
table-compositor Documentation

Release 1.0.0

Guru Devanla

Sep 02, 2022

CONTENTS

1	Introduction	3
1.1	Getting Started	3
1.2	Installation	3
2	Basics	5
2.1	Sample Data	5
2.2	A Hello World Example: Dataframe to Xlsx	5
2.3	Building the Presentation Model	7
2.4	Improving on our first iteration	9
2.5	Multi-hierarchical columns and indices	11
3	Layouts	15
4	Example of XLSX Styles	19
4.1	Style with background color	19
4.2	Style with percentage formatting	19
4.3	Style with alignment and fonts	19
4.4	Using a different XLSX Writer Engine	20
5	Example of HTML Styles	21
5.1	Style for headers	21
5.2	Style for cell holding numeric values	21
5.3	Style using the html_styles.td_style object	22
6	XLSX Examples	23
6.1	Helper Functions (Data loading routines)	23
6.2	Example 1 - DataFrame with default styles	24
6.3	Example 2 - DataFrame with custom styles	25
6.4	Example 3 - Simple DataFrame with Layouts	28
6.5	Example 4 - DataFrames with Multi-hierarchical columns and indices	30
7	HTML Examples	35
7.1	Helper Functions (Data loading routines)	35
7.2	Example 1 - DataFrame with default styles	36
7.3	Example 2 - DataFrame with custom styles	37
7.4	Example 3 - Simple DataFrame with Layouts	40
7.5	Example 4 - DataFrames with Multi-hierarchical columns and indices	44
8	API	47
8.1	Building the presentation model	47
8.2	Rendering to XLSX	49

8.3	Rendering to HTML	49
8.4	Helper XLSX Styles	49
8.5	Helper HTML Styles	52
9	Code Used in documentation	55
9.1	Basic Usage	55
9.2	XLSX Examples	59
9.3	HTML Examples	67
10	Indices and tables	77
	Index	79

Contents:

INTRODUCTION

The table-compositor library provides the API to render data stored in table-like data structures. Currently the library only supports rendering data available in a Panda's DataFrames. The DataFrame layout is used as the table layout(including single and multi hierarchical columns/indices) by the library. The table layout is rendered directly on to an xlsx sheet or to a html page. Styling and layout attributes can be used to render colorful xlsx or html reports. The library also supports rendering of multiple data frames into a single xlsx sheet or html page (with horizontal/vertical layouts). The objective of the library is to be able to use the DataFrame as the API to configure the style and layout properties of the report. Callback functions are provided to customize all styling properties. The nice thing about the callback functions are that the style properties are set on cells indexed with index/column values available in the original dataframe used during rendering.

Code: <https://github.com/InvestmentSystems/table-compositor>

Docs: <http://table-compositor.readthedocs.io>

Packages: <https://pypi.python.org/pypi/table-compositor>

1.1 Getting Started

The table-compositor library builds on the concept of Panda's DataFrame as API to render colorful reports. The various ways of providing styling attributes and choosing layouts are demonstrated with numerous examples in the documentation. Please refer to the Basics section of the documentation to get started with the HelloWorld example.

1.2 Installation

A standard setuptools installer is available via PyPI:

<https://pypi.python.org/pypi/table-compositor>

Or, install via pip3:

```
pip3 install table-compositor
```

Source code can be obtained here:

<https://github.com/InvestmentSystems/table-compositor>

BASICS

The purpose of this library is to use the Pandas DataFrame as an interface to represent the layout of a table that needs to be rendered to an xlsx file or as an html table. The library abstracts away the tedious work of working at the *cell* level of an xlsx sheet or a html table. It provides a call-back mechanism by which the user is able to provide values that need to be rendered and also the styling that needs to be used for each cell in the rendered table. The library is also capable of laying out multiple tables in the same sheet which are evenly spaced vertically or horizontally based on the layout configuration provided.

2.1 Sample Data

During the later part of this documentation, we will use the sample data from the Social Security Administration which contains the U.S. child birth name records. We choose this sample data for two reasons. We reuse some of the discussion that are outlined by Wes McKinnery's Python For Data Analysis, 2nd Edition(2017). The same data is also used in the documentation of another library *function-pipe* <<http://function-pipe.readthedocs.io/en/latest/index.html>> that the Investment Systems Group has open-sourced.

<https://www.ssa.gov/oact/babynames/names.zip>

Further more, we will assume that a flattened file from all the smaller files in the .zip file is available after we invoke the following function.

Please refer to the XLSX Examples section for code that loads this data.

2.2 A Hello World Example: Dataframe to Xlsx

Every use of this library involves four steps.

1. We build a dataframe that resembles the shape of the table that will be rendered.
2. The dataframe is passed as an argument to the function called `build_presentation_model`. This function accepts a *dataframe* and also a number of functions as arguments. We call the value returned by this function, the *presentation_model*.
3. Create a *layout* of multiple *presentation models* (if we want more than one table rendered in same xlsx sheet or same html page)
4. Call the `render_xlsx` or `render_html` functions on the respective writers. For xlsx files either `OpenPyxlCompositor`(uses *openpyxl* library) or `XlsxWriterCompositor`(uses *xlsxwriter* library). For HTML use the *HTMLWriter*.

2.2.1 A Quick Look at a Xlsx example

We will start with a simple dataframe and render the dataframe as-is to a xlsx file

```
import pandas as pd
from table_compositor.table_compositor import build_presentation_model
from table_compositor.xlsx_writer import OpenPyxlCompositor
# Note: use XlsxWriterCompositor to use xlsxwriter library

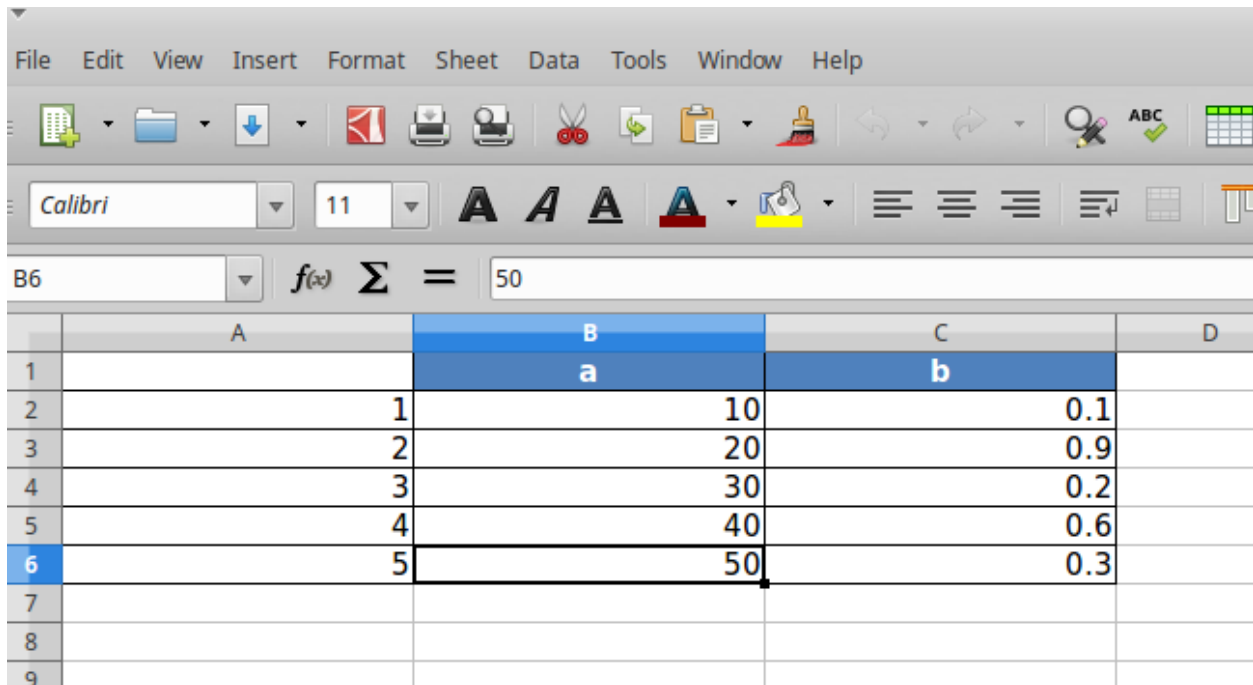
sample_df = pd.DataFrame(dict(a=[10, 20, 30, 40, 50], b=[0.1, 0.9, 0.2, 0.6, 0.3]),
    ↪ index=[1, 2, 3, 4, 5])

# create a presentation model
# defaults to engine='openpyxl'. Needs to be set to 'xlsxwriter' to use `xlsxwriter`
    ↪ library instead.
presentation_model = build_presentation_model(df=sample_df)

# create a layout, which is usually a nested list of presentation models
layout = [presentation_model]

# render to xlsx
output_fp = '/tmp/example1.xlsx'
OpenPyxlCompositor.to_xlsx(layout, output_fp=output_fp)
```

Running this code produces the following output:



	A	B	C	D
1		a	b	
2	1	10	0.1	
3	2	20	0.9	
4	3	30	0.2	
5	4	40	0.6	
6	5	50	0.3	
7				
8				
9				

In the above code snippet, we first created a dataframe called `sample_df`.

To render this *dataframe*, we first invoke `build_presentation_model`. The `build_presentation_model` accepts the *dataframe* as its first argument. In this example, we use the *defaults* provided by this method for all other arguments. The `build_presentation_model` returns an *presentation_model* object.

Before we call `OpenPyxlCompositor.to_xlsx` we create a *layout*. A *layout* is a nested list of *presentation_models*. In

our case, since we have only one *presentation_model* we create a list with a single element. Later on when we work with multiple presentation models that need to be rendered on to the same sheet, we could create nested list such as `[[model1, model2], [model3]]` etc.

2.3 Building the Presentation Model

The *build_presentation_model* function is the most important interface in this library. This function exposes all the functionality that is required to render beautiful looking excel worksheets or html tables.

We will now build up on our previous example and add styling to the report we generate. Before, we do that lets take a quick look at the signature of *build_presentation__model*.

```
table_compositor.table_compositor.build_presentation_model(*, df, output_format='xlsx',
                                                           data_value_func=None,
                                                           column_style_func=None,
                                                           data_style_func=None,
                                                           header_style_func=None,
                                                           header_value_func=None,
                                                           index_style_func=None,
                                                           index_value_func=None,
                                                           index_name_func=None,
                                                           index_name_style_func=None,
                                                           engine='openpyxl', **kwargs)
```

Construct and return the presentation model that will be used while rendering to html/xlsx formats. The returned object has all the information required to render the tables in the requested format. The details of the object is transparent to the caller. It is only exposed for certain advanced operations.

Parameters

- **df** – The dataframe representation of the table. The shape of the dataframe closely resembles the table that will be rendered in the requested format.
- **output_format** – ‘html’ or ‘xlsx’
- **data_value_func** – example: `lambda idx, col: df.loc[idx, col]`, assuming df is in the closure. This can be None, if no data transformation is required to the values already present in the source df
- **column_style_func** – the function can substitute the *data_style_func*, if the same style can be applied for the whole column. This argument should be preferred over the *data_style_func* argument. Using this option provides better performance since the fewer objects will be created internally and fewer callbacks are made to this function when compared to *data_style_func*. This argument only applies to the data contained in the dataframe and not the cell where the headers are rendered. For fine grained control at *cell* level, the *data_style_func* argument can be used. For more information on return values of this function, refer to the documentation for *data_style_func* argument.
- **data_style_func** – used to provide style at the cell level. Example: `lambda idx, col: return dict(font=Font(...))`, where *Font* is the openpyxl object and *font* is the attr available in the *cell* instance of openpyxl. For xlsx, the keys in the dict are the attrs of the *cell* object in openpyxl and the values correspond to the value of that attribute. Example are found in *xlsx_styles* module. For html, the key-value pairs are any values that go into to the style attribute of a *td*, *th* cell in html. Examples are found in *html_styles* module. example: `dict(background-color='#F8F8F8')`. When performance becomes an issue, and cell level control is not needed, it is recommended to use the *column_style_func* argument rather than this argument. If the preferred engine is *XlswWriter*, then the style dictionary returned should have key/values

compatible with the *Format* object declared in the *XlsxWriter* library. A reference can be found in `xlsx_styles.XlsxWriterStyleHelper` class`

- **header_value_func** – func that takes a object of type *IndexNode*. The *IndexNode* contains the attributes that refer to the header being rendered. The returned value from this function is displayed in place of the header in the dataframe at the location. The two properties available on the *IndexNode* object are *value* and *key*. The *key* is useful to identify the exact index and level in context while working with multi-hierarchical columns.
- **header_style_func** – func that takes a object of type *IndexNode*. The return value of this function is similar to `data_style_func`.
- **index_value_func** – func that takes a object of type *IndexNode*. The *IndexNode* contains the attributes that refer to the index being rendered. The returned value from this function is displayed in place of the index in the dataframe at the location.
- **index_style_func** – func that takes a object of type *IndexNode*. The return value of this function is similar to `data_style_func`.
- **index_name_func** – func that returns a string for index name (value to be displayed on top-left corner, above the index column)
- **index_name_style** – the style value same as `data_style_func` that will be used to style the cell
- **engine** – required while building presentation model for xlsx. Argument ignored for HTML rendering. This argument is used to provide the default callback style functions, where the style dictionary returned by the callback functions should be compatible with the engine being used.
- **kwargs** – ‘hide_index’ - if True, then hide the index column, default=False
‘hide_header, - if True, then hide the header, default=False
‘use_convert’ - if True, do some conversions from dataframe values to values excel can understand for example np.NaN are converted to NaN strings

Returns

A presentation model, to be used to create layout and provide the layout to the html or xlsx writers.

