upsetplot Documentation

Release 0.2.1

Joel Nothman

Mar 18, 2020

Contents

This is another Python implementation of UpSet plots by Lex et al. [Lex2014]. UpSet plots are used to visualise set overlaps; like Venn diagrams but more readable. Documentation is at https://upsetplot.readthedocs.io.

This `upsetplot` library tries to provide a simple interface backed by an extensible, object-oriented design.

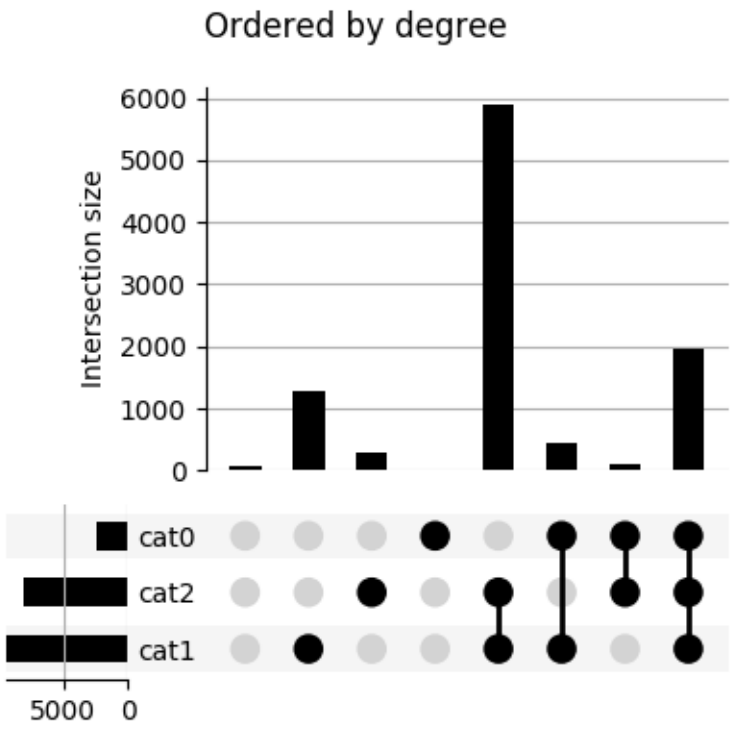
The basic input format is a `pandas.Series` containing counts corresponding to set intersection sizes. The index indicates which rows pertain to which sets, by having multiple boolean indices, like `example` in the following:

```
>>> from upsetplot import generate_data
>>> example = generate_data(aggregated=True)
>>> example
set0   set1   set2
False  False  False      56
              True      283
       True   False    1279
              True     5882
True   False  False      24
              True       90
       True   False     429
              True     1957
Name: value, dtype: int64
```

Then:

```
>>> from upsetplot import plot
>>> plot(example)
>>> from matplotlib import pyplot
>>> pyplot.show()
```

makes:

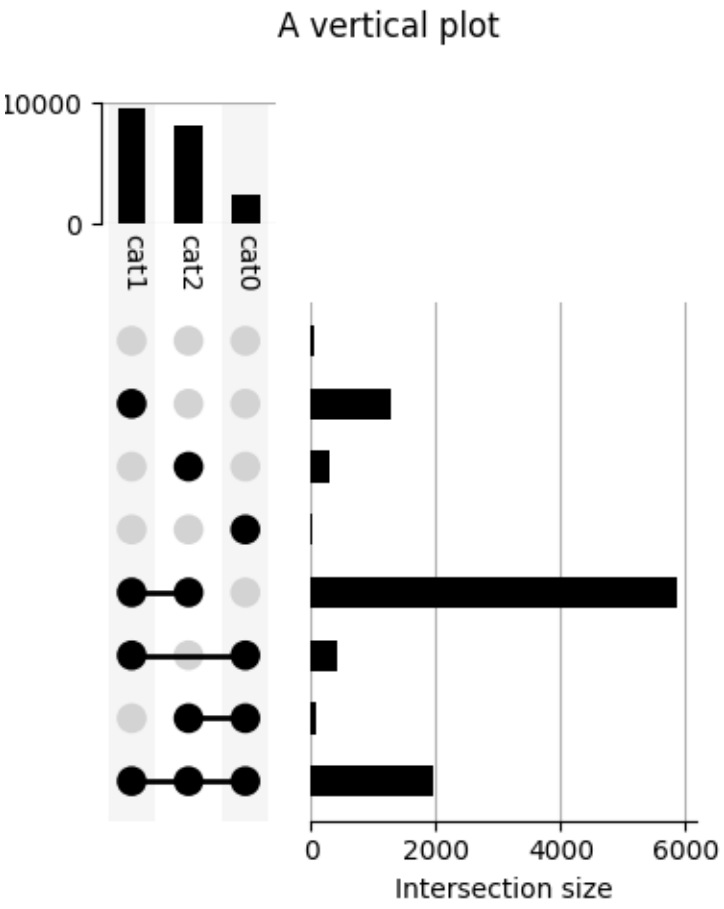


This plot shows the cardinality of every set combination seen in our data. The leftmost column counts items absent from any set. The next three columns count items only in `set1`, `set2` and `set3` respectively, with following columns showing cardinalities for items in each combination of exactly two named sets. The rightmost column counts items in all three sets.

CHAPTER 1

Rotation

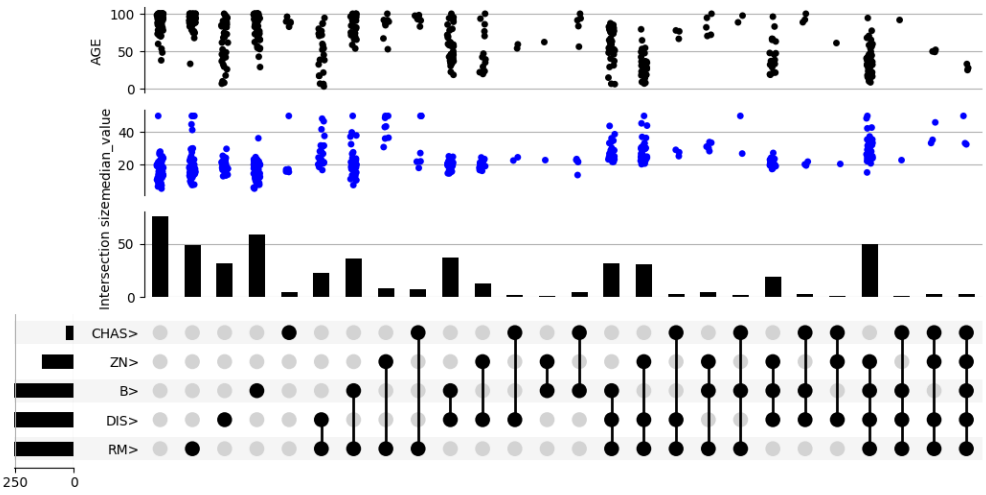
We call the above plot style “horizontal” because the set intersections are presented from left to right. Vertical plots are also supported!



CHAPTER 2

Distributions

Providing a DataFrame rather than a Series as input allows us to expressively plot the distribution of variables in each subset.



CHAPTER 3

Loading datasets

While the dataset above is randomly generated, you can prepare your own dataset for input to upsetplot. A helpful tool is *from_memberships*, which allows us to reconstruct the example above by indicating each data point's set membership:

```
>>> from upsetplot import from_memberships
>>> example = from_memberships(
...     [[],
...      ['set2'],
...      ['set1'],
...      ['set1', 'set2'],
...      ['set0'],
...      ['set0', 'set2'],
...      ['set0', 'set1'],
...      ['set0', 'set1', 'set2'],
...      ],
...      data=[56, 283, 1279, 5882, 24, 90, 429, 1957]
... )
>>> example
set0   set1   set2
False  False  False      56
               True      283
       True   False    1279
               True     5882
True   False  False      24
               True       90
       True   False     429
               True     1957
dtype: int64
```

3.1 Installation

To install the library, you can use `pip`:

```
$ pip install upsetplot
```

Installation requires:

- pandas
- matplotlib >= 2.0
- seaborn to use *UpSet.add_catplot*

It should then be possible to:

```
>>> import upsetplot
```

in Python.

3.2 Why an alternative to py-upset?

Probably for petty reasons. It appeared py-upset was not being maintained. Its input format was undocumented, inefficient and, IMO, inappropriate. It did not facilitate showing plots of each set intersection distribution as in Lex et al's work introducing UpSet plots. Nor did it include the horizontal bar plots illustrated there. It did not support Python 2. I decided it would be easier to construct a cleaner version than to fix it.

