fileQuit()

findReplace()
Find or replace

getCurrentTable()
Return the currently used table

getSampleData(*name*, *rows=None*)
Sample table

importExcel(*filename=None*)

importFile(*filename=None*)

importHDF()

importMultiple()
Import many files

importMultipleFiles(*folders=False*)
Import many files

importPickle()

importURL()
Import from URL

interpreter()
Launch python interpreter

loadMeta(*table*, *meta*)
Load meta data for a sheet/table, this includes plot options and table selections

loadPlugin(*plugin*)

Instantiate the plugin and call it's main method

loadSettings()

Load GUI settings

load_dataframe(*df*, *name=None*, *select=False*)

Load a DataFrame into a new sheet :param df: dataframe :param name: name of new sheet :param select: set new sheet as selected

load_pickle(*filename*)

Load a pickle file

mergeSheets()

Merge two sheets

newProject(*data=None*, *ask=False*)

New project

openProject(*filename=None*, *asksave=False*)

Open project file

open_url(*url=''*, *event=None*)

Open the online documentation

paste()

pasteNewSheet()

plotToScratchpad(*label=None*)

Cache the current plot so it can be viewed later

preferences()

Preferences dialog

processing_completed()

Generic process completed

progress_fn(*msg*)

refresh()

Refresh all tables

removeSheet(*index*, *ask=True*)

Remove sheet

renameSheet()

Rename the current sheet

replot()

Plot current

run_threaded_process(*process*, *on_complete*)

Execute a function in the background with a worker

saveAsProject()

Save as a new project filename

saveMeta(*tablewidget*)
Save meta data such as current plot options and certain table attributes. These are re-loaded when the sheet is opened.

saveProject(*filename=None*)
Save project

saveSettings()
Save GUI settings

saveWithProgress(*filename*)
Save with progress bar

setTheme(*theme=None*)
Change interface theme.

showErrorLog()
Show log file contents

showPlotFrame()
Show/hide the plot frame

showPlugin(*plugin*)
Add plugin as dock widget

showRecentFiles()
Populate recent files menu

showScratchpad()
Show stored plot figures

startLogging()
Logging

stateChanged(*bool*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

tabSelected(*index*)
Re-load plot widgets for current tab

tableToScratchpad()
Send table selection to scratchpad

undo()

updatePlotWidgets(*table*)
Update plot widgets from values in table

updatePluginMenu()
Update plugins

updatePlugins()
Update table for a plugin if it needs it

zoomIn()

zoomOut()

tablexplore.app.**main**()

8.1.3 tablexplore.core module

Implements core classes for tablexplore Created May 2017 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 tablexplore.core.**ColumnHeader**

Bases: QHeaderView

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.core.**DataFrameModel**(*dataframe=None*, **args*)

Bases: QAbstractTableModel

DataFrame Model class.

columnCount(*self*, *parent: PySide2.QtCore.QModelIndex = Invalid(PySide2.QtCore.QModelIndex)*) → int

data(*index*, *role=PySide2.QtCore.Qt.ItemDataRole.DisplayRole*)

Edit or display roles. Handles what happens when the Cells are edited or what appears in each cell. https://www.pythonguis.com/tutorials/pyside-qtableview-modelviews-numpy-pandas/

flags(*self*, *index: PySide2.QtCore.QModelIndex*) → PySide2.QtCore.Qt.ItemFlags

headerData(*col*, *orientation*, *role*)

What's displayed in the headers

rowCount(*self*, *parent: PySide2.QtCore.QModelIndex = Invalid(PySide2.QtCore.QModelIndex)*) → int

setData(*index*, *value*, *role=PySide2.QtCore.Qt.ItemDataRole.EditRole*)

Set data upon edits

sort(*idx*, *ascending=True*)

Sort table by given column number

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update(*df*)

class tablexplore.core.**DataFrameTable**(*parent=None*, *dataframe=None*, *font='Arial'*, *fontsize=12*, *columnwidth=80*, *timeformat='%m-%d-%Y'*, *bg='#F4F4F3'*, ***kwargs*)

Bases: QTableView

QTableView with pandas DataFrame as model.

addColumn()

Add a column

addRows()

Add n rows

changeColumnWidths(*factor=1.1*)

Set column widths

checkColumnsUnique()

