
Rayon Documentation

Release 1.4.3

Carnegie Mellon University

December 13, 2013

CONTENTS

1	Licenses	1
2	Overview	3
2.1	What is Rayon?	3
2.2	Installing	3
2.3	Command-line Tools	3
2.4	Building Visualizations in Python	4
3	Toolbox — Creating objects in Rayon	11
3.1	Toolbox Nicknames	12
4	Manipulating Data	15
4.1	Datasets	15
4.2	Columns	19
4.3	Rows	20
4.4	Melding and Flattening	21
5	Building Visualizations	23
5.1	Rendering	23
5.2	Charts	23
5.3	Plots	34
5.4	Scales	35
6	Conventions Used in Rayon	37
6.1	Specifying Colors	37
6.2	Specifying Size and Distance	38
6.3	Specifying Line Styles	38
7	Borders	39
7.1	none	39
7.2	Tickable borders	39
7.3	Labeled borders	41
7.4	Split borders	42
8	Charts	43
8.1	Layout	43
8.2	Padding	43
8.3	Charts	43
9	Data	57

9.1	Datasets	57
9.2	Columns	65
9.3	Rows	66
10	Grid lines	67
10.1	Creating grid lines	68
10.2	Adding tick marks to grid lines	69
10.3	horizontal	69
10.4	vertical	70
11	Lines	71
11.1	Lines and antialiasing	71
11.2	Creating lines	71
11.3	dashed	71
11.4	dotted	72
11.5	dotdash	72
11.6	none	72
11.7	solid	72
12	Markers	73
12.1	darrow	73
12.2	uarrow	73
12.3	larrow	74
12.4	rarrow	74
12.5	circle	74
12.6	cross	74
12.7	dot	74
12.8	hline	75
12.9	none	75
12.10	vline	75
13	Page	77
14	Plots	79
14.1	Creating Plots	79
14.2	Plots	80
15	Scales	93
15.1	Exceptions	93
15.2	Creating Scales	94
15.3	Scale Limits	94
15.4	Range Scales	94
15.5	Categorical Scales	94
15.6	Color Scales	97
15.7	Spatial Scales	99
16	Text	103
16.1	Specifying Text Properties	103
16.2	Labelers	105
16.3	Labeled Markers	105
17	Tick Sets	107
17.1	Tick sets and range scales	107
18	Toolbox	109

18.1	Creating a Toolbox	109
18.2	Using a Toolbox	109
18.3	The Toolbox object	110
19	ryhilbert	117
19.1	SYNOPSIS	117
19.2	DESCRIPTION	117
19.3	CONFIGURATION	117
19.4	REQUIRED ARGUMENTS	117
19.5	INPUT ASSOCIATION	118
19.6	SCALING DATA	118
19.7	FILTERING DATA	118
19.8	DISPLAY	119
19.9	GENERAL OPTIONS	120
19.10	EXAMPLES	120
19.11	SEE ALSO	120
20	rycategories	121
20.1	SYNOPSIS	121
20.2	DESCRIPTION	121
20.3	CONFIGURATION	121
20.4	REQUIRED ARGUMENTS	121
20.5	INPUT ASSOCIATION	122
20.6	SCALING DATA	122
20.7	DISPLAY	122
20.8	EXAMPLES	125
20.9	SEE ALSO	125
21	rypiechart	127
21.1	SYNOPSIS	127
21.2	DESCRIPTION	127
21.3	CONFIGURATION	127
21.4	REQUIRED ARGUMENTS	128
21.5	INPUT ASSOCIATION	128
21.6	DISPLAY	128
21.7	EXAMPLES	128
21.8	SEE ALSO	128
22	ryscatterplot	129
22.1	SYNOPSIS	129
22.2	DESCRIPTION	129
22.3	CONFIGURATION	129
22.4	REQUIRED ARGUMENTS	129
22.5	MARKERS	129
22.6	INPUT ASSOCIATION	130
22.7	SCALING DATA	130
22.8	FILTERING DATA	131
22.9	STATISTICS	131
22.10	DISPLAY	131
22.11	GRIDDED SCATTERPLOTS	133
22.12	DEPRECATED OPTIONS	133
22.13	EXAMPLES	134
22.14	SEE ALSO	134
23	rytimeseries	135

23.1	SYNOPSIS	135
23.2	DESCRIPTION	135
23.3	TERMINOLOGY	135
23.4	CONFIGURATION	136
23.5	READING DATA	136
23.6	VISUALIZATION STYLES	136
23.7	ARGUMENTS AND OPTIONS	137
23.8	STATISTICS	139
23.9	DISPLAY	139
23.10	EXAMPLES	144
23.11	SEE ALSO	146
24	rytools	147
24.1	DESCRIPTION	147
24.2	TOOLS	147
24.3	SEE ALSO	147
25	rydataformat	149
25.1	DESCRIPTION	149
25.2	USING SILK OUTPUT WITH RAYON	149
25.3	BASIC FORMAT	149
25.4	COMMENTS	150
25.5	HEADERS	150
25.6	SPECIAL HEADERS	150
25.7	COLUMN NAMES	151
26	ryrc	153
26.1	DESCRIPTION	153
26.2	CONFIGURATION FILE FORMAT	153
26.3	IGNORING CONFIGURATION	153
26.4	EXAMPLES	153
27	ryspecs	157
27.1	DESCRIPTION	157
27.2	COMMAND INVOCATION	157
27.3	COLUMN SPECIFICATION	157
27.4	SIZE AND DISTANCE	158
27.5	SCALES	158
27.6	TICK MARKS	158
27.7	MARKER SHAPE	159
27.8	LINE STYLES	159
27.9	COLORS	160
27.10	DATES AND TIMES	163
27.11	SEE ALSO	163
	Python Module Index	165
	Index	167

LICENSES

Copyright 2008-2011 by Carnegie Mellon University

Use of the Network Situational Awareness Python support library and related source code is subject to the terms of the following licenses:

GNU Public License (GPL) Rights pursuant to Version 2, June 1991

Government Purpose License Rights (GPLR) pursuant to DFARS 252.227.7013

NO WARRANTY

ANY INFORMATION, MATERIALS, SERVICES, INTELLECTUAL PROPERTY OR OTHER PROPERTY OR RIGHTS GRANTED OR PROVIDED BY CARNEGIE MELLON UNIVERSITY PURSUANT TO THIS LICENSE (HEREINAFTER THE “DELIVERABLES”) ARE ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, INFORMATIONAL CONTENT, NONINFRINGEMENT, OR ERROR-FREE OPERATION. CARNEGIE MELLON UNIVERSITY SHALL NOT BE LIABLE FOR INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, SUCH AS LOSS OF PROFITS OR INABILITY TO USE SAID INTELLECTUAL PROPERTY, UNDER THIS LICENSE, REGARDLESS OF WHETHER SUCH PARTY WAS AWARE OF THE POSSIBILITY OF SUCH DAMAGES. LICENSEE AGREES THAT IT WILL NOT MAKE ANY WARRANTY ON BEHALF OF CARNEGIE MELLON UNIVERSITY, EXPRESS OR IMPLIED, TO ANY PERSON CONCERNING THE APPLICATION OF OR THE RESULTS TO BE OBTAINED WITH THE DELIVERABLES UNDER THIS LICENSE.

Licensee hereby agrees to defend, indemnify, and hold harmless Carnegie Mellon University, its trustees, officers, employees, and agents from all claims or demands made against them (and any related losses, expenses, or attorney’s fees) arising out of, or relating to Licensee’s and/or its sub licensees’ negligent use or willful misuse of or negligent conduct or willful misconduct regarding the Software, facilities, or other rights or assistance granted by Carnegie Mellon University under this License, including, but not limited to, any claims of product liability, personal injury, death, damage to property, or violation of any laws or regulations.

Carnegie Mellon University Software Engineering Institute authored documents are sponsored by the U.S. Department of Defense under Contract FA8721-05-C-0003. Carnegie Mellon University retains copyrights in all material produced under this contract. The U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce these documents, or allow others to do so, for U.S. Government purposes only pursuant to the copyright license under the contract clause at 252.227.7013.

OVERVIEW

2.1 What is Rayon?

Rayon is a Python library and set of tools for generating basic two-dimensional statistical visualizations. Rayon can be used to automate reporting; provide data visualization in command-line, GUI or web applications; or do ad-hoc exploratory data analysis.

Rayon can generate visualizations in PDF, PNG, SVG and PostScript formats. It can also be used in [wxPython](#) GUI applications.

Rayon has a minimal set of dependencies, to simplify deployment in environments with significant change control requirements.

2.2 Installing

Rayon requires the following software packages:

- Python 2.4 or later
- The third party renderer of your choice. For GUI apps, Rayon uses [wxPython](#) (tested on 2.8.9+). For static output (PDF, PNG, SVG and PostScript), Rayon requires [Cairo](#) 1.2.4 or later, and [PyCairo](#) (tested with 1.2, 1.4 and 1.8).

Rayon is distributed as a tarball generated by Python distutils. To install, extract the archive and run `setup.py`, as in the following example (replacing `1.0`, with whatever Rayon version – usually the most current – you’re installing):

```
$ tar xzf rayon-1.0.tar.gz
$ cd rayon-1.0
$ python setup.py help      # to see options
$ python setup.py install  # to install
```

More detailed information on using `setup.py` is available from the Python distutils documentation.

2.3 Command-line Tools

Rayon is primarily a Python library. However, it also comes with a (currently small) suite of command-line tools that illustrate its capabilities and to provide value without writing Python code.

The following Python scripts come installed with Rayon and provide specific visualization capabilities at the command-line.

ryscatterplot Plots pairs of data elements against each other in a scatterplot.

rystripplot Presents multiple time series next to each other for easy comparison.

ryhilbert Displays a set of points on a number line (typically IP network addresses) on a space-filling Hilbert curve.

For more information on these tools, consult their man pages.

2.4 Building Visualizations in Python

The main use of Rayon is to produce visualizations using Python code. The next section shows some examples of Rayon's use, from a simple scatterplot to a more complicated example with multiple datasets and additional styling.

2.4.1 A Basic Scatterplot

The following is a basic but complete example that reads data from a file and generates an image file containing a scatterplot:

```
from rayon import toolbox

tools = toolbox.Toolbox.for_file()

infile = "sample_in.txt"
outfile = "sample_out.png"

# Read in data
indata = toolbox.new_dataset_from_filename(
    infile)

# Define the chart
chart = tools.new_chart("square")
plt = tools.new_plot("scatter")
plt.set_data(x=indata.column(0),
             y=indata.column(1))
chart.add_plot(plt)
c.set_chart_background("white")

# Draw the chart
page = tools.new_page_from_filename(
    outfile, width=400, height=400)
page.write(chart)
```

Depending on the data passed in `sample_in.txt`, *A (very) basic scatterplot* shows what the resulting scatterplot might look like.

2.4.2 Adding Borders and Titles

Although the previous example is sparse, it could be used out of the box to quickly visualize some data. However, some labels would be useful to give the data context. Also, given the sparseness of our data, we can make the dots larger for readability:

```
from rayon import toolbox

tools = toolbox.Toolbox.for_file()
```



Figure 2.1: A (very) basic scatterplot

```
# Read in data
indata = toolbox.new_dataset_from_filename(
    "sample_in.txt")

# Define the chart
chart = tools.new_chart("square")
plt = tools.new_plot("scatter")
plt.set_data(x=indata.column(0),
             y=indata.column(1))
# -- Change default marker size
plt.set_scales(marker_size=3)
chart.add_plot(plt)
c.set_chart_background("white")

# Add borders
# -- Horizontal bottom border
xmarker = tools.new_labeled_marker(
    marker=tools.new_marker('vline'),
    labeler=tools.new_labeler(halign="right"),
    label_position="e")
xticks = tools.new_tickset_from_spec(
    'default', tick_spec="n(5)",
    col=indata.column(0),
    scale=plt.get_scale("x"),
    labeledmarker=xmarker)
xborder = tools.new_border(
    "hline",
    ticksets=[xticks],
    tpad="5px")
xlabel = tools.new_border("hlabel", label="Time")
chart.add_bottom_border(xborder, height=.10)
chart.add_bottom_border(xborder, height=.06)
# -- Vertical left border
ymarker = tools.new_labeled_marker(
    marker=tools.new_marker('hline'),
    labeler=tools.new_labeler(halign="center"),
    label_position="s")
yticks = tools.new_tickset_from_spec(
    'default',
    tick_spec="n(5)",
    col=indata.column(1),
    scale=plt.get_scale("y"),
    labeledmarker=ymarker)
yborder = tools.new_border(
    "vline",
    ticksets=[yticks],
    rpad="5px")
ylabel = tools.new_border("vlabel", label="Space")
chart.add_left_border(yborder, width=.10)
chart.add_left_border(ylabel, width=.06)

# Add title
title = tools.new_border(
    "hlabel",
    label="Time versus Space",
    font_size="large")
chart.add_top_title(title, width=.10)
```

```
# Add padding
chart.add_padding("all", "20px")

# Draw the chart
page = tools.new_page_from_filename(
    outfile,
    width=400,
    height=400)
page.write(chart)
```

The visualization generated from this code would look like the the one in *A slightly more advanced scatterplot*.

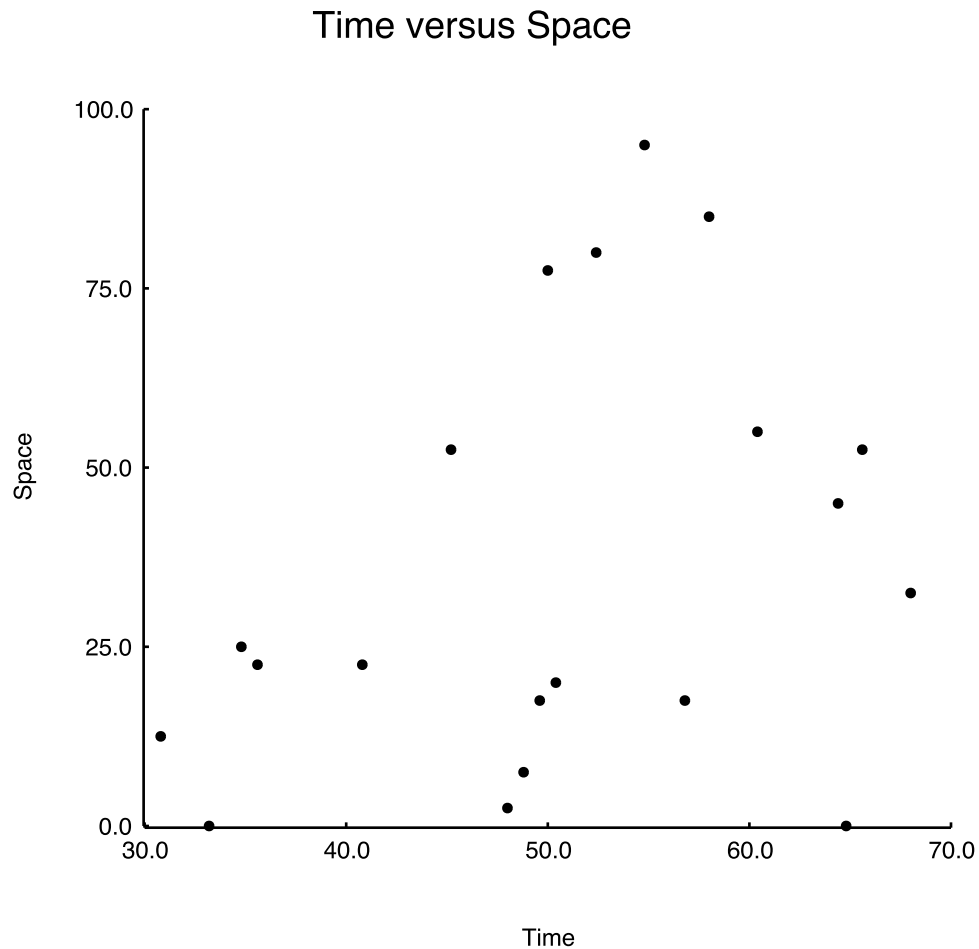


Figure 2.2: A slightly more advanced scatterplot

2.4.3 Gridlines and Additional Data

Finally, we enhance our example by plotting additional data and by adding a gray background and a grid. We'll also split out border creation (to make the code more readable), and put the main chart creation in a function of its own:

```
from rayon import toolbox

def build_axis_border(tools, bdr_type, mkr_type,
                     lbl_halign, lbl_pos, col, scale, **padding)
    marker = tools.new_labeled_marker(
        marker=tools.new_marker(mkr_type),
        labeler=tools.new_labeler(halign=lbl_halign),
        label_position=lbl)
    ticks = tools.new_tickset_from_spec(
        'default', tick_spec="n(5)",
        col=col, scale=scale,
        labeledmarker=marker)
    return tools.new_border(
        bdr_type, ticksets=[ticks],
        **padding)

def build_left_border(tools, col, scale, padding):
    return build_axis_border(
        tools, "vline", "hline", "right", "e",
        col, scale, rpad=padding)

def build_top_border(tools, col, scale, padding):
    return build_axis_border(
        tools, "hline", "vline", "center", "s",
        col, scale, tpad=padding)

def build_chart(tools, indata)
    chart = tools.new_chart('square')

    p1 = tools.new_plot('scatter')
    p1.set_data(x=indata.column(0),
               y=indata.column(1))
    p1.set_scales(marker_color='black',
                  marker_size=3)
    chart.add_plot(p1, "p1")

    p2 = tools.new_plot('scatter')
    p2.set_data(x=indata.column(0),
               y=indata.column(2))
    p2.set_scales(marker_color='red',
                  marker_size=3)
    chart.add_plot(p2, "p2")

    xborder = build_bottom_border(
        tools, indata.column(0),
        chart.get_scale("x"), "5px")
    chart.add_bottom_border(xborder, height="5px")

    yborder = build_left_border(
        tools, indata.column(1),
        chart.get_scale('x'), "5px")
    chart.add_left_border(yborder, width="5px")

    hgridlines = tools.new_gridlines_from_border(
        'horizontal', xborder)
    hgridlines.set_scales(line_color='white')
    chart.add_rear_gridlines(hgridlines)
```

```
vgridlines = tools.new_gridlines_from_border(
    'vertical', yborder)
vgridlines.set_scales(line_color='white')
chart.add_rear_gridlines(vgridlines)

chart.add_top_title(tools.new_border(
    'hlabel',
    label="Scatterplot Example")

chart.set_plot_background("gray")
chart.set_chart_background("white")
chart.set_padding(allpad="25px")
return chart

tools = toolbox.Toolbox.for_file()
indata = toolbox.new_dataset_from_filename(
    "sample_in.txt")
chart = build_chart(tools, indata)
page = tools.new_page_from_filename(
    outfile, width=400, height=400)
page.write(chart)
```

A scatterplot with a background and multiple data series shows what the resulting scatterplot might look like.

Once a function like `build_chart` in the above example is written, it can be used in a variety of contexts:

- A cron script that generates nightly reports
- A CGI script
- A GUI application
- A module where it can be available to colleagues or a broader user community

The visualization technique is available to many different audiences to use in whatever way and on whatever data they choose.