3.3 References

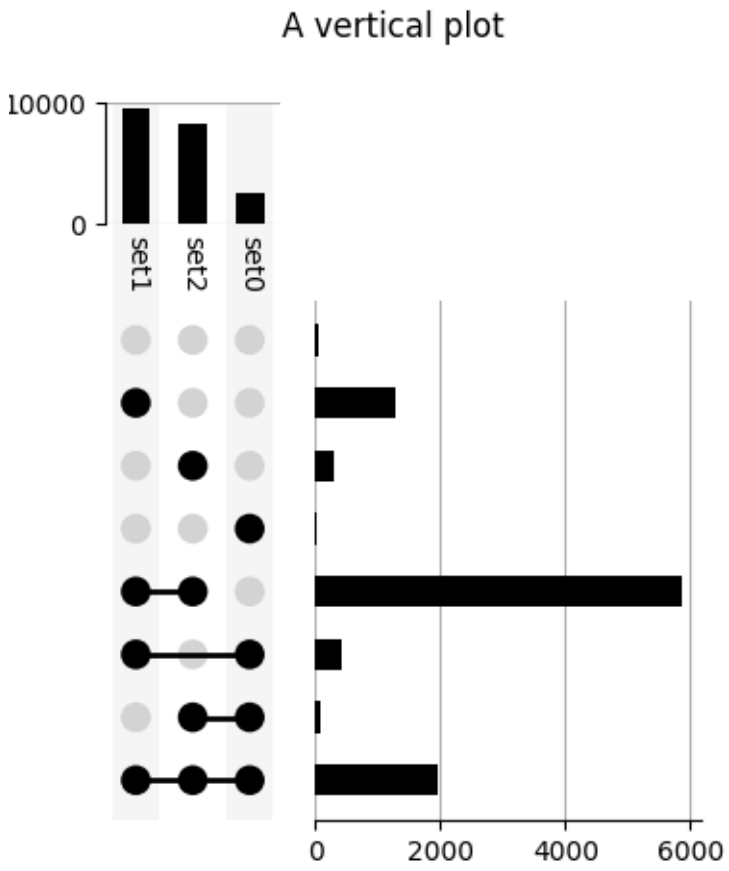
3.3.1 Examples

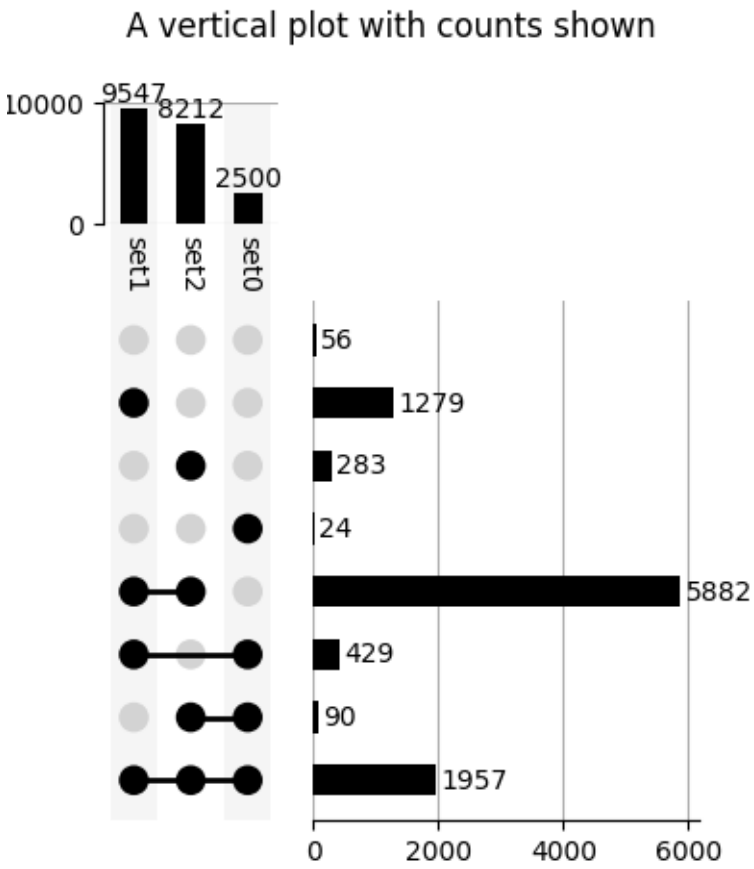
Introductory examples for upsetplot.