Check if columns are all unique

columnClicked(*col*)

columnHeaderMenu(*pos*)

Column header right click popup menu

contextMenuEvent(*event*)

Reimplemented to create context menus for cells and empty space.

deleteCells(*rows*, *cols*, *answer=None*)

Clear the cell contents

deleteColumn(*column=None*)

Delete column

deleteRows()

Delete rows

getColumnOrder()

Get column names from header in their displayed order

getColumnWidths()

getMemory()

Get memory info as string

getScrollPosition()

Get current row/col position

getSelectedColumns()

Get selected column indexes

getSelectedDataFrame()

Get selection as a dataframe

getSelectedRows()

handleDoubleClick(*item*)

keyPressEvent(*self*, *event: PySide2.QtGui.QKeyEvent*) → None

memory_usage()

refresh()

Refresh table if dataframe is changed

renameColumn(*column=None*)

resetIndex()

rowHeaderMenu(*pos*)

Row header popup menu

selectColumn(*self*, *column: int*) → None

setColumnType(*column=None*)

Change the column dtype

setColumnWidths(*widths*)

setIndex(*column*)

Set column as index

setIndexType()

Set the type of the index

setRowColor(*rowIndex*, *color*)

setScrollPosition(*row*, *col*)

Move to row/col position

setSelected(*rows*, *cols*)

Set selection programmatically from a list of rows and cols. https://doc.qt.io/archives/qtjambi-4.5.2_01/com/trolltech/qt/model-view-selection.html

showAll()

Re-show unfiltered

showSelection(*item*)

sort(*idx*, *ascending=True*)

Sort by selected columns

sortIndex(*ascending=True*)

Sort by inde,

staticMetaObject = <PySide2.QtCore.QMetaObject object>

storeCurrent()

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

undo()

Undo last change to table

updateFont()

Update the font

viewRow()

View row data

zoomIn(*fontsize=None*)

Zoom in table

zoomOut(*fontsize=None*)

Zoom out table

class tablexplore.core.**DataFrameWidget**(*parent=None*, *dataframe=None*, *app=None*, *toolbar=True*, *statusbar=True*, ***kwargs*)

Bases: QWidget

Widget containing a tableview and toolbars

addColumn()

addRows()

aggregate()
Groupby aggregate operation

applyColumnFunction(*column*)
Apply column wise functions, applies a calculation per row and ceates a new column.

applySettings(*settings*)
Settings

applyStringMethod(*column*)
Apply string operation to column(s)

applyTransformFunction(*column*)
Apply resampling and transform functions on a single column.

bin()
Split into bins using cut

cleanData()
Deal with missing data

clear()
Clear table

close()
Close events

closeSubtable()

convertColumnNames()
Reformat column names

convertDates(*column*)
Convert single or multiple columns into datetime or extract features from datetime object.

convertNumeric()
Convert cols to numeric if possible

convertTypes()

copy()
Copy to clipboard

createPlotViewer(*parent=None*)
Create a plot widget attached to this table

createToolbar()
Create toolbar

editMode(*evt=None*)
Change table edit mode

exportTable()
Export table

fillData(*column*)
Fill column with data

fillDates(*column*)
Fill with datetime

fillStrings(*column*)
Fill column with string data

filter()
Show filter dialog

findDuplicates()
Find or remove duplicates

findreplace()
Find/replace dialog

getFileType()

getSelectedDataFrame()
Get selection as a dataframe

importExcel(*filename=None*)
Import excel file

importFile(*filename=None*, *dialog=True*, ***kwargs*)
Import csv file

importHDF()
Import hdf5 file

importPickle()

importURL(*recent*)
Import hdf5 file

info()
Table info

insert()
Insert from clipboard

load()

manageColumns()
Edit columns

melt()
Melt table

merge()

paste()
Paste from clipboard

pivot()
Pivot table

plot()
Plot from selection

refresh()

resample()

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

runScript()

Run a set of python commands on the table

save()

selectAll()

Select all data

showAsText()

Show selection as text

showInterpreter()

Show the Python interpreter

showSubTable(*df=None*, *title=None*, *index=False*, *out=False*)

Add the child table

stateChanged(*idx*, *idx2*)

Run whenever table model is changed

staticMetaObject = <PySide2.QtCore.QMetaObject object>

statusBar()

