PrettyPandas Documentation

Release 0.0.4

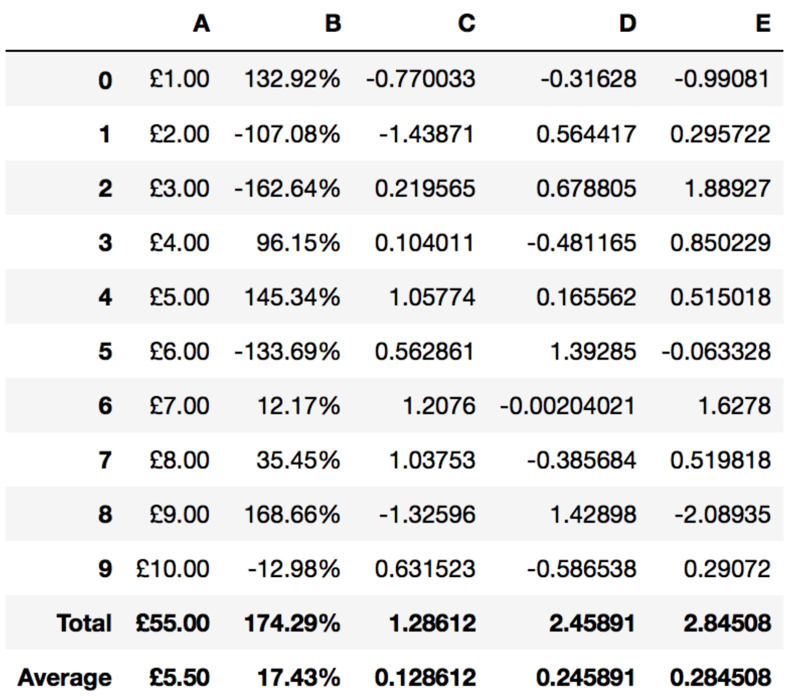
Henry Hammond

Mar 26, 2018

Contents

PrettyPandas is an extension to the Pandas DataFrame class that helps you create report qualitiy tables with a simple API.

```
(
    df
    .pipe(PrettyPandas)
    .as_currency('GBP', subset='A')
    .as_percent(subset='B')
    .total()
    .average()
)
```



	A	B	C	D	E
0	£1.00	132.92%	-0.770033	-0.31628	-0.99081
1	£2.00	-107.08%	-1.43871	0.564417	0.295722
2	£3.00	-162.64%	0.219565	0.678805	1.88927
3	£4.00	96.15%	0.104011	-0.481165	0.850229
4	£5.00	145.34%	1.05774	0.165562	0.515018
5	£6.00	-133.69%	0.562861	1.39285	-0.063328
6	£7.00	12.17%	1.2076	-0.00204021	1.6278
7	£8.00	35.45%	1.03753	-0.385684	0.519818
8	£9.00	168.66%	-1.32596	1.42898	-2.08935
9	£10.00	-12.98%	0.631523	-0.586538	0.29072
Total	**£55.00**	**174.29%**	**1.28612**	**2.45891**	**2.84508**
Average	**£5.50**	**17.43%**	**0.128612**	**0.245891**	**0.284508**

CHAPTER 1

Features

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

Note: Version 0.0.4 removes the `apply_pretty_globals` function and other custom CSS properties because Pandas and Jupyter now defaults to providing great looking html tables. If you still want custom CSS you can use the 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 Quick Start

4.1.1 Adding Summaries

PrettyPandas supports many built in summary functions, as well as providing the ability to create your own summaries. Summary functions can be applied over a DataFrame's rows or columns, or both.

The builtin summary methods are:

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

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	E
0	1	1.32921	-0.770033	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072
Total	**55**	**1.74294**	**1.28612**	**2.45891**	**2.84508**

Or additionally if you want to use Pandas fluent API:

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

PrettyPandas follows a fluent API so you can chain multiple summaries easily:

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

	A	B	C	D	E
0	1	1.32921	-0.770033	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072
Average	**5.5**	**0.174294**	**0.128612**	**0.245891**	**0.284508**

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	E	Average
0	1	1.32921	-0.770033	-0.31628	-0.99081	**0.0504176**
1	2	-1.07082	-1.43871	0.564417	0.295722	**0.0701218**
2	3	-1.6264	0.219565	0.678805	1.88927	**0.832248**
3	4	0.961538	0.104011	-0.481165	0.850229	**1.08692**
4	5	1.45342	1.05774	0.165562	0.515018	**1.63835**
5	6	-1.33694	0.562861	1.39285	-0.063328	**1.31109**
6	7	0.121668	1.2076	-0.00204021	1.6278	**1.99101**
7	8	0.354493	1.03753	-0.385684	0.519818	**1.90523**
8	9	1.68658	-1.32596	1.42898	-2.08935	**1.74005**
9	10	-0.12982	0.631523	-0.586538	0.29072	**2.04118**

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_zero(column):
    return (column > 0).sum()