About the callback functions provided as arguments:

Note that callback function provided as arguments to this function are provided with either a tuple of index, col arguments are some information regarding the index or headers being rendered. Therefore, a common pattern would be to capture the *dataframe* being rendered in a closure of this callback func before passing them as arguments.

For example:

```
df = pd.DataFrame(dict(a=[1, 2, 3]))
```

```
def data_value_func():
```

```
    def _inner(idx, col):
```

```
        return df.loc[idx, col] * 10.3
```

```
    return _inner
```

```
pm = build_presentation_model(df=df, data_value_func=data_value_func())
```

2.4 Improving on our first iteration

Now, that we got a overview of the `build_presentation_mode` function, lets try setting these arguments to improve the look of our reports.

Say, we have the following requirements:

1. Display column 'A' as in dollar format.
2. Display column 'B' as percentage values.'
3. Set back-ground color of column 'B' to red if value is less than 50%
4. Capitalize all the column headers and add a yellow background
5. Multiply all index values by 100 while rendering and add a color to the background.
6. Display a 'custom text' on the top left corner, where pandas whole usually display the index name if available.

We update our previous example to do the following:

```

1 import os
2 import tempfile
3
4 import pandas as pd
5
6 from table_compositor.table_compositor import build_presentation_model
7 from table_compositor.xlsx_styles import OpenPyxlStyleHelper
8
9 # There are equivalent classes for using xlsxwriter library. Namely,
10 # XlsxWriterCompositor and XlsxWriterStyleHelper
11 from table_compositor.xlsx_writer import OpenPyxlCompositor
12
13
14 def basic_example2():
15
16     df = pd.DataFrame(
17         dict(a=[10, 20, 30, 40, 50], b=[0.1, 0.9, 0.2, 0.6, 0.3]), index=[1, 2, 3, 4, 5]
18     )
19
20     def style_func(idx, col):
21         if col == "b":
22             return OpenPyxlStyleHelper.get_style(number_format="0.00%")
23         else:
24             # for 'a' we do dollar format
25             return OpenPyxlStyleHelper.get_style(number_format="$#,##.00")
26
27     # create a presentation model
28     # note the OpenPyxlStyleHelper function available in xlsx_styles module. But a
29     ↪return value of style function
30     # can be any dict whose keys are attributes of the OpenPyxl cell object.
31     presentation_model = build_presentation_model(
32         df=df,
33         data_value_func=lambda idx, col: df.loc[idx, col] * 10
34         if col == "a"
35         else df.loc[idx, col],

```

(continues on next page)

(continued from previous page)

```

22     data_style_func=style_func,
23     header_value_func=lambda node: node.value.capitalize(),
24     header_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
25     index_value_func=lambda node: node.value * 100,
26     index_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
27     index_name_func=lambda _: "Basic Example",
28     index_name_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
29 )
30
31 # create a layout, which is usually a nested list of presentation models
32 layout = [presentation_model]
33
34 # render to xlsx
35 output_fp = os.path.join(tempfile.gettempdir(), "basic_example2.xlsx")
36 OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=output_fp)
37
38

```

On line 3 we create the dataframe.

To satisfy the requirements we listed above we pass the callback function to the *build_presentation_model*. Note that some helper functions are available in *xlsx_style* function to create styles for openpyxl. But, any other dict with keys that are *attr* of cell object of openpyxl should work. The above example produces the output as shown below:

	A	B	C
1	Basic Example	A	B
2	100	\$100.00	10.00%
3	200	\$200.00	90.00%
4	300	\$300.00	20.00%
5	400	\$400.00	60.00%
6	500	\$500.00	30.00%
7			
8			

2.5 Multi-hierarchical columns and indices

Rendering dataframes with multi-hierarchical columns or indices are very similar to rendering the simpler dataframes. The `data_value_func` and `data_style_func` work the same way. The functions that handle *index* cell rendering and *column* header rendering can access the *IndexNode* object that is passed to those functions to determine the value and level that is currently being rendered. This becomes clearer with an example.

We demonstrate this by setting a variety of colors to each cell that holds one of the values of the hierarchical columns or indices.

Note that the *IndexNode* argument passed to the callback function has a *node.key* field that unique identifies each cell with a name that is built appending the value of each item in the index or column hierarchy.

```

1 import os
2 import tempfile
3
4 import pandas as pd
5
6 from table_compositor.table_compositor import build_presentation_model
7 from table_compositor.xlsx_styles import OpenPyxlStyleHelper
8
9 # There are equivalent classes for using xlsxwriter library. Namely,
10 # XlsxWriterCompositor and XlsxWriterStyleHelper
11 from table_compositor.xlsx_writer import OpenPyxlCompositor
12
13
14 def basic_example3():
15
16     df = pd.DataFrame(
17         dict(
18             a=[10, 20, 30, 40],
19             b=[0.1, 0.9, 0.2, 0.6],
20             d=[50, 60, 70, 80],
21             e=[200, 300, 400, 500],
22         )
23     )
24     df.columns = pd.MultiIndex.from_tuples(
25         [("A", "x"), ("A", "y"), ("B", "x"), ("B", "y")]
26     )
27     df.index = pd.MultiIndex.from_tuples([(1, 100), (1, 200), (2, 100), (2, 200)])
28     print(df)
29
30     def index_style_func(node):
31         # node.key here could be one of (1,), (1, 100), (2,), (2, 100), (2, 200)
32         bg_color = "FFFFFF"
33         if node.key == (1,) or node.key == (2,):
34             bg_color = "9E80B8"
35         elif node.key[1] == 100:
36             bg_color = "4F90C1"
37         elif node.key[1] == 200:
38             bg_color = "6DC066"
39         return OpenPyxlStyleHelper.get_style(bg_color=bg_color)

```

(continues on next page)

(continued from previous page)

```

28 def header_style_func(node):
29     bg_color = "FFFFFF"
30     if node.key == ("A",) or node.key == ("B",):
31         bg_color = "9E80B8"
32     elif node.key[1] == "x":
33         bg_color = "4F90C1"
34     elif node.key[1] == "y":
35         bg_color = "6DC066"
36     return OpenPyxlStyleHelper.get_style(bg_color=bg_color)
37
38     # create a presentation model
39     # note the OpenPyxlStyleHeloer function available in xlsx_styles module. But a
↪return value of style function
40     # can be any dict whose keys are attributes of the OpenPyxl cell object.
41     presentation_model = build_presentation_model(
42         df=df,
43         index_style_func=index_style_func,
44         header_style_func=header_style_func,
45         index_name_func=lambda _: "Multi-Hierarchy Example",
46     )
47
48     # create a layout, which is usually a nested list of presentation models
49     layout = [presentation_model]
50
51     # render to xlsx
52     output_fp = os.path.join(tempfile.gettempdir(), "basic_example3.xlsx")
53     OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=output_fp)
54
55

```

The above function gives us the *xlsx* file shown below. Note the colors used to render the indices and columns and review how the two functions, namely, *index_style_function* and *header_style_function* provide the colors based on the *IndexNode* attributes. You will notice the use of *node.key* in these functions to identify each cell uniquely.

The screenshot shows a spreadsheet application window. The menu bar includes File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, and Help. The toolbar contains icons for file operations (new, open, save, print, copy, paste, undo, redo) and formatting (font face, font size, bold, italic, underline, text color, background color, fill color, alignment). The formula bar shows '16' and a function 'f(x)' with a summation symbol and an equals sign. The spreadsheet grid has columns A through H and rows 1 through 8. The data is organized into a multi-hierarchy table with rows 1-6 and columns C-F. Row 1 has a header 'Multi-Hierarchy Example' in column A. Row 2 has sub-headers 'A' and 'B' in columns C and D. Row 3 has data '100', '10', '0.1', '50', '200' in columns C-F. Row 4 has data '200', '20', '0.9', '60', '300' in columns C-F. Row 5 has data '100', '30', '0.2', '70', '400' in columns C-F. Row 6 has data '200', '40', '0.6', '80', '500' in columns C-F. The cells are colored: row 1 is purple, row 2 is green, row 3 is blue, row 4 is green, row 5 is blue, and row 6 is green. The cells are also outlined with a thick black border.

	A	B	C	D	E	F	G	H
1			A		B			
2	Multi-Hierarchy Example		x	y	x	y		
3		100	10	0.1	50	200		
4	1	200	20	0.9	60	300		
5		100	30	0.2	70	400		
6	2	200	40	0.6	80	500		
7								
8								

LAYOUTS

Apart from providing styling and formatting facilities, the library also provides a powerful way to layout multiple tables on one sheet. In this section we will look at some examples.

We will use the same presentation model from *basic_example2()*. We will layout the presentation models with different layouts.

```
1 import os
2 import tempfile
3
4 import pandas as pd
5
6 from table_compositor.table_compositor import build_presentation_model
7 from table_compositor.xlsx_styles import OpenPyxlStyleHelper
8
9 # There are equivalent classes for using xlsxwriter library. Namely,
10 # XlsxWriterCompositor and XlsxWriterStyleHelper
11 from table_compositor.xlsx_writer import OpenPyxlCompositor
12
13
14 def layout_example1():
15
16     df = pd.DataFrame(
17         dict(a=[10, 20, 30, 40, 50], b=[0.1, 0.9, 0.2, 0.6, 0.3]), index=[1, 2, 3, 4, 5]
18     )
19
20     def style_func(idx, col):
21         if col == "b":
22             return OpenPyxlStyleHelper.get_style(number_format="0.00%")
23         else:
24             # for 'a' we do dollar format
25             return OpenPyxlStyleHelper.get_style(number_format="$#,##.00")
26
27     # create a presentation model
28     # note the OpenPyxlStyleHeloer function available in xlsx_styles module. But a
29     ↪return value of style function
30     # can be any dict whose keys are attributes of the OpenPyxl cell object.
31     presentation_model = build_presentation_model(
32         df=df,
33         data_value_func=lambda idx, col: df.loc[idx, col] * 10
34         if col == "a"
35         else df.loc[idx, col],
```

(continues on next page)

(continued from previous page)

```

22     data_style_func=style_func,
23     header_value_func=lambda node: node.value.capitalize(),
24     header_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
25     index_value_func=lambda node: node.value * 100,
26     index_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
27     index_name_func=lambda _: "Basic Example",
28     index_name_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
29 )
30
31 # start_layout_code_1
32 # create a layout, which is usually a nested list of presentation models
33 layout = [[presentation_model], [[presentation_model], [presentation_model]]]
34
35 # render to xlsx
36 output_fp = os.path.join(tempfile.gettempdir(), "layout_vertical_example1.xlsx")
37 # the default value for orientation is 'vertical'
38 OpenPyxlCompositor.to_xlsx(
39     layout=layout, output_fp=output_fp, orientation="vertical"
40 )
41
42 output_fp = os.path.join(tempfile.gettempdir(), "layout_horizontal_example1.xlsx")
43 OpenPyxlCompositor.to_xlsx(
44     layout=layout, output_fp=output_fp, orientation="horizontal"
45 )
46 print("Writing xlsx file=", output_fp)
47
48 # mutiple nesting
49 layout_complex = [
50     presentation_model,
51     [presentation_model, [presentation_model, presentation_model]],
52 ]
53
54 output_fp = os.path.join(tempfile.gettempdir(), "layout_complex_example1.xlsx")
55 OpenPyxlCompositor.to_xlsx(
56     layout=layout_complex, output_fp=output_fp, orientation="vertical"
57 )
58 print("Writing xlsx file=", output_fp)
59 # end_layout_code_1
60
61

```

In the preceding example, we create two layouts. On line 28, we have a layout defined and then rendered to two files with different *orientations*.

When the orientation is *vertical*, then each item (*presentation_model*) in the list is layed out vertically. The orientation flips between *vertical* and *horizontal* for every nested listed that is encountered. In this example, you will notice that since the second item in the outer list is a list, the two presentation models in the inner list are rendered side-by-side (i.e. with horizontal orientation)

	A	B	C	D	E	F	G	H
1	Basic Example	A	B					
2	100	\$100.00	10.00%					
3	200	\$200.00	90.00%					
4	300	\$300.00	20.00%					
5	400	\$400.00	60.00%					
6	500	\$500.00	30.00%					
7								
8	Basic Example	A	B		Basic Example	A	B	
9	100	\$100.00	10.00%		100	\$100.00	10.00%	
10	200	\$200.00	90.00%		200	\$200.00	90.00%	
11	300	\$300.00	20.00%		300	\$300.00	20.00%	
12	400	\$400.00	60.00%		400	\$400.00	60.00%	
13	500	\$500.00	30.00%		500	\$500.00	30.00%	
14								

When the value of orientation argument is changed to *horizontal*, the renderer renders the outerlist horizontally and flips the orientation of inner lists to vertical. The second output is show below.