Status bar at bottom

subTableFromSelection()

transpose()

updateStatusBar()

Update the table details in the status bar

class tablexplore.core.**HeaderProxyStyle**

Bases: QProxyStyle

drawControl(*self*, *element: PySide2.QtWidgets.QStyle.ControlElement*, *option: PySide2.QtWidgets.QStyleOption*, *painter: PySide2.QtGui.QPainter*, *widget: Union[PySide2.QtWidgets.QWidget, NoneType] = None*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.core.**HeaderView**(*parent*)

Bases: QHeaderView

" Column header class.

sectionSizeFromContents(*logicalIndex*)

Get section size from contents

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.core.**ItemEditorFactory**

Bases: QItemEditorFactory

createEditor(*self*, *userType: int*, *parent: PySide2.QtWidgets.QWidget*) → PySide2.QtWidgets.QWidget

class tablexplore.core.**RowHeader**

Bases: QHeaderView

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.core.**SubTableWidget**(*parent=None*, *dataframe=None*, ***args*)

Bases: *DataFrameWidget*

Widget for sub table

createToolbar()

Override default toolbar

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.core.**SubWidget**(*parent*, *table*)

Bases: QDockWidget

closeEvent(*self*, *event: PySide2.QtGui.QCloseEvent*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

8.1.4 tablexplore.dialogs module

Implements some dialog utilities for tablexplore Created Feb 2019 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 tablexplore.dialogs.**AggregateDialog**(*parent*, *df*, *title='Groupby-Aggregate'*)

Bases: *BasicDialog*

Qdialog with multiple inputs

apply()

Do the operation

createWidgets()

Create widgets

customButtons()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**BasicDialog**(*parent*, *df*, *title=None*, *app=None*)

Bases: QDialog

Qdialog for table operations interfaces

apply()
Override this

close(*self*) → bool

copy_to_clipboard()
Copy result to clipboard

copy_to_sheet()
Copy result to new sheet in app, if available

copy_to_subtable()
Do the operation

createButtons(*parent*)

createWidgets()
Create widgets - override this

export()
export result to file

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**ColorButton**(**args*, *color=None*, ***kwargs*)
Bases: QPushButton

Custom Qt Widget to show a chosen color.

Left-clicking the button shows the color-chooser, while right-clicking resets the color to None (no-color).

color()

colorChanged

mousePressEvent(*self*, *e: PySide2.QtGui.QMouseEvent*) → None

onColorPicker()
Show color-picker dialog to select color. Qt will use the native dialog by default.

setColor(*color*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**ComboDelegate**(*parent*, *items*)
Bases: QItemDelegate

A delegate to add QComboBox in every cell of the given column

createEditor(*self*, *parent: PySide2.QtWidgets.QWidget*, *option: PySide2.QtWidgets.QStyleOptionViewItem*, *index: PySide2.QtCore.QModelIndex*) → PySide2.QtWidgets.QWidget

currentIndexChanged()

setEditorData(*self*, *editor: PySide2.QtWidgets.QWidget*, *index: PySide2.QtCore.QModelIndex*) → None

setModelData(*self*, *editor: PySide2.QtWidgets.QWidget*, *model: PySide2.QtCore.QAbstractItemModel*, *index: PySide2.QtCore.QModelIndex*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**ConvertTypesDialog**(*parent*, *df*, *title='Convert types'*)

Bases: *BasicDialog*

Dialog to melt table

apply()

Do the operation

createButtons(*parent*)

createWidgets()

Create widgets

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**FilterBar**(*parent*, *table*)

Bases: QWidget

Single Widget based filter

createWidgets()

Create widgets

getFilter()

Get filter values for this instance

onClose(*ce*)

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**FilterDialog**(*parent*, *table*, *title=None*, *app=None*)

Bases: QWidget

Qdialog for table query/filtering

addFilter()

Add a filter using widgets

apply()

Apply filters

applyWidgetFilters(*df*, *mask=None*)

Apply the widget based filters, returns a boolean mask

copyResult()

createToolBar(*parent*)

createWidgets()

Create widgets

onClose()

refresh()

Reset the table

removeFiltered()