Note: Click *here* to download the full example code

Vertical orientation

This illustrates the effect of orientation='vertical'.


A vertical plot
10000
0
set1
set2
set0
0
2000
4000
6000



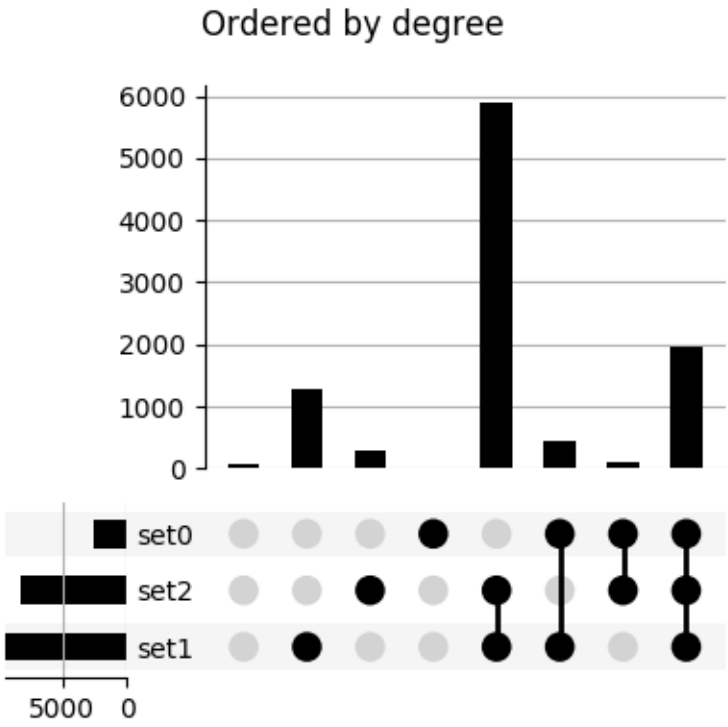

```
from matplotlib import pyplot as plt
from upsetplot import generate_data, plot

example = generate_data(aggregated=True)
plot(example, orientation='vertical')
plt.suptitle('A vertical plot')
plt.show()

plot(example, orientation='vertical', show_counts='%d')
plt.suptitle('A vertical plot with counts shown')
plt.show()
```

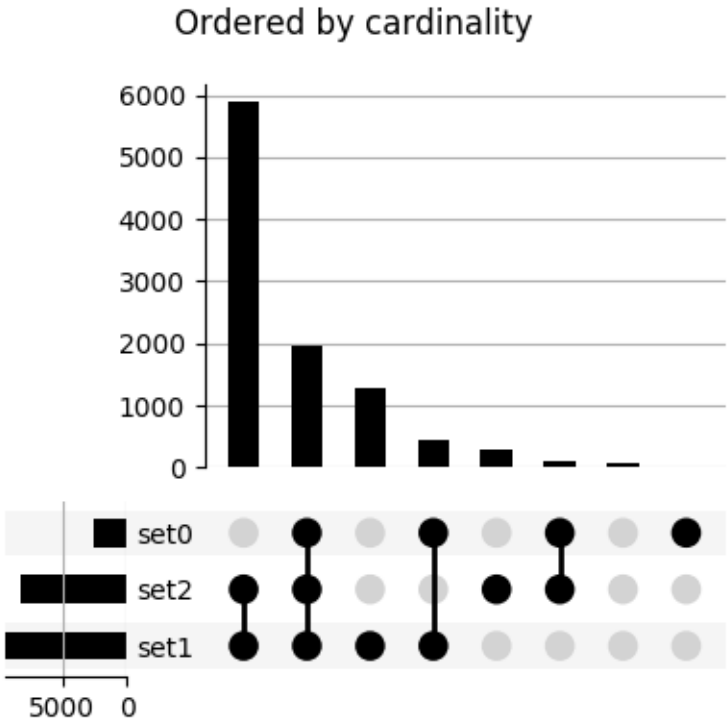
Total running time of the script: (0 minutes 1.049 seconds)

