PrettyPandas Documentation

Release 0.0.3

Henry Hammond

Mar 24, 2018

Contents

PrettyPandas is a Pandas DataFrame Styler class that helps you create report quality tables with a simple API.

```
(PrettyPandas(df, precision=0)
 .as_percent(subset=['C', 'D'], precision=3)
 .as_money(subset='A')
 .average()
 .total()
)
```

	A	B	C	D
0	$10	58.0	7.156%	39.343%
1	$3	64.0	121.872%	33.958%
2	$5	56.0	138.399%	26.289%
3	$3	47.0	45.284%	147.185%
4	$5	46.0	119.918%	208.354%
Average	**$5**	**54.0**	**86.526%**	**91.026%**
Total	**$25**	**271.0**	**432.628%**	**455.129%**

CHAPTER 1

Features

- Add summary rows and columns.
- A nice and customizable theme.
- Number formatting for currency, scientific units, and percentages.
- Chaining commands.
- Works seamlessly with Pandas Style API.

CHAPTER 2

Installation

You can install PrettyPandas using `pip` with support for Python 2.7, 3.3, 3.4, and 3.5:

```
pip install prettypandas
```

You can also install from source:

```
git clone git@github.com:HHammond/PrettyPandas.git
cd PrettyPandas
python setup.py install
```

CHAPTER 3

Contributing

The project is available on GitHub and anyone is welcome to contribute. You can use the issue tracker to report issues, bugs, or suggest improvements.

CHAPTER 4

Contents

4.1 Getting Started

4.1.1 Adding Style to a DataFrame

The PrettyPandas class takes advantage of the new Pandas Style API to create custom tables for your dataframes. If you have a dataframe `df`, this is how that might look:

```
from prettypandas import PrettyPandas

table = PrettyPandas(df)
```

	A	B	C	D
0	9.935787	58.0	0.071559	0.393433
1	2.902448	64.0	1.218719	0.339576
2	4.647678	56.0	1.383985	0.262889
3	2.629603	47.0	0.452839	1.47185
4	4.777165	46.0	1.199178	2.083545

Note: An instance of PrettyPandas is no longer the original dataframe but a presentation of the dataframe. That means you cannot use any of the standard transformations from the `pandas.DataFrame` class. The PrettyPandas instance can't change the original dataframe so there's no fear of contaminating your data.

4.1.2 Adding Summaries

PrettyPandas supports many different summary functions, as well the ability to apply summaries along rows, columns, or both. Summaries chain together so you can use multiple summaries without any headaches.

The builtin summary methods are:

- *total*
- *average*
- *median*
- *min*
- *max*

The above functions work nicely on your table, if you wanted to add a grand total to the bottom of your table the code is simple:

```
PrettyPandas(df).total()
```

	A	B	C	D
0	9.935787	58.0	0.071559	0.393433
1	2.902448	64.0	1.218719	0.339576
2	4.647678	56.0	1.383985	0.262889
3	2.629603	47.0	0.452839	1.47185
4	4.777165	46.0	1.199178	2.083545
Total	**24.892682**	**271.0**	**4.326281**	**4.551292**

And if you want to mix and match summaries:

```
PrettyPandas(df).total().average()
```

	A	B	C	D
0	9.935787	58.0	0.071559	0.393433
1	2.902448	64.0	1.218719	0.339576
2	4.647678	56.0	1.383985	0.262889
3	2.629603	47.0	0.452839	1.47185
4	4.777165	46.0	1.199178	2.083545
Total	**24.892682**	**271.0**	**4.326281**	**4.551292**
Average	**4.978536**	**54.2**	**0.865256**	**0.910258**

The `axis` parameter specifies which `numpy` style axis to apply a summary on — 0 for columns, 1 for rows, and `None` for both.

```
PrettyPandas(df).total(axis=1)
```

	A	B	C	D	Total
0	9.935787	58.0	0.071559	0.393433	**68.400779**
1	2.902448	64.0	1.218719	0.339576	**68.460743**
2	4.647678	56.0	1.383985	0.262889	**62.294552**
3	2.629603	47.0	0.452839	1.47185	**51.554292**
4	4.777165	46.0	1.199178	2.083545	**54.059888**

You can even mix and match summaries applied to different axis.