Subtract current filtered result from original table

staticMetaObject = <PySide2.QtCore.QMetaObject object>

togglecase()

update()
Update the column widgets if table has changed

class tablexplore.dialogs.**FindReplaceDialog**(*parent*, *table*, *title=None*, *app=None*)
Bases: QWidget

Qdialog for table query/filtering

clear()

createToolBar(*parent*)

createWidgets()
Create widgets

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

findAll()
Apply

findNext()
Show next cell of search results

onClose()

replace()
Replace all instances of search text

staticMetaObject = <PySide2.QtCore.QMetaObject object>

togglecase()

class tablexplore.dialogs.**ImportDialog**(*parent=None*, *filename=None*)
Bases: QDialog

Provides a dialog for import settings

createButtons(*parent*)

createWidgets()
Create widgets

doImport()
Do the import

quit()

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

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update()
Reload previews

class tablexplore.dialogs.**ManageColumnsDialog**(*parent*, *df*, *title='Manage Columns'*, *app=None*)

Bases: *BasicDialog*

Qdialog for column re-arranging

checknumeric()

Check if any cols numeric

convert()

createButtons(*parent*)

createWidgets()

Create widgets - override this

deduplicate()

Rename duplicate column names

delete()

sort()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

undo()

update()

Update list

class tablexplore.dialogs.**MeltDialog**(*parent*, *df*, *title='Melt'*)

Bases: *BasicDialog*

Dialog to melt table

apply()

Do the operation

createWidgets()

Create widgets

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**MergeDialog**(*parent*, *df*, *df2=None*, *title='Merge Tables'*, *app=None*)

Bases: *BasicDialog*

Dialog to melt table

apply()

Do the operation

createWidgets()

Create widgets

staticMetaObject = <PySide2.QtCore.QMetaObject object>

updateColumns()

class tablexplore.dialogs.**MultipleFilesDialog**(*parent*, *title='Import Multiple'*)

Bases: QDialog

createWidgets()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.MultipleInputDialog(*parent*, *options=None*, *title='Input'*, *width=400*, *height=200*)

Bases: QDialog

Qdialog with multiple inputs

accept(*self*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.PivotDialog(*parent*, *df*, *title='Pivot'*)

Bases: *BasicDialog*

Dialog to pivot table

apply()

Do the operation

createWidgets()

Create widgets

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.PlainTextEditor(*parent=None*, ***kwargs*)

Bases: QPlainTextEdit

contextMenuEvent(*self*, *e: PySide2.QtGui.QContextMenuEvent*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

zoom(*delta*)

class tablexplore.dialogs.PreferencesDialog(*parent*, *options={}*)

Bases: QDialog

Preferences dialog from config parser options

apply()

Apply options to current table

createButtons(*parent*)

createWidgets(*options*)

create widgets

reset()

Reset to defaults

setDefaults()

Populate default kwds dict

staticMetaObject = <PySide2.QtCore.QMetaObject object>

updateWidgets(*kwds=None*)

Update widgets from stored or supplied kwds

class tablexplore.dialogs.**ProgressWidget**(*parent=None*, *label=''*)

Bases: QDialog

Progress widget class

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**Renamer**

Bases: object

class tablexplore.dialogs.**SimpleDialog**(*parent*, *title=None*)

Bases: QDialog

Qdialog for generic operations

createButtons()

Create buttons

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**TextDialog**(*parent*, *text=''*, *title='Text'*, *width=400*, *height=300*)

Bases: QDialog

Text edit dialog

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.dialogs.**Worker**(*fn*, **args*, ***kwargs*)

Bases: QRunnable

Worker thread for running background tasks.

run(*self*) → None

class tablexplore.dialogs.**WorkerSignals**

Bases: QObject

Defines the signals available from a running worker thread. Supported signals are: finished

No data

error
tuple (exctype, value, traceback.format_exc())

result
object data returned from processing, anything

error

finished

progress

result

staticMetaObject = <PySide2.QtCore.QMetaObject object>

tablexplore.dialogs.**addToolBarItems**(*toolbar*, *parent*, *items*)

Populate toolbar from dict of items

tablexplore.dialogs.**dialogFromOptions**(*parent*, *opts*, *sections=None*, *wrap=2*, *section_wrap=4*, *style=None*)

Get Qt widgets dialog from a dictionary of options. :param opts: options dictionary :param sections: :param section_wrap: how many sections in one row :param style: stylesheet css if required

tablexplore.dialogs.**getName**(*parent*, *current=''*, *txt='Enter value'*)

