Cloudy Mountain Plot

Release 0.9.3

Mar 03, 2022

Contents:

A categorical data visualization inspired by Violin, Bean and Pirate Plots.

CHAPTER 1

Introduction

A Cloudy Mountain Plot is an informative RDI[1] categorical distribution plot inspired by Violin, Bean and Pirate Plots.

- Like Violin plots [Hintze_Nelson_1998], it shows smoothed kernel density curves, revealing information which would be hidden in boxplots, for example presence of multiple "peaks" ("modes") in the distribution "mountain".
- Like Bean plots [Kampstra_2008], it shows the raw data, drawn as a cloud of points. By default all data points are shown but you can optionally control this and limit the display to a subset of the data.
- Like Pirate plots [Phillips_2017], it marks confidence intervals (either from Student's T or as Bayesian Highest Density Intervals or as interquantile ranges) for the probable position of the true population mean.

Since by default it does not symmetrically mirror the density curves, it allows immediate comparisions of distributions side-by-side.

The present documentation introduces both what cloudy mountain plots are and how to create them, using a plotting function (`cmplot`) which has been coded in both Julia and Python, built on top of the freely available *Plotly* graphic library.

1.1 Elements of the plot

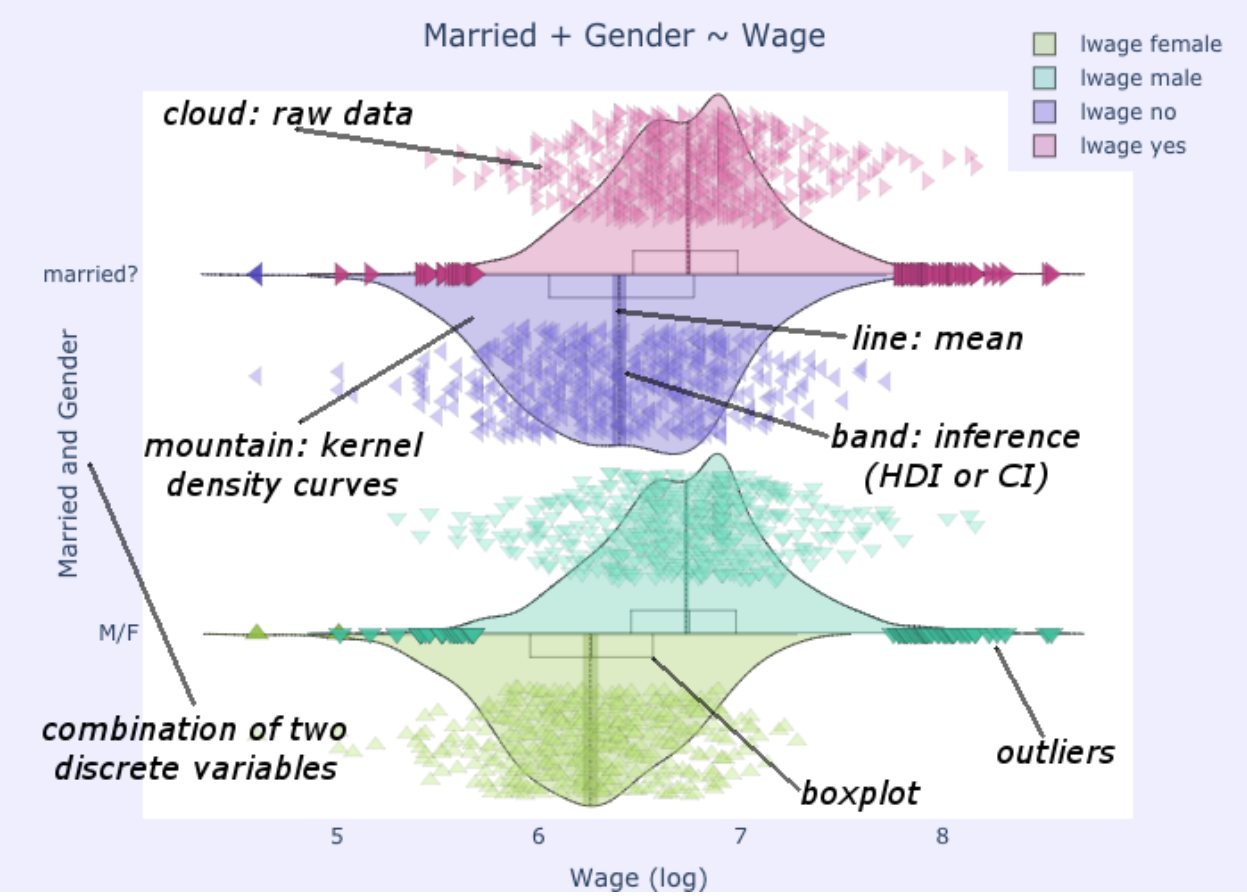
(Note: check the *Interactive example* to see how the following figure actually looks like when you create it, with the full interactive power of plotly)

cloud Marker symbols show the number and location of the raw data points. They are shown jittered for clarity. It is possible to fully control both the aspect (`opacity` and `shapes`) of the markers and their `number` (in case showing them all would prove too slow or unelegant). It is also possible `not to show` any point. For clarity, by default the points are plotted on the opposite side of the kernel density curve. They can alternatively be plotted `over the density curve`, as in the above image.

mountain Kernel density estimation curve.

line Indicates the mean of the distribution

[1] RDI: Raw data + Descriptive statistics + Inferential statistics


Married + Gender ~ Wage
lwage female
lwage male
lwage no
lwage yes
cloud: raw data
married?
line: mean
mountain: kernel
density curves
band: inference
(HDI or CI)
Married and Gender
M/F
combination of two
discrete variables
outliers
boxplot
5
6
7
8
Wage (log)

band Probable position of the true population mean, to desired level of confidence. Method used can be *specified* as either CI[2] , HDI[3] or IQR[4]. It is also possible not to show the band.

boxplot A small boxplot. It can be *shown or hidden*, as desired.

outliers The outliers are marked without jitter, on the baseline, and with less transparency. It is of course possible to choose *whether to show* the outliers.

[2] CI: Confidence Interval, from Student's T distribution
[3] HDI: Bayesian Highest Density Intervals
[4] IQR: Interquartile range

CHAPTER 2

Interactive example

(The following example only works in html documentation, as it requires plotly javascript for the interactivity)

CHAPTER 3

Download and installation

`cmplot` has no platform-specific dependencies and should thus work on all platforms.