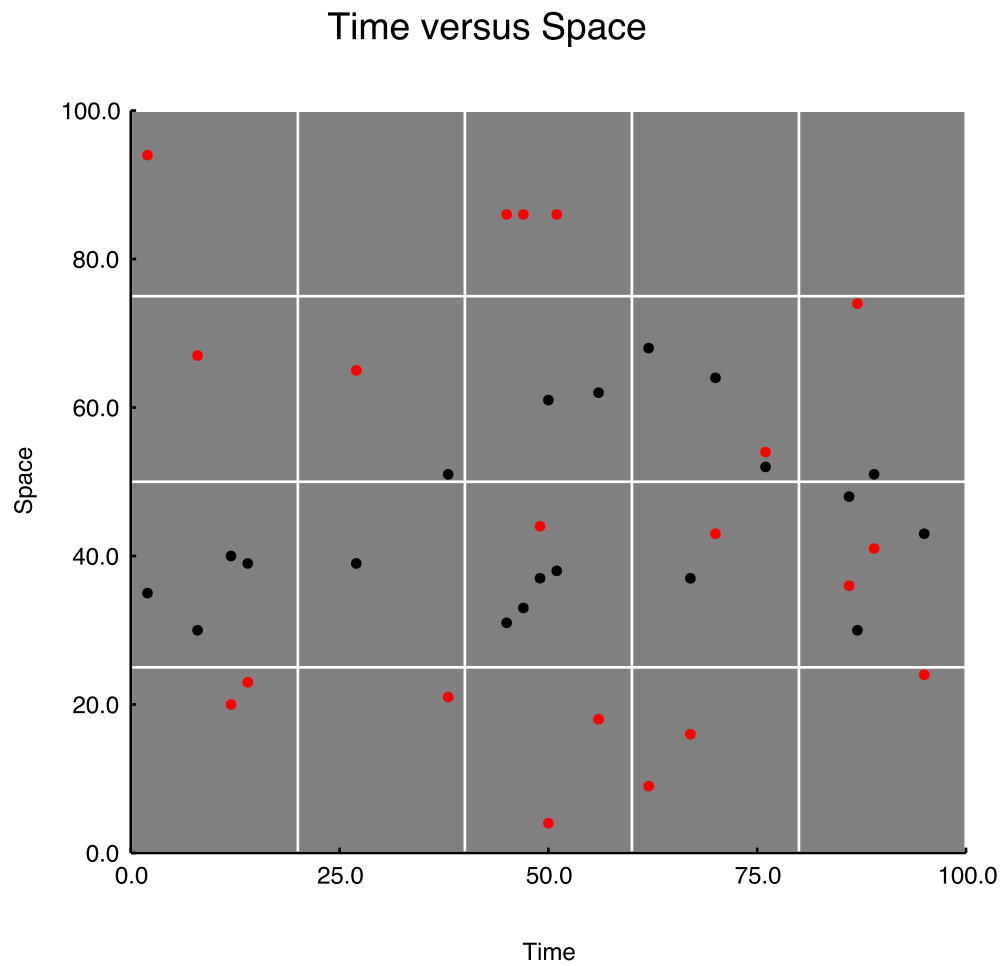


Figure 2.3: A scatterplot with a background and multiple data series

TOOLBOX — CREATING OBJECTS IN RAYON

The `Toolbox` class is how users create new Rayon objects. This may seem unusual to long-time Python users who are used to creating and using their objects directly, as in the following example:

```
from rayon import foo
d = foo.Foo("bar.txt")
...
```

This approach carries with it some advantages (notably simplicity), but it can get unwieldy when the objects created come from many different modules:

```
# Note: the actual constructors for these objects are
# private; this example simplifies them, and won't run
# as it is written here.
```

```
from rayon import (
    foo_backend,
    borders,
    charts,
    data,
    plots)

def build_chart():
    d = data.Dataset.from_filename("sample_in.txt")
    c = charts.SquareChart()

    p = plots.ScatterPlot()
    p.set_data(x=d.column(0), y=d.column(1))
    c.add_plot(p)

    b = borders.HorizontalLabeledBorder(label="Title")
    c.add_top_border(b, height=.05)

    c.set_chart_background("white")
    return c

page = foo_backend.Page.from_filename(
    "sample_out.png", 800, 600)
page.write(build_chart())
```

Several more items have been appended to the `import` statement. When additional objects must be created (to add text labels, tick marks and so on) the statement can grow quite bulky. What's more, if the developer wishes to move

Rayon-object-creating code to another module, it is now necessary to parse out which `import` statements are needed and move them over, as well.

This code has an additional problem: it hard-codes the use of a specific rendering backend (`foo_backend`), even though it is only needed for the `Page` object. It would be nice if the code that renders the chart could do so without hardcoding the backend; this gives us flexibility to add or change backends in the future.

The `Toolbox` class eases this management burden by putting all object creation in one place:

```
# This example, unlike the previous one,  
# actually works. :)  
  
from rayon import toolbox  
  
def build_chart(t):  
    d = t.dataset_from_filename("sample_in.txt")  
    c = t.new_chart("square")  
  
    p = t.new_plot("scatter")  
    p.set_data(x=d.column(0), y=d.column(1))  
    c.add_plot(p)  
  
    bl = t.new_border("hlabel", label="Title")  
    c.add_bottom_border(b, height=.05)  
  
    c.set_chart_background("white")  
    return c  
  
t = toolbox.Toolbox.for_file()  
page = t.new_page_from_filename(  
    "sample_out.png", 800, 600)  
page.write(build_chart(t))
```

The `import` statement is obviously much simpler. More importantly, if we choose to move `build_chart` to a different module, we don't have to move *any* `import` statements over; we just have to pass it a `Toolbox` object when the function is invoked. As codebases grow, this kind of flexibility can be very useful.

Are we still dependent on a particular backend? Well, the `Toolbox` still needs to know whether we intend to render our visualization to a static file, or to a more dynamic surface like a GUI panel, which can change its dimensions and handle user interface actions. However, it no longer hardcodes a reference to the `foo` backend, meaning we can use other file-based backends in the future, should they become available. The `build_chart` function, meanwhile, doesn't care at all what backend we are using, and can return charts that are usable when rendering to any backend.

3.1 Toolbox Nicknames

Many Rayon objects (e.g. markers, lines, borders) have several variants from which the user may choose. These variants can be specified via the `Toolbox` object using strings called nicknames.

To illustrate the use of nicknames, we will modify our example so it is a complete standalone script for generating scatterplots. The user can specify the input and output file. We will also frame the plot with lines on the bottom and left. The user will specify what style of line to use:

```
import sys  
from rayon import toolbox  
  
def build_chart(t, line_style):  
    d = t.dataset_from_filename("sample_in.txt")
```

```

c = t.new_chart("square")

p = t.new_plot("scatter")
p.set_data(x=d.column(0), y=d.column(1))
c.add_plot(p)

b1 = t.new_border("hlabel", label="Title")
c.add_bottom_border(b, height=.05)

l1 = t.new_line(line_style)
btm_border = t.new_border("hline", line=l1)

l2 = t.new_line(line_style)
lft_border = t.new_border("vline", line=l2)

c.set_chart_background("white")
return c

# Warning: real, non-example scripts
# would check their input...
infile, outfile, line_style = sys.argv[1:]

t = toolbox.Toolbox.for_file()
page = t.new_page_from_filename(
    "sample_out.png", 800, 600)
page.write(build_chart(t))

```

The lines on the bottom and left of the visualization are specified directly from user input. This is a good deal easier than a hypothetical alternative based on static object constructors:

```

...

def build_chart(t, line_style):
    ...
    if line_style == 'solid':
        l1 = t.new_solid_line()
    elif line_style == 'dashed':
        l1 = t.new_dashed_line()
    elif ...

```

The nicknames used to create an object variant are listed in the `Toolbox` methods that create them. Throughout the *Rayon User's Guide* and *Rayon Reference Manual*, object documentation lists the method (and nickname, where applicable) used to create them on the `Toolbox` object.

MANIPULATING DATA

In Rayon, data is organized into datasets, which are ordered collections of columns. A column is an ordered collection of data of the same type. All columns within a dataset should have the same number of members. Data in datasets can be accessed by column or by row, which is the collection of data members in each column of a dataset at a particular position, in the order of arrangement in the dataset.

4.1 Datasets

A `Dataset` object may be created from data in memory, or from data streamed from disk or the network. A dataset may also be exported to a file or stream.

4.1.1 Creating and using a Dataset object

`Dataset` objects are created from the `Toolbox` object. The `new_dataset_from_filename` and `new_dataset_from_stream` methods create datasets from external data like files or network streams. The `new_dataset_from_columns` and `new_dataset_from_rows` methods create datasets from data in memory. To create an empty `Dataset` object, pass an empty list into either the `new_dataset_from_columns` or `new_dataset_from_rows` method.

After a `Dataset` object has been instantiated, new columns can be added to it with the `append_column` method. Therefore, this:

```
# t is a Toolbox object
# c1, c2, and c3 are Column objects
d = t.new_dataset_from_columns([c1, c2, c3])
```

is equivalent to this:

```
d = t.new_dataset_from_columns([])
d.append_column(c1)
d.append_column(c2)
d.append_column(c3)
```

Headers can be added with the `set_header` method:

```
# Add header named "foo" with value "bar"
d.set_header("foo", "bar")
```

Column names can be added with the `set_column_names` method:

```
# d is a dataset object with 3 columns
d.set_column_names(["foo", "bar", "baz"])
```

4.1.2 Comparing Dataset objects

`Dataset` objects may be compared for equality. Two `Dataset` objects are considered equal to each other if and only if:

- They contain an equal number of columns
- Their lists of column names are identical (same length, contents and order).
- They have equal columns in the same order. Column equality is determined by comparing each `Column` object.

4.1.3 Accessing Data in Dataset Objects

The `Dataset` object supports row-major and column-major iteration. The default iterator is row-major; for `r` in `d` iterates over all the rows in dataset `d` in the current sort order. To iterate over the rows in native sort order (regardless of the current sort order), use `Dataset.iter_ignore_sorted`. To get an iterator over all the columns in a `Dataset` object, use `Dataset.iter_columns`.

The `Dataset` object does not support array-based direct access: `d[0]` will raise an error. To refer to a specific column or row in a dataset, use the `Dataset.get_column` or `Dataset.get_row` methods. To refer to a specific field in a dataset, get the appropriate row or column and access the field directly:

```
# d is a dataset with 10 rows
# and columns foo, bar and baz

# Gets value of column foo in 5th row
d.get_row(4).foo
# ...or...
d.get_column('foo')[4]
```

`get_column` and `get_row` return `Column` and `Row` objects. See *Columns* and *Rows* for more information on these objects.

4.1.4 Sorting and Filtering Dataset objects

It is often convenient to work with data that is sorted in different ways, or with subsets of data. Sometimes, as with top *n* lists, we want to do both.

Datasets can be sorted using the `Dataset.sort` method. This method takes a key function which takes a *Row* as input, and returns a comparable value which is actually used in the sort. The `Dataset.sort` method changes the *sort order* of the `Dataset` object, which changes the order through which the data is iterated or accessed through the `get_row` method. It does **not** change the ordering of data on-disk; the *native sort order* (or *insert order*) can always be recovered by calling the `Dataset.sort` method with no arguments.

The `Dataset` object's filtering methods also take a function. The filtering function takes a *Row* object as input, and returns `True` if the row passes the filter, `False` if it fails. (As a shortcut, `True` is equivalent to `lambda r: True` and `False` is equivalent to `lambda r: False`; these are sometimes used in conjunction with the *limit* parameter to select a portion of the dataset.) The `Dataset.filter_pass` and `Dataset.filter_fail` methods are utility methods to simplify invocations to `Dataset.filter`; if both the passing and failing sets of data are of interest, they can be captured in a single pass through the data using `Dataset.filter_both`.

To generate a top n list from a dataset, first sort the data on the desired column, then filter it with a *limit* parameter of n :

```
# Get the top n elements of d, sorted by num_hits
d.sort(key=lambda r: r.num_hits)
top_n_by_num_hits = d.filter_pass(True, limit=n)
```

4.1.5 Exporting Dataset objects

Dataset objects can be exported to disk, streams or strings in the format described in *On-disk format*. The methods for this are analogous to those described in *Creating and using a Dataset object*.

```
>>> d.to_file("bar.txt")
>>> ostrm = open("bar2.txt", 'w')
>>> d.to_stream(ostrm)
>>> print d3.to_string()
1|a
2|b
3|c
```

4.1.6 On-disk format

This section describes the format Rayon uses for data import and export. This format is fundamentally delimited text, with additional facilities for storing metadata with the dataset. The SiLK tools' text output (if `--no-titles` and `--no-columns` are passed to the tool as options) is a subset of this format, and may be passed to Rayon unchanged.

Rayon's default delimiter is the pipe character (`|`). The delimiting character must not appear in the input. An example of valid input is:

```
foo|1|2
bar|3|4
```

Whitespace around delimiters will be removed, so the following is equivalent to the above:

```
foo| 1| 2
bar| 3| 4
```

Whitespace lines are also ignored, so this is equivalent to the above:

```
foo|1|2

bar|3|4
```

The Rayon data format does not support infix or postfix comments. In the following, the text `# this is not` will be interpreted as content:

```
# this is a comment
foo | a | a
bar | b | b # this is not
```

Comments

Lines beginning with an octothorpe (`#`) are comments, and are ignored (with the exceptions outlined in *Special Headers* and *Column Names*). Therefore, this is also equivalent to the above:

```
foo| 1| 2
# this line will be ignored
bar| 3| 4
```

Headers

Headers are special comments at the top of the file (or beginning of the stream) that start with exactly two octothorpes (##). Headers are used to store metadata about a dataset, and may be accessed or changed using methods on the `Dataset` object.

A header contains a name, followed by a colon, followed by a value. There may be whitespace between the colon and either the name or value. Here is an example of a header:

```
## Description: This is an example dataset
foo| 1| 2
bar| 3| 4
```

This data set contains a header named `Description`. The header has the value `This is an example dataset`.

Header names may contain the upper- and lower-case letters, numbers, underscore (`_`) and hyphen (`-`). Header values may contain any of the ASCII character set between `0x20` and `0x7e`, inclusive. Certain header names are reserved, specifically those in *Special Headers* and names beginning with `Rayon-`; notwithstanding these restrictions, users may create arbitrary headers as they see fit. Header case will be preserved, but header lookups are case-insensitive.

The first line of data will terminate the processing of headers; any subsequent lines beginning with any number of octothorpes will be treated as a comment.

Special Headers

Some headers have special meaning when parsed. For instance, the `Delimiter` header may be used to change the delimiting character of the file:

```
## Delimiter: ,
foo,1,2
bar,3,4
```

The following header names have special meanings:

Title The title of the dataset

Delimiter A single character to be used as the delimiting character between items in a row.

Typemap A set of type names used to convert data in the file from text to a native data type.

Column-Names A list of column names, delimited by the same character as the data. (See *Column Names*)

Column Names

Column names may be specified with the `Column-Names` header:

```
## Column-Names: label/value1/value2
foo|1|2
bar|3|4
```

This input will generate a dataset with the column names “label”, “value1” and “value2”, respectively. Column names can be used to access data in the dataset, and add readability to the on-disk representation. See the [Dataset](#) documentation for more details.

As a convenience, there is an alternate syntax for specifying column names. The last comment line before the first data row may optionally specify the names of the columns in the dataset. If the last comment line before the first data row is delimited with the delimiting character and contains as many elements as the first data line of the file, its contents will be used as the names of the dataset columns. The following is equivalent to the previous example:

```
# label| value1| value2
foo|1|2
bar|3|4
```

As with headers, whitespace between the comment character and the column name designation will be ignored, but multiple comment characters will probably give unwanted results. Thus, the following is legal:

```
#label| value1| value2
foo|1|2
bar|3|4
```

The following is also legal; the dataset ignores whitespace surrounding column names:

```
#label|value1|value2
foo|1|2
bar|3|4
```

4.2 Columns

A [Column](#) object represents a single column of data in a dataset. [Column](#) objects can be extracted from [Dataset](#) objects and recombined in different ways. It is also possible to iterate over the data in a [Column](#) and, for numeric data, to compute statistics such as the mean and variance.

4.2.1 Comparing Column objects

[Column](#) objects can be compared for equality. Two [Column](#) objects are equal if each item in one column compares as equal to the item in the other column at the corresponding index.

4.2.2 Accessing Data in Column objects

[Column](#) objects can be treated like arrays:

```
>>> from rayon.toolbox import Toolbox
>>> t = Toolbox.for_file()
>>> raw = [[1,2,3], ['a', 'b', 'c']]
>>> d = t.new_dataset_from_columns(raw, colnames=['foo', 'bar'])
>>> c = d.get_column('bar')
>>> c[0]
'a'
>>> len(c)
3
>>> list(c)
['a', 'b', 'c']
```

However, [Column](#) objects are immutable, so this won't work:

```
>>> c[0] = 'z'
...
TypeError: 'Column' object does not support item assignment
```

4.2.3 Column Statistics

`Column` objects provide several statistical functions on the data. With the exception of `uniq`, these methods require that the `Column` object contain numeric data.

The following functions are available:

```
# c is a Column object
c.max()           # Largest value
c.mean()          # Average value
c.min()           # Smallest value in column
c.percentile(25)  # 25th percentile value
c.sample_stdev()  # Sample standard deviation
c.sample_variance() # Sample variance
c.stdev()         # Population standard deviation
c.variance()      # Population variance
c.uniq()          # List of all unique values
```

In general, if a statistical function is available from the `Column` object, it is better to use it than to compute it independently because the `Column` object will cache values and (where necessary) sorted order.

4.3 Rows

Rows are data containers representing one row in a dataset. They are returned from the `get_row` method and passed as an argument to the sorting and filtering functions used in `sort`, `filter_pass` and `filter_fail` methods.

The data in Row objects can be accessed either by index or, if the `Dataset` object contains column names, by name:

```
>>> from rayon.toolbox import Toolbox
>>> t = Toolbox.for_file()
>>> raw = [[1,2,3], ['a', 'b', 'c']]
>>> d = t.new_dataset_from_columns(raw, colnames=['foo', 'bar'])
>>> r = d.get_row(2)
>>> len(r)
2
>>> r[0]
3
>>> r[1]
'c'
>>> r['foo']
3
>>> r['bar']
'c'
>>> r.foo
3
>>> r.bar
'c'
```

Multiple values can be returned by passing a tuple of indices.:

```
>>> r[(0, 1)]
(3, 'c')
>>> r[('foo', 'bar')]
(3, 'c')
>>> r[(0, 'bar')]
(3, 'c')
```

It is also possible to iterate over Row objects:

```
>>> tuple(x for x in r)
(3, 'c')
>>> tuple(r)
(3, 'c')
>>> list(r)
[3, 'c']
```

Like Column objects, rows are immutable:

```
>>> r.bar = "baz"
...
TypeError: 'Row' object does not support item assignment
```

4.4 Melding and Flattening

Usually, it is best if each scalar value in a dataset is a member of its own column. Occasionally, however, we want a single column that contains multiple scalar values. For example, if we are displaying a barchart showing how many times each combination of n values was observed, the permutation will need to be a single column containing multiple values.

To get the data in the right place, we must first import the data in the normal way. Say the data on disk represents connections to either TCP or UDP ports in a set of network traces, and looks like this:

```
# proto|port|network|count
TCP|8080|A|1009
UDP|8080|A|1001388
TCP|25|A|4396
TCP|53|B|230
UDP|25|A|4
...
```

If we simply plot port against, say count, TCP and UDP ports 8080 will be plotted in the same place, which is probably not what we want. Since each observation is of the form “ W connections to port X via protocol Y on network Z), we might very well wish we had a 2-column dataset, where one column was count and the other was a composite of proto, port and network.

To get this we first import the data normally, then extract a special Column object containing the proto, port and network columns as a single unit, using the `meld` method. We can then put this in a new dataset:

```
# t is a Toolbox object
d = t.dataset_from_file("our-data.txt")
key = d.meld('proto', 'port', 'network')
count = d.get_column('count')
d2 = t.dataset_from_columns(
    [key, count],
    colnames=['key', 'count'])
```

The key column is now made up of tuples containing the original Column objects' data:

```
>>> key[0]
("TCP", 8080, "A")
```

These `Column` objects can be passed into plots that use scales which understand them. For example, they may be used as the categories in an `hbar` plot.

The `flatten` method will return a new `Dataset` object, with all melded columns reverted to multiple columns in the data. For exaple, the following will make `d3`, which is identical to `d`:

```
d3 = d2.flatten(
    new_colnames=['proto', 'port', 'network'])
d3.append_column(d2.get_column('count'), 'count')
```

BUILDING VISUALIZATIONS

This chapter describes the core building blocks of Rayon. The *chart* contains and lays out all the elements of a Rayon visualization. The most important component of a data visualization in Rayon is usually the *plot*; one or more can go in the center of the chart.

5.1 Rendering

The *Page* object represents a surface on which a chart will draw a visualization. To create a *Page* object, call the *new_page_from_buffer* or *new_page_from_filename*. The *Page* object has a single method, *Page.write*, which takes a chart and visualizes it on the drawing surface:

```
# t is a Toolbox object
c = t.new_chart('square')
# ...configure c...
p = t.new_page_from_filename(
    "foo.png", width=400, height=200)
p.write(c)
```

A chart can be written to multiple outputs by passing it to multiple *Page* objects. This can be done to save a chart in a GUI as a file, for instance.

5.2 Charts

Charts are laid out in a series of boxes, as in *The chart layout model*.

Each box can contain one or more visual components, such as a *plot* or *border*.

At the center is the plotting area; plots added to the chart drawn here; how they are laid out varies between the different chart types. The user can also insert *grid lines* and background/foreground colors or images into a *square* chart.

The corners of the chart may also contain plots, if desired. The width and height of the corners are a function of the borders along their edges; if borders don't exist along both dimensions of a corner, the corner has zero area and won't be displayed.

Around the plotting area's edges are the top, bottom, left and right border areas. borders added to the chart are laid out here in the order they were added, from the center outward. (That is, borders added via *add_top_border* are rendered above the plotting area, bottom to top; borders added via *add_bottom_border* are rendered below the plotting area, top to bottom.) There is no explicit limit to the number of borders a chart may have; the plotting area will shrink as necessary to accomodate them.