	A	B	C	D	E	F	G	H
1	Basic Example	A	B		Basic Example	A	B	
2	100	\$100.00	10.00%		100	\$100.00	10.00%	
3	200	\$200.00	90.00%		200	\$200.00	90.00%	
4	300	\$300.00	20.00%		300	\$300.00	20.00%	
5	400	\$400.00	60.00%		400	\$400.00	60.00%	
6	500	\$500.00	30.00%		500	\$500.00	30.00%	
7								
8					Basic Example	A	B	
9					100	\$100.00	10.00%	
10					200	\$200.00	90.00%	
11					300	\$300.00	20.00%	
12					400	\$400.00	60.00%	
13					500	\$500.00	30.00%	
14								

EXAMPLE OF XLSX STYLES

In the preceding examples, we have used the functions provided by *xslx_styles.OpenPyxlStyleHelper* to return the required style dictionary. Some examples of style dictionaries that can be returned by functions returning styles are provided below for reference. For details on how to build style attributes refer to the *openpyxl* documentation.

4.1 Style with background color

```
from openpyxl.styles import PatternFill
from openpyxl.styles import fills

fill = PatternFill(fgColor=Color('4f81BD'), patternType=fills.FILL_SOLID)

# note that we return a dict, whose key = `fill` which is an
# attribute of `cell` object in `openpyxl`
style = dict(fill=fill)
```

4.2 Style with percentage formatting

```
number_format = '0.00%'

fill = PatternFill(fgColor=Color('4f81BD'), patternType=fills.FILL_SOLID)

# note that we return a dict, whose key = `number_format` which is an
# attribute of `cell` object in `openpyxl`
style = dict(number_format=number_format)
```

4.3 Style with alignment and fonts

```
from openpyxl.styles import Alignment
from openpyxl.styles import Font
font=Font(bold=True, color='FFFFFF')

# note that we return a dict, whose key = `number_format` which is an
# attribute of `cell` object in `openpyxl`
style = dict(alignment=Alignment(horizontal=center, font=font)
```

4.4 Using a different XLXS Writer Engine

Note that if *xlsxwriter* library is used, the keys in the dictionary returned by the callback funcs should match the keys required to build the *Format* object declared in the *xlsxwriter* library. Some examples of these keys can be found in *xlsx_styles.XlsxWriterStyleHelper* class.

EXAMPLE OF HTML STYLES

In the Basic section, we only saw examples of how to render dataframes to a xlsx format. The same setup can be used to render dataframes to HTML using the `html_writer.HTMLWriter.to_html` function. The only thing that has to be changed is the style attributes that are being returned. We want our style providing functions to return style attributes that can be inlined into the `style` attribute of a `<td>` or `<th>` tag.

Some examples of style dictionaries that can be return by functions returning styles are provided below for reference.

5.1 Style for headers

```
style = dict(
    text_align='center',
    background_color='#4F81BD',
    color='#FFFFFF',
    font_weight='bold',
    white_space='pre',
    padding='10px',
    border=1)
```

5.2 Style for cell holding numeric values

```
numeric_style = dict(
    text_align='right',
    background_color='#FFFFFF',
    color='#000000',
    font_weight='normal',
    white_space='pre',
    padding='10px',
    border=None)
```

5.3 Style using the `html_styles.td_style` object

```
style = td_style(  
    text_align='center',  
    background_color='#4F81BD',  
    color='#FFFFFF',  
    font_weight='bold',  
    white_space='pre',  
    padding='10px',  
    border=1)
```

XLSX EXAMPLES

This page provides a list of examples that demonstrate rendering xlsx output from the given dataframe. Each example is self-contained inside a class. We have some helper functions to provide the data we need for this examples

All examples listed in this section use *OpenPyxlStyleHelper* and *OpenPyxlCompositor*. To use the *xlsxwriter* library replace these with *XlsxWriterStyleHelper* and *XlsxWriterCompositor* respectively.

If the callback funcs do not use the **StyleHelpers* and return their own *Style* objects then the objects returned should be compatible with the value *engine* provided to the engine attribute.

Examples of relevant style objects can be found respective documentations for the *engine* being used.

6.1 Helper Functions (Data loading routines)

```
1 import tempfile
2 import webbrowser
3 import zipfile
4
5 import pandas as pd
6 import requests
7
8 import table_compositor.table_compositor as tc
9 import table_compositor.xlsx_styles as xlsstyle
10 import table_compositor.xlsx_writer as xlsxw
11
```

```
1 # code snippet adapted from http://function-pipe.readthedocs.io/en/latest/usage_df.html
2 # source url
3 URL_NAMES = "https://www.ssa.gov/oact/babynames/names.zip"
4 ZIP_NAME = "names.zip"
5
6
7 def load_names_data():
8     fp = os.path.join(tempfile.gettempdir(), ZIP_NAME)
9     if not os.path.exists(fp):
10         r = requests.get(URL_NAMES)
11         with open(fp, "wb") as f:
12             f.write(r.content)
13
14     post = collections.OrderedDict()
15     with zipfile.ZipFile(fp) as zf:
```

(continues on next page)

(continued from previous page)

```

16     # get ZipInfo instances
17     for zi in sorted(zf.infolist(), key=lambda zi: zi.filename):
18         fn = zi.filename
19         if fn.startswith("yob"):
20             year = int(fn[3:7])
21             df = pd.read_csv(
22                 zf.open(zi), header=None, names=("name", "gender", "count")
23             )
24             df["year"] = year
25             post[year] = df
26
27     df = pd.concat(post.values())
28     df.set_index("name", inplace=True, drop=True)
29     return df
30
31
32 def sample_names_data():
33     df = load_names_data()
34     df = df[(df["year"] == 2015) & (df["count"] > 1000)]
35     return df.sample(100, random_state=0).sort_values("count")
36
37
38 def top_names_for_year(year=2015, gender="F", top_n=5):
39     df = load_names_data()
40     df = df[(df["year"] == year) & (df["gender"] == gender)]
41     df = df.sort_values("count")[:top_n]
42     return df
43
44

```

6.2 Example 1 - DataFrame with default styles

Demonstrates converting dataframe into html format with default styles.

```

1 class XLSXExample1:
2     """
3     Demonstrates rendering a simple dataframe to a xlsx file
4     using the default styles
5     """
6
7     @classmethod
8     def render_xlsx(cls):
9         """
10         Render the df to a xlsx file.
11         """
12
13         # load data
14         df = sample_names_data()
15         # build presentation model
16         pm = tc.build_presentation_model(df=df, output_format="xlsx")

```

(continues on next page)

(continued from previous page)

```

17     # render to xlsx
18     tmpdir = tempfile.gettempdir()
19     fp = os.path.join(tmpdir, "example1.xlsx")
20     layout = [pm]
21     print("Writing to " + fp)
22     xlszw.OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=fp)
23
24
25

```

	A	B	C	D
1	name	gender	count	year
2	Jared	M	1004	2015
3	Paislee	F	1008	2015
4	Kathryn	F	1009	2015
5	Erik	M	1012	2015
6	Daniella	F	1013	2015
7	Amanda	F	1013	2015
8	Lilah	F	1022	2015
9	Fatima	F	1041	2015
10	Finley	M	1055	2015
11	Ali	M	1059	2015
12	Elliana	F	1061	2015
13	Dante	M	1066	2015
14	Shelby	F	1071	2015
15	Marco	M	1079	2015
16	Diana	F	1081	2015
17	Haley	F	1128	2015
18	Johnny	M	1140	2015

6.3 Example 2 - DataFrame with custom styles

In this example, we format the different components of dataframe with various styling attributes

```

1 class XLSXExample2:
2     """
3     Demonstrates using call-backs that help set the display and style
4     properties of each cell in the xlsx sheet.
5     """
6
7     @staticmethod
8     def data_value_func(df):
9         def _inner(idx, col):
10             if col == "gender":
11                 if df.loc[idx, col] == "F":
12                     return "Female"
13                 return "Male"
14             return df.loc[idx, col]
15

```

(continues on next page)

(continued from previous page)

```

16         return _inner
17
18     @staticmethod
19     def data_style_func(df):
20         def _inner(idx, col):
21             bg_color = None
22             number_format = "General"
23             if col == "count":
24                 number_format = "#,##0"
25             if df.loc[idx, "gender"] == "F":
26                 bg_color = "bbdef8"
27             else:
28                 bg_color = "e3f2fd"
29             return xlsstyle.OpenPyxlStyleHelper.get_style(
30                 bg_color=bg_color, number_format=number_format
31             )
32
33         return _inner
34
35     @staticmethod
36     def index_name_value_func(value):
37         return value.capitalize()
38
39     @staticmethod
40     def index_name_style_func(value):
41         return xlsstyle.OpenPyxlStyleHelper.default_header_style()
42
43     @staticmethod
44     def header_value_func(node):
45         return node.value.capitalize()
46
47     @staticmethod
48     def header_style_func(node):
49         return xlsstyle.OpenPyxlStyleHelper.default_header_style()
50
51     @staticmethod
52     def index_value_func(node):
53         return node.value.capitalize()
54
55     @staticmethod
56     def index_style_func(df):
57         def _inner(node):
58             bg_color = None
59             if df.loc[node.value, "gender"] == "F":
60                 bg_color = "bbdef8"
61             else:
62                 bg_color = "e3f2fd"
63             return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)
64
65         return _inner
66
67     @classmethod

```

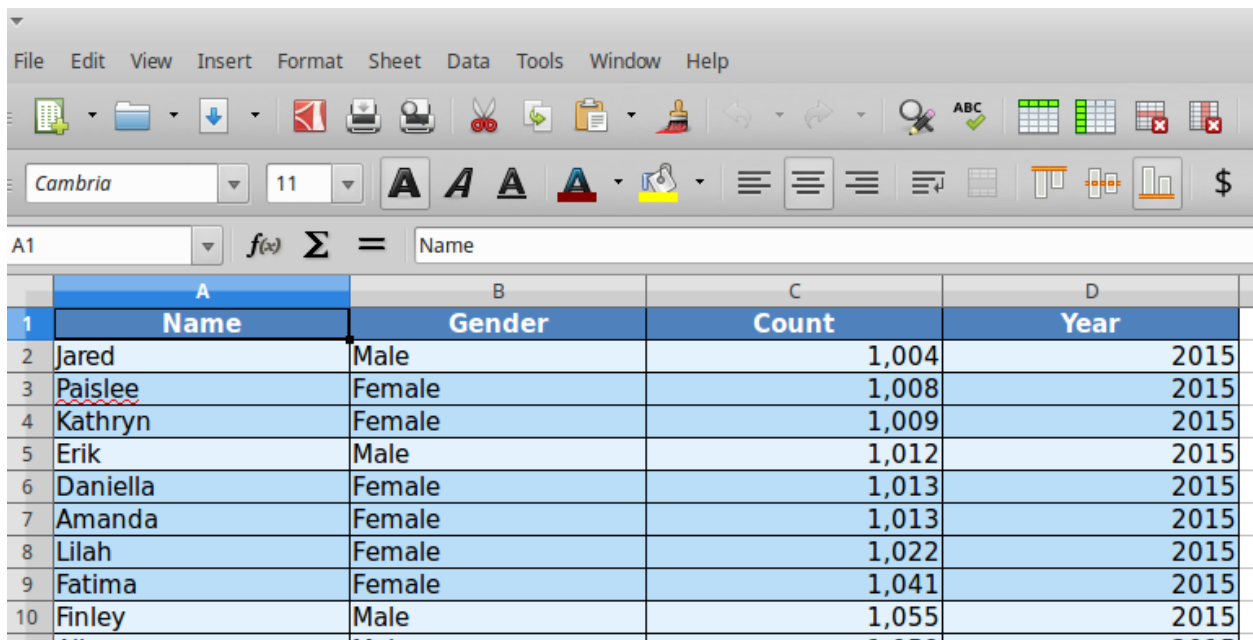
(continues on next page)

(continued from previous page)

```

68 def render_xlsx(cls):
69     # load data
70     df = sample_names_data()
71     # build presentation model
72     klass_ = XLSXExample2
73     pm = tc.build_presentation_model(
74         df=df,
75         output_format="xlsx",
76         data_value_func=klass_.data_value_func(df),
77         data_style_func=klass_.data_style_func(df),
78         header_value_func=klass_.header_value_func,
79         header_style_func=klass_.header_style_func,
80         index_style_func=klass_.index_style_func(df),
81         index_value_func=klass_.index_value_func,
82         index_name_style_func=klass_.index_name_style_func,
83         index_name_func=klass_.index_name_value_func,
84     )
85
86     # render to xlsx
87     tempdir = tempfile.gettempdir()
88     fp = os.path.join(tempdir, "example2.xlsx")
89     layout = [pm]
90     print("Writing to " + fp)
91     xlsxw.OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=fp)
92
93

```



	A	B	C	D
1	Name	Gender	Count	Year
2	Jared	Male	1,004	2015
3	Paislee	Female	1,008	2015
4	Kathryn	Female	1,009	2015
5	Erik	Male	1,012	2015
6	Daniella	Female	1,013	2015
7	Amanda	Female	1,013	2015
8	Lilah	Female	1,022	2015
9	Fatima	Female	1,041	2015
10	Finley	Male	1,055	2015

