StyleFrame Documentation

Release latest

Dec 26, 2017

Contents

1 Installation and testing **3**

2 Basic Usage Examples **5**

3 API Documentation **7**
- 3.1 utils . . . 7
- 3.2 Styler Class . . . 9
- 3.3 StyleFrame Class . . . 10

4 Commandline Interface **15**
- 4.1 General Information . . . 15
- 4.2 Usage . . . 15
- 4.3 JSON Format . . . 15

A library that wraps pandas and openpyxl and allows easy styling of dataframes in excel.

Contents:

CHAPTER 1

Installation and testing

```
$ pip install styleframe
```

To make sure everything works as expected, run StyleFrame's unittests:

```
from StyleFrame import tests

tests.run()
```

CHAPTER 2

Basic Usage Examples

StyleFrame's `init` supports all the ways you are used to initiate pandas dataframe. An existing dataframe, a dictionary or a list of dictionaries:

```
from StyleFrame import StyleFrame, Styler, utils

sf = StyleFrame({'col_a': range(100)})
```

Applying a style to rows that meet a condition using pandas selecting syntax. In this example all the cells in the *col_a* column with the value > 50 will have blue background and a bold, sized 10 font:

```
sf.apply_style_by_indexes(indexes_to_style=sf[sf['col_a'] > 50],
                          cols_to_style=['col_a'],
                          styler_obj=Styler(bg_color=utils.colors.blue, bold=True,␣
↪font_size=10))
```

Creating ExcelWriter used to save the excel:

```
ew = StyleFrame.ExcelWriter(r'C:\my_excel.xlsx')
sf.to_excel(ew)
ew.save()
```

It is also possible to style a whole column or columns, and decide whether to style the headers or not:

```
sf.apply_column_style(cols_to_style=['a'], styler_obj=Styler(bg_color=utils.colors.
↪green),
                      style_header=True)
```

CHAPTER 3

API Documentation

3.1 utils

The utils module contains the most widely used values for styling elements such as colors and border types for convenience. It is possible to directly use a value that is not present in the utils module as long as Excel recognises it.

3.1.1 utils.number_formats

```
general = 'General'
general_integer = '0'
general_float = '0.00'
percent = '0.0%'
thousands_comma_sep = '#,##0'
date = 'DD/MM/YY'
time_24_hours = 'HH:MM'
time_24_hours_with_seconds = 'HH:MM:SS'
time_12_hours = 'h:MM AM/PM'
time_12_hours_with_seconds = 'h:MM:SS AM/PM'
date_time = 'DD/MM/YY HH:MM'
date_time_with_seconds = 'DD/MM/YY HH:MM:SS'
```

3.1.2 utils.colors

```
white = op_colors.WHITE
blue = op_colors.BLUE
dark_blue = op_colors.DARKBLUE
yellow = op_colors.YELLOW
dark_yellow = op_colors.DARKYELLOW
green = op_colors.GREEN
dark_green = op_colors.DARKGREEN
```

```
black = op_colors.BLACK
red = op_colors.RED
dark_red = op_colors.DARKRED
purple = '800080'
grey = 'D3D3D3'
```

3.1.3 utils.fonts

```
aegean = 'Aegean'
aegyptus = 'Aegyptus'
aharoni = 'Aharoni CLM'
anaktoria = 'Anaktoria'
analecta = 'Analecta'
anatolian = 'Anatolian'
arial = 'Arial'
calibri = 'Calibri'
david = 'David CLM'
dejavu_sans = 'DejaVu Sans'
ellinia = 'Ellinia CLM'
```

3.1.4 utils.borders

```
dash_dot = 'dashDot'
dash_dot_dot = 'dashDotDot'
dashed = 'dashed'
dotted = 'dotted'
double = 'double'
hair = 'hair'
medium = 'medium'
medium_dash_dot = 'mediumDashDot'
medium_dash_dot_dot = 'mediumDashDotDot'
medium_dashed = 'mediumDashed'
slant_dash_dot = 'slantDashDot'
thick = 'thick'
thin = 'thin'
```

3.1.5 utils.horizontal_alignments

```
general = 'general'
left = 'left'
center = 'center'
right = 'right'
fill = 'fill'
justify = 'justify'
center_continuous = 'centerContinuous'
distributed = 'distributed'
```

3.1.6 utils.vertical_alignments

```
top = 'top'
center = 'center'
bottom = 'bottom'
justify = 'justify'
distributed = 'distributed'
```