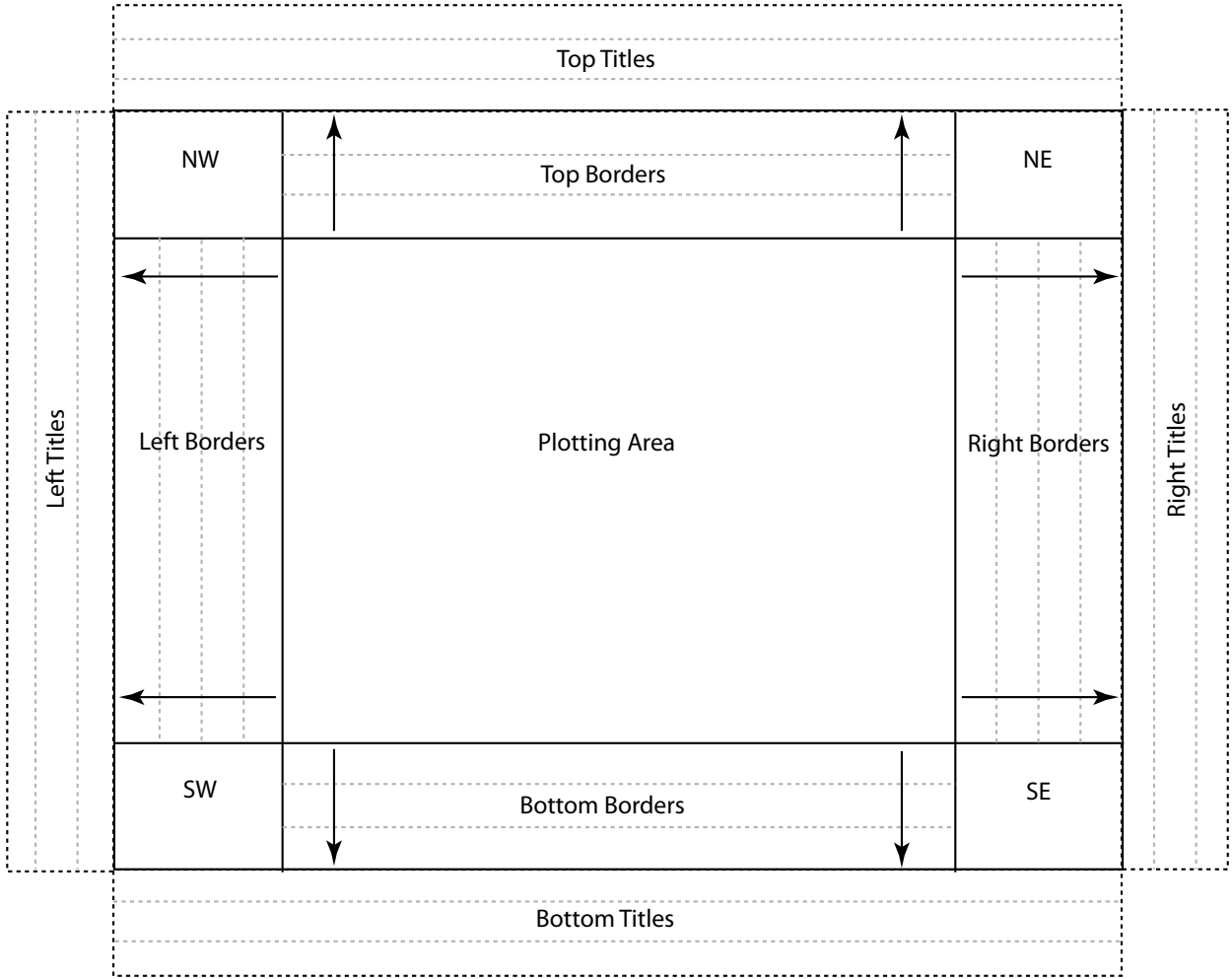


Figure 5.1: The chart layout model

Outside both the borders and corners are the title areas. These areas are identical to the border areas, but span across the corners. They are more appropriate for chart-wide titles, as borders placed in them will center across the whole chart, not just the plot area.

5.2.1 Padding

Each component placed in a box may contain padding. Padding may also be added to the chart itself.

Padding Objects in the Chart

Most methods that add objects to a *chart* take padding arguments. The four padding arguments are *tpad*, *bpad*, *lpad* and *rpadd*. These take size specifications (see *Specifying Size and Distance*) quantifying the amount of top-, bottom-, left- and right-edge padding, respectively.

The following code illustrates the use of padding arguments, and demonstrates the interaction between padding and other spacing arguments, like *width* and *height*:

```
# t is a Toolbox object
# c is a chart
b = t.new_border('hline', bpad="5px")
c.add_top_border(b, height="10px")
p = t.new_plot('scatter')
c.add_plot(p, lpad="3px")
```

In the above example, the chart will allocate ten pixels of height to the border. The border will only use five pixels of that space, reserving the rest for padding at the bottom of its bounding box. The box containing the plot will also be padded by three pixels on its left side.

Padding the Chart

The chart itself can also be padded. Use `set_padding` to change the amount of whitespace around the edges of the chart. The following puts ten pixels of whitespace on each edge of the chart:

```
c.set_padding(tpad="10px", bpad="10px",
              lpad="10px", rpadd="10px")
```

tpad, *bpad*, *lpad* and *rpadd* refer to the padding of the top, bottom, left and right edges, respectively, as when padding objects in the chart. Additionally, the *allpad* argument is a shorthand for padding on all edges. The previous example could also be written as:

```
c.set_padding(allpad="10px")
```

A chart with no padding and *A chart with padding* illustrate the use of chart padding. The first has no padding.

Note that the plot extends all the way to the right edge of the image – there is zero right border. The left side is similarly tight; some of the labels on the left (and the rightmost label on the bottom) have been clipped.

A chart with padding has 25 pixels of padding on all sides. The overall visual effect is more pleasant, and nothing is clipped.

5.2.2 Adding Borders and Titles

Borders can be added to the edges of a chart. The following is an example of adding a border to the top of a chart:

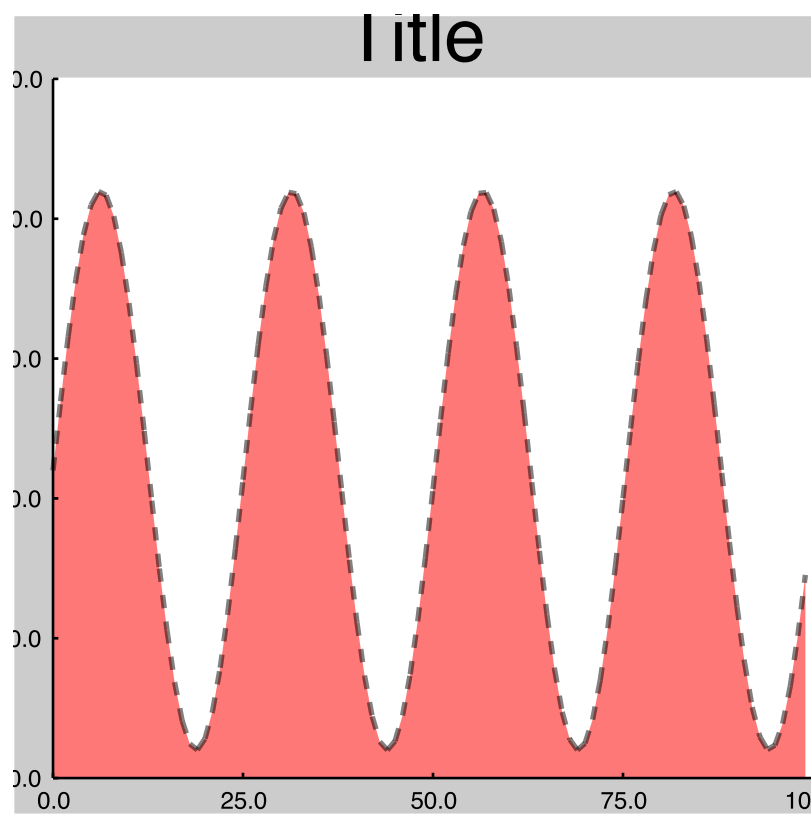


Figure 5.2: A chart with no padding

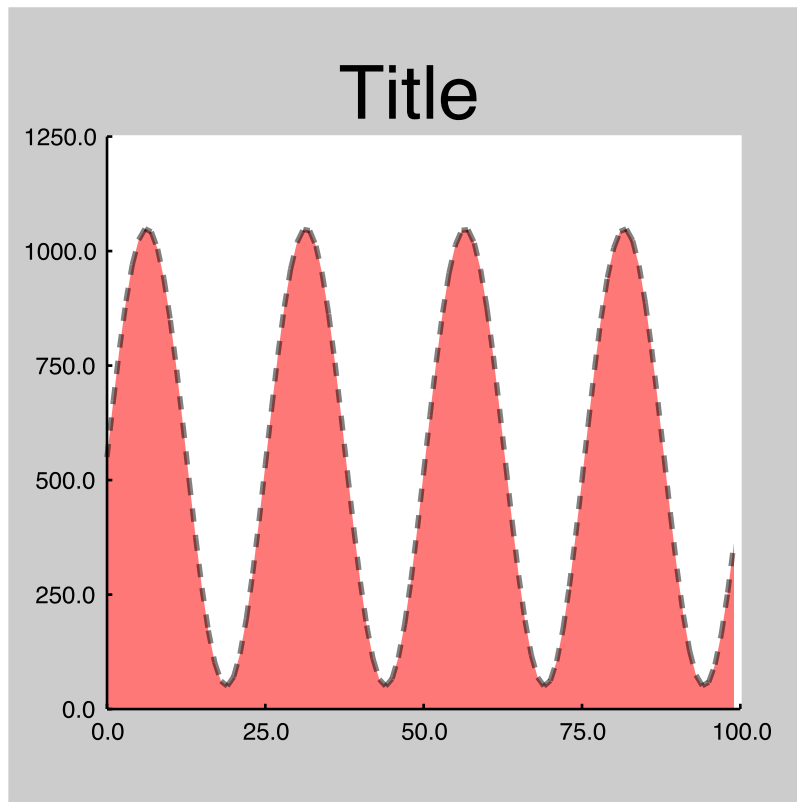


Figure 5.3: A chart with padding

```
# t is a Toolbox object
# c is a chart
b = t.new_border('hlabel', label="Title",
                 font_size="x-large")
c.add_top_border(b, height="50px")
```

`add_top_border` takes a border and a size specification of the width or height of the border being added. (See *Specifying Size and Distance*.)

A comparable set of methods exists for adding borders to the plot as titles. Title areas span across the whole chart, where border areas only span the plotting area.

The next example adds the border above as a title:

```
b = t.new_border('hlabel', label="Title",
                 font_size="x-large")
c.add_top_title(b, height=.10)
```

Title added as a border, centered over the plotting area shows the difference in layout between adding a border and adding a title. The first image contains a centered label as a border, added using `add_top_border`.

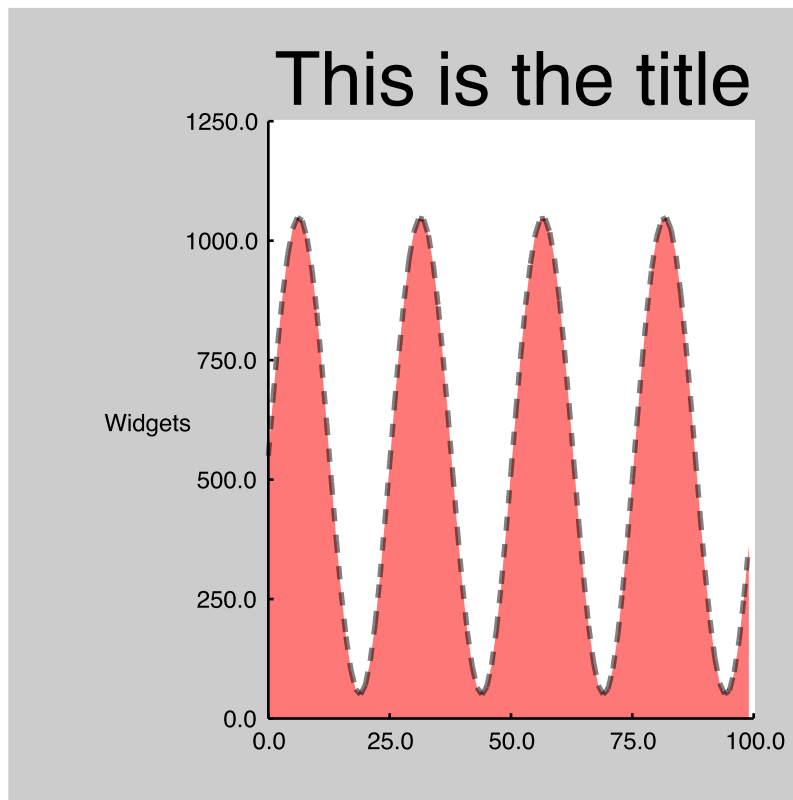


Figure 5.4: Title added as a border, centered over the plotting area

The label is centered over the plot area. To center it over the whole chart (including the left border), use `add_top_title` instead, as in *Title added as a title, centered over the whole chart*.

5.2.3 Adding Gridlines

Grid lines can be added to the plotting area to add reference lines or grids. Grid lines can be added to both the front

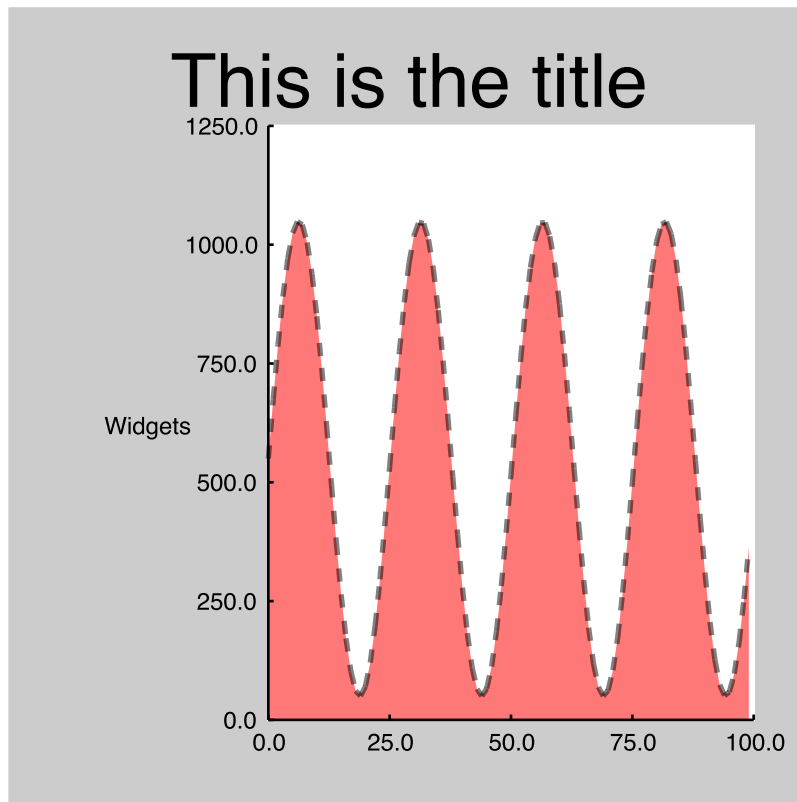


Figure 5.5: Title added as a title, centered over the whole chart

and rear of the plotting area:

```
# t is a toolbox. c is a chart. b is a border
vgrid = t.new_gridlines_from_border(b)
c.add_rear_gridlines(vgrid)
```

In *Chart with rear gridlines*, black lines have been added to the chart using `add_rear_gridlines`.

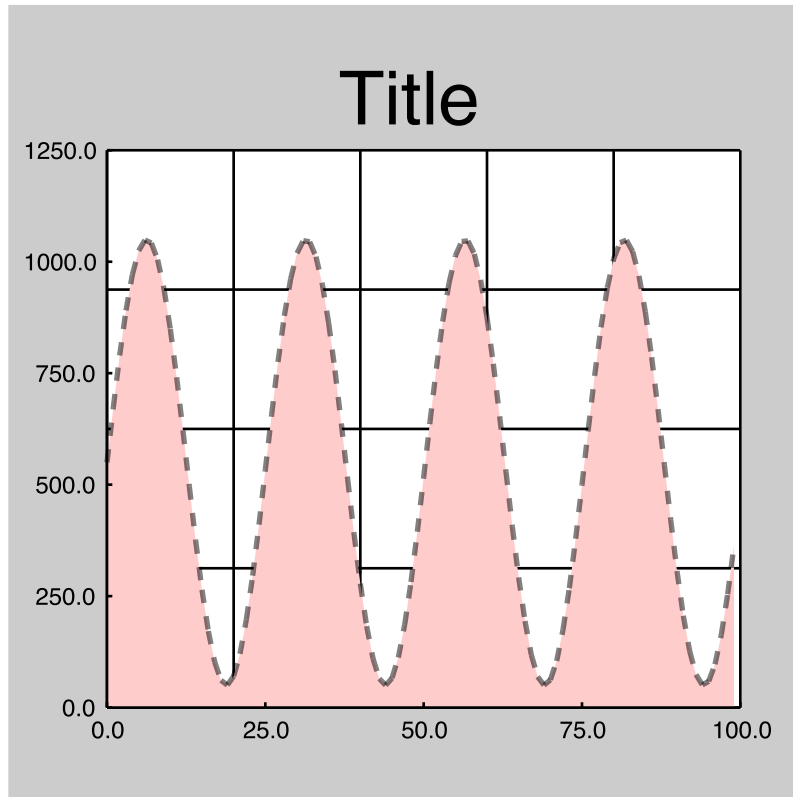


Figure 5.6: Chart with rear gridlines

Chart with front gridlines is constructed identically to the previous one, except that the lines have been added to the chart using `add_front_gridlines`.

5.2.4 Adding Plots

The `add_plot` method adds a plot to a chart's plotting area. How the plot is laid out depends on the type of chart you are using.

square charts

The `square` chart treats the plotting area as a series of layers. Successive calls to `add_plot` or `add_chart` add new plots and charts to the plotting area, to be drawn on top of previously-added objects.

Two plots in a square chart, red added first illustrates how the order in which plots are added influences display. In the first image, the red plot is added before the blue one. Contrast this figure with *Two plots in a square chart, blue added first*, the blue plot is added first.

The `square` object also has an `add_chart` method, which adds a chart to the plotting area in the same way.