6.4 Example 3 - Simple DataFrame with Layouts

Demonstrates rendering multi-dataframes in one worksheet along with common functions for styling

```

1 class XLSXExample3:
2     """
3     Demonstrates using call-backs and also rendering multiple tables to single
4     worksheet.
5     """
6
7     @staticmethod
8     def data_value_func(df):
9         def _inner(idx, col):
10             if col == "gender":
11                 if df.loc[idx, col] == "F":
12                     return "Female"
13                 return "Male"
14             return df.loc[idx, col]
15
16         return _inner
17
18     @staticmethod
19     def data_style_func(df):
20         def _inner(idx, col):
21             bg_color = None
22             number_format = "General"
23             if col == "count":
24                 number_format = "#,##0"
25             if df.loc[idx, "gender"] == "F":
26                 bg_color = "bbdef8"
27             else:
28                 bg_color = "e3f2fd"
29             return xlsstyle.OpenPyxlStyleHelper.get_style(
30                 bg_color=bg_color, number_format=number_format
31             )
32
33         return _inner
34
35     @staticmethod
36     def index_name_value_func(value):
37         return value.capitalize()
38
39     @staticmethod
40     def index_name_style_func(value):
41         return xlsstyle.OpenPyxlStyleHelper.default_header_style()
42
43     @staticmethod
44     def header_value_func(node):
45         return node.value.capitalize()
46
47     @staticmethod
48     def header_style_func(node):
49         return xlsstyle.OpenPyxlStyleHelper.default_header_style()

```

(continues on next page)

(continued from previous page)

```

50
51 @staticmethod
52 def index_value_func(node):
53     return node.value.capitalize()
54
55 @staticmethod
56 def index_style_func(df):
57     def _inner(node):
58         bg_color = None
59         if df.loc[node.value, "gender"] == "F":
60             bg_color = "bbdef8"
61         else:
62             bg_color = "e3f2fd"
63         return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)
64
65     return _inner
66
67 @classmethod
68 def render_xlsx(cls):
69     # Prepare first data frame (same as in render_xlsx)
70     df = sample_names_data()
71     # build presentation model
72     klass_ = XLSXExample3
73     pm_all = tc.build_presentation_model(
74         df=df,
75         output_format="xlsx",
76         data_value_func=klass_.data_value_func(df),
77         data_style_func=klass_.data_style_func(df),
78         header_value_func=klass_.header_value_func,
79         header_style_func=klass_.header_style_func,
80         index_style_func=klass_.index_style_func(df),
81         index_value_func=klass_.index_value_func,
82         index_name_style_func=klass_.index_name_style_func,
83         index_name_func=klass_.index_name_value_func,
84     )
85
86     male_df = top_names_for_year(gender="M")
87     pm_top_male = tc.build_presentation_model(
88         df=male_df,
89         output_format="xlsx",
90         data_value_func=klass_.data_value_func(male_df),
91         data_style_func=klass_.data_style_func(male_df),
92         header_value_func=klass_.header_value_func,
93         header_style_func=klass_.header_style_func,
94         index_style_func=klass_.index_style_func(male_df),
95         index_value_func=klass_.index_value_func,
96         index_name_style_func=klass_.index_name_style_func,
97         index_name_func=klass_.index_name_value_func,
98     )
99
100     female_df = top_names_for_year(gender="F")
101     pm_top_female = tc.build_presentation_model(

```

(continues on next page)

(continued from previous page)

```

102     df=female_df,
103     output_format="xlsx",
104     data_value_func=klass_.data_value_func(female_df),
105     data_style_func=klass_.data_style_func(female_df),
106     header_value_func=klass_.header_value_func,
107     header_style_func=klass_.header_style_func,
108     index_style_func=klass_.index_style_func(female_df),
109     index_value_func=klass_.index_value_func,
110     index_name_style_func=klass_.index_name_style_func,
111     index_name_func=klass_.index_name_value_func,
112 )
113
114 layout = [pm_all, [pm_top_female, pm_top_male]]
115 # render to xlsx
116 tempdir = tempfile.gettempdir()
117 fp = os.path.join(tempdir, "example3.xlsx")
118 print("Writing to " + fp)
119 xlsxw.OpenPyxlCompositor.to_xlsx(
120     layout=layout, output_fp=fp, orientation="horizontal"
121 )
122
123

```

	A	B	C	D		F	G	H	I
	Name	Gender	Count	Year		Name	Gender	Count	Year
1	Jared	Male	1,004	2015		Zyrielle	Female	5	2015
2	Paislee	Female	1,008	2015		Falak	Female	5	2015
3	Kathryn	Female	1,009	2015		Fairy	Female	5	2015
4	Erik	Male	1,012	2015		Faige	Female	5	2015
5	Daniella	Female	1,013	2015		Faeryn	Female	5	2015
6	Amanda	Female	1,013	2015					
7	Lilah	Female	1,022	2015					
8	Fatima	Female	1,041	2015					
9	Finley	Male	1,055	2015					
10	Ali	Male	1,059	2015					
11	Eliana	Female	1,061	2015					
12	Dante	Male	1,066	2015					
13	Shelby	Female	1,071	2015					
14	Marco	Male	1,079	2015					

6.5 Example 4 - DataFrames with Multi-hierarchical columns and indices

Demonstrates rendering dataframes with multi-hierarchical indices and multi-hierarchical columns

```

1 class XLSXExample4:
2     """
3     Demonstrate styling and rendering of multi-hierarchical indexed dataframe
4     into a xlsx file.
5     """
6
7     @staticmethod
8     def data_style_func(df):
9         def _inner(idx, col):

```

(continues on next page)

(continued from previous page)

```

10         bg_color = None
11         number_format = "General"
12         if col == "count":
13             number_format = "#,##0"
14         if idx[1] == "F":
15             bg_color = "bbdef8"
16         else:
17             bg_color = "e3f2fd"
18         return xlsstyle.OpenPyxlStyleHelper.get_style(
19             bg_color=bg_color, number_format=number_format
20         )
21
22     return _inner
23
24 @staticmethod
25 def index_name_value_func(value):
26     return "Max By Year"
27
28 @staticmethod
29 def index_name_style_func(value):
30     return xlsstyle.OpenPyxlStyleHelper.default_header_style()
31
32 @staticmethod
33 def header_value_func(node):
34     return node.value.capitalize()
35
36 @staticmethod
37 def header_style_func(node):
38     return xlsstyle.OpenPyxlStyleHelper.default_header_style()
39
40 @staticmethod
41 def index_value_func(node):
42     if isinstance(node.value, str):
43         return node.value.capitalize()
44     return node.value
45
46 @staticmethod
47 def index_style_func(df):
48     def _inner(node):
49         bg_color = None
50         if len(node.key) == 1:
51             bg_color = "4f81bd"
52         elif node.key[1] == "F":
53             bg_color = "bbdef8"
54         else:
55             bg_color = "e3f2fd"
56         return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)
57
58     return _inner
59
60 @classmethod
61 def render_xlsx(cls):

```

(continues on next page)

(continued from previous page)

```
62     # Prepare first data frame (same as in render_xlsx)
63     data_df = load_names_data()
64     data_df = data_df[data_df["year"] >= 2000]
65     g = data_df.groupby(("year", "gender"))
66     df = g.max()
67
68
69     klass_ = cls
70     pm = tc.build_presentation_model(
71         df=df,
72         output_format="xlsx",
73         # data_value_func=None, # use default
74         data_style_func=klass_.data_style_func(df),
75         header_value_func=klass_.header_value_func,
76         header_style_func=klass_.header_style_func,
77         index_style_func=klass_.index_style_func(df),
78         index_value_func=klass_.index_value_func,
79         index_name_style_func=klass_.index_name_style_func,
80         index_name_func=klass_.index_name_value_func,
81     )
82
83     layout = [pm]
84     # render to xlsx
85     tempdir = tempfile.gettempdir()
86     fp = os.path.join(tempdir, "example4.xlsx")
87     print("Writing to " + fp)
88     xlszw.OpenPyxlCompositor.to_xlsx(
89         layout=layout, output_fp=fp, orientation="horizontal"
90     )
91
92
```

The screenshot shows a spreadsheet application interface. The menu bar includes File, Edit, View, Insert, Format, Sheet, Data, Tools, Window, and Help. The toolbar contains various icons for file operations, editing, and formatting. The font settings are set to Cambria, size 11. The active cell is A1, and the formula bar shows the text "Max By Year".

	A	B	C	D	E
1	Max By Year		Count		
2	2000	F	25,953		
3		M	34,467		
4	2001	F	25,052		
5		M	32,531		
6	2002	F	24,459		
7		M	30,558		
8	2003	F	25,685		
9		M	29,620		
10	2004	F	25,028		
11		M	27,873		
12	2005	F	23,930		
13		M	25,822		
14	2006	F	21,393		
15		M	24,832		

HTML EXAMPLES

This page provides a list of examples that demonstrate rendering html tables from the given dataframe. Each example is self-contained inside a class. We have some helper functions to provide the data we need for this examples

7.1 Helper Functions (Data loading routines)

```
1 import tempfile
2 import webbrowser
3 import zipfile
4
5 import pandas as pd
6 import requests
7
8 import table_compositor.html_styles as html_style
9 import table_compositor.html_writer as htmlw
10 import table_compositor.table_compositor as tc
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
26
```

(continued from previous page)

```

22         zf.open(zi), header=None, names=("name", "gender", "count")
23     )
24     df["year"] = year
25     post[year] = df
26
27     df = pd.concat(post.values())
28     df.set_index("name", inplace=True, drop=True)
29     return df
30
31
32 def sample_names_data():
33     df = load_names_data()
34     df = df[(df["year"] == 2015) & (df["count"] > 1000)]
35     return df.sample(50, random_state=0).sort_values("count")
36
37
38 def top_names_for_year(year=2015, gender="F", top_n=5):
39     df = load_names_data()
40     df = df[(df["year"] == year) & (df["gender"] == gender)]
41     df = df.sort_values("count")[:top_n]
42     return df
43
44

```

7.2 Example 1 - DataFrame with default styles

Demonstrates converting dataframe into html format with default styles.

```

1 class HTMLExample1:
2     """
3     Demonstrate rendering of a simple dataframe into html
4     """
5
6     @classmethod
7     def render_html(cls):
8
9         # load data
10        df = load_names_data()
11        df = df[:100]
12
13        # build presentation model
14        pm = tc.build_presentation_model(df=df, output_format="html")
15
16        # render to xlsx
17        tempdir = tempfile.gettempdir()
18        fp = os.path.join(tempdir, "example_1.html")
19        layout = [pm]
20        print("Writing to " + fp)
21        html = htmlw.HTMLWriter.to_html(layout, border=1)
22        output_fp = os.path.join(tempfile.gettempdir(), "example1.html")

```

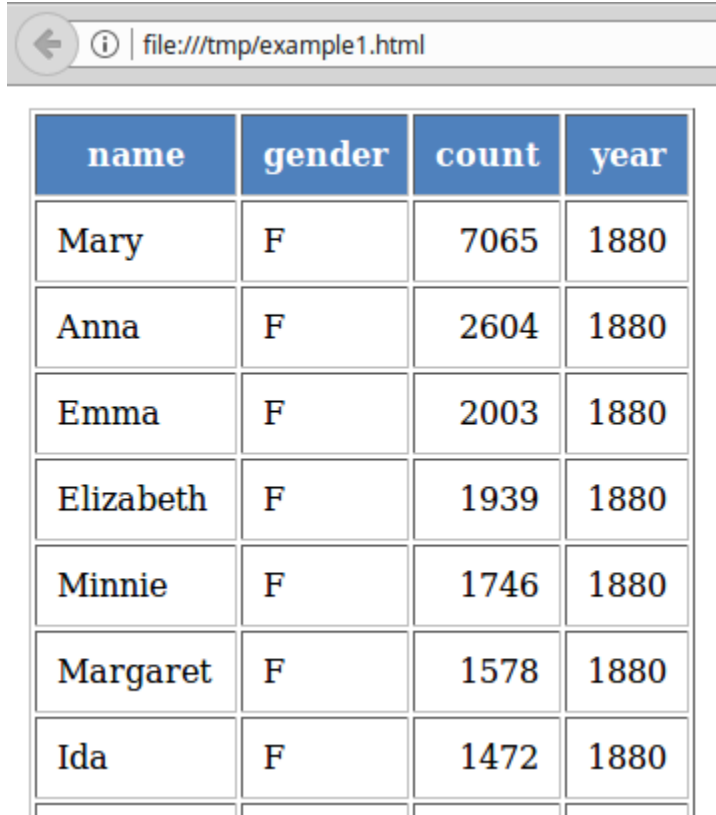
(continues on next page)

(continued from previous page)

```

23     with open(output_fp, "w") as f:
24         f.write(html)
25
26

```



The screenshot shows a web browser window with the address bar displaying 'file:///tmp/example1.html'. The browser content is a table with four columns: 'name', 'gender', 'count', and 'year'. The table contains seven rows of data, all with a year of 1880. The names listed are Mary, Anna, Emma, Elizabeth, Minnie, Margaret, and Ida. The counts for each name are 7065, 2604, 2003, 1939, 1746, 1578, and 1472, respectively.

name	gender	count	year
Mary	F	7065	1880
Anna	F	2604	1880
Emma	F	2003	1880
Elizabeth	F	1939	1880
Minnie	F	1746	1880
Margaret	F	1578	1880
Ida	F	1472	1880

7.3 Example 2 - DataFrame with custom styles

In this example, we format the different components of dataframe with various styling attributes

```

1  class HTMLExample2:
2      """
3      Demonstrate rendering of a simple dataframe into html
4      """
5
6      @staticmethod
7      def data_value_func(df):
8          def _inner(idx, col):
9              if col == "gender":
10                 if df.loc[idx, col] == "F":
11                     return "Female"
12                 return "Male"
13             return df.loc[idx, col]
14
15         return _inner

```

(continues on next page)