3.1.7 utils.underline

```
single = 'single'
double = 'double'
```

3.1.8 utils.fill_pattern_types

```
solid = 'solid'
dark_down = 'darkDown'
dark_gray = 'darkGray'
dark_grid = 'darkGrid'
dark_horizontal = 'darkHorizontal'
dark_trellis = 'darkTrellis'
dark_up = 'darkUp'
dark_vertical = 'darkVertical'
gray0625 = 'gray0625'
gray125 = 'gray125'
light_down = 'lightDown'
light_gray = 'lightGray'
light_grid = 'lightGrid'
light_horizontal = 'lightHorizontal'
light_trellis = 'lightTrellis'
light_up = 'lightUp'
light_vertical = 'lightVertical'
medium_gray = 'mediumGray'
```

3.2 Styler Class

Used to represent a style.

3.2.1 Init Arguments

```
Styler(bg_color=None, bold=False, font=utils.fonts.arial, font_size=12, font_
↪color=None,
       number_format=utils.number_formats.general, protection=False, underline=None,
       border_type=utils.borders.thin, horizontal_alignment=utils.horizontal_
↪alignments.center,
       vertical_alignment=utils.vertical_alignments.center, wrap_text=True, shrink_to_
↪fit=True,
       fill_pattern_type=utils.fill_pattern_types.solid, indent=0)
```

bg_color (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) The background color

bold (bool) If true, a bold typeface is used

font (str: one of *utils.fonts* or other font name Excel supports) The font to use

font_size (int) The font size

font_color (str: one of *utils.colors*, hex string or color name ie *'yellow'* Excel supports) The font color

number_format (str: one of *utils.number_formats* or any other format Excel supports) The format of the cell's value

protection (bool) If true, the cell/column will be write-protected

underline (str: one of *utils.underline* or any other underline Excel supports) The underline type

border_type (str: one of *utils.borders* or any other border type Excel supports) The border type

horizontal_alignment (str: one of *utils.horizontal_alignments* or any other horizontal alignment Excel supports) Text's horizontal alignment

vertical_alignment (str: one of *utils.vertical_alignments* or any other vertical alignment Excel supports) Text's vertical alignment

wrap_text (bool)

shrink_to_fit (bool)

fill_pattern_type (str: one of *utils.fill_pattern_types* or any other fill pattern type Excel supports) Cells's fill pattern type

indent (int)

3.2.2 Methods

create_style

arguments None

returns *openpyxl* style object.

3.3 StyleFrame Class

Represent a stylized dataframe

3.3.1 Init Arguments

```
StyleFrame(obj, styler_obj=None)
```

obj Any object that pandas' dataframe can be initialized with: an existing dataframe, a dictionary, a list of dictionaries or another StylerFrame.

styler_obj (Styler) A Styler object. Will be used as the default style of all cells.

3.3.2 Methods

apply_style_by_indexes

arguments

indexes_to_style The StyleFrame indexes to style. This usually passed as pandas selecting syntax. For example, `sf[sf['some_col'] = 20]`

styler_obj (Styler) The *Styler* object that represent the style

cols_to_style=None (str | list | tuple) The column names to apply the provided style to. If `None` all columns will be styled.

height=None (int) If provided, the new height for the matched indexes.

returns self

apply_column_style

arguments

cols_to_style (str | list | tuple) The column names to style.

styler_obj (Styler) A *Styler* object.

style_header=False (bool) If True, the column(s) header will also be styled.

use_default_formats=True (bool) If True, the default formats for date and times will be used.

width=None (int) If provided, the new width for the specified columns.

returns self

apply_headers_style

arguments

styler_obj (Styler) A *Styler* object.

returns self

style_alternate_rows

arguments

styles (list | tuple) List or tuple of *Styler* objects to be applied to rows in an alternating manner

returns self

rename

arguments

columns=None (dict) A dictionary from old columns names to new columns names.

inplace=False (bool) If False, a new StyleFrame object will be returned. If True, renames the columns inplace.

returns self if inplace is *True*, new StyleFrame object is *False*

set_column_width

arguments

columns (str | list| tuple) Column name(s).

width (int) The new width for the specified columns.

returns self

set_column_width_dict

arguments

col_width_dict (dict) A dictionary from column names to width.

returns self

set_row_height

arguments

rows (int| list | tuple) Row(s) index.

height (int) The new height for the specified indexes.

returns self

set_row_height_dict

arguments

row_height_dict (dict) A dictionary from row indexes to height.

returns self

read_excel

arguments

path (str) The path to the Excel file to read.

sheetname (str) The sheet name to read from.