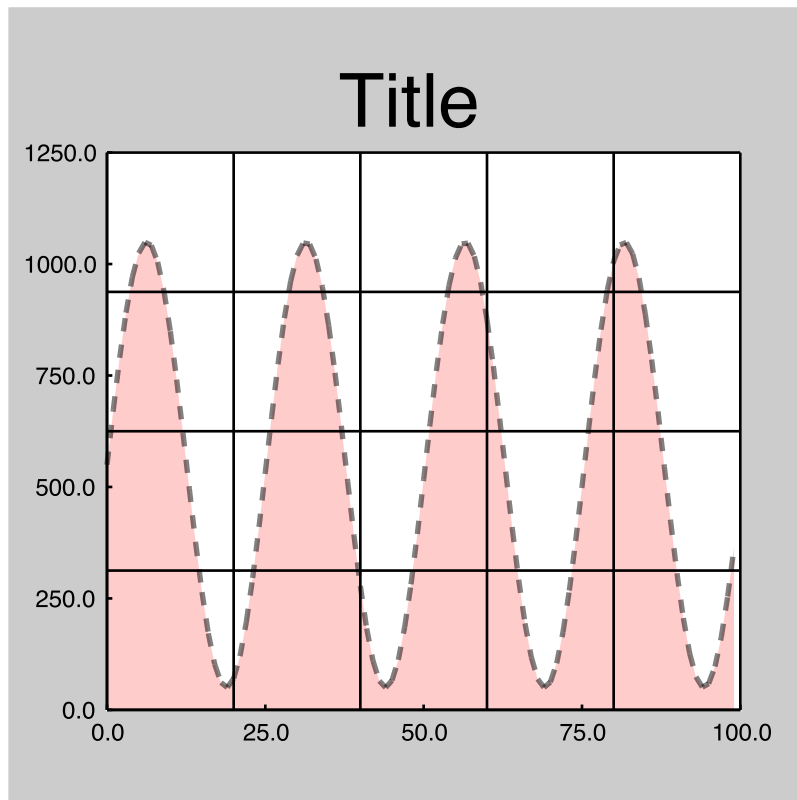


Figure 5.7: Chart with front gridlines

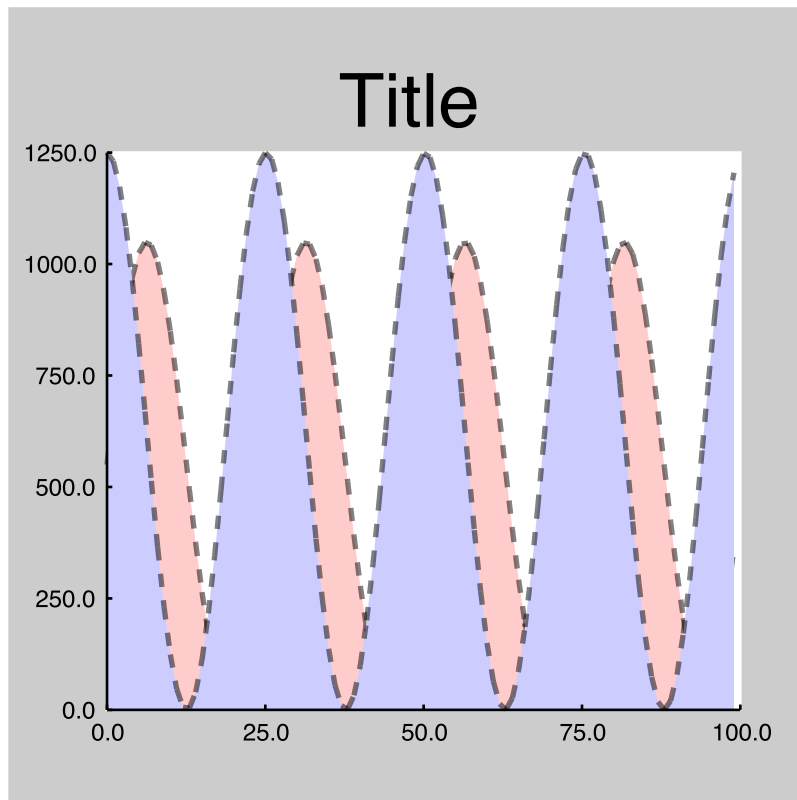


Figure 5.8: Two plots in a `square` chart, red added first

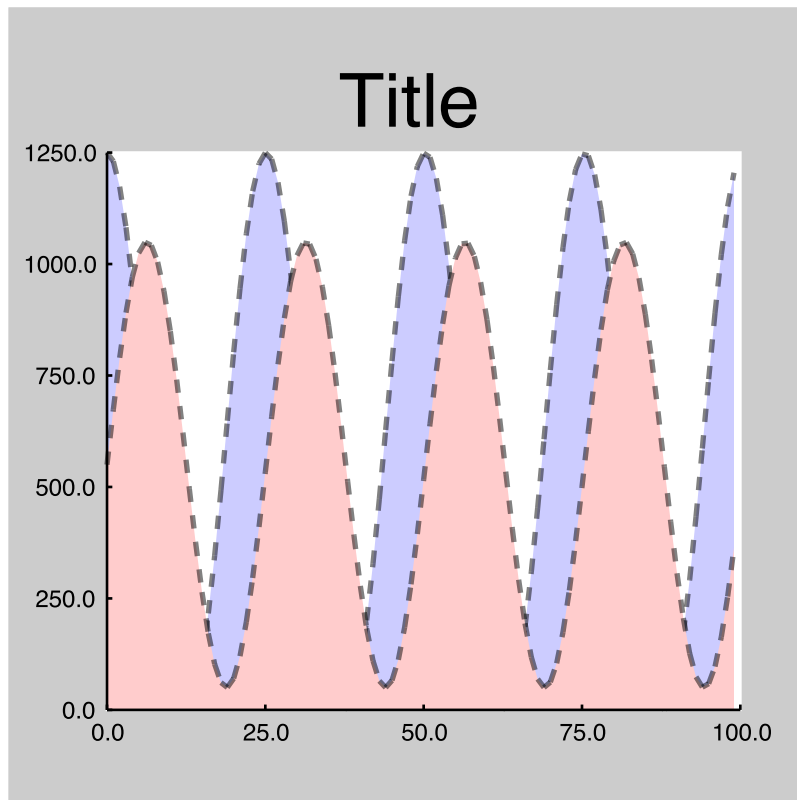


Figure 5.9: Two plots in a `square` chart, blue added first

tilted and tilted_adv charts

The `tilted` and `tilted_adv` charts divide the plotting area into a series of cells, filling each cell with plots and charts in the order in which they were added. It is generally more useful to add charts rather than plots to `tilted` and `tilted_adv` charts, as you can associate gridlines, borders and other decorations with the individual subcharts.

Adding charts to a tiled chart illustrates adding a `square` chart to a `tilted` chart. Axis lines and labels have been added to each subchart; the first subchart also shows the scale limits (which are shared by all the subcharts).

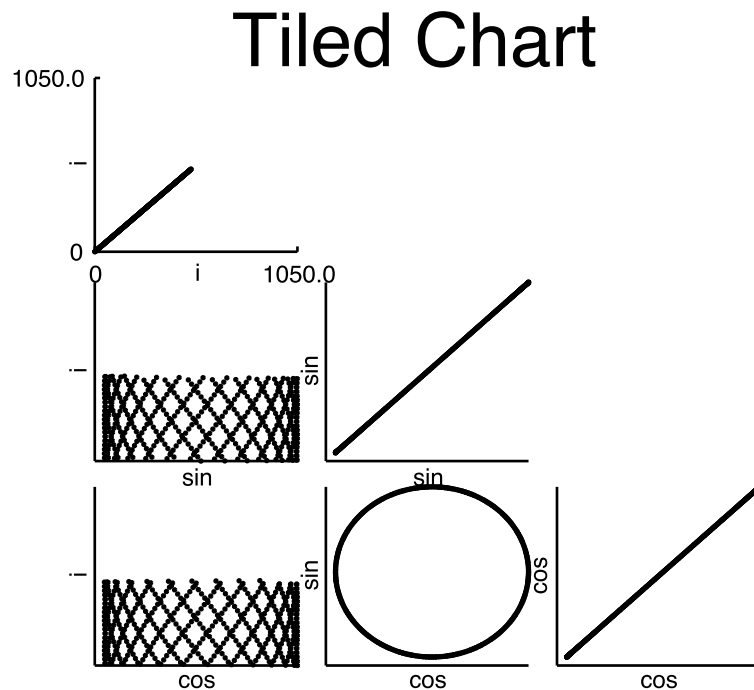


Figure 5.10: Adding charts to a tiled chart

5.3 Plots

Plots are the main way to visualize data in Rayon. A plot contains a set of axes. A fully-specified axis consists of a name (or axis label), a column of data, a scale for the data. It also contains the logic to draw a picture using the axes.

The following is the simplest use of a object (in this case, a `scatter` plot); associate data with axes, and add the plot to a `chart`:

```
# t is a toolbox. d is a dataset. c is a chart.
p = t.new_plot('scatter',
               x_data=d.column(0),
               y_data=d.column(1))
c.add_plot(p)
```

5.3.1 Setting data and scales

In order to use a plot, it is necessary to configure the plot with data to display and scales to transform the data into something drawable.

There are two ways to set axis data and scales. When a plot is created, axis data and scales can be set by passing special arguments to `rayon.toolbox.Toolbox.new_plot`. Each argument consists of an axis label and either the words `scale` or `data`, separated by an underscore. In the following example, `x_data` and `y_data` are setting the data component of the `x` and `y` axes:

```
# t is a toolbox. d is a dataset.
p = t.new_plot('scatter',
               x_data=d.get_column(0),
               y_data=d.get_column(1))
```

Data and scales may also be associated with axes after the object is created, with the `set_data` and `set_scales` methods:

```
# t is a toolbox. d is a dataset.
p = t.new_plot('scatter')
p.set_data(x=d.get_column(0), y=d.get_column(1))
# Add a log scale to Y
p.set_scales(y=t.new_scale('log'))
```

A scale argument will also accept a constant value that will be used for every point. The following example sets all points in the `scatter` plot to blue:

```
# t is a toolbox. d is a dataset.
p = t.new_plot('scatter',
               x_data=d.column(0),
               y_data=d.column(1),
               marker_color_scale="blue")
```

5.3.2 More Information

A full list of the plot objects that come with Rayon is available in the *Plots* section of the *Rayon Reference Manual*.

5.4 Scales

Scales are used to convert data in a visualization into a set of coordinates. These coordinates are used by plots and borders to draw points on the chart.

Rayon provides several scales that commonly appear in visualization – linear and log scales, categorical scales and scales that map numeric ranges to color gradients, among others. However, scales can be functions. A scale function takes an argument (a piece of data) and returns a value that can be used by the plot to display data on the associated axis. The following function implements a simple linear scale that could be used as the `x` or `y` axis of a `scatter` plot:

```
# input_max is the largest
# value of x we plan to see
def linear_scale_func(x):
    return float(x) / float(input_max)
```

To determine what kind of data a plot expects on a given axis, consult the documentation for that object. For instance, the `x` axis of the `scatter` plot takes a value between 0 and 1, so `linear_scale_func` would work for all

positive values of x less than or equal to `input_max`. (For a more robust version of this scale that can handle negative numbers, see the [linear](#) scale.)

5.4.1 Changing Limits

Many scales have natural limits to the data they can transform. The [scales](#) that come with Rayon will, by default, grow and shrink their limits to cover the data associated with it. This behavior can be changed in two ways. The first is to specify limits when the scale is created:

```
# t is a toolbox
s = t.new_scale('linear', 10, 20)
```

This will create a [linear](#) scale whose lowest point is 10 and highest is 20. These limits will not change, even if the data exceeds them; it is a *fixed* scale.

To provide initial limits to a scale, but allow the scale to adjust its limits if necessary, pass it the *fixed* argument with a value of `False`:

```
s = t.new_scale('linear', 10, 20, fixed=False)
```

Rayon will not create a fixed scale unless the limits are fully specified when the scale is created; the following three statements create unfixed scales, even though no *fixed* argument is passed:

```
s = t.new_scale('linear')
s = t.new_scale('linear', 10)
s = t.new_scale('linear', input_min=20)
```

5.4.2 More Information

A full list of the scale objects that come with Rayon is available in the [Scales](#) section of the *Rayon Reference Manual*.

CONVENTIONS USED IN RAYON

This chapter describes some conventions that are used throughout Rayon to specify colors, sizes, text and line styles.

6.1 Specifying Colors

Colors can be specified in two main ways in Rayon. The first is as a triple or 4-tuple of integers ranging from 0-255. The elements of the tuple represent the proportion of red, green, blue and (if specified) alpha in the color. (An alpha value of 255 represents full opacity.) For example:

```
# Add a pure blue background. (c is a chart.)
c.add_plot_background((0, 0, 255, 255))
```

If no alpha is specified, it is assumed to be 255. The following is equivalent to the above:

```
c.add_plot_background((0, 0, 255))
```

Color may also be specified as a string. Rayon follows the format described in <http://www.w3.org/TR/CSS2/syndata.html#color-units> and <http://www.w3.org/TR/css3-color>, with two exceptions:

- Alpha channels are supported in hexadecimal notation as well as in functional notation.
- The `hsl(...)` and `hsl(...)` specification forms are not supported.

For example, here are all the ways to specify that a blue background should be added to a plot in a chart using a string, disregarding case (which is ignored):

```
# name. All named colors are fully opaque,
# i.e. alpha is 255.
c.add_plot_background("blue")
# rgb/rgba
c.add_plot_background("#00f")
c.add_plot_background("#00ff")
# rrggbb/rrggbbaa
c.add_plot_background("#0000ff")
c.add_plot_background("#0000ffff")
# rgb(...)
c.add_plot_background("rgb(0, 0, 255)")
c.add_plot_background("rgb(0%, 0%, 100%)")
# rgba(...) (Note how alpha is specified)
c.add_plot_background("rgba(0, 0, 255, 1.0)")
c.add_plot_background("rgba(0%, 0%, 100%, 1.0)")
```

6.2 Specifying Size and Distance

Size may be specified in absolute or relative values. If a size specification is a number between 0 and 1, it is presumed to be a proportion of available space (width or height, as appropriate), as in the following example:

```
# Add a border that is 3% of chart height.
# (c is a chart, b is a border.)
c.add_top_border(b, height=.03)
```

To specify an absolute size, supply a string containing the number, and a valid unit specifier:

```
# Add three pixels of padding to
# the top of the chart
c.add_padding('top', '3px')
# Add three points
c.add_padding('top', '3pt')
```

At this time, there is no practical difference between points and pixels in Rayon, and they may be used interchangeably. In PNG images and GUI canvases, 3px will represent three pixels; in PDF and SVG images, it will represent three points. The same is true of 3pt.

6.3 Specifying Line Styles

The configuration of lines in Rayon may be created from string specifications. The format for a line specification is similar to the CSS specification's shorthand for defining borders, described in <http://www.w3.org/TR/CSS2/box.html#border-shorthand-properties>: width style color, where width is an integer width in device units (points or pixels), style is one of the valid line styles that can be passed to the `new_line` method of the `Toolbox` object (i.e., solid, dashed, dotted and dotdash) and color is a valid color specification, as described in *Specifying Colors*.

BORDERS

Borders are used to decorate the edges of the plotting area in a chart. Borders can be used to frame the plotting area; add tickmarks to a chart; or annotate a chart with labels, titles and captions.

7.1 none

`toolbox.new_border('none'[, <padding>])`

A border which does nothing. Can be used as a placeholder where a border is expected.

See [Padding](#) for more information on padding.

7.2 Tickable borders

The following border objects can decorate a chart with tick marks from a tickset. For more information on creating and using ticksets, see [Tick Sets](#).

7.2.1 hline

`toolbox.new_border('hline'[, ticksets, line, y_offset=0, <padding>])`

Draws a horizontal line on the border.

If supplied, *ticksets* is an iterable of [tick sets](#) that will be drawn along the border. Tick sets may be added after object creation with the `add_tickset` method.

If supplied, *line* is a [line](#) that will be drawn horizontally across the width of the border.

If supplied, *y_offset* describes how far from the top of the border region the line will be drawn. With a *y_offset* of 0, the line will be drawn on the top edge of the border region; with a *y_offset* of 1, it will be drawn at the bottom.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

`add_tickset(tickset)`

Adds *tickset* to the tick sets that will be drawn on this border. Tick sets will be drawn in the order in which they are added.

`iter_ticksets()`

Returns an iterator over the tick sets stored in this object.

get_tickset (*idx*)
Returns the tick set stored at *idx*.

7.2.2 horizontal

toolbox.new_border ('horizontal'[, *ticksets*, <*padding*>])
Draws a simple horizontal border with no decoration.

If supplied, *ticksets* is a an iterable of [tick sets](#) that will be drawn along the border. Tick sets may be added after object creation with the [add_tickset](#) method.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_tickset (*tickset*)
Adds *tickset* to the tick sets that will be drawn on this border. Tick sets will be drawn in the order in which they are added.

iter_ticksets ()
Returns an iterator over the tick sets stored in this object.

get_tickset (*idx*)
Returns the tick set stored at *idx*.

7.2.3 vline

toolbox.new_border ('vline'[, *ticksets*, *line*, *y_offset*=0, <*padding*>])
Draws a vertical line on the border.

If supplied, *ticksets* is a an iterable of [tick sets](#) that will be drawn along the border. Tick sets may be added after object creation with the [add_tickset](#) method.

If supplied, *line* is a [line](#) that will be drawn vertically across the height of the border.

If supplied, *x_offset* describes how far from the right edge of the border region the line will be drawn. With a *x_offset* of 0, the line will be drawn on the far right edge of the border region; with a *x_offset* of 1, it will be drawn on the far left edge.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_tickset (*tickset*)
Adds *tickset* to the tick sets that will be drawn on this border. Tick sets will be drawn in the order in which they are added.

iter_ticksets ()
Returns an iterator over the tick sets stored in this object.

get_tickset (*idx*)
Returns the tick set stored at *idx*.

7.2.4 vertical

toolbox.new_border ('vertical'[, *ticksets*, <*padding*>])
Draws a simple vertical border with no decoration.

If supplied, *ticksets* is a an iterable of [tick sets](#) that will be drawn along the border. Tick sets may be added after object creation with the [add_tickset](#) method.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_tickset (*tickset*)

Adds *tickset* to the tick sets that will be drawn on this border. Tick sets will be drawn in the order in which they are added.

iter_ticksets ()

Returns an iterator over the tick sets stored in this object.

get_tickset (*idx*)

Returns the tick set stored at *idx*.

7.3 Labeled borders

Labeled borders are used to add axis labels, titles, and captions to charts.

7.3.1 hlabel

```
toolbox.new_border ('hlabel', label[, font_size="normal", font_family="default",
                                font_style="normal", font_weight="normal", halign="center",
                                valign="center", color="black", <padding> ])
```

Draws a single static text label (e.g., a title or caption) horizontally in a border.

label is the text label to apply to the border.

font_size, *font_family* and *font_style* take the values described in [Specifying Text Properties](#).

angle is an optional angle of rotation, relative to the label's default horizontal, left-to-right orientation.

halign and *valign* specify, respectively, the horizontal and vertical alignment of the label within the border region.

color is the color in which to draw the text.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

7.3.2 vlabel

```
toolbox.new_border ('vlabel', label[, font_size="normal", font_family="default",
                                font_style="normal", font_weight="normal", halign="center",
                                valign="center", color="black", <spacing_args> ])
```

Draws a single static text label (e.g., a title or caption) vertically in a border.

label is the text label to apply to the border.

font_size, *font_family* and *font_style* take the values described in [Specifying Text Properties](#).

angle is an optional angle of rotation, relative to the label's default vertical, bottom-to-top orientation.

halign and *valign* specify, respectively, the horizontal and vertical alignment of the label within the border region.

color is the color in which to draw the text.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

7.4 Split borders

Split borders subdivide a border into two equally-sized subdivisions. Each subdivision can, itself, contain another border. This can be used in some situations to introduce vertical or horizontal alignment where it is not otherwise natural to do so using the normal layout mechanism.

Note: It is normally a better idea to use the layout facilities documented in [Charts](#) to achieve layout effects; split borders are useful, however, for aligning elements across the layout sections of a chart. (Even then, data and scales should be used if the alignment naturally arises from the data; these classes should only be used to align elements with purely presentational layout, such as the width of columns.)

7.4.1 `hsplit`

`toolbox.new_border('hsplit', children[, <padding>])`

Draws a horizontal border, split into cells arranged from top to bottom. Each contains a child border, which is drawn on the portion of the region allocated to it.

children is a list of borders which will be drawn in the subregions, from left to right. There will be as many subregions as there are children.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

7.4.2 `vsplit`

`toolbox.new_border('vsplit', children[, <padding>])`

Draws a vertical border, split into cells arranged from left to right. Each contains a child border, which is drawn on the portion of the region allocated to it.

children is a list of borders which will be drawn in the subregions, from left to right. There will be as many subregions as there are children.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

CHARTS

A complete visualization consists of one or more plots and possibly other elements, such as tick marks, labels, titles and captions. In Rayon, the *chart* is responsible for Laying these components out visually on the space allocated to it.

8.1 Layout

Rayon charts generally divide the allotted space into *borders*, *titles*, *corners* and the *plotting area*. The plotting area and corners may contain plots, grid lines, backgrounds, foregrounds, or other chart objects. For more information on this layout model, see [Charts](#).

Different charts lay out the plotting area in different ways. See the descriptions of the `square`, `tiled` and `tiled_adv` charts for details.

8.2 Padding

Items added to the plotting area or corners may be associated with padding. This reduces the effective layout area of that element in order to introduce empty space. There are four padding arguments:

- *tpad* indicates how much padding to add to the top of the element
- *bpad* indicates how much padding to add to the bottom of the element
- *lpad* indicates how much padding to add to the left of the element
- *rpadd* indicates how much padding to add to the right of the element

See [Specifying Size and Distance](#) for information on the format of these arguments.

8.3 Charts

Most of the methods on chart objects concern either adding elements to the chart or accessing elements on the chart. Methods for associating elements with parts of the chart that may contain many elements (e.g., borders) begin with the prefix `add_`. Methods for associating elements with parts of the chart that may contain only one element (e.g., backgrounds) begin with the prefix `set_`. Methods for getting elements from the chart are prefixed with `get_`.