3.1 Python instructions

The latest version of cmplot can be installed by typing either:

```
pip3 install --upgrade cmplot
```

(from Python Package Index)

or:

```
pip3 install git+git://github.com/g-insana/cmplot.py.git
```

(from GitHub).

3.2 Julia instructions

```
julia> import Pkg; Pkg.add("CMPlot")
```

(from official Julia registry)

or:

```
julia> ]
(v1.1) pkg> dev https://github.com/g-insana/CMPlot.jl.git
```

(from CMPlot GitHub).

CHAPTER 4

Quickstart

All you need is a dataframe with your data[1] and a single call to the plotting function, as here detailed.

(See Dataframe for few lines of code to construct your dataframe in case you don't have one to start with)

- In Python:

```
>>> import plotly.graph_objects as go
>>> from cmplot import cmplot

 #call the cmplot directly inside a plotly Figure function as:

>>> go.Figure(*cmplot(mydataframe,xcol="xsymbol"))

 #alternatively get traces and layout as separate variables, so that you can modify␣
↪them or combine with others before passing them to Figure() function:

>>> (traces,layout)=(cmplot(mydataframe,xcol="xsymbol"))

 #[...] do something with traces/layout

>>> go.Figure(traces,layout) #plot it
```

- In Julia:

```
julia> using CMPlot

julia> using PlotlyJS

#call the cmplot directly inside a plotly Figure function as:

julia> plot(cmplot(mydataframe,xcol=:xsymbol)...)
```

(continues on next page)

[1] a dataframe with at least one column holding a categorical independent variable - we'll refer to it as xcol - and at least one column holding a continuous dependent variable - which we'll term ycol

(continued from previous page)

```
#alternatively get traces and layout as separate variables, so that you can modify␣
↪them or combine with others before passing them to Figure() function:

julia> traces,layout=cmplot(mydataframe,xcol=:xsymbol)

julia> # [...] do something with traces/layout

julia> plot(traces,layout) # plot it
```

CHAPTER 5

Tutorial

Example usages of CMPlot, with plots and options used are presented below (e.g. *Heights in families*).

An introduction to the several features is also offered by the provided jupyter notebook. This can be previewed via nbviewer.jupyter.org, or directly edited and run locally.

5.1 Hands-on

In fact probably the best way to try out the several features of cmplot is to clone from github, or to directly `download the Julia` or the `Python jupyter notebook` and play with it, tweaking options, trying different data combinations and expecially using your own data or other publicly available datasets.

The notebook examples use the datasets from Iris [Anderson_Edgar_Fisher_1935_1936] and Wages [Cornwell_Rupert_1988] but it is immediate to switch to other datasets via the included calls to RDatasets - for Julia - and PyDatasets - for Python).

5.2 Heights in families

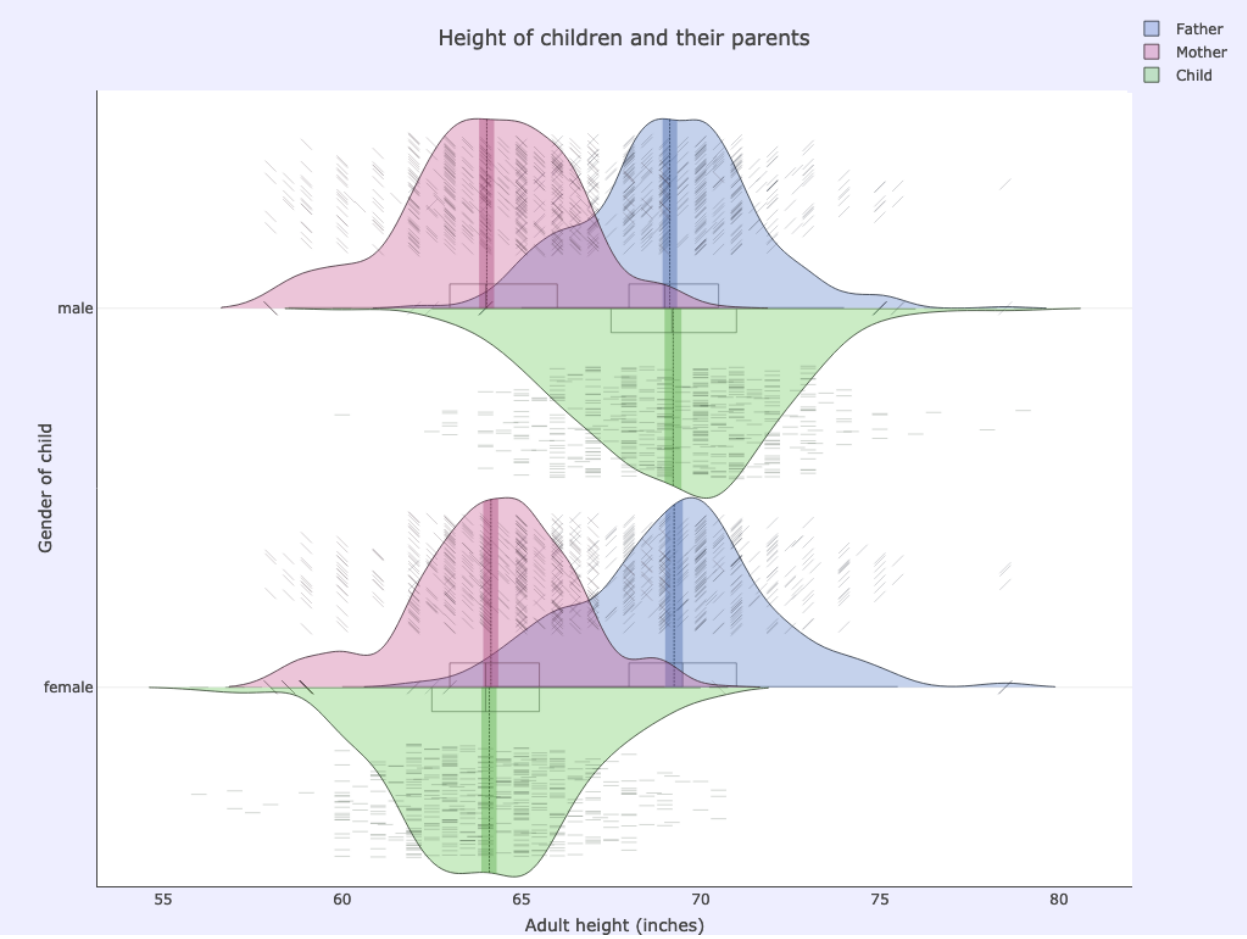
For this example we'll use the dataset from Galton on the heights of parents and their children [Galton_1886].