Creating a Custom Summary

The *summary* method creates a custom summary from a function which takes an array-like structure as a list.

```
def count_greater_than_five(items):
    return sum(item > 5 for item in items)

PrettyPandas(df).summary(count_greater_than_five, title="> 5")
```

	A	B	C	D
0	9.935787	58.0	0.071559	0.393433
1	2.902448	64.0	1.218719	0.339576
2	4.647678	56.0	1.383985	0.262889
3	2.629603	47.0	0.452839	1.47185
4	4.777165	46.0	1.199178	2.083545
> 5	**1.0**	**5.0**	**0.0**	**0.0**

4.1.3 Formatting Numbers

Most reports use at least some units of measurement. PrettyPandas currently supports percentages, money, and a more general unit method.

- *as_percent*
- *as_currency*
- *as_unit*

The `as_unit` method takes a positional `unit` argument which indicates the string representing the unit to be used and a `location` argument to specify whether the unit should be a prefix or suffix to the value.

The `as_currency` and `as_percent` methods are localized to use whatever units your Python distribution thinks are best for you. If you aren't getting the correct units use the *`set_locale`* method to specify your locale.

If you need to use a different currency, just pass it to `currency='...'` to change it.

The `as_money` method takes optional `currency` and `location` arguments which work just like the `as_unit` method. By default the currency is in dollars.

Note: Python 2 doesn't support unicode literals by default. You can use unicode literals (e.g. `u'€'`) or import the unicode literal behaviour from Python 3:

```
from __future__ import unicode_literals
```

Formatting Columns

By default the formatting methods apply to the entire dataframe. When you need to format just a few columns you can use the *subset* argument to specify a single column, or multiple columns.

```
PrettyPandas(df).as_percent(subset='A')  # Format just column A
```

	A	B	C	D
0	993.578709%	58	0.071559	0.393433
1	290.244824%	64	1.218719	0.339576
2	464.767819%	56	1.383985	0.262889
3	262.960342%	47	0.452839	1.47185
4	477.716520%	46	1.199178	2.083545

```
PrettyPandas(df).as_percent(subset=['A', 'B'])  # Format columns A and B
```

	A	B	C	D
0	993.578709%	5800.000000%	0.071559	0.393433
1	290.244824%	6400.000000%	1.218719	0.339576
2	464.767819%	5600.000000%	1.383985	0.262889
3	262.960342%	4700.000000%	0.452839	1.47185
4	477.716520%	4600.000000%	1.199178	2.083545

Formatting Rows and Complex Formatting

Formatting rows is more complicated than formatting columns. The *subset* argument needs to take in a *pandas.Index* to specify the row.

```
# Format the row with row-index 3
PrettyPandas(df, precision=2).as_percent(subset=pd.IndexSlice[3,:])
```

	A	B	C	D
0	9.94	58	0.07	0.39
1	2.9	64	1.22	0.34
2	4.65	56	1.38	0.26
3	262.96%	4700.00%	45.28%	147.18%
4	4.78	46	1.2	2.08

For multi-index dataframes subsetting is more complicated. You will need to use multiple `pandas.IndexSlice` objects to get the correct rows.

The following example shows how to select rows in a multi-index:

```
idx = pd.IndexSlice
first_row_idx = idx[:, 1]    # select all with index like (*, 1)
second_row_idx = idx[:, 2]   # select all with index like (*, 2)

(PrettyPandas(df2)
 .as_money(subset=idx[first_row_idx, :])
 .as_percent(subset=idx[second_row_idx, :])
 )
```

		A	B	C	D
1	1	$9.94	$58.00	$0.07	$0.39
1	2	290.24%	6400.00%	121.87%	33.96%
2	1	$4.65	$56.00	$1.38	$0.26
2	2	262.96%	4700.00%	45.28%	147.18%
3	1	$4.78	$46.00	$1.20	$2.08

For more info on Pandas indexing, read Pandas Indexing and Pandas Advanced Indexing.

4.1.4 The Magic Function

The `apply_pretty_globals()` function will patch your notebook so that all tables are styled the same. This injects HTML into the notebook (which some hosts don't allow).

4.2 prettypandas package

class prettypandas.**PrettyPandas**(*data*, *summary_rows=None*, *summary_cols=None*, *formatters=None*, **args*, ***kwargs*)

Bases: pandas.io.formats.style.Styler

Pretty pandas dataframe Styles.