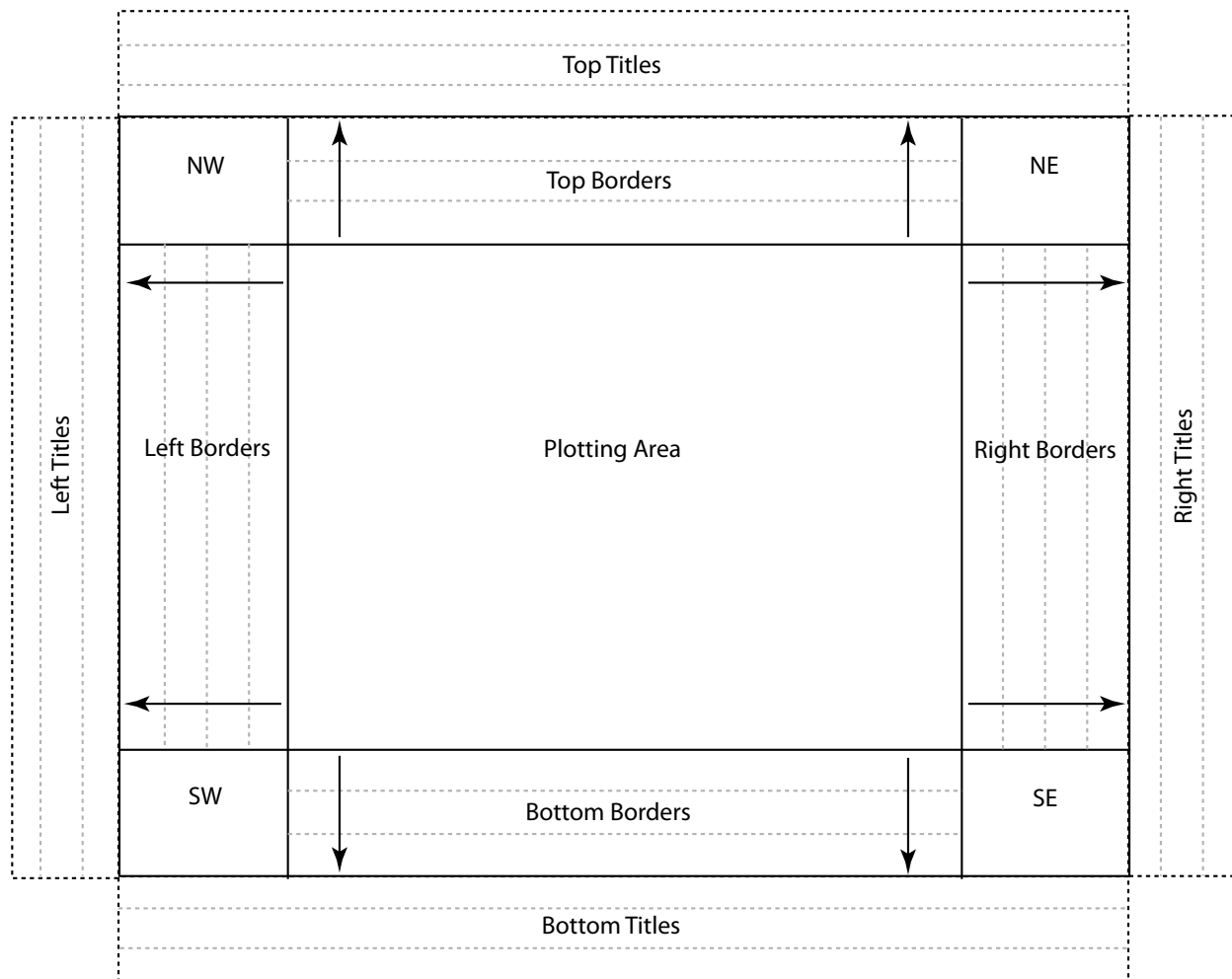


Figure 8.1: Chart layout model

8.3.1 tiled

`toolbox.new_chart('tiled', [num_columns, num_rows, orientation="auto"])`

Subdivides the plotting area into a tiled grid with the specified number of columns and rows, and renders plots and charts into tiles in the grid. All cells have equal width and height. To control the width and height of columns and rows in the grid, use the `tiled_adv` chart.

num_columns, if supplied, is the number of columns in the grid. If not supplied, the grid will have as many columns as needed to tile the required number of cells into *num_rows*.

num_rows, if supplied, is the number of rows in the grid. If not supplied, the grid will have as many rows as needed to tile the required number of cells into *num_columns*.

One of *num_columns* or *num_rows* must be supplied. If both are supplied, the maximum number of cells is the product of *num_columns* and *num_rows*; attempting to add more than that will generate an error.

orientation determines the order in which cells in the grid will be filled, and must be one of the strings `horizontal`, `vertical` or `auto`. Horizontal tiling fills the top row first, from left to right, then the next row left-to-right, and so on until all the cells are filled or no more objects remain to be added. Vertical tiling fills the leftmost column first, from top to bottom, then the next column to the right top-to-bottom, and so on. `auto` indicates that vertical tiling should be used if *col_weights* is specified, and horizontal tiling should be used if *row_weights* is specified. If both *col_weights* and *row_weights* are supplied and *orientation* is `auto`, horizontal tiling will be used.

add_bottom_border (*border*, *height*)

Adds a border to the bottom edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see `add_bottom_title`.

add_bottom_title (*title_border*, *height*)

Adds a border to the bottom edge of the chart. Unlike `add_bottom_border`, this border runs the entire width of the chart, and under any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

add_chart (*chart* [, *name*, <padding>])

Adds a chart object to the plotting area.

chart is the chart to add.

If supplied, *name* is a string that can be used later to retrieve *chart* from this object using `get_chart`.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_left_border (*border*, *width*)

Adds a border to the left edge of the chart.

border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see `add_left_title`.

add_left_title (*title_border*, *width*)

Adds a border to the left edge of the chart. Unlike `add_left_border`, this border runs the entire height of the chart, and to the left of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

add_ne_corner (*element*[, <*padding*>])

Adds an element to the northeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_nw_corner (*element*[, <*padding*>])

Adds an element to the northwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_plot (*plot* : *plot*[, *name* : *str*, <*padding*>])

Adds a plot to the plotting area.

plot is the plot to add.

If supplied, *name* is a string that can be used later to retrieve *plot* from this object using [get_plot](#).

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_right_border (*border*, *width*)

Adds a border to the right edge of the chart.

border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see [add_right_title](#).

add_right_title (*title_border*, *width*)

Adds a border to the right edge of the chart. Unlike [add_right_border](#), this border runs the entire height of the chart, and to the right of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

add_se_corner (*element*[, <*padding*>])

Adds *element* to the southeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_sw_corner (*element*[, <*padding*>])

Adds *element* to the southwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_top_border (*border*, *height*)

Adds a border to the top edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see [add_top_title](#).

add_top_title (*title_border*, *height*)

Adds a border to the top edge of the chart. Unlike `add_top_border`, this border runs the entire width of the chart, and over any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

clear_padding ()

Remove all padding from the chart. This does not remove the padding from the interior components of a chart, only its outer edges.

get_chart (*name*) → chart

Returns a chart associated with this chart in the plotting area.

name is the name given to the associated chart when it was added using `add_chart`. If no chart corresponding to *name* is associated with this chart, this method returns `None`.

get_charts () → list

Gets all chart objects associated with this chart's plotting area.

get_event_source (*gui_event_source* [, *activate=True*]) → event source

Creates and returns a Rayon event source that maps this chart to *gui_event_source*. See [events](#) for more information on event handling in Rayon.

Note: Event handling is only implemented in interactive rendering backends. Non-interactive backends provide this function for consistency, but it does nothing.

gui_event_source is a GUI window or panel which will receive and dispatch UI events. If the backend is non-interactive, the type of *event_source* is unimportant.

If *activate* is `True`, Rayon event source will be created in an activated state – i.e., it will immediately begin dispatching events from *event_source* to the chart. If *activate* is `False`, the user must activate the event source using the source's `activate` method.

get_plot (*name*) → plot

Returns a plot associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_element (*name*) → plot or chart

Returns a plot or chart associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot or chart corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_elements () → list

Returns a list of all chart and plot objects associated with the plotting area.

get_plots () → list

Returns a list of all plot objects associated with the plotting area.

set_chart_background (*background*)

Defines the background image or color for the entire chart. This image or color will be drawn behind any other element, and will cover the entire drawable space for the chart, including borders and titles.

background is a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_chart_foreground (*foreground*)

Defines a foreground image or color for the entire chart. This image or color will be drawn in front of any other element, and will cover the entire drawable space of the chart, including borders and titles. Alpha transparency is observed in both images and colors.

foreground is a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_padding ([*allpad* : *size spec*, <*padding*>])

Sets the amount of padding around the edges of the chart.

If specified, *allpad* is a single size specification (see [Specifying Size and Distance](#)). The amount of padding specified by *allpad* will be applied to all four edges of the canvas. The remaining arguments will be ignored.

tpad, *bpad*, *lpad* and *rpadd* are size specifications indicating the amount of padding that should be applied to the top, bottom, left and right edges of the canvas, respectively.

set_parent (*parent*)

Sets the parent of a chart to *parent* for the purposes of event handling in an interactive rendering backend. Does nothing if the backend is non-interactive.

set_plot_background (*background*[, <*padding*>])

Defines a background image for the plotting area. This image or color will be drawn behind any other element.

background is either a Rayon background object or a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_plot_foreground (*foreground*[, <*padding*>])

Defines a foreground image for the plotting area. This image or color will be drawn behind any other element. Alpha transparency is observed in both images and colors.

foreground is either a Rayon background object or a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

8.3.2 tiled_adv

`toolbox.new_chart` ('tiled_adv'[, *col_weights* : *iterable*, *row_weights* : *iterable*, *orientation*="vertical")

Subdivides the plotting area into a tiled grid according to a set of user-supplied weights, and renders plots and charts into tiles in the grid. This chart offers the user more flexibility in creating the grid than the `tiled` chart, but has a more complicated interface.

If supplied, *col_weights* is an iterable containing the integer weights each column should have. A list `[1, 1, 1]` indicates three columns of equal weight; `[2, 1, 1]` indicates three columns, where the first (left-most) is twice as wide as the other two. If *col_weights* is not supplied, the grid will have as many equally-weighted columns as needed to tile the required number of cells, given the number of rows.

If supplied, *row_weights* is an iterable of integer weights each row should have, with the same semantics as *col_weights*. If not supplied, the grid will have as many equally-weighted rows as needed to tile the required number of cells, given the number of columns.

One of *row_weights* or *col_weights* must be supplied. If both are supplied, the number of rows and columns is fixed, and the maximum number of cells is the product of the number of columns and the number of rows; attempting to add more than that will generate an error.

orientation determines the order in which cells in the grid will be filled, and must be one of the strings `horizontal`, `vertical` or `auto`. Horizontal tiling fills the top row first, from left to right, then the next row left-to-right, and so on until all the cells are filled or no more objects remain to be added. Vertical tiling fills the leftmost column first, from top to bottom, then the next column to the right top-to-bottom, and so on. `auto` indicates that vertical tiling should be used if *col_weights* is specified, and horizontal tiling should be used if *row_weights* is specified. If both *col_weights* and *row_weights* are supplied and *orientation* is `auto`, horizontal tiling will be used.

add_bottom_border (*border*, *height*)

Adds a border to the bottom edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see [add_bottom_title](#).

add_bottom_title (*title_border*, *height*)

Adds a border to the bottom edge of the chart. Unlike [add_bottom_border](#), this border runs the entire width of the chart, and under any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

add_chart (*chart*[, *name*, <*padding*>])

Adds a chart object to the plotting area.

chart is the chart to add.

If supplied, *name* is a string that can be used later to retrieve *chart* from this object using [get_chart](#).

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_left_border (*border*, *width*)

Adds a border to the left edge of the chart.

border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see [add_left_title](#).

add_left_title (*title_border*, *width*)

Adds a border to the left edge of the chart. Unlike [add_left_border](#), this border runs the entire height of the chart, and to the left of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

add_ne_corner (*element*[, <*padding*>])

Adds an element to the northeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_nw_corner (*element*[, <*padding*>])

Adds an element to the northwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_plot (*plot* : *plot*[, *name* : *str*, <*padding*>])

Adds a plot to the plotting area.

plot is the plot to add.

If supplied, *name* is a string that can be used later to retrieve *plot* from this object using `get_plot`.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_right_border (*border*, *width*)

Adds a border to the right edge of the chart.

border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see `add_right_title`.

add_right_title (*title_border*, *width*)

Adds a border to the right edge of the chart. Unlike `add_right_border`, this border runs the entire height of the chart, and to the right of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

add_se_corner (*element*[, <*padding*>])

Adds *element* to the southeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_sw_corner (*element*[, <*padding*>])

Adds *element* to the southwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_top_border (*border*, *height*)

Adds a border to the top edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see `add_top_title`.

add_top_title (*title_border*, *height*)

Adds a border to the top edge of the chart. Unlike `add_top_border`, this border runs the entire width of the chart, and over any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

clear_padding ()

Remove all padding from the chart. This does not remove the padding from the interior components of a chart, only its outer edges.

get_chart (*name*) → *chart*

Returns a chart associated with this chart in the plotting area.

name is the name given to the associated chart when it was added using `add_chart`. If no chart corresponding to *name* is associated with this chart, this method returns `None`.

get_charts () → list

Gets all chart objects associated with this chart's plotting area.

get_event_source (*gui_event_source* [, *activate=True*]) → event source

Creates and returns a Rayon event source that maps this chart to *gui_event_source*. See *events* for more information on event handling in Rayon.

Note: Event handling is only implemented in interactive rendering backends. Non-interactive backends provide this function for consistency, but it does nothing.

gui_event_source is a GUI window or panel which will receive and dispatch UI events. If the backend is non-interactive, the type of *event_source* is unimportant.

If *activate* is `True`, Rayon event source will be created in an activated state – i.e., it will immediately begin dispatching events from *event_source* to the chart. If *activate* is `False`, the user must activate the event source using the source's `activate` method.

get_plot (*name*) → plot

Returns a plot associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_element (*name*) → plot or chart

Returns a plot or chart associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot or chart corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_elements () → list

Returns a list of all chart and plot objects associated with the plotting area.

get_plots () → list

Returns a list of all plot objects associated with the plotting area.

set_chart_background (*background*)

Defines the background image or color for the entire chart. This image or color will be drawn behind any other element, and will cover the entire drawable space for the chart, including borders and titles.

background is a color specification. (See *Specifying Colors*.)

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

set_chart_foreground (*foreground*)

Defines a foreground image or color for the entire chart. This image or color will be drawn in front of any other element, and will cover the entire drawable space of the chart, including borders and titles. Alpha transparency is observed in both images and colors.

foreground is a color specification. (See *Specifying Colors*.)

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

set_padding ([*allpad* : *size spec*, <*padding*>])

Sets the amount of padding around the edges of the chart.

If specified, *allpad* is a single size specification (see *Specifying Size and Distance*). The amount of padding specified by *allpad* will be applied to all four edges of the canvas. The remaining arguments will be ignored.

tpad, *bpad*, *lpad* and *rpad* are size specifications indicating the amount of padding that should be applied to the top, bottom, left and right edges of the canvas, respectively.

set_parent (*parent*)

Sets the parent of a chart to *parent* for the purposes of event handling in an interactive rendering backend. Does nothing if the backend is non-interactive.

set_plot_background (*background*[, <*padding*>])

Defines a background image for the plotting area. This image or color will be drawn behind any other element.

background is either a Rayon background object or a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_plot_foreground (*foreground*[, <*padding*>])

Defines a foreground image for the plotting area. This image or color will be drawn behind any other element. Alpha transparency is observed in both images and colors.

foreground is either a Rayon background object or a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

8.3.3 square

toolbox.new_chart ('square')

Renders each plot in a series of layers, each of which has the full dimension of the plotting area. Plots will be drawn in the order in which they were added, with more recently-added ones drawn “on top”.

add_bottom_border (*border*, *height*)

Adds a border to the bottom edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see [add_bottom_title](#).

add_bottom_title (*title_border*, *height*)

Adds a border to the bottom edge of the chart. Unlike [add_bottom_border](#), this border runs the entire width of the chart, and under any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

add_chart (*chart*[, *name*, <*padding*>])

Adds a chart object to the plotting area.

chart is the chart to add.

If supplied, *name* is a string that can be used later to retrieve *chart* from this object using [get_chart](#).

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_front_gridlines (*grid*[, <*padding*>])

Adds grid lines to the plotting area. The grid lines will appear on top of any other elements in the plotting area (except grid lines that are subsequently added with this method). Grid lines will be drawn in the order they were added to the chart, with the most recently added ones drawn on top.

grid is a Rayon grid lines object.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

See Also:*Grid lines***add_left_border** (*border*, *width*)

Adds a border to the left edge of the chart.

border is a Rayon border object. (See *Borders*.)

width is a size specification (see *Specifying Size and Distance*) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see `add_left_title`.

add_left_title (*title_border*, *width*)

Adds a border to the left edge of the chart. Unlike `add_left_border`, this border runs the entire height of the chart, and to the left of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See *Borders*.)

width is a size specification (see *Specifying Size and Distance*) of the border width.

add_ne_corner (*element*[, <*padding*>])

Adds an element to the northeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

add_nw_corner (*element*[, <*padding*>])

Adds an element to the northwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

add_plot (*plot* : *plot*[, *name* : *str*, <*padding*>])

Adds a plot to the plotting area.

plot is the plot to add.

If supplied, *name* is a string that can be used later to retrieve *plot* from this object using `get_plot`.

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

add_rear_gridlines (*grid*[, <*padding*>])

Adds grid lines to the plotting area. The grid lines will appear behind any other elements in the plotting area (except grid lines that have already been added with this method). Grid lines will be drawn in the order they were added to the chart, with the most recently added ones drawn on top.

grid is a Rayon grid lines object.

padding represents zero or more padding arguments. (See *Padding* for more information on padding.)

See Also:*Grid lines***add_right_border** (*border*, *width*)

Adds a border to the right edge of the chart.

border is a Rayon border object. (See *Borders*.)

width is a size specification (see *Specifying Size and Distance*) of the border width.

This border runs from the top edge of the plot area to the bottom edge. To add a border that spans the whole height of the chart, see `add_right_title`.

add_right_title (*title_border*, *width*)

Adds a border to the right edge of the chart. Unlike `add_right_border`, this border runs the entire height of the chart, and to the right of any top or bottom borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

width is a size specification (see [Specifying Size and Distance](#)) of the border width.

add_se_corner (*element*[, <*padding*>])

Adds *element* to the southeast corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_sw_corner (*element*[, <*padding*>])

Adds *element* to the southwest corner of the chart.

element is a Rayon plot or chart.

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

add_top_border (*border*, *height*)

Adds a border to the top edge of the chart.

border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

This border runs from the left edge of the plot area to the right edge. To add a border that spans the whole width of the chart, see `add_top_title`.

add_top_title (*title_border*, *height*)

Adds a border to the top edge of the chart. Unlike `add_top_border`, this border runs the entire width of the chart, and over any left or right borders the chart may have.

title_border is a Rayon border object. (See [Borders](#).)

height is a size specification (see [Specifying Size and Distance](#)) of the border height.

clear_padding ()

Remove all padding from the chart. This does not remove the padding from the interior components of a chart, only its outer edges.

get_chart (*name*) → chart

Returns a chart associated with this chart in the plotting area.

name is the name given to the associated chart when it was added using `add_chart`. If no chart corresponding to *name* is associated with this chart, this method returns `None`.

get_charts () → list

Gets all chart objects associated with this chart's plotting area.

get_event_source (*gui_event_source*[, *activate=True*]) → event source

Creates and returns a Rayon event source that maps this chart to *gui_event_source*. See [events](#) for more information on event handling in Rayon.

Note: Event handling is only implemented in interactive rendering backends. Non-interactive backends provide this function for consistency, but it does nothing.

gui_event_source is a GUI window or panel which will receive and dispatch UI events. If the backend is non-interactive, the type of *event_source* is unimportant.

If *activate* is `True`, Rayon event source will be created in an activated state – i.e., it will immediately begin dispatching events from *event_source* to the chart. If *activate* is `False`, the user must activate the event source using the source’s `activate` method.

get_plot (*name*) → plot

Returns a plot associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_element (*name*) → plot or chart

Returns a plot or chart associated with this chart in the plotting area.

name is the name given to the plot or chart when it was added using `add_plot`. If no plot or chart corresponding to *name* is associated with this chart, this method returns `None`.

get_plot_elements () → list

Returns a list of all chart and plot objects associated with the plotting area.

get_plots () → list

Returns a list of all plot objects associated with the plotting area.

set_chart_background (*background*)

Defines the background image or color for the entire chart. This image or color will be drawn behind any other element, and will cover the entire drawable space for the chart, including borders and titles.

background is a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_chart_foreground (*foreground*)

Defines a foreground image or color for the entire chart. This image or color will be drawn in front of any other element, and will cover the entire drawable space of the chart, including borders and titles. Alpha transparency is observed in both images and colors.

foreground is a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_padding ([*allpad* : size spec, <padding>])

Sets the amount of padding around the edges of the chart.