His data consists of the heights of 930 adult children and of their respective parentages, for a total of 205 families.

```
>>> import plotly.graph_objects as go
>>> from cmplot import cmplot
>>> from pydataset import data
>>> df = data("GaltonFamilies")
```

With a simple initial plot we can see that all males (sons and all fathers) occupy a very similar range of heights and all females (daughters and all mothers) likewise, and with very similar distribution:

```
>>> cmplot(df, xcol="gender", ycol=["father", "childHeight", "mother"],␣
↪pointsoverdens=True, pointshapes=["line-nw", "line-ne", "line-ew"])
```


Height of children and their parents
Father
Mother
Child
male
female
Gender of child
55
60
65
70
75
80
Adult height (inches)

- xcol: gender
- ycol: father, childHeight, mother
- Options used: *pointsoverdens* = True, *pointshapes* = ["line-nw", "line-ne", "line-ew"]

Sometimes a boxplot or even a violinplot can be less informative or harder to decode than a simple histogram as shown in the top part of the next picture:

Furthermore the boxplot also suffers from a very well known drawback: it hides any multimodal distribution (the two *peaks* - modes - at 5 and 8 number of children per family; as seen in the bottom half of the picture).

Showing the raw data alongside the kernel density curve helps in seeing things more clearly:

- ycol: childNum, children
- Options used: *pointsoverdens* = True, *spanmode* = 'hard'

After binning the total number of siblings in a family and the birth order (which child is born earlier or later), we can further explore the data.

For example plotting child height vs family size we can see a lower average height for children of very numerous families (with 9 or more siblings):

- xcol: childrenBinned
- ycol: childHeight
- Options used: *pointsoverdens* = True, *side* = 'pos', *ycolorgroups* = False

Being born late in the family was apparently even less conducive to stature: there is much lower average height for late born children compared to earlier born ones *(Note that binning on birth order in this dataset needs to be adjusted to factor for gender due to how the birth order was originally recorded)*:

- xcol: childNumBinned
- ycol: childHeight
- Options used: *pointsoverdens* = True, *side* = 'pos', *ycolorgroups* = False

Cloudy Mountain Plots make it is easy to separate according to combinations of two or more categorical variables. For example we can plot children's height according to both their gender and their birth order:

```
>>> cmplot(df, xcol=["childNumBinned", "gender"], ycol="childHeight",␣
↪ycolorgroups=False, xsuperimposed=True)
```

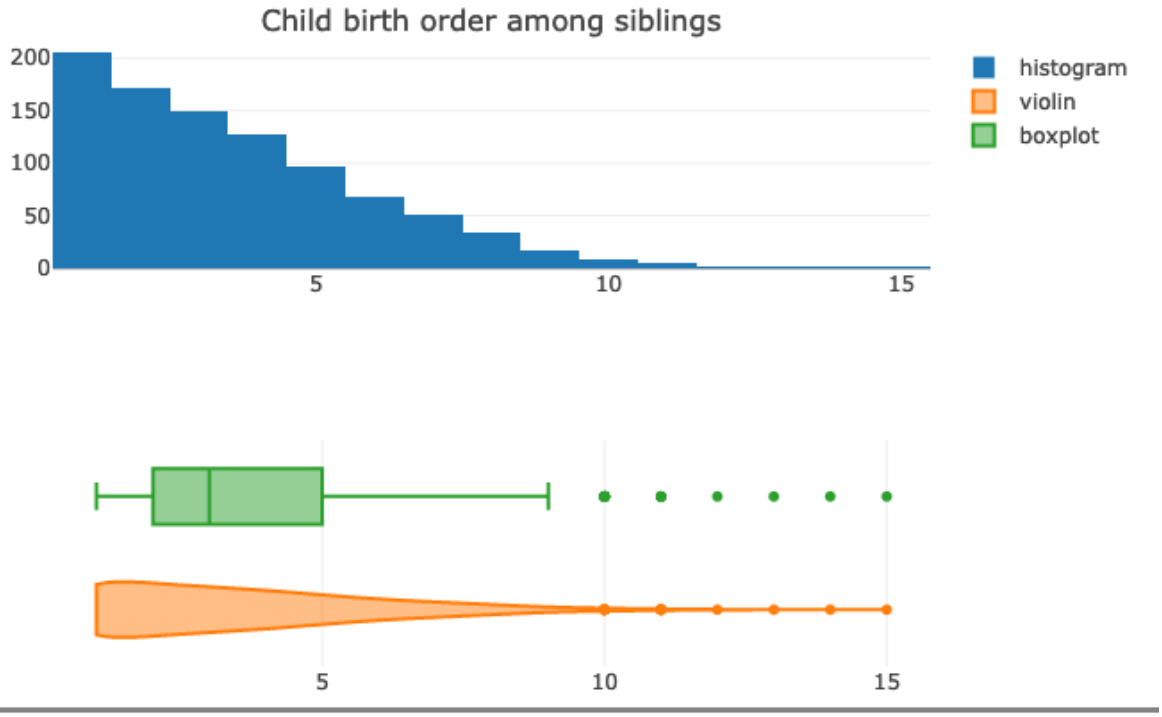
- xcol: childNumBinned, gender
- ycol: childHeight
- Options used: *side* = 'pos', *ycolorgroups* = False, *xsuperimposed* = True,

