

---

**styleframe**

**unknown**

**Nov 14, 2023**



# CONTENTS

- 1 Installation and testing** **3**
- 2 Basic Usage Examples** **5**
- 3 API Documentation** **7**
  - 3.1 utils module . . . . . 7
  - 3.2 styler module . . . . . 10
  - 3.3 style\_frame module . . . . . 12
- 4 Commandline Interface** **19**
  - 4.1 General Information . . . . . 19
  - 4.2 Usage . . . . . 19
  - 4.3 JSON Format . . . . . 19



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

Contents:



## INSTALLATION AND TESTING

```
$ pip install styleframe
```

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

```
from StyleFrame import tests  
tests.run()
```





## 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)
```



## API DOCUMENTATION

### 3.1 utils module

This 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'
```

#### decimal\_with\_num\_of\_digits

##### arguments

##### num\_of\_digits

(int) Number of digits after the decimal point

##### returns

A format string that represents a floating point number with the provided number of digits after the decimal point. For example, `utils.number_formats.decimal_with_num_of_digits(2)` will return `'0.00'`

### 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'
```

### 3.1.9 `utils.conditional_formatting_types`

```
num = 'num'
percent = 'percent'
max = 'max'
min = 'min'
formula = 'formula'
percentile = 'percentile'
```

## 3.2 styler module

This module contains classes that represent styles.

### 3.2.1 Styler Class

Used to represent a style.

```
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, comment_author=None,
↪comment_text=None)
```

**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)

**comment\_author**

(str)

**comment\_text**

(str)

## Methods

### combine

A classmethod used to combine *Styler Class* objects. The right-most object has precedence. For example: `Styler.combine(Styler(bg_color='yellow', font_size=24), Styler(bg_color='blue'))` will return `Styler(bg_color='blue', font_size=24)`

**arguments****styles**

Arbitrary number of *Styler* objects

**returns**

*Styler Class* object

### to\_openpyxl\_style

**arguments**

None

**returns**

*openpyxl* style object.

## 3.3 style\_frame module

### 3.3.1 StyleFrame Class

Represent a stylized dataframe

```
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 StyleFrame.

**styler\_obj**

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

#### Methods

#### apply\_style\_by\_indexes

**arguments**

**indexes\_to\_style**

(list | tuple | int | Container) The StyleFrame indexes to style. This usually passed as pandas selecting syntax. For example, `sf[sf['some_col'] = 20]`

**styler\_obj**

(*Styler Class*) Styler object that contains the style which will be applied to indexes in *indexes\_to\_style*

**cols\_to\_style=None**

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

**height=None**

(None | int | float) If provided, height for rows whose indexes are in *indexes\_to\_style*.

**complement\_style=None**

(None | *Styler Class*) Styler object that contains the style which will be applied to indexes not in *indexes\_to\_style*

**complement\_height=None**

(None | int | float) Height for rows whose indexes are not in *indexes\_to\_style*. If not provided then *height* will be used (if provided).

**overwrite\_default\_style=True**

(bool) If *True*, the default style (the style used when initializing StyleFrame) will be overwritten. If *False* then the default style and the provided style will be combined using Styler.combine method.

**returns**

self



## apply\_column\_style

### arguments

**cols\_to\_style**

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

**styler\_obj**

(*Styler Class*) 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**

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

**overwrite\_default\_style=True**

(bool) If *True*, the default style (the style used when initializing *StyleFrame*) will be overwritten. If *False* then the default style and the provided style will be combined using *Styler.combine* method.

**returns**

self

## apply\_headers\_style

### arguments

**styler\_obj**

(*Styler Class*) A *Styler* object.

**style\_index\_header**

(bool) If *True* then the style will also be applied to the header of the index column

**returns**

self

## style\_alternate\_rows

### arguments

**styles**

(list | tuple | set) List or tuple of *Styler Class* 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 | float) 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 | set) Row(s) index.

**height**

(int | float) 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

## add\_color\_scale\_conditional\_formatting

### arguments

#### start\_type

(str: one of *utils.conditional\_formatting\_types* or any other type Excel supports) The type for the minimum bound

#### start\_value

The threshold for the minimum bound

#### start\_color

(str: one of *utils.colors*, hex string or color name ie 'yellow' Excel supports) The color for the minimum bound

#### end\_type

(str: one of *utils.conditional\_formatting\_types* or any other type Excel supports) The type for the maximum bound

#### end\_value

The threshold for the maximum bound

#### end\_color

(str: one of *utils.colors*, hex string or color name ie 'yellow' Excel supports) The color for the maximum bound

#### mid\_type=None

(None | str: one of *utils.conditional\_formatting\_types* or any other type Excel supports) The type for the middle bound

#### mid\_value=None

The threshold for the middle bound

#### mid\_color=None

(None | str: one of *utils.colors*, hex string or color name ie 'yellow' Excel supports) The color for the middle bound

#### columns\_range=None

(None | list | tuple) A two-elements list or tuple of columns to which the conditional formatting will be added to. If not provided at all the conditional formatting will be added to all columns. If a single element is provided then the conditional formatting will be added to the provided column. If two elements are provided then the conditional formatting will start in the first column and end in the second. The provided columns can be a column name, letter or index.

### returns

self

### read\_excel

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

---

**Note:** `read_excel` also accepts all arguments that `pandas.read_excel` accepts as kwargs.

---

#### arguments

##### **path**

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

##### **sheetname**

Deprecated since version 1.6: Use `sheet_name` instead.