(continued from previous page)

```

16
17 @staticmethod
18 def data_style_func(df):
19     def _inner(idx, col):
20         color = "#FFFFFF"
21         text_align = "left"
22         if col == "count":
23             text_align = "right"
24         if df.loc[idx, "gender"] == "F":
25             color = "#bbdef8"
26         else:
27             color = "#e3f2fd"
28         return html_style.td_style(
29             text_align=text_align,
30             background_color=color,
31             color="#000000",
32             font_weight="normal",
33             white_space="pre",
34             padding="10px",
35             border=None,
36         )
37
38     return _inner
39
40 @staticmethod
41 def index_name_value_func(value):
42     return value.capitalize()
43
44 @staticmethod
45 def header_value_func(node):
46     return node.value.capitalize()
47
48 @staticmethod
49 def header_style_func(node):
50     return html_style.td_style(
51         text_align="center",
52         background_color="#4F81BD",
53         color="#FFFFFF",
54         font_weight="bold",
55         white_space="pre",
56         padding="10px",
57         border=1,
58     )
59
60 @staticmethod
61 def index_value_func(node):
62     return node.value.capitalize()
63
64 @staticmethod
65 def index_style_func(node):
66     return html_style.td_style(
67         text_align="center",

```

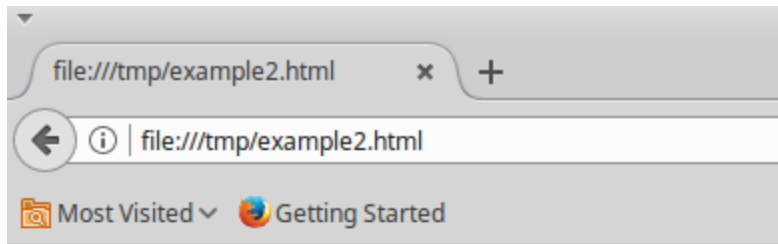
(continues on next page)

(continued from previous page)

```

68         background_color="#4F81BD",
69         color="#FFFFFF",
70         font_weight="bold",
71         white_space="pre",
72         padding="10px",
73         border=1,
74     )
75
76     @classmethod
77     def render_html(cls):
78         # load data
79         df = sample_names_data()
80         # build presentation model
81         klass_ = HTMLExample2
82         pm = tc.build_presentation_model(
83             df=df,
84             output_format="html",
85             data_value_func=klass_.data_value_func(df),
86             data_style_func=klass_.data_style_func(df),
87             header_value_func=klass_.header_value_func,
88             header_style_func=klass_.header_style_func,
89             index_style_func=klass_.index_style_func,
90             index_value_func=klass_.index_value_func,
91             index_name_func=klass_.index_name_value_func,
92         )
93
94         layout = [pm]
95         html = htmlw.HTMLWriter.to_html(layout, border=1)
96         output_fp = os.path.join(tempfile.gettempdir(), "example2.html")
97         print("Writing to =", output_fp)
98         with open(output_fp, "w") as f:
99             f.write(html)
100
101

```



Name	Gender	Count	Year
Paislee	Female	1008	2015
Kathryn	Female	1009	2015
Erik	Male	1012	2015
Lilah	Female	1022	2015
Fatima	Female	1041	2015
Dante	Male	1066	2015
Shelby	Female	1071	2015
Marco	Male	1079	2015
Diana	Female	1081	2015
Haley	Female	1128	2015

7.4 Example 3 - Simple DataFrame with Layouts

Demonstrates rendering dataframes with multi-hierarchical indices and multi-hierarchical columns

```

1 class HTMLExample3:
2     """
3     Demonstrate styling and rendering of multiple multi-hierarchical indexed dataframe
4     into a html file
5     """
6
7     @staticmethod
8     def data_value_func(df):

```

(continues on next page)

(continued from previous page)

```

9     def _inner(idx, col):
10         if col == "gender":
11             if df.loc[idx, col] == "F":
12                 return "Female"
13             return "Male"
14         return df.loc[idx, col]
15
16     return _inner
17
18 @staticmethod
19 def data_style_func(df):
20     def _inner(idx, col):
21         color = "#FFFFFF"
22         text_align = "left"
23         if col == "count":
24             text_align = "right"
25         if df.loc[idx, "gender"] == "F":
26             color = "#bbdef8"
27         else:
28             color = "#e3f2fd"
29         return html_style.td_style(
30             text_align=text_align,
31             background_color=color,
32             color="#000000",
33             font_weight="normal",
34             white_space="pre",
35             padding="10px",
36             border=None,
37         )
38
39     return _inner
40
41 @staticmethod
42 def index_name_value_func(value):
43     return "Max By Year"
44
45 @staticmethod
46 def header_value_func(node):
47     return node.value.capitalize()
48
49 @staticmethod
50 def header_style_func(node):
51     return html_style.td_style(
52         text_align="center",
53         background_color="#4F81BD",
54         color="#FFFFFF",
55         font_weight="bold",
56         white_space="pre",
57         padding="10px",
58         border=1,
59     )
60

```

(continues on next page)

(continued from previous page)

```

61 @staticmethod
62 def index_value_func(node):
63     if isinstance(node.value, str):
64         return node.value.capitalize()
65     return node.value
66
67 @staticmethod
68 def index_style_func(node):
69     return html_style.td_style(
70         text_align="center",
71         background_color="#4F81BD",
72         color="#FFFFFF",
73         font_weight="bold",
74         white_space="pre",
75         padding="10px",
76         border=1,
77     )
78
79 @classmethod
80 def render_html(cls):
81
82     # Prepare first data frame (same as in render_xlsx)
83     df = sample_names_data()
84     # build presentation model
85     klass_ = HTMLExample4
86     pm_all = tc.build_presentation_model(
87         df=df,
88         output_format="html",
89         data_value_func=klass_.data_value_func(df),
90         data_style_func=klass_.data_style_func(df),
91         header_value_func=klass_.header_value_func,
92         header_style_func=klass_.header_style_func,
93         index_style_func=klass_.index_style_func,
94         index_value_func=klass_.index_value_func,
95         index_name_func=lambda _: "Sample Data",
96     )
97
98     male_df = top_names_for_year(gender="M")
99     pm_top_male = tc.build_presentation_model(
100         df=male_df,
101         output_format="html",
102         data_value_func=klass_.data_value_func(male_df),
103         data_style_func=klass_.data_style_func(male_df),
104         header_value_func=klass_.header_value_func,
105         header_style_func=klass_.header_style_func,
106         index_style_func=klass_.index_style_func,
107         index_value_func=klass_.index_value_func,
108         index_name_func=lambda _: "Max by Year",
109     )
110
111     female_df = top_names_for_year(gender="F")
112     pm_top_female = tc.build_presentation_model(

```

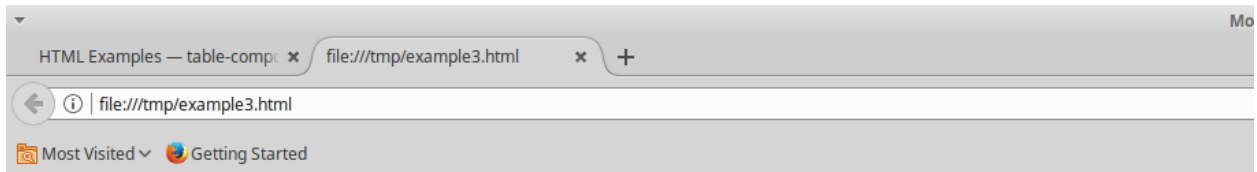
(continues on next page)

(continued from previous page)

```

113     df=female_df,
114     output_format="html",
115     data_value_func=klass_.data_value_func(female_df),
116     data_style_func=klass_.data_style_func(female_df),
117     header_value_func=klass_.header_value_func,
118     header_style_func=klass_.header_style_func,
119     index_style_func=klass_.index_style_func,
120     index_value_func=klass_.index_value_func,
121     index_name_func=lambda _: "Max by Year",
122 )
123
124 layout = [pm_all, [pm_top_female, pm_top_male]]
125 # render to xlsx
126 html = htmlw.HTMLWriter.to_html(layout, border=1, orientation="horizontal")
127 output_fp = os.path.join(tempfile.gettempdir(), "example3.html")
128 print("Writing to =", output_fp)
129 with open(output_fp, "w") as f:
130     f.write(html)
131
132

```



Sample Data	Gender	Count	Year	Max by Year	Gender	Count	Year
Paislee	Female	1008	2015	Zyrielle	Female	5	2015
Kathryn	Female	1009	2015	Falak	Female	5	2015
Erik	Male	1012	2015	Fairy	Female	5	2015
Lilah	Female	1022	2015	Faige	Female	5	2015
Fatima	Female	1041	2015	Faeryn	Female	5	2015
Dante	Male	1066	2015				
Shelby	Female	1071	2015	Max by Year	Gender	Count	Year
Marco	Male	1079	2015	Zyus	Male	5	2015
Diana	Female	1081	2015	Greyton	Male	5	2015
Haley	Female	1128	2015	Greyton	Male	5	2015
Johnny	Male	1140	2015	Grigoriy	Male	5	2015
Kali	Female	1179	2015	Gurtej	Male	5	2015

7.5 Example 4 - DataFrames with Multi-hierarchical columns and indices

Demonstrates rendering dataframes with multi-hierarchical indices and multi-hierarchical columns

```

1 class HTMLExample4:
2     """
3     Demonstrate styling and rendering of multi-hierarchical indexed dataframe
4     into a html file.
5     """
6
7     @staticmethod
8     def data_value_func(df):
9         def _inner(idx, col):
10             if col == "gender":
11                 if df.loc[idx, col] == "F":
12                     return "Female"
13                 return "Male"
14             return df.loc[idx, col]
15
16         return _inner
17
18     @staticmethod
19     def data_style_func(df):
20         def _inner(idx, col):
21             color = "#FFFFFF"
22             text_align = "left"
23             if col == "count":
24                 text_align = "right"
25             if idx[1] == "F":
26                 color = "#bbdef8"
27             else:
28                 color = "#e3f2fd"
29
30             return html_style.td_style(
31                 text_align=text_align,
32                 background_color=color,
33                 color="#000000",
34                 font_weight="normal",
35                 white_space="pre",
36                 padding="10px",
37                 border=None,
38             )
39
40         return _inner
41
42     @staticmethod
43     def index_name_value_func(value):
44         return "Max By Year"
45
46     @staticmethod
47     def header_value_func(node):

```

(continues on next page)

(continued from previous page)

```

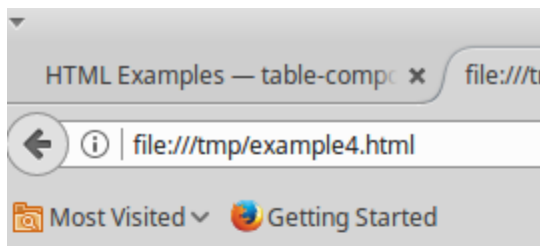
48     return node.value.capitalize()
49
50 @staticmethod
51 def header_style_func(node):
52     return html_style.td_style(
53         text_align="center",
54         background_color="#4F81BD",
55         color="#FFFFFF",
56         font_weight="bold",
57         white_space="pre",
58         padding="10px",
59         border=1,
60     )
61
62 @staticmethod
63 def index_value_func(node):
64     if isinstance(node.value, str):
65         return node.value.capitalize()
66     return node.value
67
68 @staticmethod
69 def index_style_func(node):
70     return html_style.td_style(
71         text_align="center",
72         background_color="#4F81BD",
73         color="#FFFFFF",
74         font_weight="bold",
75         white_space="pre",
76         padding="10px",
77         border=1,
78     )
79
80 @classmethod
81 def render_html(cls):
82
83     # Prepare first data frame (same as in render_xlsx)
84     data_df = load_names_data()
85     data_df = data_df[data_df["year"] >= 2000]
86     g = data_df.groupby(("year", "gender"))
87     df = g.max()
88
89     klass_ = cls
90     pm = tc.build_presentation_model(
91         df=df,
92         output_format="html",
93         data_value_func=klass_.data_value_func(df),
94         data_style_func=klass_.data_style_func(df),
95         header_value_func=klass_.header_value_func,
96         header_style_func=klass_.header_style_func,
97         index_style_func=klass_.index_style_func,
98         index_value_func=klass_.index_value_func,
99         index_name_func=klass_.index_name_value_func,

```

(continues on next page)

(continued from previous page)

```
100     )
101
102     layout = [pm]
103     # render to xlsx
104     html = htmlw.HTMLWriter.to_html(layout, border=1)
105     output_fp = os.path.join(tempfile.gettempdir(), "example4.html")
106     print("Writing to =", output_fp)
107     with open(output_fp, "w") as f:
108         f.write(html)
109
110
```



Max By Year		Count
2000	F	25953
	M	34467
2001	F	25052
	M	32531
2002	F	24459
	M	30558
2003	F	25685
	M	29620
2004	F	25028
	M	27873

8.1 Building the presentation model

```
table_compositor.table_compositor.build_presentation_model(*, df, output_format='xlsx',  
                                                           data_value_func=None,  
                                                           column_style_func=None,  
                                                           data_style_func=None,  
                                                           header_style_func=None,  
                                                           header_value_func=None,  
                                                           index_style_func=None,  
                                                           index_value_func=None,  
                                                           index_name_func=None,  
                                                           index_name_style_func=None,  
                                                           engine='openpyxl', **kwargs)
```

Construct and return the presentation model that will be used while rendering to html/xlsx formats. The returned object has all the information required to render the tables in the requested format. The details of the object is transparent to the caller. It is only exposed for certain advanced operations.