If specified, *allpad* is a single size specification (see [Specifying Size and Distance](#)). The amount of padding specified by *allpad* will be applied to all four edges of the canvas. The remaining arguments will be ignored.

tpad, *bpad*, *lpad* and *rpadd* are size specifications indicating the amount of padding that should be applied to the top, bottom, left and right edges of the canvas, respectively.

set_parent (*parent*)

Sets the parent of a chart to *parent* for the purposes of event handling in an interactive rendering backend. Does nothing if the backend is non-interactive.

set_plot_background (*background*[, <padding>])

Defines a background image for the plotting area. This image or color will be drawn behind any other element.

background is either a Rayon background object or a color specification. (See [Specifying Colors](#).)

padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

set_plot_foreground (*foreground*[, <padding>])

Defines a foreground image for the plotting area. This image or color will be drawn behind any other element. Alpha transparency is observed in both images and colors.

foreground is either a Rayon background object or a color specification. (See [Specifying Colors](#).)
padding represents zero or more padding arguments. (See [Padding](#) for more information on padding.)

DATA

This chapter describes the objects Rayon provides for importing, exporting and manipulating data. These objects were not designed for general-purpose numeric analysis on large datasets, but rather for moving data into and out of Rayon for visualization with reasonable efficiency.

For more information on datasets (including the syntax of Rayon’s text-based data format), see *Manipulating Data* in *Building Visualizations*.

9.1 Datasets

```
toolbox.new_dataset_from_stream(stream[, colnames : list, typemap : dict, delimiter="|"])
toolbox.new_dataset_from_filename(filename[, colnames : list, typemap : dict, delimiter="|"])
toolbox.new_dataset_from_columns(columns[, colnames : list, header : list, sortkey : function])
toolbox.new_dataset_from_rows(rows[, colnames : list, header : list, sortkey : function])
```

`Dataset` objects are collections of columnar data, represented by `Column` objects. One can iterate over the data either row-wise or column-wise; `Row` objects provide a row-based view of a `Dataset` object’s data. `Dataset` objects may be created using the `new_dataset_from_filename` (for filenames) or `new_dataset_from_stream` (for file or file-like objects like the `StringIO.StringIO` object) methods of the `Toolbox` object.

`Dataset` objects are mutable; new columns can be added or removed. Since `Column` objects are immutable, however, rows cannot be modified. It is possible to create new `Dataset` objects from existing ones with filtered data.

`Dataset` objects may also be non-destructively sorted by adding sort key functions. The native sort order of the object is the order of the underlying columns.

Metadata about `Dataset` objects is stored in key-value pairs called *headers*. The keys and values for headers must both be strings. Headers can be added, removed or modified as desired.

append_column (*new_col* : *Column* or *iterable* [, *name* : *str*])

Append *new_col* to this object. *new_col* must be either a `Column` object or an iterable sequence (list, tuple, iterator, etc.).

If this `Dataset` object already contains columns, *new_col* must have the same number of elements as the existing columns. *new_col* is appended to the “right” of the other columns, e.g.:

```
>>> # t is a Toolbox object
>>> raw = [[1,2,3], ['a', 'b', 'c'], [3,2,1]]
>>> d = t.new_dataset_from_columns(raw)
>>> d.append_column([4,5,6])
>>> d.get_num_columns()
```

```
4
>>> [i for i in d.get_column(3)]
[4, 5, 6]
```

It is an error to append a column to a `Dataset` object without a column name if the `Dataset` object already contains column names:

```
>>> # t is a Toolbox object
>>> raw = [[1,2,3], ['a', 'b', 'c'], [3,2,1]]
>>> colnames = ['foo', 'bar', 'baz']
>>> d = t.new_dataset_from_columns(raw, colnames=colnames)
>>> d.append_column([4,5,6])
RayonDataException: must specify a column name
```

The column will be inserted in native sort order, regardless of the current sort order:

```
>>> # t is a Toolbox object
>>> raw = [[1,2,3], ['a', 'b', 'c'], [3,2,1]]
>>> d = t.new_dataset_from_columns(raw)
>>> d.sort(lambda r: r[2])
>>> d.append_column([4,5,6])
>>> d.sort()
>>> [[i for i in r] for r in d]
[[3, 'c', 1, 6], [2, 'b', 2, 5], [1, 'a', 3, 4]]
```

To append a column to a sorted `Dataset` object and have it appear in sorted order, first clone the `Dataset` object using `filter`, then append the column to the new `Dataset`:

```
>>> from rayon.data import *
>>> raw = [[1,2,3], ['a', 'b', 'c'], [3,2,1]]
>>> d = Dataset.from_columns(raw)
>>> d.sort(lambda r: r[2])
>>> d2 = d.filter(True)
>>> d2.append_column(Column[4,5,6])
>>> [[i for i in r] for r in d2]
[[3, 'c', 1, 4], [2, 'b', 2, 5], [1, 'a', 3, 6]]
```

del_header (*name* : *str*)

Remove the header named *name* from the list of headers. If *name* does not exist, this method does nothing.

filter_both (*filter_fn* [, *insert_sorted*=*False*, *copy_sort_key*=*True*, *copy_header*=*True*, *copy_colnames*=*True*, *limit* : *int*, *fail_limit* : *int*]) → 2-tuple of `Dataset`

Produces a tuple of two new `Dataset` objects based on this one, with rows passing *filter_fn* in the first, and rows failing *filter_fn* in the second. This method is equivalent to calling `a, b = (data.filter_pass(fn, ...), data.filter_fail(fn, ...))`, but is more efficient.

filter_fn is a function taking a *row* object and returning `True` or `False`; *filter_fn* may also be one of the values `True` or `False`, meaning to pass (`True`) or fail (`False`) every row in the dataset.

If *insert_sorted* is `True`, the native sort order of the filtered datasets will be the parent dataset's current sort order. If *insert_sorted* is `False`, the native sort order of the filtered datasets will be the parent dataset's native sort order, regardless of the current sort order.

If *copy_sort_key* is `True`, the current sort order from the parent dataset will be copied into the filtered datasets. This does not change the native sort order.

If `copy_header` is `True`, the parent dataset's headers will be copied into the output datasets. Otherwise, the output datasets will have no headers.

If `copy_colnames` is `True`, the parent dataset's column names will be copied into the output datasets. Otherwise, the output datasets will have no column names.

If supplied, `limit` is the maximum number of passing rows to return from this filter. By default, all rows matching `filter_fn` will be returned in the first `Dataset` object.

If supplied, `fail_limit` is the maximum number of failing rows to return from this filter. By default, all rows failing `filter_fn` will be returned in the second `Dataset` object.

See Also:

`filter_fail`, `filter_pass`

filter_fail (`filter_fn`[, `insert_sorted=False`, `copy_sort_key=True`, `copy_header=True`, `copy_colnames=True`, `limit: int`]) → `Dataset`

Produces a new `Dataset` based on this one, containing only rows failing `filter_fn`.

`filter_fn` is a function taking a `row` object and returning `True` or `False`; `filter_fn` may also be one of the values `True` or `False`, meaning to pass (`True`) or fail (`False`) every row in the dataset.

If `insert_sorted` is `True`, the native sort order of the filtered dataset will be the parent dataset's current sort order. If `insert_sorted` is `False`, the native sort order of the filtered dataset will be the parent dataset's native sort order, regardless of the current sort order.

If `copy_sort_key` is `True`, the current sort order from the parent dataset will be copied into the filtered dataset. This does not change the native sort order.

If `copy_header` is `True`, the parent dataset's headers will be copied into the output dataset. Otherwise, the output dataset will have no headers.

If `copy_colnames` is `True`, the parent dataset's column names will be copied into the output dataset. Otherwise, the output dataset will have no column names.

If supplied, `limit` is the maximum number of rows to return from this filter. By default, all rows matching `filter_fn` will be returned.

See Also:

`filter_both`, `filter_pass`

filter_pass (`filter_fn`[, `insert_sorted=False`, `copy_sort_key=True`, `copy_header=True`, `copy_colnames=True`, `limit: int`]) → `Dataset`

Produces a new `Dataset` based on this one, containing only rows passing `filter_fn`.

`filter_fn` is a function taking a `row` object and returning `True` or `False`; `filter_fn` may also be one of the values `True` or `False`, meaning to pass (`True`) or fail (`False`) every row in the dataset.

If `insert_sorted` is `True`, the native sort order of the filtered dataset will be the parent dataset's current sort order. If `insert_sorted` is `False`, the native sort order of the filtered dataset will be the parent dataset's native sort order, regardless of the current sort order.

If `copy_sort_key` is `True`, the current sort order from the parent dataset will be copied into the filtered dataset. This does not change the native sort order.

If `copy_header` is `True`, the parent dataset's headers will be copied into the output dataset. Otherwise, the output dataset will have no headers.

If `copy_colnames` is `True`, the parent dataset's column names will be copied into the output dataset. Otherwise, the output dataset will have no column names.

If supplied, *limit* is the maximum number of rows to return from this filter. By default, all rows matching *filter_fn* will be returned.

See Also:

`filter_both`, `filter_fail`

flatten (*new_colnames* : *iterable*) → Dataset

Creates a copy of this dataset, converting any composite `Column` objects into one `Column` object for each element in the composite. Note that this may change the column indices of subsequent columns.

If *new_colnames* is supplied, it should be an iterable of strings, which will be used as column names for the new dataset, with the first new column getting the first name from *new_colnames*, and so on. If the number of column names in *new_colnames* does not match the number of columns in the flattened dataset, a `RayonDataException` will be raised. If *new_colnames* is `None` (the default), this method will try to use the names of the parent dataset as the names of the output dataset. (This will only be possible if the `Dataset` object contains no melded columns.) If that isn't possible, the output dataset will not contain column names.

See Also:

`meld`

get_column (*colname_or_index* : *int or str* [, *ignore_sort=False*]) → Column

Given a column index or name, returns the corresponding `Column` object.

colname_or_index is either the index (an integer) or name (a string) referring to the column.

If *ignore_sort* is `False`, return the column in the order of the current sort. Otherwise, the column will be sorted in insert order, even if the dataset is currently sorted by some key.

get_column_index_from_name (*name* : *int or str*)

Returns the numeric index corresponding to a column referenced by *name*. *name* can be numeric index, in which case it is simply returned if it refers to an index in this dataset. (Consequently, this method can be used to convert a value that may be a name or an index to an index.)

It is an error to call this method on an empty dataset.

get_column_name_from_index (*index* : *int or str*)

Returns the name corresponding to a column referenced by *idx*. *idx* can be a name, in which case it is simply returned if it refers to a column in this dataset. (Consequently, this method can be used to convert a value that may be a name or an index to a name. Note, however, that it is an error to call this method if the dataset has no column names.)

get_column_names () → list of str

Returns a list of names of all columns in the dataset, in order from lowest index to highest. If no column names are specified, this method returns an empty list.

If *synthesize_indices* is `True`, this method will return a list of synthetic indices that can be used to access the columns in the dataset. This guarantees that the return value of this method may be used to access the columns of the dataset.

get_header (*name* : *str*) → str

Gets the value of the header named *name*

get_headers () → iterator

Returns an iterator of 2-tuples over the headers of this dataset. Each 2-tuple is a (*name*, *value*) pair representing one header. Order is undefined.

get_num_columns () → int

Returns the number of columns in the dataset

get_num_rows() → int

Returns the number of rows in the dataset.

get_row(index: int[, ignore_sort=False]) → Row

Returns the row in the dataset corresponding to *index*. Negative numbers index from the end of the dataset, as in Python lists.

If *ignore_sort* is `False`, return the row corresponding to *index* according to the dataset's current sort order. If *ignore_sort* is `True`, return the row corresponding to *index* according to the dataset's native sort order.

iter_columns() → iterator

Iterate over the columns in the dataset, in order from the lowest index to the highest.

iter_ignore_sorted() → iterator

Iterate over the rows in the dataset according to the dataset's native sort order, regardless of the current sort order. (To get an iterator over the rows of the dataset in the current sort order, use Python's `iter` function.)

meld([*names_or_indices]) → MultiColumn

Creates a single composite `Column` object from the multiple columns referenced in *names_or_indices*. This is useful, for example, in data where several columns together form a unique key.

If exactly one name/index is supplied, this method returns a normal `Column` object, and is equivalent to calling the `get_column` method with that name/index.

If *names_or_indices* is not supplied, a `Column` object will be created consisting of all columns in the dataset.

partition(key_colname_or_index[, insert_sorted=True, copy_header=True, copy_colnames=True]) → [(keyval, Dataset)]

Creates several new `Dataset` objects, partitioned along the unique values of one or more of the parent dataset's columns. The new datasets are returned as a tuple of pairs; the first pair is the common value in the partitioning column; the second is a new dataset containing the matching rows.

The following examples illustrate the use of `partition` on a dataset *d* with the following rows:

```
red   | 1 | 0
red   | 1 | 1
green | 1 | 2
green | 1 | 3
blue  | 1 | 4
blue  | 1 | 5
```

The following example partitions *d* on the first column, producing three datasets (one for each color), each containing two rows:

```
>>> from pprint import pprint
>>> p0 = d.partition(0)
>>> pprint(p0)
(('red', <rayon.data.Dataset object at 0x...>),
 ('green', <rayon.data.Dataset object at 0x...>),
 ('blue', <rayon.data.Dataset object at 0x...>))
>>> print(p0[0][1].to_string())
red|1|0
red|1|1
```

Partitioning *d* on the second column produces a single dataset containing all the rows:

```
>>> p1 = d.partition(1)
>>> pprint(p1)
((1, <rayon.data.Dataset object at 0x10058eb90>),)
>>> print p1[0][1].to_string()
red|1|0
red|1|1
green|1|2
green|1|3
blue|1|4
blue|1|5
```

As each value in the third column is unique, partitioning *d* on this column produces produces six datasets of one row each:

```
>>> p2 = d.partition(2)
>>> pprint(p2)
((0, <rayon.data.Dataset object at 0x...>),
 (1, <rayon.data.Dataset object at 0x...>),
 (2, <rayon.data.Dataset object at 0x...>),
 (3, <rayon.data.Dataset object at 0x...>),
 (4, <rayon.data.Dataset object at 0x...>),
 (5, <rayon.data.Dataset object at 0x...>))
>>> print p2[0][1].to_string()
red|1|0
>>> print p2[1][1].to_string()
red|1|1
>>> print p2[2][1].to_string()
green|1|2
```

You can partition on multiple columns. To do so, pass in a sequence of column names or indices. In this case, the key will be a tuple of unique combinations of the requested key columns, in the order in which they were passed in:

```
>>> p3 = d.partition((0, 1))
>>> pprint(p3)
((('red', 1), <rayon.data.Dataset object at 0x...>),
 (('green', 1), <rayon.data.Dataset object at 0x...>),
 (('blue', 1), <rayon.data.Dataset object at 0x...>))
>>> p4 = d.partition((1, 0))
>>> pprint(p4)
((1, 'red'), <rayon.data.Dataset object at 0x...>),
(1, 'green'), <rayon.data.Dataset object at 0x...>),
(1, 'blue'), <rayon.data.Dataset object at 0x...>))
```

The result of partitioning a *Dataset* object on a zero-length sequence is a single dataset containing all the rows of the original:

```
>>> d.partition(tuple())
(((), <rayon.data.Dataset object at 0x1006a4690>),)
>>> print d.partition(tuple())[0][1].to_string()
red|1|0
red|1|1
green|1|2
green|1|3
```

```
blue|1|4
blue|1|5
```

key_colname_or_index is the name or index of the column on which to partition, or a tuple of names or indices, as described above.

If *insert_sorted* is `True`, the rows will be inserted into the partitioned datasets in the order indicated by the parent dataset's current sort order. If *insert_sorted* is `False`, the rows will be inserted into the partitioned datasets according to the parent dataset's native sort order, regardless of the current sort order.

If *copy_header* is `True`, the parent dataset's headers will be copied into the output dataset. Otherwise, the output dataset will have no headers.

If *copy_colnames* is `True`, the parent dataset's column names will be copied into the output dataset. Otherwise, the output dataset will have no column names.

This function returns a list of 2-tuples (*keyval*, *dataset*), where *keyval* is one of the unique values in the column referred to by *key_colname_or_index*, and *dataset* is a `Dataset` object containing the rows in the parent dataset that contain that value in *key_colname_or_index*.

set_column_names (*colnames* : list of str)

Replaces the `Dataset` object's column names (if they exist) with *colnames*.

colnames is an iterable of strings. The number of elements in *colnames* should correspond to the number of columns in the dataset.

set_header (*name* : str, *value* : str)

Sets the value of the header named *name* to *value*

sort ([*key* : function])

Changes the current sort order according to the *key*. If *key* is `None`, revert to native sort order.

key is a function (or callable object) taking a *row* object and returning a comparable value. (That is, a value for which the `<`, `>`, `<=`, `>=` and `==` operators have meaning, that can be compared with others of its type to determine ordering.) *key* can also be a class whose constructor takes a *row* object and returns an instance suitable for comparison. (i.e., defining `__lt__`, `__gt__`, etc.)

Some examples:

```
# Sorts on the first 2 columns of d
d.sort(key=lambda r: (r[0], r[1]))
```

```
# This is equivalent to the above...
```

```
class Comparator(object):
    def __init__(self, r):
        self.to_cmp = (r[0], r[1])
    def __lt__(self, other):
        return self < other
    def __gt__(self, other):
        return self > other
    def __le__(self, other):
        return self <= other
    def __ge__(self, other):
        return self >= other
    def __eq__(self, other):
        return self == other
    def __ne__(self, other):
        return self != other
```

```
d.sort(key=Comparator)

# ...as is this
def CallableComparator(object):
    def __call__(self, row):
        return r[0], r[1]

d.sort(key=CallableComparator())
```

to_file (*fname* : str[, *delimiter*="|"])

Writes the contents of this dataset to a file. The file will be created if it does not exist, and will be overwritten if it does exist. The format of the output is described in *On-disk format*.

Generally, this method supports writing any opaque datatype that can be represented as a string, provided that the string representation does not contain the character used as the field delimiter.

At this time, composite columns (such as those returned by the `meld` method) are *not* supported; to write datasets with melded columns to disk, first flatten them using the `flatten` method.

This method is equivalent to calling:

```
# d is the dataset
return d.to_stream(open(fname, 'w'))
```

fname is the name of the file to write.

delimiter should be a single character that will be used in the stream to delimit fields in a record.

to_stream (*stream* : fobj[, *delimiter*="|"])

Writes the contents of a dataset to a stream. The format of the output is described in *On-disk format*.

Generally, this method supports writing any opaque datatype that can be represented as a string, provided that the string representation does not contain the character used as the field delimiter.

At this time, composite columns (such as those returned by the `meld` method) are *not* supported; to write datasets with melded columns to disk, first flatten them using the `flatten` method.

stream is a file-like object that supports writing.

delimiter should be a single character that will be used in the stream to delimit fields in a record.

to_string ([*delimiter*="|"]) → str

Writes the contents of this dataset to a string, then returns the string.

Generally, this method supports writing any opaque datatype that can be represented as a string, provided that the string representation does not contain the character used as the field delimiter.

At this time, composite columns (such as those returned by the `meld` method) are *not* supported; to write datasets with melded columns to disk, first flatten them using the `flatten` method.

This method is equivalent to calling:

```
# d is the dataset
sio = StringIO()
d.to_stream(sio)
return sio
```

delimiter should be a single character that will be used in the stream to delimit fields in a record.

9.2 Columns

`toolbox.new_column_from_data(raw_data : iterable)`

`toolbox.new_column_from_constant(constant, length : int)`

`Column` objects are immutable collections of columnar data. `Column` objects are used by the `Dataset` object to store its data and are returned from some methods of the `Dataset` object. `Column` objects may also be created using the `new_column_from_data` and `new_column_from_constant` methods of the `Toolbox` object.

For purposes of item access, membership testing and iteration, `Column` objects can be treated as Python tuples. `Column` objects may also be added to each other if their data types match. If the data in the `Column` object is numeric, several methods can provide statistical information.

max () → number

Returns the largest element in the column data.

If the data in this column is not numeric, this method will raise an error.

mean () → number

Returns the mean of the data in the column.

If the data in this column is not numeric, this method will raise an error.

min () → number

Returns the smallest element in the column data.

If the data in this column is not numeric, this method will raise an error.

percentile (*p* : int) → number

Returns the value of the *p* th percentile element in the column data. *p* should be an integer between 1 and 100.

If the data in this column is not numeric, this method will raise an error.

sample_stdev () → number

Returns the sample standard deviation of the column data. (The data is assumed to represent a sample of a larger population.)