read_style=False (bool) If *True* the sheet's style will be loaded to the returned StyleFrame object.

kwargs Any keyword argument pandas' *read_excel* supports.

returns StyleFrame object

A classmethod used to create a StyleFrame object from an existing Excel.

to_excel

arguments

allow_protection=False (bool) Allow to protect the cells that specified as protected. If used `protection=True` in a Styler object this must be set to *True*.

right_to_left=False (bool) Makes the sheet right-to-left.

columns_to_hide=None (str | list | tuple) Columns names to hide.

row_to_add_filters=None (int) Add filters to the given row index, starts from 0 (which will add filters to header row).

columns_and_rows_to_freeze=None (str) Column and row string to freeze. For example "C3" will freeze columns: A, B and rows: 1, 2.

returns self

CHAPTER 4

Commandline Interface

4.1 General Information

Starting with version 1.1 StyleFrame offers a commandline interface that lets you create an xlsx file from a json file.

4.2 Usage

Flag	Explanation
`-v`	Displays the installed versions of StyleFrame and its dependencies
`--json_path`	Path to the json file
`--json`	json string
`--output_path`	Path to the output xlsx file. If not provided defaults to `output.xlsx`

4.2.1 Usage Examples

```
$ styleframe --json_path data.json --output_path data.xlsx
```

```
$ styleframe --json "[{\"sheet_name\":  \"sheet_1\", \"columns\":
[{\"col_name\":  \"col_a\", \"cells\":  [{\"value\":  1}]}]}]"
```

Note: You may need to use different syntax to pass a JSON string depending on your OS and terminal application.

4.3 JSON Format

The input JSON should be thought of as an hierarchy of predefined entities, some of which correspond to a Python class used by StyleFrame. The top-most level should be a list of `sheet` entities (see below).

An example JSON:

```
[
  {
    "sheet_name": "Sheet1",
    "default_styles": {
      "headers": {
        "font_size": 17,
        "bg_color": "yellow"
      },
      "cells": {
        "bg_color": "red"
      }
    },
    "columns": [
      {
        "col_name": "col_a",
        "style": {"bg_color": "blue", "font_color": "yellow"},
        "width": 30,
        "cells": [
          {
            "value": 1
          },
          {
            "value": 2,
            "style": {
              "bold": true,
              "font": "Arial",
              "font_size": 30,
              "font_color": "green",
              "border_type": "double"
            }
          }
        ]
      },
      {
        "col_name": "col_b",
        "cells": [
          {
            "value": 3
          },
          {
            "value": 4,
            "style": {
              "bold": true,
              "font": "Arial",
              "font_size": 16
            }
          }
        ]
      }
    ],
    "row_heights": {
      "3": 40
    },
    "extra_features": {
      "row_to_add_filters": 0,
      "columns_and_rows_to_freeze": "A7",
      "startrow": 5
```

```
    }
  }
]
```

4.3.1 style

Corresponds to: *Styler Class*.

This entity uses the arguments of `Styler.__init__()` as keys. Any missing keys in the JSON will be given the same default values.

`"style": {"bg_color": "yellow", "bold": true}`

4.3.2 cell

This entity represents a single cell in the sheet.

Required keys:

`"value"` - The cell's value.

Optional keys:

`"style"` - The `style` entity for this cell. If not provided, the `style` provided to the `coloumn` entity will be used. If that was not provided as well, the default `Styler.__init__()` values will be used.

`{"value": 42, "style": {"border": "double"}}`

4.3.3 column

This entity represents a column in the sheet.

Required keys:

`"col_name"` - The column name.

`"cells"` - A list of `cell` entities.

Optional keys:

`"style"` - A style used for the entire column. If not provided the default `Styler.__init__()` values will be used.

`"width"` - The column's width. If not provided Excel's default column width will be used.

4.3.4 sheet

This entity represents the entire sheet.

Required keys:

`"sheet_name"` - The sheet's name.

`"columns"` - A list of `column` entities.

Optional keys:

`"default_styles"` - A JSON object with items as keys and `style` entities as values. Currently supported items: `headers` and `cells`.

`"default_styles": {"headers": {"bg_color": "blue"}}`

`"row_heights"` - A JSON object with rows indexes as keys and heights as value.

`"extra_features"` - A JSON that contains the same arguments as the `to_excel` method, such as `"row_to_add_filters"`, `"columns_and_rows_to_freeze"`, `"columns_to_hide"`, `"right_to_left"` and `"allow_protection"`. You can also use other arguments that Pandas' `"to_excel"` accepts.