Parameters

- **df** – The dataframe representation of the table. The shape of the dataframe closely resembles the table that will be rendered in the requested format.
- **output_format** – ‘html’ or ‘xlsx’
- **data_value_func** – example: `lambda idx, col: df.loc[idx, col]`, assuming `df` is in the closure. This can be `None`, if no data transformation is required to the values already present in the source `df`
- **column_style_func** – the function can substitute the `data_style_func`, if the same style can be applied for the whole column. This argument should be preferred over the `data_style_func` argument. Using this option provides better performance since the fewer objects will be created internally and fewer callbacks are made to this function when compared to `data_style_func`. This argument only applies to the data contained in the dataframe and not the cell where the headers are rendered. For fine grained control at *cell* level, the `data_style_func` argument can be used. For more information on return values of this function, refer to the documentation for `data_style_func` argument.
- **data_style_func** – used to provide style at the cell level. Example: `lambda idx, col: return dict(font=Font(...))`, where `Font` is the `openpyxl` object and `font` is the attr available in the *cell* instance of `openpyxl`. For `xlsx`, the keys in the dict are the attrs of the *cell* object in `openpyxl` and the values correspond to the value of that attribute. Example are found in `xlsx_styles` module. For `html`, the key-value pairs are any values that go into to the style attribute of a

td, th cell in html. Examples are found in `html_styles` module. example: `dict(background-color='#F8F8F8')`. When performance becomes an issue, and cell level control is not needed, it is recommended to use the `column_style_func` argument rather than this argument. If the preferred engine is `XlswWriter`, then the style dictionary returned should have key/values compatible with the `Format` object declared in the `XlsxWriter` library. A reference can be found in `xlsx_styles.XlsxWriterStyleHelper` class`

- **header_value_func** – func that takes a object of type *IndexNode*. The *IndexNode* contains the attributes that refer to the header being rendered. The returned value from this function is displayed in place of the header in the dataframe at the location. The two properties available on the *IndexNode* object are *value* and *key*. The *key* is useful to identify the exact index and level in context while working with multi-hierarchical columns.
- **header_style_func** – func that takes a object of type *IndexNode*. The return value of this function is similar to `data_style_func`.
- **index_value_func** – func that takes a object of type *IndexNode*. The *IndexNode* contains the attributes that refer to the index being rendered. The returned value from this function is displayed in place of the index in the dataframe at the location.
- **index_style_func** – func that takes a object of type *IndexNode*. The return value of this function is similar to `data_style_func`.
- **index_name_func** – func that returns a string for index name (value to be displayed on top-left corner, above the index column)
- **index_name_style** – the style value same as `data_style_func` that will be used to style the cell
- **engine** – required while building presentation model for `xlsx`. Argument ignored for HTML rendering. This argument is used to provide the default callback style functions, where the style dictionary returned by the callback functions should be compatible with the engine being used.
- **kwargs** – ‘hide_index’ - if True, then hide the index column, default=False
‘hide_header’, - if True, then hide the header, default=False
‘use_convert’ - if True, do some conversions from dataframe values to values excel can understand for example `np.NaN` are converted to `NaN` strings

Returns

A presentation model, to be used to create layout and provide the layout to the `html` or `xlsx` writers.

About the callback functions provided as arguments:

Note that callback function provided as arguments to this function are provided with either a tuple of index, col arguments are some information regarding the index or headers being rendered. Therefore, a common pattern would be to capture the *dataframe* being rendered in a closure of this callback func before passing them as arguments.

For example:

```
df = pd.DataFrame(dict(a=[1, 2, 3]))
```

```
def data_value_func():
```

```
    def _inner(idx, col):
```

```
        return df.loc[idx, col] * 10.3
```

```
    return _inner
```

```
pm = build_presentation_model(df=df, data_value_func=data_value_func())
```

8.2 Rendering to XLSX

8.3 Rendering to HTML

```
class table_compositor.html_writer.HTMLWriter
```

```
static to_html(layout, orientation='vertical', **kwargs)
```

Take a layout which contains a list of presentation models builds using the `build_presentation_model` function.

Parameters

- **layout** – An nested list of `presentation_models`, examples: `[presentation_model]` or `[presentation_model1, presentation_model2]`. Not all nested layouts work very well in HTML, currently
- **orientation** – if vertical, the top level presentation model elements are rendered vertically, and for every nested level the orientation is flipped. if horizontal, then the behavior is inverse
- **kwargs** – all key-value pairs available in `kwargs` are directly set as value of the style attribute of `table` tag. example `dict(background-color='#FF88FF')`, is used as `<table style='background-color:#FF88FF'>..</table>`

Returns

Return a HTML formatted string. The outermost tag of the returned string is the `<table>`

8.4 Helper XLSX Styles

```
class table_compositor.xlsx_styles.OpenPyxlStyleHelper
```

```
static default_header_style(*, alignment='center', font=<openpyxl.styles.fonts.Font object>
                             Parameters: name=None, charset=None, family=None, b=True, i=False,
                             strike=None, outline=None, shadow=None, condense=None,
                             color=<openpyxl.styles.colors.Color object> Parameters:
                             rgb='00FFFFFF', indexed=None, auto=None, theme=None, tint=0.0,
                             type='rgb', extend=None, sz=None, u=None, vertAlign=None,
                             scheme=None, bgColor='4F81BD', border=<object object>)
```

Provides styles for default headers for OpenPyxl engine

Parameters

- **alignment** – 'center', 'left', 'right' used for horizontal alignment
- **font** – an `openpyxl.Font` instance
- **bgColor** – hex color that will be used as background color in the fill pattern
- **border** – an `openpyxl.Border` instance, defaults to thin white border

Returns

A dict of key-values pairs, where each key is a attr of the `cell` object in `openpyxl` and value is valid value of that attr.

static `get_style(number_format='General', bg_color=None, border=None, font=None)`

Helper method to return a openpyxl Style

Parameters

- **number_format** – an xlsx compatible number format string
- **bg_color** – hex color that will be used as background color in the fill pattern
- **border** – an openpyxl.Border instance, defaults to thin white border

Returns

A dict of key-values pairs, where each key is a attr of the *cell* object in openpyxl and value is valid value of that attr.

class `table_compositor.xlsx_styles.XlsxWriterStyleHelper`

Class provides style objects for XlsxWriter library uses to render xlsx files

static `default_header_style(*, number_format='General', alignment='center', font=None, bg_color='#4F81BD', border=<object object>)`

Provides styles for default headers for XlsxWriter engine

Parameters

- **alignment** – 'center', 'left', 'right' used for horizontal alignment
- **font** – an openpyxl.Font instance
- **bg_color** – hex color that will be used as background color in the fill pattern
- **border** – an openpyxl.Border instance, defaults to thin white border

Returns

A dict of key-values pairs, where each key is a attr of the *cell* object in openpyxl and value is valid value of that attr.

static `get_style(number_format='General', bg_color=None, border=<object object>)`

Helper method to return Style dictionary for XlsxWriter engine

Parameters

- **number_format** – an xlsx compatible number format string
- **bg_color** – hex color that will be used as background color in the fill pattern
- **border** – an openpyxl.Border instance, defaults to thin white border

Returns

A dict of key-values pairs, where each key/value is compatible with the *Format* object in XlsxWriter library.

class `table_compositor.xlsx_styles.XLSXWriterDefaults`

Class provides defaults callback funcs that can be used while calling the `build_presentation_model`.

static `data_style_func(df)`

Default value that can be used as callback for `data_style_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes `idx`, `col` as arguments and returns a openpyxl compatible style dictionary

static data_value_func(df)

Default value that can be used as callback for data_value_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

A function that takes *idx*, *col* as arguments and returns the `df.loc[idx, col]` value

static header_style_func(df)

Default value that can be used as callback for data_style_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as arguments and returns a openpyxl compatible style dictionary

static header_value_func(df)

Default value that can be used as callback for data_header_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as arguments and returns `node.value`

static index_name_style_func(df)

Default value that can be used as callback for index_name_style_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes `index.name` as arguments and returns a openpyxl compatible style dictionary

static index_name_value_func(df)

Default value that can be used as callback for index_name_value_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes `index.name` as arguments and returns `index.name` if not None, else ""

static index_style_func(df)

Default value that can be used as callback for index_style_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as arguments and returns a openpyxl compatible style dictionary

static index_value_func(df)

Default value that can be used as callback for index_header_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as arguments and returns `node.value`

8.5 Helper HTML Styles

class `table_compositor.html_styles.HTMLWriterDefaults`

Class provides defaults callback funcs that can be used while calling the `build_presentation_model`.

static `data_style_func(df)`

Default value that can be used as callback for `data_style_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes `idx`, `col` as arguments and returns a dictionary of html style attributes

static `data_value_func(df, dollar_columns=None)`

Default value that can be used as callback for `data_value_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

A function that takes `idx`, `col` as arguments and returns the `df.loc[idx, col]` value

static `header_style_func(df)`

Default value that can be used as callback for `header_style_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as argument and returns a dictionary of html style attributes

static `header_value_func(df)`

Default value that can be used as callback for `header_value_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

A function that takes *node* as arguments and returns the `node.value`

static `index_name_style_func(df)`

Default value that can be used as callback for `index_name_style_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes `index.name` as argument and returns a dictionary of html style attributes

static `index_name_value_func(df)`

Default value that can be used as callback for `index_name_value_func`

Parameters

df – the dataframe that will be used to build the presentation model

Returns

A function that takes `index.name` as argument and return `index.name` if not None else ''

static index_style_func(df)

Default value that can be used as callback for index_style_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

a function table takes *node* as argument and returns a dictionary of html style attributes

static index_value_func(df)

Default value that can be used as callback for index_value_func

Parameters

df – the dataframe that will be used to build the presentation model

Returns

A function that takes node as arguments and returns the node.value

CODE USED IN DOCUMENTATION

9.1 Basic Usage

```
# start_imports
import os
import tempfile

import pandas as pd

from table_compositor.table_compositor import build_presentation_model
from table_compositor.xlsx_styles import OpenPyxlStyleHelper

# There are equivalent classes for using xlsxwriter library. Namely,
# XlsxWriterCompositor and XlsxWriterStyleHelper
from table_compositor.xlsx_writer import OpenPyxlCompositor

# end_imports

# start_basic_example_2
def basic_example2():

    df = pd.DataFrame(
        dict(a=[10, 20, 30, 40, 50], b=[0.1, 0.9, 0.2, 0.6, 0.3]), index=[1, 2, 3, 4, 5]
    )

    def style_func(idx, col):
        if col == "b":
            return OpenPyxlStyleHelper.get_style(number_format="0.00%")
        else:
            # for 'a' we do dollar format
            return OpenPyxlStyleHelper.get_style(number_format="$#,##.00")

    # create a presentation model
    # note the OpenPyxlStyleHelper function available in xlsx_styles module. But a_
    ↪return value of style function
    # can be any dict whose keys are attributes of the OpenPyxl cell object.
    presentation_model = build_presentation_model(
        df=df,
        data_value_func=lambda idx, col: df.loc[idx, col] * 10
        if col == "a"
```

(continues on next page)

(continued from previous page)

```

    else df.loc[idx, col],
    data_style_func=style_func,
    header_value_func=lambda node: node.value.capitalize(),
    header_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
    index_value_func=lambda node: node.value * 100,
    index_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
    index_name_func=lambda _: "Basic Example",
    index_name_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
)

# create a layout, which is usually a nested list of presentation models
layout = [presentation_model]

# render to xlsx
output_fp = os.path.join(tempfile.gettempdir(), "basic_example2.xlsx")
OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=output_fp)

# end_basic_example_2

# start_basic_example_3
def basic_example3():

    df = pd.DataFrame(
        dict(
            a=[10, 20, 30, 40],
            b=[0.1, 0.9, 0.2, 0.6],
            d=[50, 60, 70, 80],
            e=[200, 300, 400, 500],
        )
    )
    df.columns = pd.MultiIndex.from_tuples(
        [("A", "x"), ("A", "y"), ("B", "x"), ("B", "y")]
    )
    df.index = pd.MultiIndex.from_tuples([(1, 100), (1, 200), (2, 100), (2, 200)])
    print(df)

    def index_style_func(node):
        # node.key here could be one of (1,), (1, 100), (2,), (2, 100), (2, 200)
        bg_color = "FFFFFF"
        if node.key == (1,) or node.key == (2,):
            bg_color = "9E80B8"
        elif node.key[1] == 100:
            bg_color = "4F90C1"
        elif node.key[1] == 200:
            bg_color = "6DC066"
        return OpenPyxlStyleHelper.get_style(bg_color=bg_color)

    def header_style_func(node):
        bg_color = "FFFFFF"
        if node.key == ("A",) or node.key == ("B",):
            bg_color = "9E80B8"

```

(continues on next page)