If the data in this column is not numeric, this method will raise an error.

sample_variance () → number

Returns the sample variance of the data. (The data is assumed to represent a sample of a larger population.)

If the data in this column is not numeric, this method will raise an error.

stdev () → number

Returns the population standard deviation of the column data. (The data is assumed to represent the total population.)

If the data in this column is not numeric, this method will raise an error.

uniq () → list

Returns a list of all unique values in column. If `return_sorted` is `True`, return the items in sorted order. Otherwise, the are returned in the order in which they appear in the column.

variance () → number

Returns the population variance of the data in the column data. (The data is assumed to represent the total population.)

If the data in this column is not numeric, this method will raise an error.

9.3 Rows

Row objects provide a row-major “view” on a `Dataset` object’s data. They provide access to data by index and (when a dataset has column names) by name, e.g.:

```
# d is a dataset with three columns,  
# named "foo", "bar", and "baz".  
# the first row of d is ['a', 1, 'y']  
>>> r.get_row(0)  
>>> r[0]  
'a'  
>>> r.foo  
'a'
```

GRID LINES

Grid lines add reference lines to the plotting area. *A chart with grid lines* is an example of a chart with grid lines:

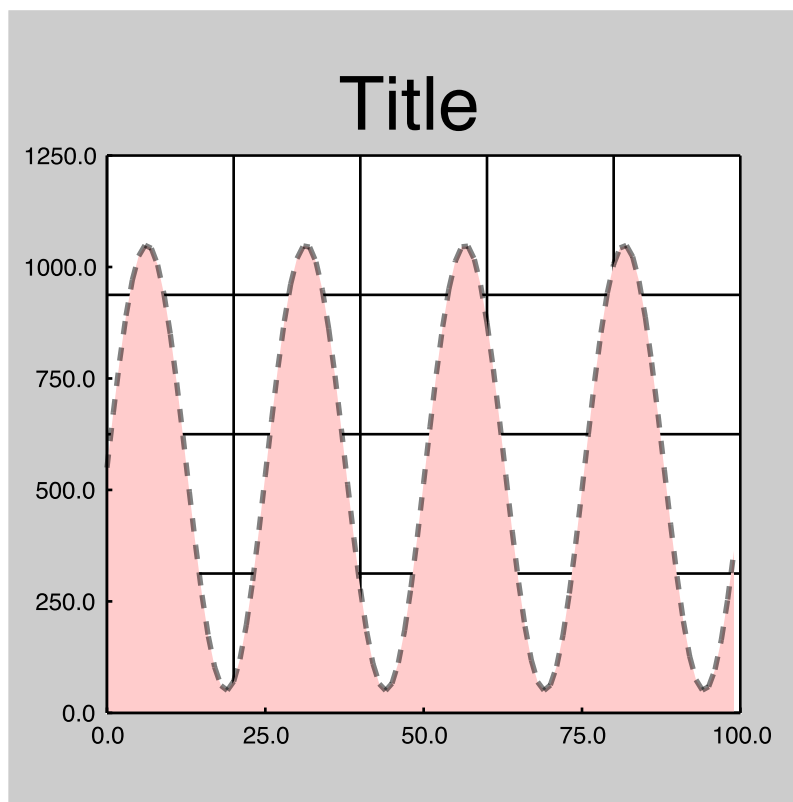


Figure 10.1: A chart with grid lines

Note that two sets of grid lines are used in this chart – one for the horizontal lines and one for the vertical lines.

Grid lines are actually special *plots* that draw lines perpendicular to its one spatial axis. Grid lines have the following axes:

Table 10.1: Axes

Axis	Value	Rendering	Default Scale
pos	A real number between 0 and 1	Spatial position	No default
line_style	String	Style of line to use (equivalent to nicknames passed to the <code>new_line</code> method).	solid
line_width	Number	Width of line in points/pixels	1
line_color	A color specification (see <i>Specifying Colors</i>)	Color of line	#000000FF
ticksets	List	<i>Tick sets</i> to draw on the line	Empty list

Grid lines are added to the `chart` via the `add_front_gridlines` and `add_back_gridlines` methods

10.1 Creating grid lines

The most straightforward way to create grid lines is via the `new_gridlines` method. The resulting grid lines require further configuration; at a minimum, the user must supply a scale (and probably data) for the `pos` axis. A minimal example:

```
vgrid = tools.new_gridlines(  
    'vertical',  
    pos_data=[10,20,30],  
    pos_scale=tools.new_scale('linear'))
```

To customize the display of grid lines, supply different scales (or data) to the grid line's axes:

```
vgrid = tools.new_gridlines(  
    'vertical',  
    pos_data=[10,20,30],  
    pos_scale=tools.new_scale('linear'),  
    line_style_scale="dashed",  
    line_width_scale=5)
```

10.1.1 Creating grid lines from tick marks

It is common to put grid lines in regular positions along an axis, often the same places where tick marks are placed. In this case, it may be easier to create grid lines using `new_gridlines_from_tick_tuples`:

```
# ticks is a Rayon tick set  
vgrid = tools.new_gridlines_from_tick_tuples(  
    'vertical',  
    scale=ticks.get_scale(),  
    tick_tuples=iter(ticks))
```

10.1.2 Creating grid lines from a border

The `new_gridlines_from_border` method takes a tickable border and puts grid lines at the positions of a tickset associated with the border. This further simplifies a common case, where grid lines parallel tick marks on a

border:

```
# b is a Rayon border
vgrid = tools.new_gridlines_from_border(
    'vertical', b)
```

10.2 Adding tick marks to grid lines

Grid lines can not only be created from tick marks; tick marks can also be associated with grid lines. The simplest way to associate tick sets is by specifying a constant scale to the `ticksets` axis. More than one tick set may be specified in this way, leading Rayon to draw multiple sets of tick marks on the same line:

```
# t1 and t2 are tick sets
vgrid = tools.new_gridlines(..., ticksets_scale=[t1, t2])
```

One drawback to this approach is that the same sets of ticks will be drawn on every line in the grid lines. To draw different ticks on different lines, we must specify data and a scale for the `ticksets` axis:

```
# col is a column with two elements
# t1 and t2 are tick sets.
vgrid = tools.new_gridlines(...,
    pos_data=col,
    ticksets_data = [t1, t2],
    ticksets_scale = lambda x: x)
```

In the above example, the scale merely passes the data through without transformation. It is easy to conceive of more complicated scales. For instance, the following returns an empty list (no tick sets) if the tickset data is `None`:

```
def ticksetscale(x):
    if x is None: return []
    return x
```

10.3 horizontal

```
toolbox.new_gridlines('horizontal', [...])
toolbox.new_gridlines_from_border('horizontal', border[, tickset_idx=0, ...])
toolbox.new_gridlines_from_tick_tuples('horizontal', scale, tick_tuples[, ...])
    Draws horizontal lines across the plotting area.
```

Table 10.2: Axes

Axis	Value	Rendering	Default Scale
pos	A real number between 0 and 1	Spatial position on the Y (vertical) axis	No default
line_style	String	Style of line to use (equivalent to nicknames passed to the <code>new_line</code> method.)	solid
line_width	Number	Width of line in points/pixels	1
line_color	A color specification (see <i>Specifying Colors</i>)	Color of line	#000000FF
ticksets	List	<i>Tick sets</i> to draw on the line	Empty list

10.4 vertical

```

toolbox.new_gridlines ('vertical', ...)
toolbox.new_gridlines_from_border ('vertical', border[, tickset_idx=0, ...])
toolbox.new_gridlines_from_tick_tuples ('vertical', scale, tick_tuples[, ...])

```

Draws vertical lines across the plotting area.

Table 10.3: Axes

Axis	Value	Rendering	Default Scale
pos	A real number between 0 and 1	Spatial position on the X (vertical) axis	No default
line_style	String	Style of line to use (equivalent to nicknames passed to the <code>new_line</code> method.)	solid
line_width	Number	Width of line in points/pixels	1
line_color	A color specification (see <i>Specifying Colors</i>)	Color of line	#000000FF
ticksets	List	<i>Tick sets</i> to draw on the line	Empty list

LINES

Rayon line objects encode information about where and how line segments should be drawn. They are used throughout Rayon, notably in *borders*, *grid lines* and *plots*.

Lines are created using the `new_line` method.

11.1 Lines and antialiasing

The rendering backends Rayon uses generally perform antialiasing on images to remove distracting artifacts. This results in smooth diagonal lines, corners and curves.

When drawing straight vertical and horizontal lines, however, antialiasing can cause a line's edges to be blurred. This can sometimes be avoided (e.g., by aligning the edges of the line exactly on a pixel boundary).

If the *fixup* parameter is specified in the `new_line` method and the line is perfectly horizontal or vertical, Rayon will attempt to adjust the line's position to avoid anti-aliasing. Note, however, that this may introduce problems of its own, for instance by introducing gaps between objects that should be contiguous.

11.2 Creating lines

Lines can be created using one of two methods on the `Toolbox` object. The first is the `new_line` method, which creates a line from a label and set of arguments, similarly to most methods in the toolbox.

Alternatively, the `new_line_from_spec` method takes a string line specification (see *Specifying Line Styles*) and uses that to create a line. The string specification method is useful for programs which create lines based on user input.

11.3 dashed

```
toolbox.new_line('dashed', color="#000000FF", width=1, fixup=False)
```

Renders a dashed line.

color is the color of the line.

width is the width of the line in pixels/points.

If *fixup* is `True`, Rayon will attempt to “fix up” perfectly horizontal and vertical lines to avoid anti-aliasing. This behavior is disabled if *fixup* is `False`.

11.4 dotted

```
toolbox.new_line('dotted', color="#000000FF", width=1, fixup=False)
```

Renders a dotted line.

color is the color of the line.

width is the width of the line in pixels/points.

If *fixup* is `True`, Rayon will attempt to “fix up” perfectly horizontal and vertical lines to avoid anti-aliasing. This behavior is disabled if *fixup* is `False`.

11.5 dotdash

```
toolbox.new_line('dotdash', color="#000000FF", width=1, fixup=False)
```

Renders a line that is a mixture of dots and dashes.

color is the color of the line.

width is the width of the line in pixels/points.

If *fixup* is `True`, Rayon will attempt to “fix up” perfectly horizontal and vertical lines to avoid anti-aliasing. This behavior is disabled if *fixup* is `False`.

11.6 none

```
toolbox.new_line('null', color="#000000FF", width=1, fixup=False)
```

Can be used like a line, but doesn't actually draw a line. All arguments to

11.7 solid

```
toolbox.new_line('solid', color="#000000FF", width=1, fixup=False)
```

MARKERS

Markers are very simple objects; they make marks on the drawing canvas. Markers draw marks relative to a point, usually around the center of the mark. This mark is referred to as the *origin* of the mark for the rest of this chapter.

Markers are categorized by the shape of the marks they draw. Markers are most often used indirectly, by specifying a marker label. An example of this is the `marker_shape` axis of the `scatter` plot. Markers are created directly when creating *labeled markers*, or in more advanced applications, such as when writing a custom plot.

12.1 `darrow`

```
toolbox.new_marker('darrow',[ rotation=0, angle=90, size=5, stroke_color="#000000FF",  
                             stroke_width=1])  
toolbox.new_marker('arrow',[ rotation=0, angle=90, size=5, stroke_color="#000000FF",  
                             stroke_width=1])
```

Draws a chevron, or simple arrow head – two line segments coming together in an angle. Arrows pointing to the left, right, top or bottom with varying degree of “sharpness” may be obtained by manipulating the *rotation* and *angle* parameters.

rotation is a measure (in degrees) of rotation the arrow should have from its default position (with the point of the arrow pointing straight up).

angle is the measure (in degrees) of the lower angle formed by the two legs of the chevron. Values greater than 180 degrees will result in an inverted v-shape, with the point facing up. Values less than 180 degrees will result in a v-shape, with the point facing down.

For horizontal orientation, *angle* refers to the right-hand angle of the arrow head; thus, values less than 180 degrees will produce a left-pointing arrow and values greater than 180 will produce a right-pointing arrow.

size is the size in points/pixels of the edges of the arrow head; *stroke_width* is their width in points/pixels. *stroke_color* is a color specification (see *Specifying Colors*) indicating the color of the arrow.

12.2 `uarrow`

```
toolbox.new_marker('uarrow',[angle=90, size=5, stroke_color="#000000FF", stroke_width=1])
```

An `Arrow` class customized for pointing straight up.

12.3 larrow

```
toolbox.new_marker('larrow', [angle=90, size=5, stroke_color="#000000FF", stroke_width=1])
```

An [Arrow](#) class customized for pointing to the left.

12.4 rarrow

```
toolbox.new_marker('rarrow', [angle=90, size=5, stroke_color="#000000FF", stroke_width=1])
```

An [Arrow](#) class customized for pointing to the right.

12.5 circle

```
toolbox.new_marker('circle', [radius=.5, stroke_color="#000000FF", fill_color="#00000000",  
                             stroke_width=1])
```

Draws a circle. The origin of the mark is the center of the circle.

radius is the radius of the circle in device units (points/pixels).

stroke_color is the color of the edge of the circle.

fill_color is the color of the circle's interior.

stroke_width is the width (in points or pixels) of the edge of the circle.

12.6 cross

```
toolbox.new_marker('cross', [below=5, above=5, before=5, after=5, color="#000000FF", width=1])
```

Draws a cross. (i.e., An equally-sized horizontal line and a vertical line crossing at their respective midpoints.) The origin of the mark is the point at which the vertical and horizontal line segments intersect.

below is the length of the vertical line segment below the origin, in points/pixels.

above is the length of the vertical line segment above the origin, in points/pixels.

before is the length of the horizontal line segment to the left of the origin, in points/pixels.

after is the length of the horizontal line segment to the right of the origin, in points/pixels.

color is a color specification (see [Specifying Colors](#)) of both the horizontal and vertical line segments.

width is the width of both the horizontal and vertical line segments in points/pixels.

12.7 dot

```
toolbox.new_marker('dot', [radius=.5, color="#000000FF"])
```

Draws a filled circle. The origin is the center of the circle.

radius is the radius of the circle in device units (points/pixels).

color is a color specification (see [Specifying Colors](#)) of the dot.

12.8 hline

```
toolbox.new_marker('hline',[before=5,after=5,color="#000000FF",width=1])
```

Draws a horizontal line segment. The origin of the mark is a point along the line. (The exact point depends on the values of the *before* and *after* parameters.)

before is the length of the line segment to the left of the origin, in points/pixels.

after is the length of the line segment to the right of the origin, in points/pixels.

color is a color specification (see [Specifying Colors](#)) of both line segments.

width is the width of both line segments in points/pixels.

12.9 none

```
toolbox.new_marker('none')
```

Makes no mark. This can be used as the *marker* argument a `LabeledMarker`, which can in turn be passed to something which expects a `LabeledMarker`, if the intent is to only draw a label.

12.10 vline

```
toolbox.new_marker('vline',[below=5,above=5,color="#000000FF",width=1])
```

Draws a vertical line segment. The origin of the mark is a point along the line. (The exact point depends on the values of the *below* and *above* parameters.)

below is the length of the line segment below the origin, in points/pixels.

above is the length of the line segment above the origin, in points/pixels.

color is a color specification (see [Specifying Colors](#)) of both line segments.

width is the width of both line segments in points/pixels.

PAGE

The `Page` object controls the rendering of a visualization to a canvas, which can be a file or an element in a GUI. They are created using the `new_page_from_buffer` and `new_page_from_filename` methods.

```
toolbox.new_page_from_buffer(buff)
toolbox.new_page_from_filename(filename, width, height)
```

```
write(chart)
    Output chart to the canvas.
```


PLOTS

Plots are the core visualization object in Rayon. Plots are composed of a set of axes, which are themselves pairings of data with scales that map the data into the visualization space. Plots also contain logic for using these axes to draw a visualization. See *Plots* in *Building Visualizations* for more details.

14.1 Creating Plots

All plots are created from the `Toolbox` object using the `new_plot` method, supplying the label and arguments for the desired plot. The plots in this chapter are organized by the label used to create them in the `new_plot` method.

In addition to the label, calls to `new_plot` also take a number of arguments of the form `<axis>_scale` or `<axis>_data`, where `<axis>` is the name of one of the plot's axes. The value of an `<axis>_scale` argument should be a *scale*; the value of an `<axis>_data` argument should be a *column* or an array of data of the same length as the data associated with every other axis. The following example creates a `scatter` plot named `scatter` and associates the `x` and `y` axes with scales and data:

```
scatter = tools.new_plot('scatter',
                        x_data=[1, 2, 3],
                        x_scale=tools.new_scale('linear'),
                        y_data=[10, 20, 30],
                        y_scale=tools.new_scale('linear'))
```

14.1.1 Default Scales and Data

If data is omitted from an axis specification, it is treated as a column of data whose values are all `None`. Consequently, data may only be omitted from an axis if the scale for that axis can tolerate `None` as an input; otherwise, an error will be generated when the visualization is rendered.

Scales may also be omitted from an axis specification, if that axis has a default scale. For instance, a `scatter` plot's default scales for the `x` and `y` axes are both `linear` scales, so the example above could be shortened to:

```
scatter = tools.new_plot('scatter',
                        x_data=[1, 2, 3],
                        y_data=[10, 20, 30])
```

14.1.2 Constant Scales

If the value of an `<axis>_scale` argument is not callable, it is implicitly converted into a scale that transforms any input into that value. This is referred to elsewhere in this document as specifying a *constant scale* for the axis. For instance, the following example specifies that all points in a `scatter` plot be black dots 4 pixels in diameter:

```
scatter = tools.new_plot('scatter',
                        x_data=[1, 2, 3],
                        y_data=[10, 20, 30],
                        marker_size_scale=4,
                        marker_color_scale="black",
                        marker_shape_scale="dot")
```

(Note the `marker_` arguments above are using default data. Constant scales can take `None` as input (because they ignore it). They are usually used with default data.)

The default scales for a `scatter` plot's `marker_size`, `marker_color` and `marker_shape` axes are all constant, so it is necessary to supply neither data nor scales for them if the default values are acceptable.

14.1.3 Additional Arguments

A few plots take additional arguments to the ones described above. These additional arguments are described as appropriate in the plot descriptions that follow.

14.2 Plots

This section describes the plots provided by Rayon. Each subsection consists of a text description, an illustration of the plot, and a table describing the plot's axes. Each row in the table contains the following information about an axis:

Axis The axis name.

Value What sort of value it expects as output from its scale.

Rendering What that value means when the plot is rendered.

Default Scale The default scale for the axis. If the default scale is constant, the output value of the constant scale.

14.2.1 bar

```
toolbox.new_plot('bar', y_origin=<auto>[, ...])
toolbox.new_plot('vbar', y_origin=<auto>[, ...])
```

Draws a number of vertical bars, with the number and height of each bar determined by the data. Data can also be mapped onto the bars' color, height, width and border color/width.

`y_origin` is the point at which to begin drawing the base of the bar. If data is less than this, this point on the Y axis will represent the top edge of the bar; otherwise it will represent the bottom edge. This value must only be supplied if the user is using a scale function for the Y axis; if a scale class such as `linear` is used, by default the origin will be the scaled value of 0 on the scale.

See Also:

`hbar`

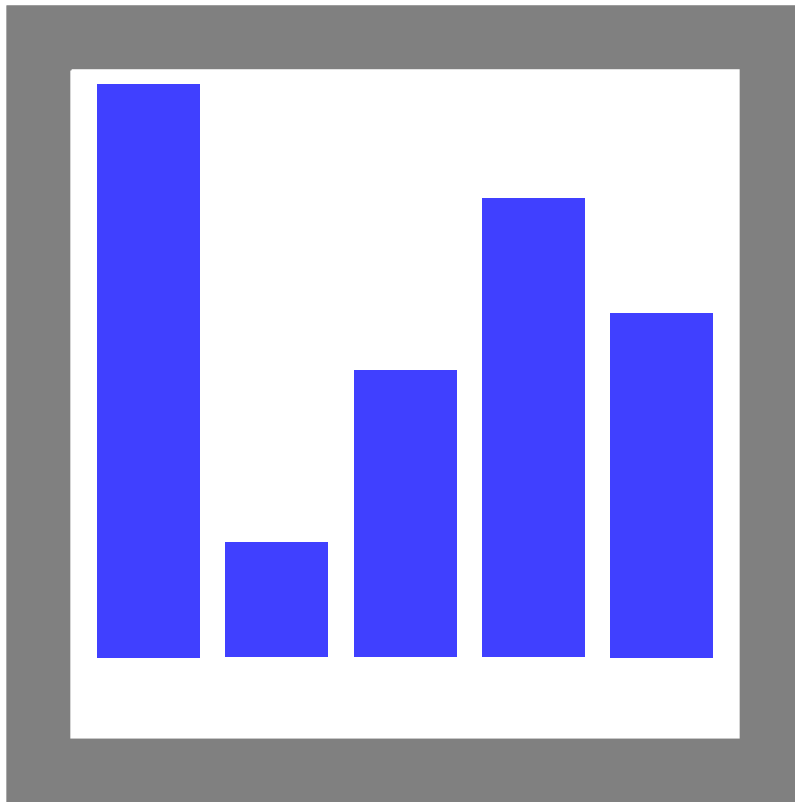


Figure 14.1: Example of a bar plot.