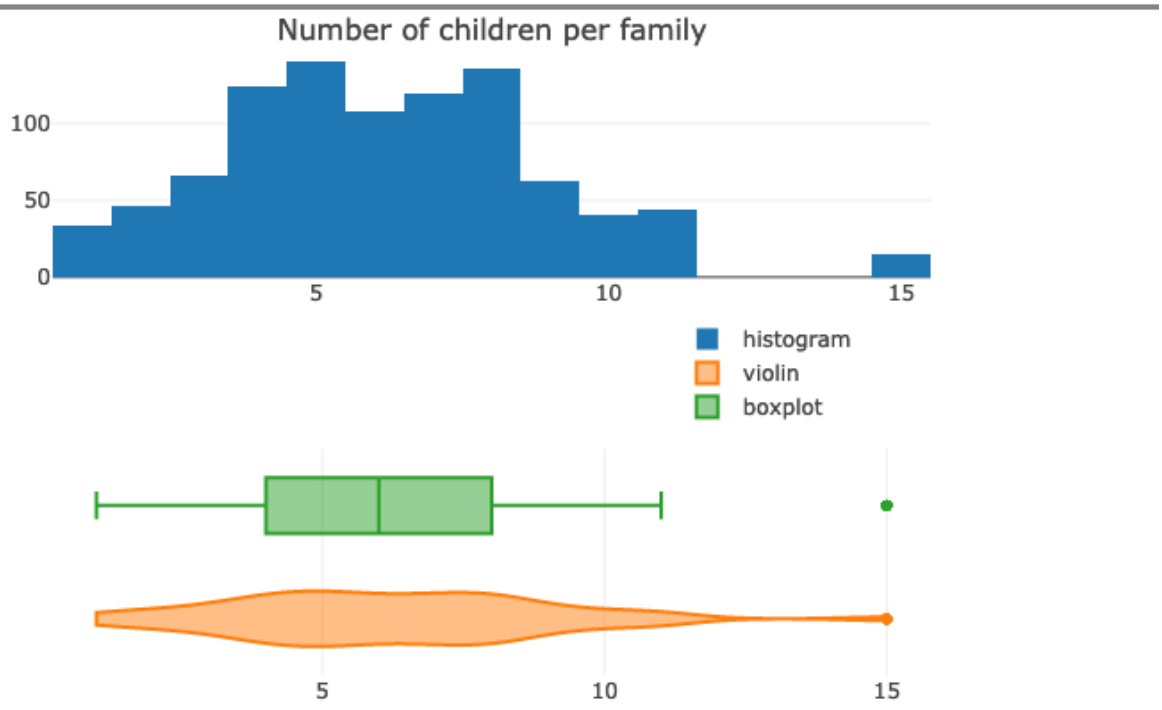
The gender component is obviously the dominant one, but still the plot makes very obvious that children born later than their siblings are on average smaller than those born earlier.

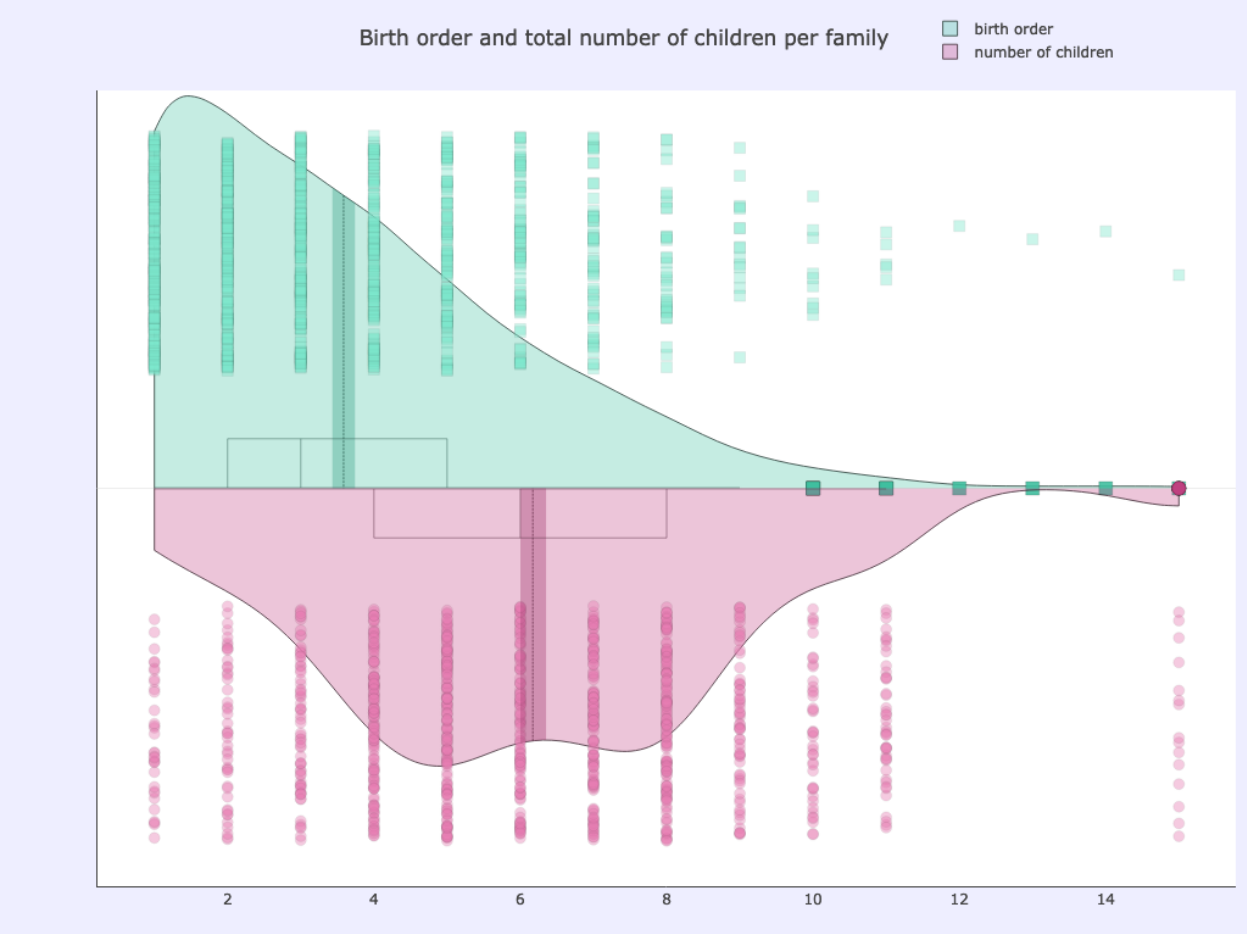
We can also plot the height of parents and of their children according to birth order among siblings.