(continued from previous page)

```

        elif node.key[1] == "x":
            bg_color = "4F90C1"
        elif node.key[1] == "y":
            bg_color = "6DC066"
        return OpenPyxlStyleHelper.get_style(bg_color=bg_color)

    # create a presentation model
    # note the OpenPyxlStyleHeloer function available in xlsx_styles module. But a
    ↪return value of style function
    # can be any dict whose keys are attributes of the OpenPyxl cell object.
    presentation_model = build_presentation_model(
        df=df,
        index_style_func=index_style_func,
        header_style_func=header_style_func,
        index_name_func=lambda _: "Multi-Hierarchy Example",
    )

    # create a layout, which is usually a nested list of presentation models
    layout = [presentation_model]

    # render to xlsx
    output_fp = os.path.join(tempfile.gettempdir(), "basic_example3.xlsx")
    OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=output_fp)

# end_basic_example_3

# start_layout_example_1
def layout_example1():

    df = pd.DataFrame(
        dict(a=[10, 20, 30, 40, 50], b=[0.1, 0.9, 0.2, 0.6, 0.3]), index=[1, 2, 3, 4, 5]
    )

    def style_func(idx, col):
        if col == "b":
            return OpenPyxlStyleHelper.get_style(number_format="0.00%")
        else:
            # for 'a' we do dollar format
            return OpenPyxlStyleHelper.get_style(number_format="$#,##.00")

    # create a presentation model
    # note the OpenPyxlStyleHeloer function available in xlsx_styles module. But a
    ↪return value of style function
    # can be any dict whose keys are attributes of the OpenPyxl cell object.
    presentation_model = build_presentation_model(
        df=df,
        data_value_func=lambda idx, col: df.loc[idx, col] * 10
        if col == "a"
        else df.loc[idx, col],
        data_style_func=style_func,

```

(continues on next page)

(continued from previous page)

```

        header_value_func=lambda node: node.value.capitalize(),
        header_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
        index_value_func=lambda node: node.value * 100,
        index_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
        index_name_func=lambda _: "Basic Example",
        index_name_style_func=lambda _: OpenPyxlStyleHelper.default_header_style(),
    )

    # start_layout_code_1
    # create a layout, which is usually a nested list of presentation models
    layout = [[presentation_model], [[presentation_model], [presentation_model]]]

    # render to xlsx
    output_fp = os.path.join(tempfile.gettempdir(), "layout_vertical_example1.xlsx")
    # the default value for orientation is 'vertical'
    OpenPyxlCompositor.to_xlsx(
        layout=layout, output_fp=output_fp, orientation="vertical"
    )

    output_fp = os.path.join(tempfile.gettempdir(), "layout_horizontal_example1.xlsx")
    OpenPyxlCompositor.to_xlsx(
        layout=layout, output_fp=output_fp, orientation="horizontal"
    )
    print("Writing xlsx file=", output_fp)

    # mutiple nesting
    layout_complex = [
        presentation_model,
        [presentation_model, [presentation_model, presentation_model]],
    ]

    output_fp = os.path.join(tempfile.gettempdir(), "layout_complex_example1.xlsx")
    OpenPyxlCompositor.to_xlsx(
        layout=layout_complex, output_fp=output_fp, orientation="vertical"
    )
    print("Writing xlsx file=", output_fp)
    # end_layout_code_1

# end_layout_example_1

if __name__ == "__main__":
    basic_example2()
    basic_example3()
    layout_example1()

```

9.2 XLSX Examples

```

"""
This module is referred to by the Sphinx documentation. If you need to run this
module, install table_compositor in an separate environment and then run this module
in that environment. This helps the imports find the modules in the right place
"""

import collections
import os

# start_imports
import tempfile
import webbrowser
import zipfile

import pandas as pd
import requests

import table_compositor.table_compositor as tc
import table_compositor.xlsx_styles as xlsstyle
import table_compositor.xlsx_writer as xlsxw

# end_imports

# start_data_routine
# code snippet adapted from http://function-pipe.readthedocs.io/en/latest/usage_df.html
# source url
URL_NAMES = "https://www.ssa.gov/oact/babynames/names.zip"
ZIP_NAME = "names.zip"

def load_names_data():
    fp = os.path.join(tempfile.gettempdir(), ZIP_NAME)
    if not os.path.exists(fp):
        r = requests.get(URL_NAMES)
        with open(fp, "wb") as f:
            f.write(r.content)

    post = collections.OrderedDict()
    with zipfile.ZipFile(fp) as zf:
        # get ZipInfo instances
        for zi in sorted(zf.infolist(), key=lambda zi: zi.filename):
            fn = zi.filename
            if fn.startswith("yob"):
                year = int(fn[3:7])
                df = pd.read_csv(
                    zf.open(zi), header=None, names=("name", "gender", "count")
                )
                df["year"] = year
                post[year] = df

```

(continues on next page)

(continued from previous page)

```

        df = pd.concat(post.values())
        df.set_index("name", inplace=True, drop=True)
        return df

def sample_names_data():
    df = load_names_data()
    df = df[(df["year"] == 2015) & (df["count"] > 1000)]
    return df.sample(100, random_state=0).sort_values("count")

def top_names_for_year(year=2015, gender="F", top_n=5):
    df = load_names_data()
    df = df[(df["year"] == year) & (df["gender"] == gender)]
    df = df.sort_values("count")[:top_n]
    return df

# end_data_routine

# start_XLSXExample1
class XLSXExample1:
    """
    Demonstrates rendering a simple dataframe to a xlsx file
    using the default styles
    """

    @classmethod
    def render_xlsx(cls):
        """
        Render the df to a xlsx file.
        """

        # load data
        df = sample_names_data()
        # build presentation model
        pm = tc.build_presentation_model(df=df, output_format="xlsx")

        # render to xlsx
        tempdir = tempfile.gettempdir()
        fp = os.path.join(tempdir, "example1.xlsx")
        layout = [pm]
        print("Writing to " + fp)
        xlsxw.OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=fp)

# end_XLSXExample1

# start_XLSXExample2
class XLSXExample2:
    """

```

(continues on next page)

(continued from previous page)

Demonstrates using call-backs that help set the display and style properties of each cell in the xlsx sheet.

```

"""

@staticmethod
def data_value_func(df):
    def _inner(idx, col):
        if col == "gender":
            if df.loc[idx, col] == "F":
                return "Female"
            return "Male"
        return df.loc[idx, col]

    return _inner

@staticmethod
def data_style_func(df):
    def _inner(idx, col):
        bg_color = None
        number_format = "General"
        if col == "count":
            number_format = "#,##0"
        if df.loc[idx, "gender"] == "F":
            bg_color = "bbdef8"
        else:
            bg_color = "e3f2fd"
        return xlsstyle.OpenPyxlStyleHelper.get_style(
            bg_color=bg_color, number_format=number_format
        )

    return _inner

@staticmethod
def index_name_value_func(value):
    return value.capitalize()

@staticmethod
def index_name_style_func(value):
    return xlsstyle.OpenPyxlStyleHelper.default_header_style()

@staticmethod
def header_value_func(node):
    return node.value.capitalize()

@staticmethod
def header_style_func(node):
    return xlsstyle.OpenPyxlStyleHelper.default_header_style()

@staticmethod
def index_value_func(node):
    return node.value.capitalize()

```

(continues on next page)

(continued from previous page)

```

@staticmethod
def index_style_func(df):
    def _inner(node):
        bg_color = None
        if df.loc[node.value, "gender"] == "F":
            bg_color = "bbdef8"
        else:
            bg_color = "e3f2fd"
        return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)

    return _inner

@classmethod
def render_xlsx(cls):
    # load data
    df = sample_names_data()
    # build presentation model
    klass_ = XLSXExample2
    pm = tc.build_presentation_model(
        df=df,
        output_format="xlsx",
        data_value_func=klass_.data_value_func(df),
        data_style_func=klass_.data_style_func(df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func(df),
        index_value_func=klass_.index_value_func,
        index_name_style_func=klass_.index_name_style_func,
        index_name_func=klass_.index_name_value_func,
    )

    # render to xlsx
    tempdir = tempfile.gettempdir()
    fp = os.path.join(tempdir, "example2.xlsx")
    layout = [pm]
    print("Writing to " + fp)
    xlsxw.OpenPyxlCompositor.to_xlsx(layout=layout, output_fp=fp)

# end_XLSXExample2

# start_XLSXExample3
class XLSXExample3:
    """
    Demonstrates using call-backs and also rendering multiple tables to single
    worksheet.
    """

    @staticmethod
    def data_value_func(df):
        def _inner(idx, col):
            if col == "gender":

```

(continues on next page)

(continued from previous page)

```

        if df.loc[idx, col] == "F":
            return "Female"
        return "Male"
    return df.loc[idx, col]

    return _inner

@staticmethod
def data_style_func(df):
    def _inner(idx, col):
        bg_color = None
        number_format = "General"
        if col == "count":
            number_format = "#,##0"
        if df.loc[idx, "gender"] == "F":
            bg_color = "bbdef8"
        else:
            bg_color = "e3f2fd"
        return xlsstyle.OpenPyxlStyleHelper.get_style(
            bg_color=bg_color, number_format=number_format
        )

    return _inner

@staticmethod
def index_name_value_func(value):
    return value.capitalize()

@staticmethod
def index_name_style_func(value):
    return xlsstyle.OpenPyxlStyleHelper.default_header_style()

@staticmethod
def header_value_func(node):
    return node.value.capitalize()

@staticmethod
def header_style_func(node):
    return xlsstyle.OpenPyxlStyleHelper.default_header_style()

@staticmethod
def index_value_func(node):
    return node.value.capitalize()

@staticmethod
def index_style_func(df):
    def _inner(node):
        bg_color = None
        if df.loc[node.value, "gender"] == "F":
            bg_color = "bbdef8"
        else:
            bg_color = "e3f2fd"

```

(continues on next page)

(continued from previous page)

```

        return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)

    return _inner

@classmethod
def render_xlsx(cls):
    # Prepare first data frame (same as in render_xlsx)
    df = sample_names_data()
    # build presentation model
    klass_ = XLSXExample3
    pm_all = tc.build_presentation_model(
        df=df,
        output_format="xlsx",
        data_value_func=klass_.data_value_func(df),
        data_style_func=klass_.data_style_func(df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func(df),
        index_value_func=klass_.index_value_func,
        index_name_style_func=klass_.index_name_style_func,
        index_name_func=klass_.index_name_value_func,
    )

    male_df = top_names_for_year(gender="M")
    pm_top_male = tc.build_presentation_model(
        df=male_df,
        output_format="xlsx",
        data_value_func=klass_.data_value_func(male_df),
        data_style_func=klass_.data_style_func(male_df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func(male_df),
        index_value_func=klass_.index_value_func,
        index_name_style_func=klass_.index_name_style_func,
        index_name_func=klass_.index_name_value_func,
    )

    female_df = top_names_for_year(gender="F")
    pm_top_female = tc.build_presentation_model(
        df=female_df,
        output_format="xlsx",
        data_value_func=klass_.data_value_func(female_df),
        data_style_func=klass_.data_style_func(female_df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func(female_df),
        index_value_func=klass_.index_value_func,
        index_name_style_func=klass_.index_name_style_func,
        index_name_func=klass_.index_name_value_func,
    )

    layout = [pm_all, [pm_top_female, pm_top_male]]

```

(continues on next page)

(continued from previous page)

```

    # render to xlsx
    tempdir = tempfile.gettempdir()
    fp = os.path.join(tempdir, "example3.xlsx")
    print("Writing to " + fp)
    xlszw.OpenPyxlCompositor.to_xlsx(
        layout=layout, output_fp=fp, orientation="horizontal"
    )

# end_XLSXExample3

# start_XLSXExample4
class XLSXExample4:
    """
    Demonstrate styling and rendering of multi-hierarchical indexed dataframe
    into a xlsx file.
    """

    @staticmethod
    def data_style_func(df):
        def _inner(idx, col):
            bg_color = None
            number_format = "General"
            if col == "count":
                number_format = "#,##0"
            if idx[1] == "F":
                bg_color = "bbdef8"
            else:
                bg_color = "e3f2fd"
            return xlsstyle.OpenPyxlStyleHelper.get_style(
                bg_color=bg_color, number_format=number_format
            )

        return _inner

    @staticmethod
    def index_name_value_func(value):
        return "Max By Year"

    @staticmethod
    def index_name_style_func(value):
        return xlsstyle.OpenPyxlStyleHelper.default_header_style()

    @staticmethod
    def header_value_func(node):
        return node.value.capitalize()

    @staticmethod
    def header_style_func(node):
        return xlsstyle.OpenPyxlStyleHelper.default_header_style()

    @staticmethod

```

(continues on next page)

(continued from previous page)

```

def index_value_func(node):
    if isinstance(node.value, str):
        return node.value.capitalize()
    return node.value

@staticmethod
def index_style_func(df):
    def _inner(node):
        bg_color = None
        if len(node.key) == 1:
            bg_color = "4f81bd"
        elif node.key[1] == "F":
            bg_color = "bbdef8"
        else:
            bg_color = "e3f2fd"
        return xlsstyle.OpenPyxlStyleHelper.get_style(bg_color=bg_color)

    return _inner

@classmethod
def render_xlsx(cls):

    # Prepare first data frame (same as in render_xlsx)
    data_df = load_names_data()
    data_df = data_df[data_df["year"] >= 2000]
    g = data_df.groupby(("year", "gender"))
    df = g.max()

    klass_ = cls
    pm = tc.build_presentation_model(
        df=df,
        output_format="xlsx",
        # data_value_func=None, # use default
        data_style_func=klass_.data_style_func(df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func(df),
        index_value_func=klass_.index_value_func,
        index_name_style_func=klass_.index_name_style_func,
        index_name_func=klass_.index_name_value_func,
    )

    layout = [pm]
    # render to xlsx
    tempdir = tempfile.gettempdir()
    fp = os.path.join(tempdir, "example4.xlsx")
    print("Writing to " + fp)
    xlsxw.OpenPyxlCompositor.to_xlsx(
        layout=layout, output_fp=fp, orientation="horizontal"
    )