Parameters

- **data** – Series or DataFrame
- **precision** – int precision to round floats to, defaults to pd.options.display.precision
- **table_styles** – list-like, default None list of {selector: (attr, value)} dicts. These values overwrite the default style.
- **uuid** – str, default None a unique identifier to avoid CSS collisons; generated automatically
- **caption** – str, default None caption to attach to the table
- **summary_rows** – list of single-row dataframes to be appended as a summary
- **summary_cols** – list of single-row dataframes to be appended as a summary

DEFAULT_BACKGROUND = u'#eee'

DEFAULT_BORDER_COLOUR = u'#c0c0c0'

DEFAULT_LOCALE = Locale('en_US')

HEADER_PROPERTIES = [(u'background', u'#eee'), (u'font-weight', u'500')]

STYLES = [{u'props': [(u'background', u'#eee'), (u'font-weight', u'500')], u'selector

SUMMARY_PROPERTIES = [(u'background', u'#eee'), (u'font-weight', u'500')]

as_currency(*subset=None*, *currency=u'USD'*, *locale=None*)

Represent subset of dataframe as currency.

Parameters

- **subset** – Pandas slice to convert to percentages
- **currency** – Currency or currency symbol to be used
- **locale** – Locale to be used (e.g. 'en_US')

as_money(*subset=None*, *precision=None*, *currency=u'$'*, *location=u'prefix'*)

[DEPRECATED] Represent subset of dataframe as currency.

Parameters

- **precision** – int Number of decimal places to round to
- **subset** – Pandas slice to convert to percentages
- **currency** – Currency string
- **location** – 'prefix' or 'suffix' indicating where the currency symbol should be.

as_percent(*subset=None*, *precision=None*, *locale=None*)

Represent subset of dataframe as percentages.

Parameters

- **subset** – Pandas slice to convert to percentages

- **precision** – int Number of decimal places to round to
- **locale** – Locale to be used (e.g. 'en_US')

as_unit(*unit*, *subset=None*, *precision=None*, *location=u'prefix'*)
Represent subset of dataframe as a special unit.

Parameters

- **unit** – string representing unit to be used.
- **subset** – Pandas slice to convert to percentages
- **precision** – int Number of decimal places to round to
- **location** – 'prefix' or 'suffix' indicating where the currency symbol should be.

average(*title=u'Average'*, ***kwargs*)
Add a mean summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.mean`.

max(*title=u'Maximum'*, ***kwargs*)
Add a maximum summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.max`.

median(*title=u'Median'*, ***kwargs*)
Add a median summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.median`.

min(*title=u'Minimum'*, ***kwargs*)
Add a minimum summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.min`.

multi_summary(*funcs*, *titles*, *axis=0*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **funcs** – Iterable of functions to be used for a summary.
- **titles** – Iterable of titles in the same order as the functions.
- **axis** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **kwargs** – Keyword arguments passed to all the functions.

classmethod set_locale(*locale*)
Set the PrettyPandas default locale.

summary (*func=<function sum>*, *title=u'Total'*, *axis=0*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **func** – Iterable of functions to be used for a summary.
- **titles** – Iterable of titles in the same order as the functions.
- **axis** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **kwargs** – Keyword arguments passed to all the functions.

The results of summary can be chained together.

total (*title=u'Total'*, ***kwargs*)
Add a total summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.sum`.

prettypandas.**apply_pretty_globals**()
Apply global CSS to make dataframes pretty.

This function injects HTML and CSS code into the notebook in order to make tables look pretty. Third party hosts of notebooks advise against using this and some don't support it. As long as you are okay with HTML injection in your notebook, go ahead and use this. Otherwise use the `PrettyPandas` class.

4.2.1 Submodules

prettypandas.formatters module

prettypandas.formatters.**as_money** (*v*, *precision=2*, *currency='$'*, *location='prefix'*)
[DEPRECATED] Convert value to currency.

Parameters

- **v** – numerical value
- **precision** – int decimal places to round to
- **currency** – string representing currency
- **location** – 'prefix' or 'suffix' representing where the currency symbol falls relative to the value

prettypandas.formatters.**as_unit** (*v*, *unit*, *precision=2*, *location='suffix'*)
Convert value to unit.