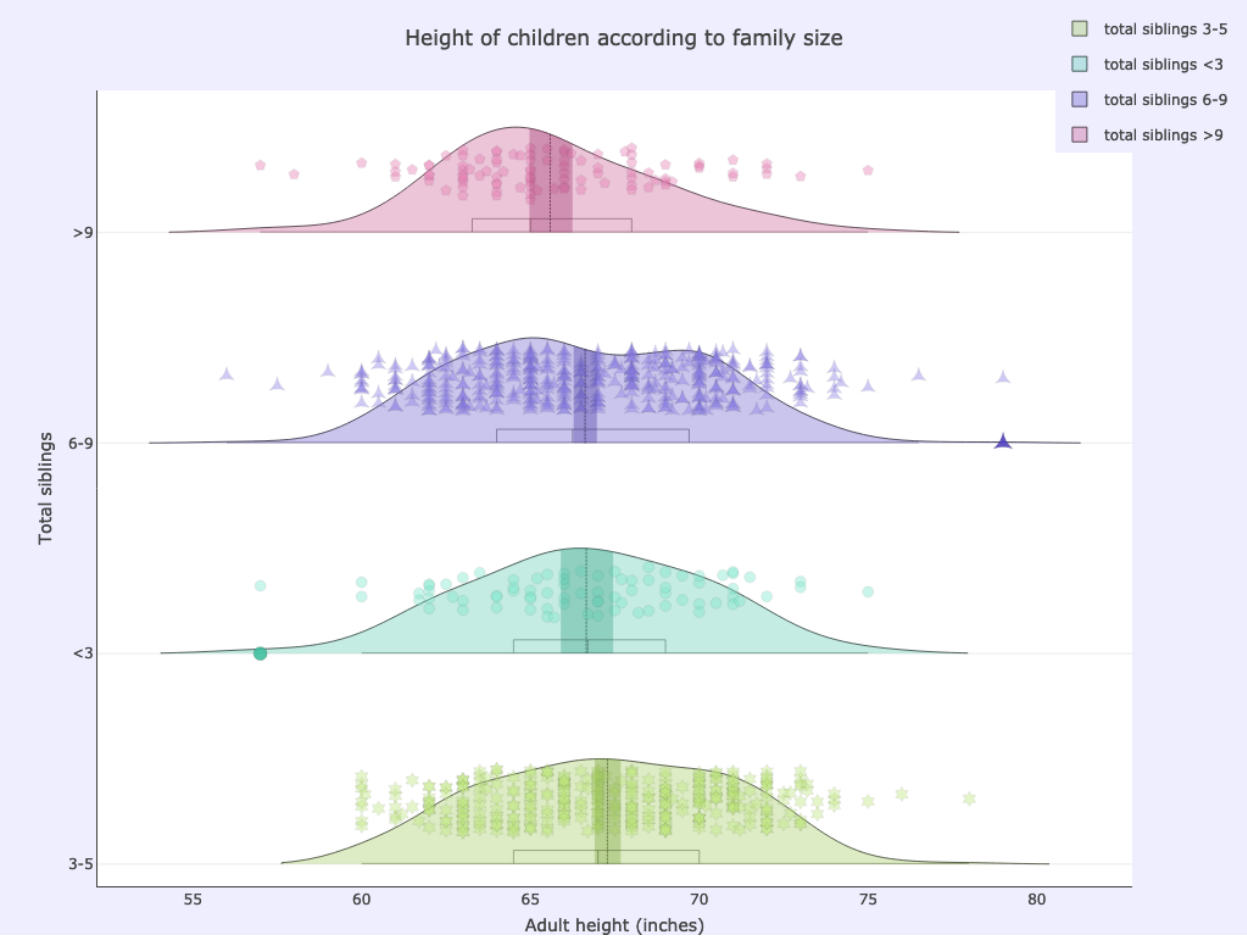
- xcol: childNumBinned
- ycol: father, childHeight, mother
- Options used: *pointsoverdens* = True, *pointshapes* = ["line-nw", "line-ne", "line-ew"]

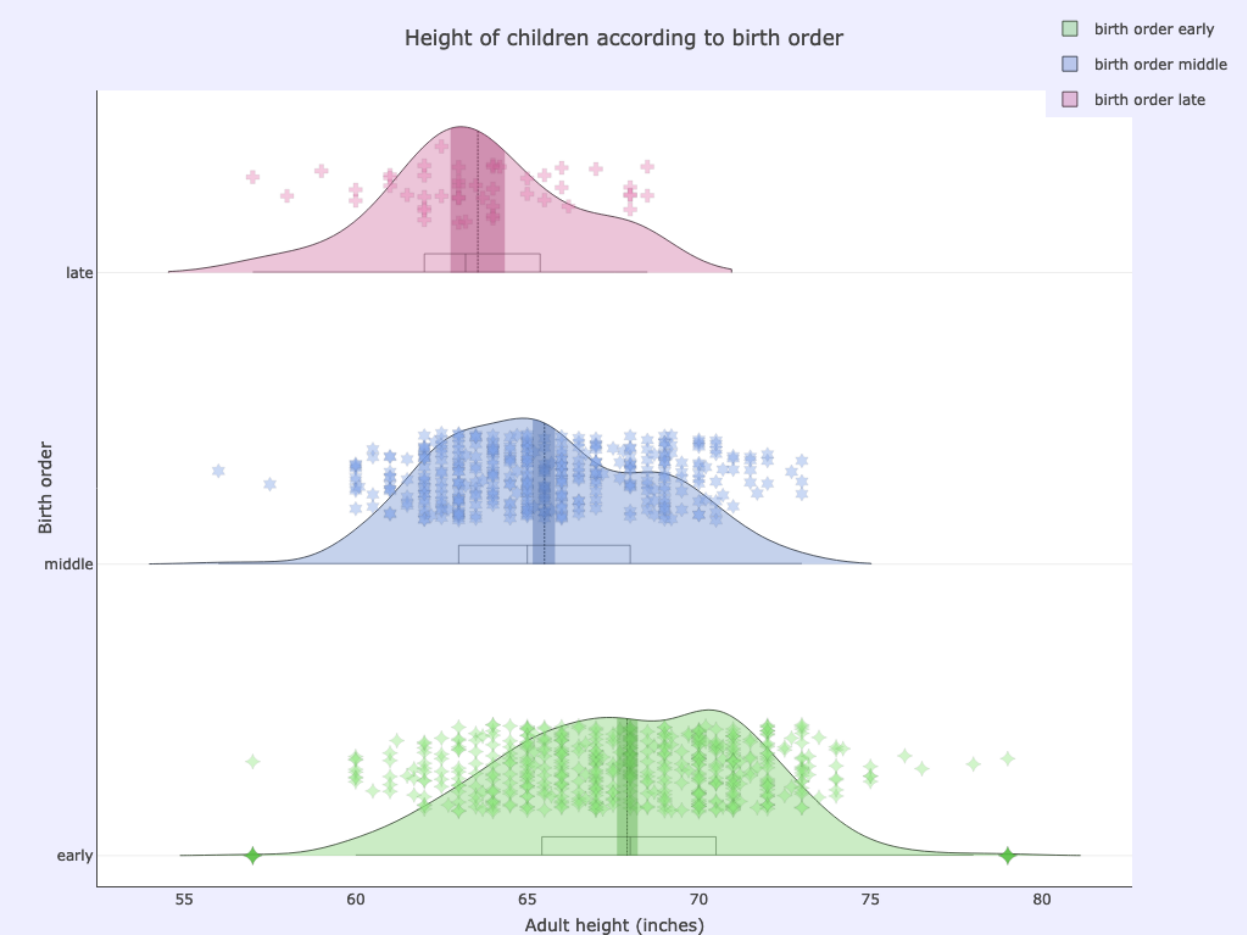
Separating by gender as well as birth order shows that, for the late born, sons are on average shorter than their fathers and daughters shorter than their mothers; for early born ones the situation is reversed, with - on average - sons taller than fathers and daughters taller than mothers:


Child birth order among siblings
200
150
100
50
0
5
10
15
histogram
violin
boxplot
5
10
15

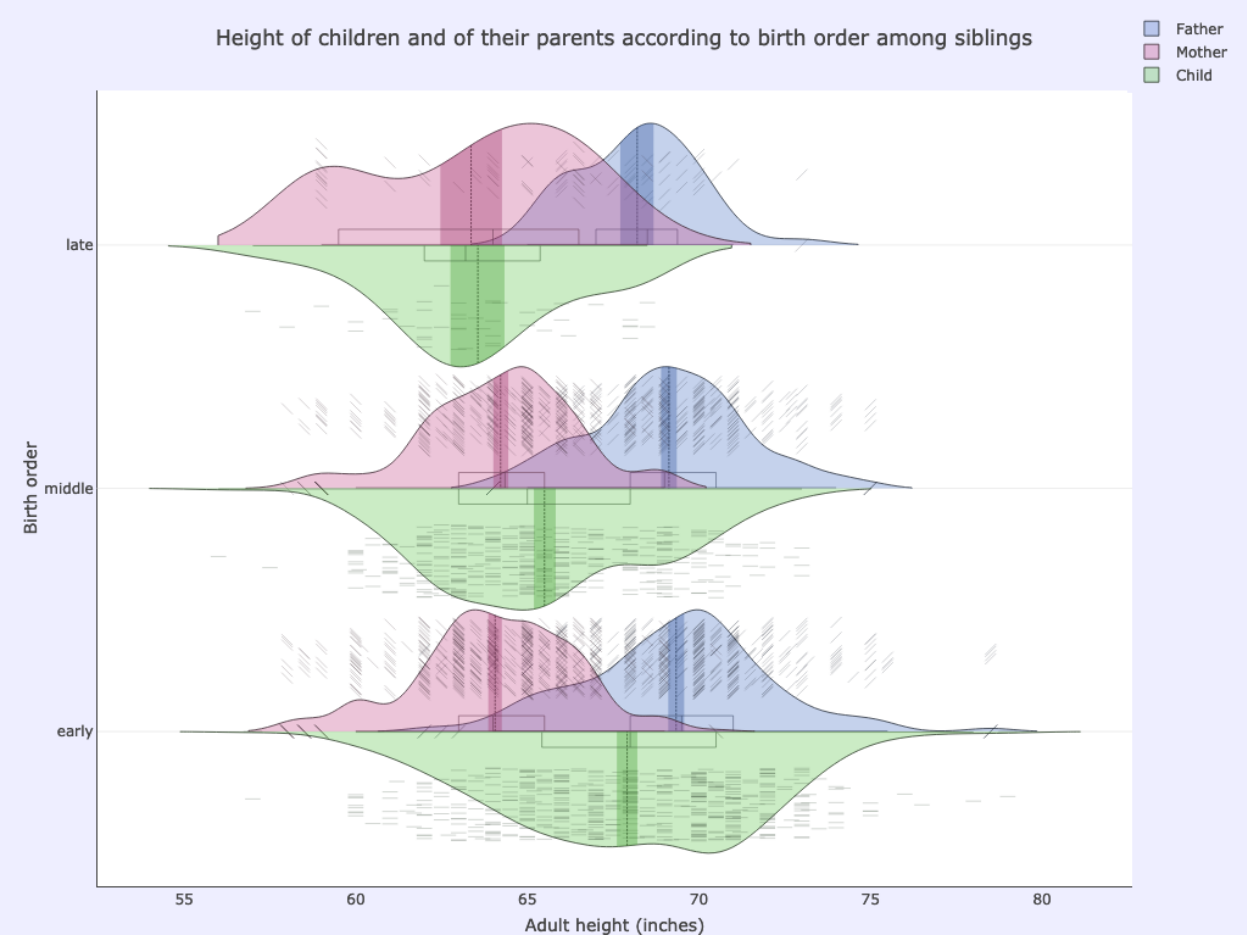

Number of children per family
100
50
0
5
10
15
histogram
violin
boxplot
5
10
15


Birth order and total number of children per family
birth order
number of children
2
4
6
8
10
12
14


Height of children according to family size
total siblings 3-5
total siblings <3
total siblings 6-9
total siblings >9
Total siblings
>9
6-9
<3
3-5
55
60
65
70
75
80
Adult height (inches)


Height of children according to birth order
birth order early
birth order middle
birth order late
Birth order
late
middle
early
55
60
65
70
75
80
Adult height (inches)


Height of children according to their gender and birth order
ChildHeight early&male
ChildHeight early&female
ChildHeight middle&female
ChildHeight middle&male
ChildHeight late&female
ChildHeight late&male
late
middle
early
Gender & Birth order
55
60
65
70
75
80
Adult height (inches)