```

(continues on next page)

(continued from previous page)

```
# end_XLSXExample4

def main():
    XLSXExample1.render_xlsx()
    XLSXExample2.render_xlsx()
    XLSXExample3.render_xlsx()
    XLSXExample4.render_xlsx()

if __name__ == "__main__":
    main()
```

9.3 HTML Examples

```
"""
This module is referred to by the Sphinx documentation. If you need to run this
module, install table_compositor in an separate environment and then run this module
in that environment. This helps the imports find the modules in the right place
"""

import collections
import os

# start_imports
import tempfile
import webbrowser
import zipfile

import pandas as pd
import requests

import table_compositor.html_styles as html_style
import table_compositor.html_writer as htmlw
import table_compositor.table_compositor as tc

# end_imports

# start_data_routine
# code snippet adapted from http://function-pipe.readthedocs.io/en/latest/usage\_df.html
# source url
URL_NAMES = "https://www.ssa.gov/oact/babynames/names.zip"
ZIP_NAME = "names.zip"

def load_names_data():
    fp = os.path.join(tempfile.gettempdir(), ZIP_NAME)
    if not os.path.exists(fp):
        r = requests.get(URL_NAMES)
```

(continues on next page)

(continued from previous page)

```

        with open(fp, "wb") as f:
            f.write(r.content)

post = collections.OrderedDict()
with zipfile.ZipFile(fp) as zf:
    # get ZipInfo instances
    for zi in sorted(zf.infolist(), key=lambda zi: zi.filename):
        fn = zi.filename
        if fn.startswith("yob"):
            year = int(fn[3:7])
            df = pd.read_csv(
                zf.open(zi), header=None, names=("name", "gender", "count")
            )
            df["year"] = year
            post[year] = df

df = pd.concat(post.values())
df.set_index("name", inplace=True, drop=True)
return df

def sample_names_data():
    df = load_names_data()
    df = df[(df["year"] == 2015) & (df["count"] > 1000)]
    return df.sample(50, random_state=0).sort_values("count")

def top_names_for_year(year=2015, gender="F", top_n=5):
    df = load_names_data()
    df = df[(df["year"] == year) & (df["gender"] == gender)]
    df = df.sort_values("count")[:top_n]
    return df

# end_data_routine

# start_HTMLExample1
class HTMLExample1:
    """
    Demonstrate rendering of a simple dataframe into html
    """

    @classmethod
    def render_html(cls):

        # load data
        df = load_names_data()
        df = df[:100]

        # build presentation model
        pm = tc.build_presentation_model(df=df, output_format="html")

```

(continues on next page)

(continued from previous page)

```

    # render to xlsx
    tempdir = tempfile.gettempdir()
    fp = os.path.join(tempdir, "example_1.html")
    layout = [pm]
    print("Writing to " + fp)
    html = htmlw.HTMLWriter.to_html(layout, border=1)
    output_fp = os.path.join(tempfile.gettempdir(), "example1.html")
    with open(output_fp, "w") as f:
        f.write(html)

# end_HTMLExample1

# start_HTMLExample2
class HTMLExample2:
    """
    Demonstrate rendering of a simple dataframe into html
    """

    @staticmethod
    def data_value_func(df):
        def _inner(idx, col):
            if col == "gender":
                if df.loc[idx, col] == "F":
                    return "Female"
                return "Male"
            return df.loc[idx, col]

        return _inner

    @staticmethod
    def data_style_func(df):
        def _inner(idx, col):
            color = "#FFFFFF"
            text_align = "left"
            if col == "count":
                text_align = "right"
            if df.loc[idx, "gender"] == "F":
                color = "#bbdef8"
            else:
                color = "#e3f2fd"
            return html_style.td_style(
                text_align=text_align,
                background_color=color,
                color="#000000",
                font_weight="normal",
                white_space="pre",
                padding="10px",
                border=None,
            )

        return _inner

```

(continues on next page)

(continued from previous page)

```

@staticmethod
def index_name_value_func(value):
    return value.capitalize()

@staticmethod
def header_value_func(node):
    return node.value.capitalize()

@staticmethod
def header_style_func(node):
    return html_style.td_style(
        text_align="center",
        background_color="#4F81BD",
        color="#FFFFFF",
        font_weight="bold",
        white_space="pre",
        padding="10px",
        border=1,
    )

@staticmethod
def index_value_func(node):
    return node.value.capitalize()

@staticmethod
def index_style_func(node):
    return html_style.td_style(
        text_align="center",
        background_color="#4F81BD",
        color="#FFFFFF",
        font_weight="bold",
        white_space="pre",
        padding="10px",
        border=1,
    )

@classmethod
def render_html(cls):
    # load data
    df = sample_names_data()
    # build presentation model
    klass_ = HTMLExample2
    pm = tc.build_presentation_model(
        df=df,
        output_format="html",
        data_value_func=klass_.data_value_func(df),
        data_style_func=klass_.data_style_func(df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func,
        index_value_func=klass_.index_value_func,

```

(continues on next page)

(continued from previous page)

```

        index_name_func=klass_.index_name_value_func,
    )

    layout = [pm]
    html = htmlw.HTMLWriter.to_html(layout, border=1)
    output_fp = os.path.join(tempfile.gettempdir(), "example2.html")
    print("Writing to =", output_fp)
    with open(output_fp, "w") as f:
        f.write(html)

# end_HTMLExample2

# start_HTMLExample3
class HTMLExample3:
    """
    Demonstrate styling and rendering of multiple multi-hierarchical indexed dataframe
    into a html file
    """

    @staticmethod
    def data_value_func(df):
        def _inner(idx, col):
            if col == "gender":
                if df.loc[idx, col] == "F":
                    return "Female"
                return "Male"
            return df.loc[idx, col]

        return _inner

    @staticmethod
    def data_style_func(df):
        def _inner(idx, col):
            color = "#FFFFFF"
            text_align = "left"
            if col == "count":
                text_align = "right"
            if df.loc[idx, "gender"] == "F":
                color = "#bbdef8"
            else:
                color = "#e3f2fd"
            return html_style.td_style(
                text_align=text_align,
                background_color=color,
                color="#000000",
                font_weight="normal",
                white_space="pre",
                padding="10px",
                border=None,
            )

```

(continues on next page)

(continued from previous page)

```

        return _inner

    @staticmethod
    def index_name_value_func(value):
        return "Max By Year"

    @staticmethod
    def header_value_func(node):
        return node.value.capitalize()

    @staticmethod
    def header_style_func(node):
        return html_style.td_style(
            text_align="center",
            background_color="#4F81BD",
            color="#FFFFFF",
            font_weight="bold",
            white_space="pre",
            padding="10px",
            border=1,
        )

    @staticmethod
    def index_value_func(node):
        if isinstance(node.value, str):
            return node.value.capitalize()
        return node.value

    @staticmethod
    def index_style_func(node):
        return html_style.td_style(
            text_align="center",
            background_color="#4F81BD",
            color="#FFFFFF",
            font_weight="bold",
            white_space="pre",
            padding="10px",
            border=1,
        )

    @classmethod
    def render_html(cls):

        # Prepare first data frame (same as in render_xlsx)
        df = sample_names_data()
        # build presentation model
        klass_ = HTMLExample4
        pm_all = tc.build_presentation_model(
            df=df,
            output_format="html",
            data_value_func=klass_.data_value_func(df),

```

(continues on next page)

(continued from previous page)

```

        data_style_func=klass_.data_style_func(df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func,
        index_value_func=klass_.index_value_func,
        index_name_func=lambda _: "Sample Data",
    )

    male_df = top_names_for_year(gender="M")
    pm_top_male = tc.build_presentation_model(
        df=male_df,
        output_format="html",
        data_value_func=klass_.data_value_func(male_df),
        data_style_func=klass_.data_style_func(male_df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func,
        index_value_func=klass_.index_value_func,
        index_name_func=lambda _: "Max by Year",
    )

    female_df = top_names_for_year(gender="F")
    pm_top_female = tc.build_presentation_model(
        df=female_df,
        output_format="html",
        data_value_func=klass_.data_value_func(female_df),
        data_style_func=klass_.data_style_func(female_df),
        header_value_func=klass_.header_value_func,
        header_style_func=klass_.header_style_func,
        index_style_func=klass_.index_style_func,
        index_value_func=klass_.index_value_func,
        index_name_func=lambda _: "Max by Year",
    )

    layout = [pm_all, [pm_top_female, pm_top_male]]
    # render to xlsx
    html = htmlw.HTMLWriter.to_html(layout, border=1, orientation="horizontal")
    output_fp = os.path.join(tempfile.gettempdir(), "example3.html")
    print("Writing to =", output_fp)
    with open(output_fp, "w") as f:
        f.write(html)

# end_HTMLExample3

# start_HTMLExample4
class HTMLExample4:
    """
    Demonstrate styling and rendering of multi-hierarchical indexed dataframe
    into a html file.
    """

```

(continues on next page)

(continued from previous page)

```

@staticmethod
def data_value_func(df):
    def _inner(idx, col):
        if col == "gender":
            if df.loc[idx, col] == "F":
                return "Female"
            return "Male"
        return df.loc[idx, col]

    return _inner

@staticmethod
def data_style_func(df):
    def _inner(idx, col):
        color = "#FFFFFF"
        text_align = "left"
        if col == "count":
            text_align = "right"
        if idx[1] == "F":
            color = "#bbdef8"
        else:
            color = "#e3f2fd"

        return html_style.td_style(
            text_align=text_align,
            background_color=color,
            color="#000000",
            font_weight="normal",
            white_space="pre",
            padding="10px",
            border=None,
        )

    return _inner

@staticmethod
def index_name_value_func(value):
    return "Max By Year"

@staticmethod
def header_value_func(node):
    return node.value.capitalize()

@staticmethod
def header_style_func(node):
    return html_style.td_style(
        text_align="center",
        background_color="#4F81BD",
        color="#FFFFFF",
        font_weight="bold",
        white_space="pre",
        padding="10px",
    )

```

(continues on next page)

(continued from previous page)

```

        border=1,
    )

    @staticmethod
    def index_value_func(node):
        if isinstance(node.value, str):
            return node.value.capitalize()
        return node.value

    @staticmethod
    def index_style_func(node):
        return html_style.td_style(
            text_align="center",
            background_color="#4F81BD",
            color="#FFFFFF",
            font_weight="bold",
            white_space="pre",
            padding="10px",
            border=1,
        )

    @classmethod
    def render_html(cls):

        # Prepare first data frame (same as in render_xlsx)
        data_df = load_names_data()
        data_df = data_df[data_df["year"] >= 2000]
        g = data_df.groupby(("year", "gender"))
        df = g.max()

        klass_ = cls
        pm = tc.build_presentation_model(
            df=df,
            output_format="html",
            data_value_func=klass_.data_value_func(df),
            data_style_func=klass_.data_style_func(df),
            header_value_func=klass_.header_value_func,
            header_style_func=klass_.header_style_func,
            index_style_func=klass_.index_style_func,
            index_value_func=klass_.index_value_func,
            index_name_func=klass_.index_name_value_func,
        )

        layout = [pm]
        # render to xlsx
        html = htmlw.HTMLWriter.to_html(layout, border=1)
        output_fp = os.path.join(tempfile.gettempdir(), "example4.html")
        print("Writing to =", output_fp)
        with open(output_fp, "w") as f:
            f.write(html)

```

(continues on next page)

(continued from previous page)

```
# end_HTMLExample4
```

```
def main():
    HTMLExample1.render_html()
    HTMLExample2.render_html()
    HTMLExample3.render_html()
    HTMLExample4.render_html()
```

```
if __name__ == "__main__":
    main()
```


INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

INDEX

B

`build_presentation_model()` (in module `table_compositor.table_compositor`), 7, 47

D

`data_style_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`data_style_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 50

`data_value_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`data_value_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 50

`default_header_style()` (`table_compositor.xlsx_styles.OpenPyxlStyleHelper` static method), 49

`default_header_style()` (`table_compositor.xlsx_styles.XlsxWriterStyleHelper` static method), 50

G

`get_style()` (`table_compositor.xlsx_styles.OpenPyxlStyleHelper` static method), 49

`get_style()` (`table_compositor.xlsx_styles.XlsxWriterStyleHelper` static method), 50

H

`header_style_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`header_style_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

`header_value_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`header_value_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

`HTMLWriter` (class in `table_compositor.html_writer`), 49

`HTMLWriterDefaults` (class in `table_compositor.html_styles`), 52

`index_name_style_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`index_name_style_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

`index_name_value_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`index_name_value_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

`index_style_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 52

`index_style_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

`index_value_func()` (`table_compositor.html_styles.HTMLWriterDefaults` static method), 53

`index_value_func()` (`table_compositor.xlsx_styles.XLSXWriterDefaults` static method), 51

O

`OpenPyxlStyleHelper` (class in `table_compositor.xlsx_styles`), 49

T

`to_html()` (`table_compositor.html_writer.HTMLWriter` static method), 49

X

XLSXWriterDefaults (class in ta-
ble_compositor.xlsx_styles), [50](#)

XlsxWriterStyleHelper (class in ta-
ble_compositor.xlsx_styles), [50](#)