PrettyPandas(df).summary(count_greater_than_zero, title="> 0")
```

	A	B	C	D	E
0	1	1.32921	-0.770033	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	4	0.961538	0.104011	-0.481165	0.850229
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072
> 0	**10**	**6**	**7**	**5**	**7**

4.1.2 Converting Back to Pandas DataFrame

`.to_frame()`

After adding summary rows or columns you can get a DataFrame with your changes applied by calling the `._to_frame`.

For example the following code would add a total to your DataFrame and return it back to a Pandas native DataFrame.

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

`.style`

The `.style` property allows you to drop right into the Pandas Style API. This code would allow you to compute a summary, format the table using percentages, and apply a backgrouned gradient to a table:

```
(
    df.pipe(PrettyPandas)
    .as_percent(precision=0)
    .median()
    .style
    .background_gradient()
)
```

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	E
0	100.00%	1.32921	-0.770033	-0.31628	-0.99081
1	200.00%	-1.07082	-1.43871	0.564417	0.295722
2	300.00%	-1.6264	0.219565	0.678805	1.88927
3	400.00%	0.961538	0.104011	-0.481165	0.850229
4	500.00%	1.45342	1.05774	0.165562	0.515018
5	600.00%	-1.33694	0.562861	1.39285	-0.063328
6	700.00%	0.121668	1.2076	-0.00204021	1.6278
7	800.00%	0.354493	1.03753	-0.385684	0.519818
8	900.00%	1.68658	-1.32596	1.42898	-2.08935
9	1000.00%	-0.12982	0.631523	-0.586538	0.29072

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

	A	B	C	D	E
0	100.00%	132.92%	-0.770033	-0.31628	-0.99081
1	200.00%	-107.08%	-1.43871	0.564417	0.295722
2	300.00%	-162.64%	0.219565	0.678805	1.88927
3	400.00%	96.15%	0.104011	-0.481165	0.850229
4	500.00%	145.34%	1.05774	0.165562	0.515018
5	600.00%	-133.69%	0.562861	1.39285	-0.063328
6	700.00%	12.17%	1.2076	-0.00204021	1.6278
7	800.00%	35.45%	1.03753	-0.385684	0.519818
8	900.00%	168.66%	-1.32596	1.42898	-2.08935
9	1000.00%	-12.98%	0.631523	-0.586538	0.29072

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).as_percent(subset=pd.IndexSlice[3,:], precision=2)
```

	A	B	C	D	E
0	1	1.32921	-0.770033	-0.31628	-0.99081
1	2	-1.07082	-1.43871	0.564417	0.295722
2	3	-1.6264	0.219565	0.678805	1.88927
3	400.00%	96.15%	10.40%	-48.12%	85.02%
4	5	1.45342	1.05774	0.165562	0.515018
5	6	-1.33694	0.562861	1.39285	-0.063328
6	7	0.121668	1.2076	-0.00204021	1.6278
7	8	0.354493	1.03753	-0.385684	0.519818
8	9	1.68658	-1.32596	1.42898	-2.08935
9	10	-0.12982	0.631523	-0.586538	0.29072

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:

```
first_row_idx = pd.IndexSlice[0, :]
second_row_idx = pd.IndexSlice[1, :]

(
    df.pipe(PrettyPandas)
    .as_currency(subset=first_row_idx)
    .as_percent(subset=second_row_idx)
    .total(axis=1)
)
```

	A	B	C	D	E	Total
0	$1.00	$1.33	-$0.77	-$0.32	-$0.99	**$0.25**
1	200.00%	-107.08%	-143.87%	56.44%	29.57%	**35.06%**
2	3	-1.6264	0.219565	0.678805	1.88927	**4.16124**
3	4	0.961538	0.104011	-0.481165	0.850229	**5.43461**
4	5	1.45342	1.05774	0.165562	0.515018	**8.19174**
5	6	-1.33694	0.562861	1.39285	-0.063328	**6.55545**
6	7	0.121668	1.2076	-0.00204021	1.6278	**9.95503**
7	8	0.354493	1.03753	-0.385684	0.519818	**9.52615**
8	9	1.68658	-1.32596	1.42898	-2.08935	**8.70025**
9	10	-0.12982	0.631523	-0.586538	0.29072	**10.2059**

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

4.2 Testing

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

```
py.test test
```

4.3 prettypandas package

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

Bases: `object`

Parameters

- **data** – DataFrame.
- **summary_rows** – list of Aggregate objects to be appended as a summary.
- **summary_cols** – list of Aggregate objects to be appended as a summary.
- **formatters** – List of Formatter objects to format.

as_currency(*currency=u'USD'*, *locale=Locale('en_US')*, **args*, ***kwargs*)

Format subset as currency

Parameters

- **currency** – Currency
- **locale** – Babel locale for currency formatting
- **subset** – Pandas subset

as_percent(*precision=2*, **args*, ***kwargs*)

Format subset as percentages

Parameters

- **precision** – Decimal precision
- **subset** – Pandas subset

as_unit(*unit*, *location=u'suffix'*, **args*, ***kwargs*)

Format subset as with units

Parameters

- **unit** – string to use as unit
- **location** – prefix or suffix
- **subset** – Pandas subset

average(*title=u'Average'*, ***kwargs*)

Add a mean summary to this table.