Wrapper for text inpuit dialog

tablexplore.dialogs.**getWidgetValue**(*w*)

Get value from any kind of widget

tablexplore.dialogs.**getWidgetValues**(*widgets*)

Get values back from a set of widgets

tablexplore.dialogs.**setWidgetValues**(*widgets*, *values*)

Set values for a set of widgets from a dict

tablexplore.dialogs.**showMessage**(*parent*, *msg*, *type='error'*)

Show an error message

8.1.5 tablexplore.interpreter module

Implements a Python interpreter From original code from https://github.com/col-one/thonside

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 tablexplore.interpreter.**Interpreter**(*extra_context={}*, *stream_out=True*, *stream_err=True*, *table=None*, *app=None*)

Bases: `InteractiveConsole`

interact(*banner=None*, *exitmsg=None*)

Starting point for the interpreter. It is override for avoid while loop as classic shell. In this context interpreter doesn't use this functionality. :param banner: starter text :param exitmsg: no used :return:

raw_input(*prompt=''*)

Override InteractiveConsole.raw_input method to add a 'slot' connection, useful for different view implementation. :param prompt: str prompt to write :return:

run(*code*)

Manage run code, like continue if : or (with prompt switch >>> to ... :param code: str code to run :return:

runcode(*code*)

Override InteractiveConsole.runcode method to manage stdout as a buffer. Useful for view implementation. :param code: str code to run :return:

write(*data*)

Override InteractiveConsole.write method to add a 'slot' connection, useful for different view implementation. :param data: str data to write :return:

class tablexplore.interpreter.**Streamer**(*queue*)

Bases: object

flush()

write(*text*)

class tablexplore.interpreter.**TerminalPython**(*parent=None*, *table=None*, *app=None*)

Bases: *Terminal*

staticMetaObject = <PySide2.QtCore.QMetaObject object>

8.1.6 tablexplore.plotting module

tableexplore plotting module Created May 2017 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 tablexplore.plotting.**AnnotationOptions**

Bases: *BaseOptions*

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

applyOptions()

Set the plot kwd arguments from the tk variables

class tablexplore.plotting.**AxesOptions**

Bases: *BaseOptions*

Class for additional formatting options like styles

class tablexplore.plotting.**BaseOptions**

Bases: object

Class to generate widget dialog for dict of options

apply()

applyOptions()

Set the plot kwd arguments from the widgets

increment(*key*, *inc*)

Increase the value of a widget

randomSettings()

Get random settings

setDefaults()

Populate default kwds dict

setWidgetValue(*key*, *value*)

showDialog(*parent*, *wrap=2*, *section_wrap=2*, *style=None*)

Auto create widgets for corresponding options and
and return the dialog.

Parameters

- **parent** – parent frame
- **wrap** – wrap for internal widgets

updateWidgets(*kwds=None*)

Update widgets from stored or supplied kwds

class tablexplore.plotting.**FormatOptions**

Bases: *BaseOptions*

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

class tablexplore.plotting.**MPLBaseOptions**

Bases: *BaseOptions*

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

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

update(*df*)

Update data widget(s) when dataframe changes

class tablexplore.plotting.**PlotViewer**(*table*, *parent=None*)

Bases: QWidget

Plot viewer class

addPlotWidget()

Create mpl plot canvas and toolbar

applyPlotoptions()

Apply the current plotter/options

autoscale(*axis='y'*)

Set all subplots to limits of largest range

bar3D(*data*, *ax*, *kwds*)

3D bar plot

canvasResize(*pad=50*)

Trigger resize canvas

checkColumnNames(*cols*)

Check length of column names

check_kwds(*kwds*, *kind*)

clear()

Clear plot

colorsfromColormap(*df*, *cmap*)

Column colors from cmap

contourData(*data*)

Get data for contour plot

createOptions()

Create option attributes for plotter

createWidgets()

Create widgets. Plot on left and dock for tools on right.

customPlot()

plot custom series

customiseSeries()

Show custom options for current plot series. Allows plot types to be specified per series and uses a custom plot function.