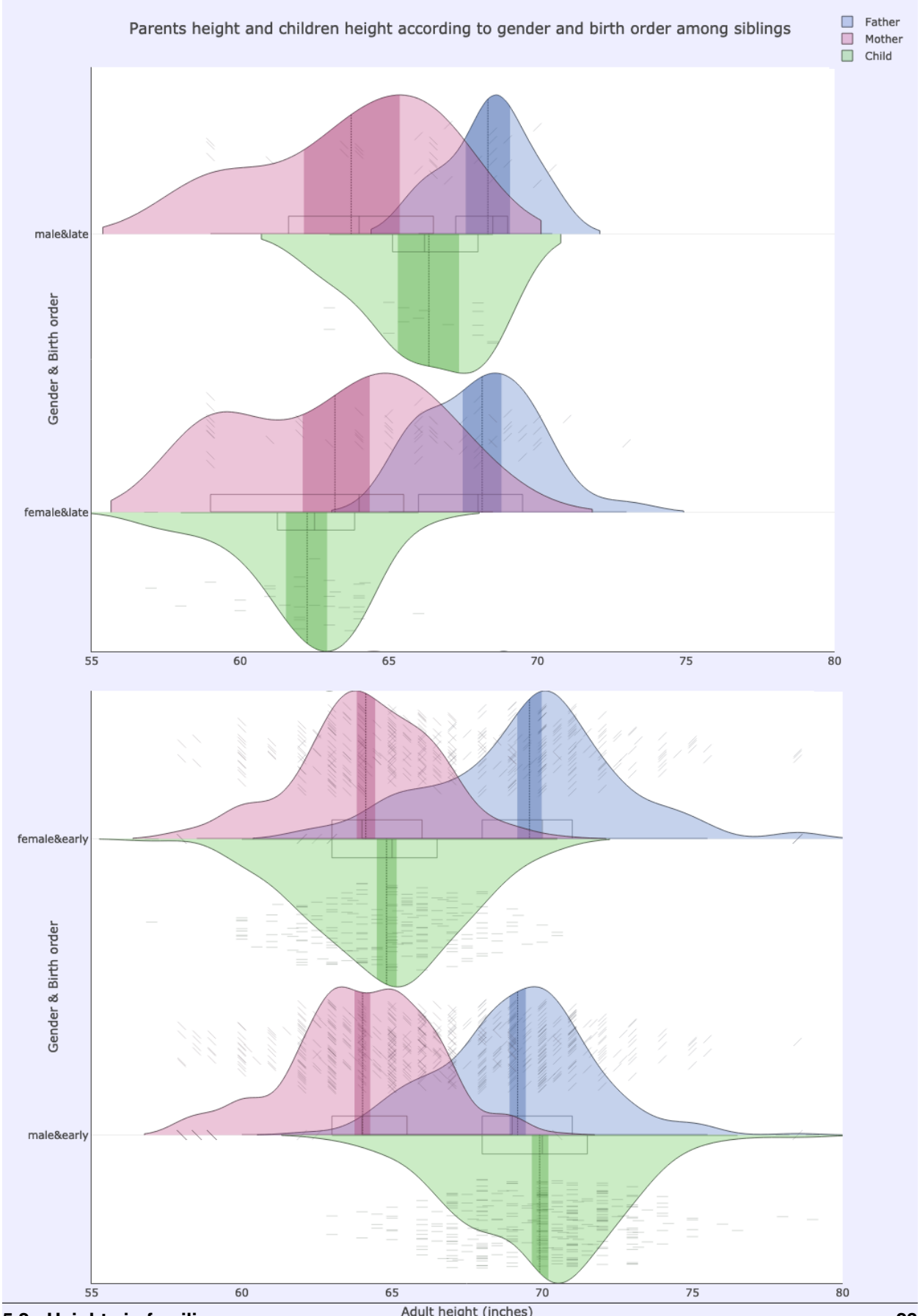

Height of children and of their parents according to birth order among siblings
Father
Mother
Child
Birth order
late
middle
early
55
60
65
70
75
80
Adult height (inches)

```
>>> cmplot(df, xcol=["childNumBinned", "gender"], ycol=["childHeight", "mother"],␣
↪ycolorgroups=False, xsuperimposed=True)
```

- xcol: childNumBinned, gender
- ycol: father, childHeight, mother
- Options used: *pointsoverdens* = True, *pointshapes* = ["line-nw", "line-ne", "line-ew"]

Note how the plotted clouds of raw data points caution us that there is much less data (much sparser data point clouds) for the late born, as these are the children belonging to very numerous families, which are less abundant than the smaller families.

Still the trend is there and the plot is informative and helps us to dig into the data and to make relevant information surface.


Parents height and children height according to gender and birth order among siblings
Father
Mother
Child
male&late
female&late
female&early
male&early
Gender & Birth order
55
60
65
70
75
80
Adult height (inches)

CHAPTER 6

Options

The only mandatory arguments for cmplot are a dataframe containing the data and either a string or a list of strings which label the columns containing the discrete independent variables in the dataframe, as shown in the *Quickstart* section.

Several additional optional arguments can be specified to customize the result, both in terms of content and of form.

xcol
: a string or an array of strings, column name(s) of the dataframe that you wish to plot as "x".

  This should be the categorical independent variable. If more than one column name is given, the combination of these will be used as "x". See examples for interpretation. e.g. xcol="Species"

ycol
: a string or an array of strings, column name(s) of the dataframe that you wish to plot as "y". Optional.

  These should be the continuous dependent variables. If ycol is not specified, then the function will plot all the columns of the dataframe except those specified in xcol.

  e.g. ycol=["Sepal.Length","Sepal.Width"] would plot sepals' length and width as a function of the flower species

orientation
: 'h' | 'v', default is 'h'

  Orientation of the plot (horizontal or vertical)

xsuperimposed
: boolean, default is False

  The default behaviour is to plot each value of the categorical variable (or each combination of values for multiple categorical variables) in a separate position. Set to True to superimpose the plots. This is useful in combination with "side='alt'" to create asymmetrical plots and comparing combinations of categorical variables (e.g. Married + Gender ~ Wage).

xlabel
: string or list of strings

Override for labelling (and placing) the plots of the categorical variables. Only relevant when using xsuperimposed

ylabel
string or list of strings

Override for labelling the dependent variables. If not specified, the labels for the dataframe ycol are used.

title
string

If not specified, the plot title will be automatically created from the names of the variables plotted.

e.g. title="Length of petals for the three species"

side
'pos' | 'neg' | 'both' | 'alt', default is 'alt'