##### **sheet\_name=0**

(str | int) The sheet name to read. If an integer is provided then it be used as a zero-based sheet index. Default is 0.

##### **read\_style=False**

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

##### **use\_openpyxl\_styles=True**

(bool) If *True* (and *read\_style* is also *True*) then the styles in the returned StyleFrame object will be Openpyxl's style objects. If *False*, the styles will be *Styler Class* objects. Defaults to *True* for backward compatibility.

---

**Note:** Using `use_openpyxl_styles=False` is useful if you are going to filter columns or rows by style, for example:

```
sf = sf[[col for col in sf.columns if col.style.font == utils.fonts.arial]]
```

---

##### **read\_comments=False**

(bool) If *True* (and *read\_style* is also *True*) cells' comments will be loaded to the returned StyleFrame object. Note that reading comments without reading styles is currently not supported.

#### returns

StyleFrame object

### to\_excel

---

**Note:** `to_excel` also accepts all arguments that `pandas.DataFrame.to_excel` accepts as kwargs.

---

#### arguments

##### **excel\_writer='output.xlsx'**

(str | pandas.ExcelWriter) File path or existing ExcelWriter

##### **sheet\_name='Sheet1'**

(str) Name of sheet the StyleFrame will be exported to

**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**

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

**row\_to\_add\_filters=None**

(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**

(None | str) Column and row string to freeze. For example “C3” will freeze columns: A, B and rows: 1, 2.

**best\_fit=None**

(None | str | list | tuple | set) single column, list, set or tuple of columns names to attempt to best fit the width for.

---

**Note:** `best_fit` will attempt to calculate the correct column-width based on the longest value in each provided column. However this isn't guaranteed to work for all fonts (works best with monospaced fonts). The formula used to calculate a column's width is equivalent to

$$(\text{len}(\text{longest\_value\_in\_column}) + \text{A\_FACTOR}) * \text{P\_FACTOR}$$

The default values for `A_FACTOR` and `P_FACTOR` are 13 and 1.3 respectively, and can be modified before calling `StyleFrame.to_excel` by directly modifying `StyleFrame.A_FACTOR` and `StyleFrame.P_FACTOR`

---

**returns**

self



## 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
<code>-v</code>	Displays the installed versions of StyleFrame and its dependencies
<code>--json_path</code>	Path to the <code>json</code> file
<code>--json</code>	<code>json</code> string
<code>--output_path</code>	Path to the output <code>xlsx</code> file. If not provided defaults to <code>output.xlsx</code>

#### 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).

The provided `JSON` is validated against the following schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "sheets",
  "definitions": {
    "Sheet": {
```

(continues on next page)

(continued from previous page)

```
"$id": "#sheet",
"title": "sheet",
"type": "object",
"properties": {
  "sheet_name": {
    "type": "string"
  },
  "columns": {
    "type": "array",
    "items": {
      "$ref": "#/definitions/Column"
    },
    "minItems": 1
  },
  "row_heights": {
    "type": "object"
  },
  "extra_features": {
    "type": "object"
  },
  "default_styles": {
    "type": "object",
    "properties": {
      "headers": {
        "$ref": "#/definitions/Style"
      },
      "cells": {
        "$ref": "#/definitions/Style"
      }
    }
  },
  "additionalProperties": false
},
"required": [
  "sheet_name",
  "columns"
],
"Column": {
  "$id": "#column",
  "title": "column",
  "type": "object",
  "properties": {
    "col_name": {
      "type": "string"
    },
    "style": {
      "$ref": "#/definitions/Style"
    },
    "width": {
      "type": "number"
    }
  },
```

(continues on next page)



(continued from previous page)

```

        "cells": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/Cell"
            }
        },
        "required": [
            "col_name",
            "cells"
        ]
    },
    "Cell": {
        "$id": "#cell",
        "title": "cell",
        "type": "object",
        "properties": {
            "value": {},
            "style": {
                "$ref": "#/definitions/Style"
            }
        },
        "required": [
            "value"
        ],
        "additionalProperties": false
    },
    "Style": {
        "$id": "#style",
        "title": "style",
        "type": "object",
        "properties": {
            "bg_color": {
                "type": "string"
            },
            "bold": {
                "type": "boolean"
            },
            "font": {
                "type": "string"
            },
            "font_size": {
                "type": "number"
            },
            "font_color": {
                "type": "string"
            },
            "number_format": {
                "type": "string"
            },
            "protection": {
                "type": "boolean"
            }
        }
    }

```

(continues on next page)

(continued from previous page)

```

        },
        "underline": {
            "type": "string"
        },
        "border_type": {
            "type": "string"
        },
        "horizontal_alignment": {
            "type": "string"
        },
        "vertical_alignment": {
            "type": "string"
        },
        "wrap_text": {
            "type": "boolean"
        },
        "shrink_to_fit": {
            "type": "boolean"
        },
        "fill_pattern_type": {
            "type": "string"
        },
        "indent": {
            "type": "number"
        }
    },
    "additionalProperties": false
},
{
    "type": "array",
    "items": {
        "$ref": "#/definitions/Sheet"
    },
    "minItems": 1
}

```

An example JSON:

```

[
  {
    "sheet_name": "Sheet1",
    "default_styles": {
      "headers": {
        "font_size": 17,
        "bg_color": "yellow"
      },
      "cells": {
        "bg_color": "red"
      }
    },
    "columns": [
      {

```

(continues on next page)

(continued from previous page)

```
"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 column 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.