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

Dot plot

getFigureSize()

getSeries(*data*)

getView()

getcmap(*name*)

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

Plot heatmap

meshData(*x*, *y*, *z*)

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

plot()

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

plotBySeries()

Plot different types depending on series

plotCurrent(*redraw=True*)

Plot the current data

replot(*data=None*)

Replot with given dataframe

savePlot(*filename=None*)

Save the current plot

scatter(*df*, *ax*, *axes_layout='single'*, *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

setFigure(*figure*)

Recreate canvas with given figure

setFigureOptions(*axs*, *kwds*)

Set axis wide options such as ylabels, title

setStyle()

Apply style

showTools()

Show/hide tools dock

showWarning(*text='plot error'*, *ax=None*)

Show warning message in the plot window

simple_plot(*df*)

test plot

staticMetaObject = <PySide2.QtCore.QMetaObject object>

updateData()

Update data widgets

updateSeries(*event=None*)

Update series with new plots

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 tablexplore.plotting.**PlotWidget**(*parent=None*, *figure=None*, *dpi=100*, *hold=False*)

Bases: FigureCanvasQTAgg

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.plotting.**SeriesOptions**(*plotviewer*)

Bases: *BaseOptions*

Class for selecting custom plot types for each series

close()

createButtons(*parent*)

createSeriesWidgets(*parent*)

showDialog(*parent=None*)

Auto create widgets for corresponding options and
and return the dialog.

Parameters

- **parent** – parent frame
- **wrap** – wrap for internal widgets

update(*event=None*)

Update series options from widgets and replot

tablexplore.plotting.**addFigure**(*parent*, *figure=None*, *resize_callback=None*)

Create a tk figure and canvas in the parent frame

tablexplore.plotting.**defaultOptions**()

Get default plotting options

tablexplore.plotting.**load_colormaps**()

tablexplore.plotting.**update_colormaps**()

Load stored colormaps

8.1.7 tablexplore.plugin module

TablExplore plugin class Created January 2021 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 tablexplore.plugin.**Plugin**(*parent=None*)

Bases: object

Base Plugin class, should be inherited by any plugin

capabilities = []

createWidgets(*width=600*, *height=600*)

Create main widget with GUI elements. This will be specific to the plugin so it must be overrriden.

main(*parent=None*)

menuentry = ''

quit(*evt=None*)

requires = []

tablexplore.plugin.**describe_class**(*obj*)

Describe the class object passed as argument, including its methods

tablexplore.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

tablexplore.plugin.**find_plugins**()

tablexplore.plugin.**get_plugins_classes**(*capability*)

Returns classes of available plugins

tablexplore.plugin.**get_plugins_instances**(*capability*)

Returns instances of available plugins

tablexplore.plugin.**init_plugin_system**(*folders*)

Find available plugins

tablexplore.plugin.**load_plugins**(*plugins*)

Load plugins

tablexplore.plugin.**parsefolder**(*folder*)

Parse for all .py files in plugins folder or zip archive

8.1.8 tablexplore.qt module

TablExplore app Created January 2021 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

8.1.9 tablexplore.terminal module

Implements some dialog utilities for tableexplore Created Feb 2019 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 tablexplore.terminal.**ExecThread**

Bases: QObject

cmd = None

def_to_run = None

finished

run()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.terminal.**QueueReceiver**(*queue*, **args*, ***kwargs*)

Bases: QObject

run()

sent

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class tablexplore.terminal.**Terminal**(*parent=None*, *hist_file=None*)

Bases: QPlainTextEdit

active_queue_thread(*queue*)

autocomplete(*command*)

Ask different possibility from command arg, proposition is limited by AUTOCOMPLETE_LIMIT constant :param command: str :return: list of proposition

closeEvent(*self*, *event: PySide2.QtGui.QCloseEvent*) → None

contextMenuEvent(*self*, *e: PySide2.QtGui.QContextMenuEvent*) → None

count_cursor_lines()

Slot def keep tracking cursor position line number. Useful to compare position to know if it is an editable line or not. :return:

exec_code(*cmd*)

get_command()

Get command, read last line and remove prompt length. :return: str command

get_cursor_position()

Get cursor position :return: int line cursor position.

get_last_line()

Get last terminal line. :return: str last line

get_next_history()

Get next history in the readline GNU history file. :return: str history

get_previous_history()

Get previous history in the readline GNU history file. :return: str history

init_history(*hist_file*)