Parameters **title** – Title to be displayed.

frame

Add summaries and convert back to DataFrame

max(*title=u'Maximum'*, ***kwargs*)

Add a maximum summary to this table.

Parameters **title** – Title to be displayed.

median(*title=u'Median'*, ***kwargs*)

Add a median summary to this table.

Parameters **`title`** – Title to be displayed.

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

Parameters **`title`** – Title to be displayed.

`multi_summary` (*funcs*, *titles*, *axis=0*, **args*, ***kwargs*)

`render` ()

`style`
Add summaries and convert to Pandas Styler

`summary` (*func=<operator.methodcaller object>*, *title=u'Total'*, *axis=0*, *subset=None*, **args*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **`func`** – function to be used for a summary.
- **`titles`** – Title for this summary column.
- **`axis`** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **`args`** – Positional arguments passed to all the functions.
- **`kwargs`** – Keyword arguments passed to all the functions.

The results of summary can be chained together.

`to_frame` ()
Add summaries and convert back to DataFrame

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

Parameters **`title`** – Title to be displayed.

`prettypandas.`**`as_currency`** (*currency='USD'*, *locale=Locale('en_US')*)

`prettypandas.`**`as_percent`** (*precision=2*, ***kwargs*)
Convert number to percentage string.

Parameters

- **`v`** – numerical value to be converted
- **`precision`** – int decimal places to round to

`prettypandas.`**`as_unit`** (*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

4.3.1 Submodules

prettypandas.summarize module

class `prettypandas.summarizer.`**Aggregate**(*title*, *func*, *subset=None*, *axis=0*, **args*, ***kwargs*)

Bases: `object`

Aggreagte

Wrapper to calculate aggregate row on datafame.

Parameters

- **title** – Aggregate row title
- **func** – Function to be passed to DataFrame.agg
- **subset** – Subset of DataFrame to compute aggregate on
- **axis** – Pandas axis to compute over
- **args** – Positionsal arguments to DataFrame.agg
- **kwargs** – Keyword arguments to DataFrame.agg

apply(*df*)

Compute aggregate over DataFrame

class `prettypandas.summarizer.`**Formatter**(*formatter*, *args*, *kwargs*)

Bases: `object`

Wrapper to apply formatting to datafame.

Parameters

- **formatter** – Function to be passed to Pandas Styler.format
- **args** – Positionsal arguments to Styler.format
- **kwargs** – Keyword arguments to Styler.format

apply(*styler*)

Apply Summary over Pandas Styler

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

Bases: `object`

Parameters

- **data** – DataFrame.
- **summary_rows** – list of Aggregate objects to be appended as a summary.
- **summary_cols** – list of Aggregate objects to be appended as a summary.
- **formatters** – List of Formatter objects to format.

as_currency(*currency=u'USD'*, *locale=Locale('en_US')*, **args*, ***kwargs*)

Format subset as currency

Parameters

- **currency** – Currency
- **locale** – Babel locale for currency formatting

- **`subset`** – Pandas subset

`as_percent` (*precision=2*, **args*, ***kwargs*)
Format subset as percentages

Parameters

- **`precision`** – Decimal precision
- **`subset`** – Pandas subset

`as_unit` (*unit*, *location=u'suffix'*, **args*, ***kwargs*)
Format subset as with units

Parameters

- **`unit`** – string to use as unit
- **`location`** – prefix or suffix
- **`subset`** – Pandas subset

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

Parameters **`title`** – Title to be displayed.

`frame`
Add summaries and convert back to DataFrame

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

Parameters **`title`** – Title to be displayed.

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

Parameters **`title`** – Title to be displayed.

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

Parameters **`title`** – Title to be displayed.

`multi_summary` (*funcs*, *titles*, *axis=0*, **args*, ***kwargs*)

`render` ()

`style`
Add summaries and convert to Pandas Styler

`summary` (*func=<operator.methodcaller object>*, *title=u'Total'*, *axis=0*, *subset=None*, **args*, ***kwargs*)
Add multiple summary rows or columns to the dataframe.

Parameters

- **`func`** – function to be used for a summary.
- **`titles`** – Title for this summary column.
- **`axis`** – Same as numpy and pandas axis argument. A value of None will cause the summary to be applied to both rows and columns.
- **`args`** – Positional arguments passed to all the functions.
- **`kwargs`** – Keyword arguments passed to all the functions.

The results of summary can be chained together.

`to_frame`()
Add summaries and convert back to DataFrame

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

Parameters **`title`** – Title to be displayed.

prettypandas.formatters module

`prettypandas.formatters.`**`as_currency`**(*currency='USD'*, *locale=Locale('en_US')*)

`prettypandas.formatters.`**`as_percent`**(*precision=2*, ***kwargs*)
Convert number to percentage string.

Parameters

- **`v`** – numerical value to be converted
- **`precision`** – int decimal places to round to

`prettypandas.formatters.`**`as_unit`**(*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

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index