'pos' would create kernel density curves rising towards the positive end of the axis, 'neg' towards the negative, 'both' creates symmetric curves (like violin/bean/pirate plots). 'alt' will alternate between 'pos' and 'neg' in case where multiple ycol are plotted.

e.g. side='both'

altsidesflip
boolean, default is False

Set to True to flip the order of alternation between sides for the kernel density curves. Only relevant when side='alt'

ycolorgroups
boolean, default is True

Set to False to have the function assign a separate colour when plotting different values of the categorical variable. Leave as True if all should be coloured the same.

spanmode
'soft' | 'hard', default is 'soft'

Controls the rounding of the kernel density curves or their sharp drop at their extremities. With 'hard' the span goes from the sample's minimum to its maximum value and no further.

pointsoverdens
boolean, default is False

Set to True to plot the raw data points over the kernel density curves. This is obviously the case when side='both', but otherwise by default points are plotted on the opposite side.

showpoints
boolean, default is True

Set to False to avoid plotting the *cloud* of data points

pointsopacity
float, range 0-1, default is 0.4

The default is to plot the data points at 40% opacity. 1 would make points completely opaque and 0 completely transparent (in that case you'd be better served by setting showpoints to False).

inf
'hdi' | 'ci' | 'iqr' | 'none', default is 'hdi'

To select the method to use for calculating the confidence interval for the inference *band* around the mean. 'hdi' for Bayesian Highest Density Interval, 'ci' for Confidence Interval based on Student's T, 'iqr' for Inter Quantile Range. Use 'none' to avoid plotting the inference band.

conf_level
float, range 0-1, default is 0.95

Confidence level to use when inf='ci', credible mass for inf='hdi'

hdi_iter
integer, default is 10000

Iterations to use when performing Bayesian t-test when inf='hdi'

showboxplot
boolean, default is True

Set to False to avoid displaying the mini *boxplot*

markoutliers
boolean, default is True

Set to False to avoid marking the *outliers*

pointshapes
array of strings

You can specify manually which symbols to use for each distribution plotted. If not specified, a random symbol is chosen for each distribution.

pointsdistance
float, range 0-1, default is 0.6

Distance at which data points will be plotted, measured from the base of the density curve. 0 is at the base, 1 is at the top.

pointsmaxdisplayed
integer, default is 0

This option sets the maximum number of points to be drawn on the graph. The default value '0' corresponds to no limit (plot all points). This option can be useful when the data amount is massive and would prove inefficient or inelegant to plot.

colorrange
integer, default is None

By default, the distribution will be coloured independently, with the colours automatically chosen as needed for a single plot, maximising the difference in hue across the colour spectrum. You can override this by specifying a number to accomodate. This is useful when joining different plots together. E.g. if the total number of colours to be accomodating, after joining two plots, would equal 4, then set colorrange=4

colorshift
integer, default is 0

This option is used in combination with colorrange to skip a certain amount of colours when they are to be assigned to the distributions to be plotted. This is useful when joining different plots together, to avoid having distributions plotted with the same colour.

CHAPTER 7

Dataframe

Cloudy mountain plots will be created from data contained in a dataframe.

What if you don't have a dataframe to start with?

Simply create one from scratch with few lines of code:

- In Python:

```
>>> import pandas #load the module

>>> mydata={} #create an empty hash

>>> mydata['Species']=["setosa", "setosa", .... "virginica"] #load data for␣
↪categorical variable

>>> mydata['PetalWidth']=[0.2, 0.2, .... 1.8] #load data for dependent variable(s)

>>> df=pandas.DataFrame(mydata) #create the dataframe
```

- In Julia:

```
julia> using DataFrames #load the module

julia> df=DataFrame() #create empty dataframe

julia> df.Species=["setosa", "setosa", .... "virginica"] #load data for categorical␣
↪variable

julia> df.PetalWidth=[0.2, 0.2, .... 1.8] #load data for dependent variable(s)
```

You can refer to the excellent Pandas (for Python) or DataFrames (for Julia) online documentation for other ways to construct or modify your dataframe.

CHAPTER 8

Plotly

The plotting function for cloudy mountain plots is built on top of the freely available Plot.ly graphic library.

You can refer to the excellent online documentation to modify the layout, create subplots, retouch the subtraces or directly modify the cloudy mountain plot source code to add your own features to it:

- Python plotly documentation and figure reference
- Julia plotly documentation

8.1 Saving your plots

Orca (or more recently Kaleido) is used to export plot.ly plots as images (see Static image export support).

- In Python:

```
>>> from pathlib import Path
>>> p1=go.Figure(*cmplot(mydataframe,xcol="xsymbol"))
>>> homedir = str(Path.home())+'/'
>>> p1.write_image(homedir+'output_filename.pdf')
```

- In Julia:

```
julia> using ORCA
julia> p1=plot(cmplot(mydataframe,xcol=:xsymbol)...)
julia> savefig(p1::Union{Plot,PlotlyJS.SyncPlot}, joinpath(homedir(),"output_filename.
↪pdf"))
```

Note: You can simply use .svg or .png extension in the output_filename string in order to export the plot in svg or png format)*

CHAPTER 9

Author

- Dr Giuseppe Insana
  - (website)
  - (email)
  - (github)
  - (pgp/gpg key)
  - (linkedin)

CHAPTER 10

References

CHAPTER 11

Contribute

- Python Source Code on GitHub
- Julia Source Code on GitHub

CHAPTER 12

Support

- Python Issue Tracker
- Julia Issue Tracker

CHAPTER 13

Copyright

Licensed under the GNU Affero General Public License.

Copyright © 2019- Giuseppe Insana

Bibliography

[Hintze_Nelson_1998] Hintze, J. L., Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. The American Statistician 52, 181–184. [PDF]

[Kampstra_2008] Kampstra, P. (2008). Beanplot: A boxplot alternative for visual comparison of distributions. Journal of Statistical Software 28, 1–9. [PDF]

[Phillips_2017] Phillips, N. (2017). YaRrr! The pirate's guide to R. The Observer. [HTML]

[Anderson_Edgar_Fisher_1935_1936] Anderson, Edgar (1935) Fisher, R. A. (1936) Iris data

[Cornwell_Rupert_1988] Cornwell, C. and P. Rupert (1988) Individual wages, US, 1976 to 1982

[Galton_1886] Galton, F. (1886). Regression Towards Mediocrity in Hereditary Stature. Journal of the Anthropological Institute 15, 246-263 [PDF]

Index

T

X

Y