History initialisation with readline GNU, and use hook atexit for save history when program closing. :param hist_file: history file path :return:

keyPressEvent(*event*)

Override to manage key board event. :param event: event key. :return:

press_enter

raw_input(*prompt=None*)

Use to write the prompt command. :param prompt: str prompt :return:

remove_last_command()

Remove current command. Useful for display history navigation. :return:

remove_last_line()

save_history(*hist_file*)

Hook def execute by atexit. :param hist_file: history file path :return:

setStyle(*self*, *arg__1: PySide2.QtWidgets.QStyle*) → None

staticMetaObject = <PySide2.QtCore.QMetaObject object>

write(*data*)

Append text to the Terminal. And keep cursor at the end. :param data: str data to write. :return:

write_autocomplete(*command*)

Prepare text to write. :param command: str command :return:

write_prompt(*data*)

Append text to the Terminal. And keep cursor at the end. :param data: str data to write. :return:

zoom(*delta*)

8.1.10 tablexplore.util module

Implements the utility methods for tableexplore 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

tablexplore.util.**adjustColorMap**(*cmap*, *minval=0.0*, *maxval=1.0*, *n=100*)

Adjust colormap to avoid using white in plots

tablexplore.util.**checkDict**(*d*)

Check a dict recursively for non serializable types

tablexplore.util.**checkOS**()

Check the OS we are in

tablexplore.util.**check_multiindex**(*index*)

Check if index is a multiindex

tablexplore.util.**colorScale**(*hex_color*, *brightness_offset=1*)

Takes a hex color and produces a lighter or darker variant. :returns: new color in hex format

tablexplore.util.**gen_colors**(*cmap*, *n*, *reverse=False*)

Generates n distinct color from a given colormap. :param cmap: The name of the colormap you want to use.

Refer https://matplotlib.org/stable/tutorials/colors/colormaps.html to choose Suggestions: For Metallicity in Astrophysics: Use coolwarm, bwr, seismic in reverse For distinct objects: Use gnuplot, brg, jet,turbo.

Parameters

- **n** (*int*) – Number of colors you want from the cmap you entered.
- **reverse** (*bool*) – False by default. Set it to True if you want the cmap result to be reversed.

Returns
A list with hex values of colors.

Return type
colorlist(list)

Taken from the mycolorpy package by binodbhttr see also https://matplotlib.org/stable/tutorials/colors/colormaps.html

tablexplore.util.**gen_lower**(*n=2*)

Generate lower case words

tablexplore.util.**gen_upper**(*n=2*)

Generate upper case words

tablexplore.util.**gen_word**(*n=2*)

Generate words with strings or symbols

tablexplore.util.**getAttributes**(*obj*)

Get non hidden and built-in type object attributes that can be persisted

tablexplore.util.**getEmptyData**(*rows=10*, *columns=4*)

tablexplore.util.**getFonts**()

Get the current list of system fonts

tablexplore.util.**getPresetData**(*name*)

Get iris dataset

tablexplore.util.**getSampleData**(*rows=400*, *cols=5*, *namelen=2*)

Generate sample data

tablexplore.util.**get_user_config_directory**()

Returns a platform-specific root directory for user config settings.

tablexplore.util.**hex_to_rgb**(*value*)

tablexplore.util.**random_colors**(*n=10*, *seed=1*)

Generate random hex colors as list of length n.

tablexplore.util.**setAttributes**(*obj*, *data*)

Set attributes from a dict. Used for restoring settings in tables

tablexplore.util.**show_colors**(*colors*, *ax=None*)

display a list of colors

tablexplore.util.**valueToBool**(*value*)

8.1.11 tablexplore.widgets module

Implements various widgets for tablexplore Created Oct 2021 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 tablexplore.widgets.**ScratchPad**(*parent=None*)

Bases: QWidget

Temporary storage widget for plots and other items. Currently supports storing text, mpl figures and dataframes

clear()

Clear plots

closeEvent(*event*)

Close

createWidgets()

Create widgets. Plot on left and dock for tools on right.

newText()

Add a text editor

remove(*idx*)

Remove selected tab and item widget

save()

Save selected item

saveAll()

Save all figures in a folder

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update(*items*)

Display a dict of stored objects

8.1.12 Module contents

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z