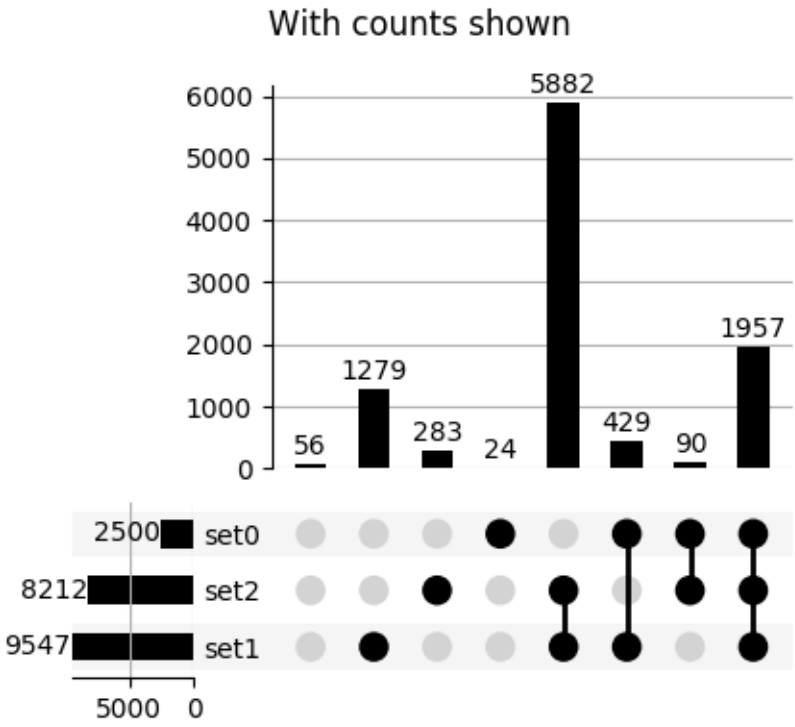
Note: Click *here* to download the full example code

Plotting with generated data

This example illustrates basic plotting functionality using generated data.


Ordered by degree
6000
5000
4000
3000
2000
1000
0
set0
set2
set1
5000
0


Ordered by cardinality
6000
5000
4000
3000
2000
1000
0
set0
set2
set1
5000
0



-

Out:

```
set0   set1   set2
False  False  False      56
                True      283
       True   False    1279
                True     5882
True   False  False      24
                True       90
       True   False     429
                True     1957
Name: value, dtype: int64
```

```
from matplotlib import pyplot as plt
from upsetplot import generate_data, plot

example = generate_data(aggregated=True)
print(example)

plot(example)
plt.suptitle('Ordered by degree')
plt.show()

plot(example, sort_by='cardinality')
plt.suptitle('Ordered by cardinality')
plt.show()
```

(continues on next page)

(continued from previous page)

```
plot(example, show_counts='%d')
plt.suptitle('With counts shown')
plt.show()
```

Total running time of the script: (0 minutes 1.912 seconds)

Note: Click *here* to download the full example code

Above-average features in Boston

Explore above-average neighborhood characteristics in the Boston dataset.

Here we take some features correlated with house price, and look at the distribution of median house price when each of these features is above average.

The most correlated features are:

ZN proportion of residential land zoned for lots over 25,000 sq.ft.

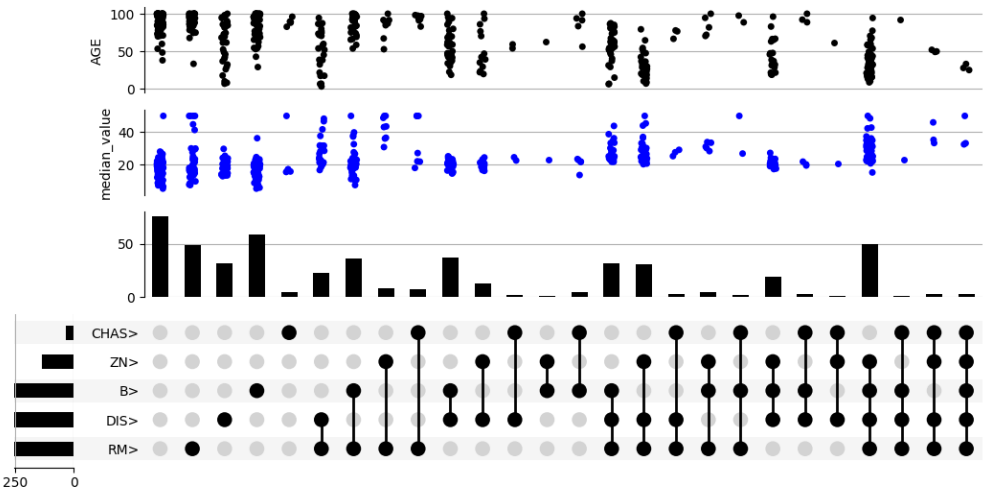
CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