Parameters

- **v** – numerical value
- **unit** – string of unit
- **precision** – int decimal places to round to
- **location** – 'prefix' or 'suffix' representing where the currency symbol falls relative to the value

prettypandas.formatters.**format_number**(*v*, *number_format*, *prefix=''*, *suffix=''*)
Format a number to a string.

prettypandas.styler module

class prettypandas.styler.**Formatter**(*subset*, *function*)
Bases: tuple

function
Alias for field number 1

subset
Alias for field number 0

class prettypandas.styler.**PrettyPandas**(*data*, *summary_rows=None*, *summary_cols=None*, *formatters=None*, **args*, ***kwargs*)
Bases: pandas.io.formats.style.Styler

Pretty pandas dataframe Styles.

Parameters

- **data** – Series or DataFrame
- **precision** – int precision to round floats to, defaults to pd.options.display.precision
- **table_styles** – list-like, default None list of {selector: (attr, value)} dicts. These values overwrite the default style.
- **uuid** – str, default None a unique identifier to avoid CSS collisons; generated automatically
- **caption** – str, default None caption to attach to the table
- **summary_rows** – list of single-row dataframes to be appended as a summary
- **summary_cols** – list of single-row dataframes to be appended as a summary

DEFAULT_BACKGROUND = u'#eee'

DEFAULT_BORDER_COLOUR = u'#c0c0c0'

DEFAULT_LOCALE = Locale('en_US')

HEADER_PROPERTIES = [(u'background', u'#eee'), (u'font-weight', u'500')]

STYLES = [{u'props': [(u'background', u'#eee'), (u'font-weight', u'500')], u'selector

SUMMARY_PROPERTIES = [(u'background', u'#eee'), (u'font-weight', u'500')]

as_currency(*subset=None*, *currency=u'USD'*, *locale=None*)
Represent subset of dataframe as currency.

Parameters

- **subset** – Pandas slice to convert to percentages
- **currency** – Currency or currency symbol to be used
- **locale** – Locale to be used (e.g. 'en_US')

as_money(*subset=None*, *precision=None*, *currency=u'$'*, *location=u'prefix'*)
[DEPRECATED] Represent subset of dataframe as currency.

Parameters

- **precision** – int Number of decimal places to round to

- **subset** – Pandas slice to convert to percentages
- **currency** – Currency string
- **location** – 'prefix' or 'suffix' indicating where the currency symbol should be.

as_percent(*subset=None*, *precision=None*, *locale=None*)
Represent subset of dataframe as percentages.

Parameters

- **subset** – Pandas slice to convert to percentages
- **precision** – int Number of decimal places to round to
- **locale** – Locale to be used (e.g. 'en_US')

as_unit(*unit*, *subset=None*, *precision=None*, *location=u'prefix'*)
Represent subset of dataframe as a special unit.

Parameters

- **unit** – string representing unit to be used.
- **subset** – Pandas slice to convert to percentages
- **precision** – int Number of decimal places to round to
- **location** – 'prefix' or 'suffix' indicating where the currency symbol should be.

average(*title=u'Average'*, ***kwargs*)
Add a mean summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.mean`.

max(*title=u'Maximum'*, ***kwargs*)
Add a maximum summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.max`.

median(*title=u'Median'*, ***kwargs*)
Add a median summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.median`.

min(*title=u'Minimum'*, ***kwargs*)
Add a minimum summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.min`.

multi_summary(*funcs*, *titles*, *axis=0*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **funcs** – Iterable of functions to be used for a summary.
- **titles** – Iterable of titles in the same order as the functions.
- **axis** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **kwargs** – Keyword arguments passed to all the functions.

classmethod set_locale(*locale*)
Set the PrettyPandas default locale.

summary(*func=<function sum>*, *title=u'Total'*, *axis=0*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **func** – Iterable of functions to be used for a summary.
- **titles** – Iterable of titles in the same order as the functions.
- **axis** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **kwargs** – Keyword arguments passed to all the functions.

The results of summary can be chained together.

total(*title=u'Total'*, ***kwargs*)
Add a total summary to this table.

Parameters

- **title** – Title to be displayed.
- **kwargs** – Keyword arguments passed to `numpy.sum`.

`prettypandas.styler.`**apply_pretty_globals**()
Apply global CSS to make dataframes pretty.

This function injects HTML and CSS code into the notebook in order to make tables look pretty. Third party hosts of notebooks advise against using this and some don't support it. As long as you are okay with HTML injection in your notebook, go ahead and use this. Otherwise use the `PrettyPandas` class.

4.3 Testing

Tests use pytest for testing. After downloading the repository from GitHub run the following:

```
py.test test
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

T