RM average number of rooms per dwelling

DIS weighted distances to five Boston employment centres

B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

This kind of dataset analysis may not be a practical use of UpSet, but helps to illustrate the `UpSet.add_catplot()` feature.




```
import pandas as pd
from sklearn.datasets import load_boston
from matplotlib import pyplot as plt
from upsetplot import UpSet
```

(continues on next page)

(continued from previous page)

```
# Load the dataset into a DataFrame
boston = load_boston()
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)

# Get five features most correlated with median house value
correls = boston_df.corrwith(pd.Series(boston.target),
                             method='spearman').sort_values()
top_features = correls.index[-5:]

# Get a binary indicator of whether each top feature is above average
boston_above_avg = boston_df > boston_df.median(axis=0)
boston_above_avg = boston_above_avg[top_features]
boston_above_avg = boston_above_avg.rename(columns=lambda x: x + '>')

# Make this indicator mask an index of boston_df
boston_df = pd.concat([boston_df, boston_above_avg],
                      axis=1)
boston_df = boston_df.set_index(list(boston_above_avg.columns))

# Also give us access to the target (median house value)
boston_df = boston_df.assign(median_value=boston.target)

# UpSet plot it!
upset = UpSet(boston_df, sum_over=False, intersection_plot_elements=3)
upset.add_catplot(value='median_value', kind='strip', color='blue')
upset.add_catplot(value='AGE', kind='strip', color='black')
upset.plot()
plt.show()
```

Total running time of the script: (0 minutes 2.826 seconds)

3.3.2 API Reference

Plotting

upsetplot.**plot**(*data*, *fig=None*, ***kwargs*)

Make an UpSet plot of data on fig

Parameters

data [pandas.Series or pandas.DataFrame] Values for each set to plot. Should have multi-index where each level is binary, corresponding to set membership. If a DataFrame, `sum_over` must be a string or False.

fig [matplotlib.figure.Figure, optional] Defaults to a new figure.

kwargs Other arguments for *UpSet*

Returns

subplots [dict of matplotlib.axes.Axes] Keys are 'matrix', 'intersections', 'totals', 'shading'

class upsetplot.**UpSet**(*data*, *orientation='horizontal'*, *sort_by='degree'*, *sort_sets_by='cardinality'*, *sum_over=None*, *facecolor='black'*, *with_lines=True*, *element_size=32*, *intersection_plot_elements=6*, *totals_plot_elements=2*, *show_counts=''*)

Manage the data and drawing for a basic UpSet plot

Primary public method is *plot()*.

Parameters

data [pandas.Series or pandas.DataFrame] Values for each set to plot. Should have multi-index where each level is binary, corresponding to set membership. If a DataFrame, `sum_over` must be a string or False.

orientation [{'horizontal' (default), 'vertical'}] If horizontal, intersections are listed from left to right.

sort_by [{'cardinality', 'degree'}] If 'cardinality', set intersections are listed from largest to smallest value. If 'degree', they are listed in order of the number of sets intersected.

sort_sets_by [{'cardinality', None}] Whether to sort the overall sets by total cardinality, or leave them in the provided order.

sum_over [str, False or None (default)] Must be specified when `data` is a DataFrame. If False, the intersection plot will show the count of each subset. Otherwise, it shows the sum of the specified field.

facecolor [str] Color for bar charts and dots.

with_lines [bool] Whether to show lines joining dots in the matrix, to mark multiple sets being intersected.

element_size [float or None] Side length in pt. If None, size is estimated to fit figure

intersection_plot_elements [int] The intersections plot should be large enough to fit this many matrix elements.

totals_plot_elements [int] The totals plot should be large enough to fit this many matrix elements.

show_counts [bool or str, default=False] Whether to label the intersection size bars with the cardinality of the intersection. When a string, this formats the number. For example, '%d' is equivalent to True.

Methods

add_catplot(self, kind[, value, elements])	Add a seaborn catplot over subsets when *plot()* is called.
make_grid(self[, fig])	Get a SubplotSpec for each Axes, accounting for label text width
plot(self[, fig])	Draw all parts of the plot onto fig or a new figure
plot_intersections(self, ax)	Plot bars indicating intersection size
plot_matrix(self, ax)	Plot the matrix of intersection indicators onto ax
plot_totals(self, ax)	Plot bars indicating total set size

plot_shading

add_catplot(*self*, *kind*, *value=None*, *elements=3*, ***kw*)

Add a seaborn catplot over subsets when *plot()* is called.

Parameters

kind [str] One of {"point", "bar", "strip", "swarm", "box", "violin", "boxen"}

value [str, optional] Column name for the value to plot (i.e. y if orientation='horizontal'), required if `data` is a DataFrame.

elements [int, default=3] Size of the axes counted in number of matrix elements.

****kw** [dict] Additional keywords to pass to `seaborn.catplot()`.

Our implementation automatically determines 'ax', 'data', 'x', 'y' and 'orient', so these are prohibited keys in `kw`.

Returns

None

make_grid(*self*, *fig=None*)
Get a SubplotSpec for each Axes, accounting for label text width

plot(*self*, *fig=None*)
Draw all parts of the plot onto fig or a new figure

Parameters

fig [matplotlib.figure.Figure, optional] Defaults to a new figure.

Returns

subplots [dict of matplotlib.axes.Axes] Keys are 'matrix', 'intersections', 'totals', 'shading'

plot_intersections(*self*, *ax*)
Plot bars indicating intersection size

plot_matrix(*self*, *ax*)
Plot the matrix of intersection indicators onto ax

plot_totals(*self*, *ax*)
Plot bars indicating total set size

Dataset loading and generation

upsetplot.**from_memberships**(*memberships*, *data=None*)
Load data where each sample has a collection of set names

The output should be suitable for passing to *UpSet* or *plot*.

Parameters

memberships [sequence of collections of strings] Each element corresponds to a data point, indicating the sets it is a member of. Each set is named by a string.

data [Series-like or DataFrame-like, optional] If given, the index of set memberships is attached to this data. It must have the same length as `memberships`. If not given, the series will contain the value 1.

Returns

DataFrame or Series `data` is returned with its index indicating set membership. It will be a Series if `data` is a Series or 1d numeric array. The index will have levels ordered by set names.

Examples

```
>>> from upsetplot import from_memberships
>>> from_memberships([
...     ['set1', 'set3'],
...     ['set2', 'set3'],
...     ['set1'],
...     []
... ])  # doctest: +ELLIPSIS, +NORMALIZE_WHITESPACE
set1   set2   set3
True   False  True     1
False  True   True     1
True   False  False    1
False  False  False    1
Name: ones, dtype: ...
>>> # now with data:
>>> import numpy as np
>>> from_memberships([
...     ['set1', 'set3'],
...     ['set2', 'set3'],
...     ['set1'],
...     []
... ], data=np.arange(12).reshape(4, 3))  # doctest: +NORMALIZE_WHITESPACE
                  0   1   2
set1  set2  set3
True  False True   0   1   2
False True  True   3   4   5
True  False False  6   7   8
False False False  9  10  11
```

upsetplot.**generate_data**(*seed=0*, *n_samples=10000*, *n_sets=3*, *aggregated=False*)

3.3.3 Changelog

What's new in version 0.2.1

- Return a Series (not a DataFrame) from *from_memberships* if data is 1-dimensional.

What's new in version 0.2

- Added *from_memberships* to allow a more convenient data input format.
- *plot* and *UpSet* now accept a `pandas.DataFrame` as input, if the `sum_over` parameter is also given.
- Added an *add_catplot* method to *UpSet* which adds Seaborn plots of set intersection data to show more than just set size or total.
- Shading of subset matrix is continued through to totals.
- Added a `show_counts` option to show counts at the ends of bar plots. (#5)
- Defined `_repr_html_` so that an *UpSet* object will render in Jupyter notebooks. (#36)
- Fix a bug where an error was raised if an input set was empty.

Bibliography

[Lex2014] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, Hanspeter Pfister, *UpSet: Visualization of Intersecting Sets*, IEEE Transactions on Visualization and Computer Graphics (InfoVis '14), vol. 20, no. 12, pp. 1983–1992, 2014. doi: doi.org/10.1109/TVCG.2014.2346248

Index