Table 14.1: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A catrange scale
y	A real number between 0 and 1	Y (vertical) spatial axis	A linear scale
bar_alignment	A real number between 0 and 1. 0 == full left, 1 == full right.	Alignment of bar in allotted region.	.5
bar_color	A color specification (see Specifying Colors)	Interior color of the bar.	#0000FFC0
border_color	A color specification (see Specifying Colors)	The border color of the bar.	#000000FF
bar_width	A real number between 0 and 1	The proportion of the range specified by x that should be filled with the bar. 0 == no width; 1 == full range.	1
border_width	Integer size (in points/pixels)	Size of the border.	0
bar_fixup	True or False	If True, align bar edges on pixels	True

14.2.2 field

```
toolbox.new_plot('field', fill_color="#9999eecc", ...)
```

Draws a visualization similar to that drawn by a [line](#) plot. However, a [field](#) plot draws two lines. Each line shares values for the X axis, but has different values for Y. The area between the two lines (the “field”) is shaded.

fill_color is a color specification (see [Specifying Colors](#)) of the color of the shading.

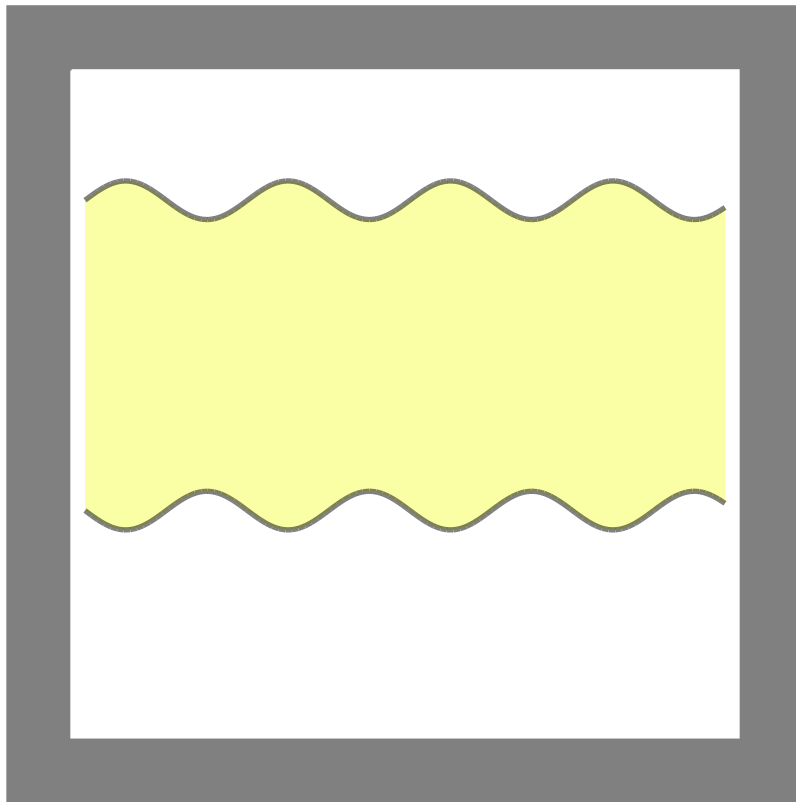


Figure 14.2: Example of a field plot.

Table 14.2: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale
y1	A real number between 0 and 1	The Y (vertical)) axis for the first line	A linear scale
y2	A real number between 0 and 1	The Y (vertical)) axis for the second line	A linear scale
line_color	A color specification (see Specifying Colors)	Color of the line segment drawn between the last point and this one. The same line color is used for the line segment drawn for the y1 data and the one drawn for the y2 data.	#0000007F
line_width	Integer size (in points/pixels)	Width of line segment ending in this point. The same line width is used for the line segment drawn for the y1 data and the one drawn for the y2 data.	2
line_style	A line style (see Lines)	Style of line segment ending in this point	dashed

14.2.3 filledline

```
toolbox.new_plot('filledline'[, fill_color="#ee999cc", ... ])
```

Draws a line based on data specified for the X and Y axes in the style of a [line](#) plot. The area “beneath” the line (that is, in the direction of smaller values on the scale), is shaded. See also the [field](#) plot.

fill_color is a color specification (see [Specifying Colors](#)) of the shading beneath the line.

Table 14.3: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale
y	A real number between 0 and 1	Y (vertical)) spatial axis	A linear scale
line_color	A color specification (see Specifying Colors)	Color of the line segment drawn between the last point and this one	#0000007F
line_width	Integer size (in points/pixels)	Width of line segment ending in this point	2
line_style	A line style (see Lines)	Style of line segment ending in this point	dashed

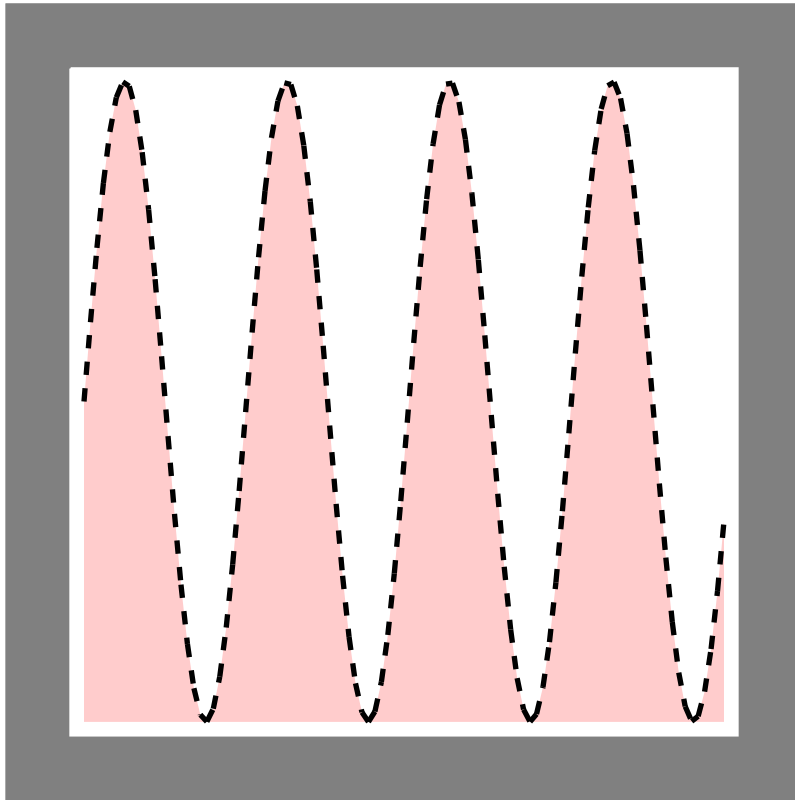


Figure 14.3: Example of a filled line plot.

14.2.4 hbar

```
toolbox.new_plot('hbar', x_origin=<auto>[, ...])
```

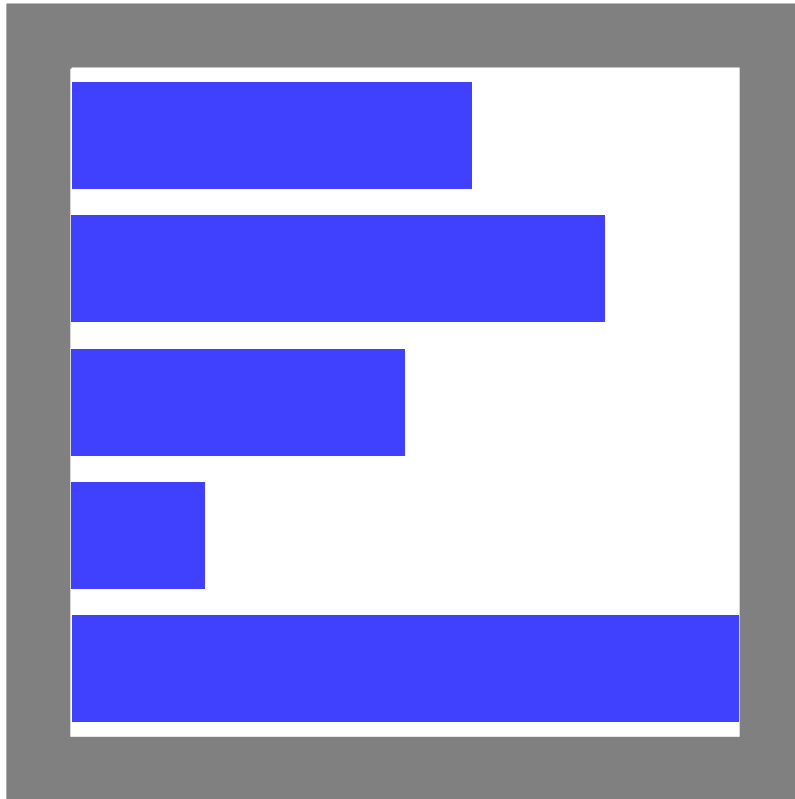


Figure 14.4: Example of a horizontal bar plot.

Plots data similarly to the `bar` plot, but orients the bars horizontally, from top to bottom.

Note that bar “width” in the `hbar` plot is the vertical dimension of the bar. The horizontal dimension is the bar “height.”

`x_origin` is the point at which to begin drawing the base of the bar. If data is less than this, this point on the X axis will represent the right edge of the bar; otherwise it will represent the left edge. This value must only be supplied if the user is using a scale function for the X axis; if a scale class such as `linear` is used, by default the origin will be the scaled value of 0 on the scale.

See Also:

`bar`

Table 14.4: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale
y	A real number between 0 and 1	Y (vertical) spatial axis	A linear scale
bar_alignment	A real number between 0 and 1. 0 == full left, 1 == full right.	Alignment of bar in allotted region.	.5
bar_color	A color specification (see Specifying Colors)	Interior color of the bar.	#0000FFC0
border_color	A color specification (see Specifying Colors)	The border color of the bar.	#000000FF
bar_width	A real number between 0 and 1	The proportion of the range specified by u that should be filled with the bar. 0 == no width; 1 == full range.	1
border_width	Integer size (in points/pixels)	Size of the border.	0
bar_fixup	True or False	If True, align bar edges on pixels	True

14.2.5 labeledscatter

```
toolbox.new_plot('labeledscatter'[, ...])
```

Draws a plot similar to that drawn by a [scatter](#) plot, but the markers may also be labeled.

By default, label data is converted to a string using `str`. If the label value is `None` or the input value cannot be converted to a string, the output value is an empty string (meaning no label is drawn).

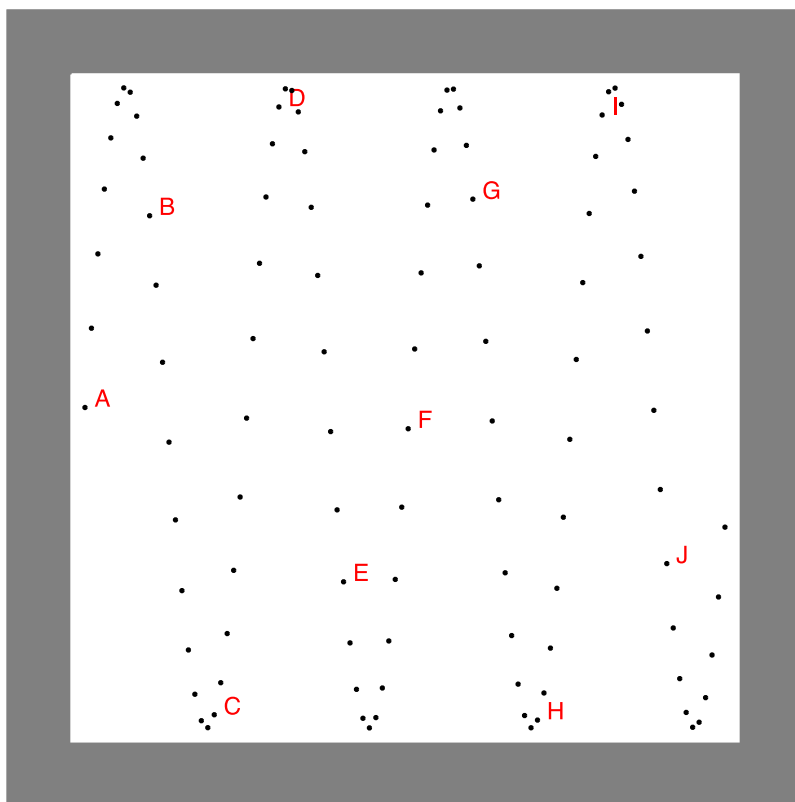


Figure 14.5: Example of a labeled scatterplot.

Table 14.5: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale
y	A real number between 0 and 1	Y (vertical) spatial axis	A linear scale
marker_color	A color specification	Color of the mark representing the point	#000000FF
marker_shape	A marker shape	Shape of the mark representing the point	dot
marker_size	Integer size (in points/pixels)	Size of the mark representing the point	1
text	String	Text of the label accompanying the marker	See description
label_position	n, s, e, or w (See Labeled Markers)	Position of the label accompanying the marker, relative to the marker	ne
label_spacing	Integer size (in points/pixels)	Space between label and marker, in points/pixels	1
font_properties	dictionary	Dictionary of properties of the label text	{ } (Empty dictionary)
border_properties	dictionary	Dictionary of properties of the label border	{ } (Empty dictionary)
bgcolor	A color specification (see Specifying Colors)	Background color of the label box	#00000000

14.2.6 line

```
toolbox.new_plot('line'[, ...])
```

Draws a series of Cartesian points as a “line.” (More accurately, as a connected chain of line segments. The points represent the endpoints of segments in the chain.)

Table 14.6: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale.
y	A real number between 0 and 1	Y (vertical) spatial axis	A linear scale.
line_color	A color specification (see Specifying Colors)	Color of the line segment drawn between the last point and this one	#000000FF
line_width	Integer size (in points/pixels)	Width of line segment ending in this point	2
line_style	A line style (see Lines)	Style of line segment ending in this point	solid

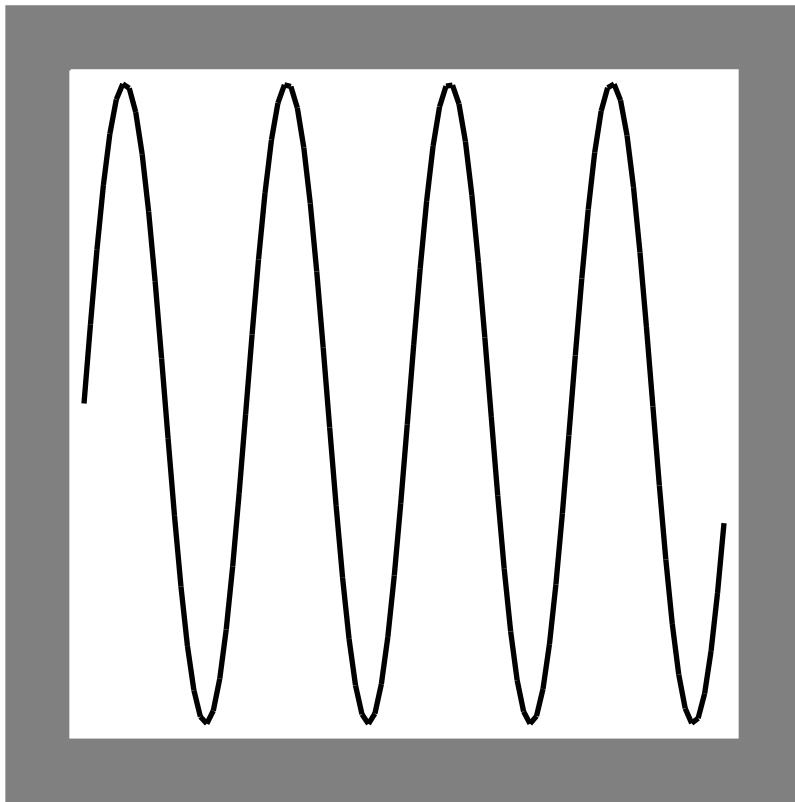


Figure 14.6: Example of a line plot.

14.2.7 scatter

```
toolbox.new_plot('scatter'[, ...])
```

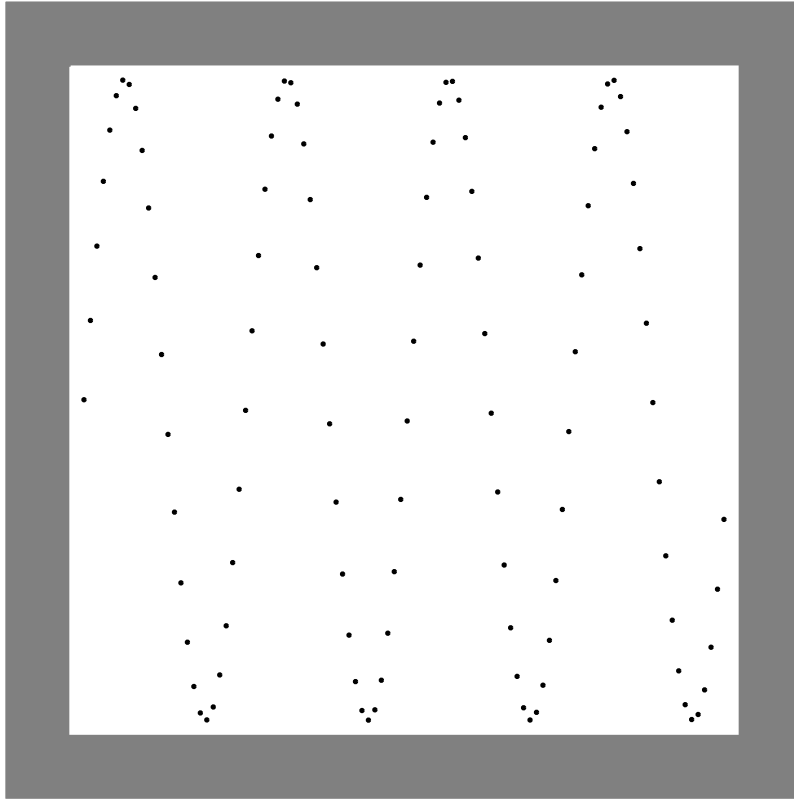


Figure 14.7: Example of a scatterplot.

Draws a plot in which data are mapped onto the X and Y axes of a Cartesian coordinate system. At these coordinates, marks are made whose shape, color and size can also be determined based on data.

Table 14.7: Axes

Axis	Value	Rendering	Default Scale
x	A real number between 0 and 1	X (horizontal) spatial axis	A linear scale
y	A real number between 0 and 1	Y (vertical) spatial axis	A linear scale
<i>marker_color</i>	A color specification (see Specifying Colors)	Color of the mark representing the point	#000000FF
<i>marker_shape</i>	A marker shape	Shape of the mark representing the point	dot
<i>marker_size</i>	Integer size (in points/pixels)	Size of the mark representing the point	1

SCALES

Scales are used in Rayon to map one set of data into another – for instance, mapping numeric values into color values or spatial coordinates. They are most often associated with data in plots or ticksets.

The simplest scale is a function that takes as its sole argument a point in the input space and returns a point in the output space. For instance, the following scale function coerces data into one of three positions on a numeric range from 0 to 1:

```
def step_scale(in_val):
    if in_val < 10:
        return 0.0
    elif in_val < 20:
        return .5
    else:
        return 1.0
```

Functions of this sort may be used wherever scales are called for. See *Scales* for more information.

This chapter describes *scale objects*, which may be used as scales. In addition to transforming data, scale objects can infer the range of input from sets of data automatically when they are associated with data in plots or ticksets.

Advanced Rayon users may wish to create custom scale objects. For more information on this, see *customizing-scale*.

15.1 Exceptions

Rayon scales may raise the following exceptions.

class `rayon.scales.RayonScaleException`

Bases: `rayon.RayonException`

Base class of scale-related exceptions.

class `rayon.scales.RayonLimitException`

Bases: `rayon.scales.RayonScaleException`

Indicates that the user tried to generate an invalid scale, or to use the limits of a scale without properly specifying them. If a scale is not fixed and no data has been associated with it, a call to a scale's `get_input_min` or `get_output_max` methods will raise this exception.

class `rayon.scales.RayonRangeException`

Bases: `rayon.scales.RayonScaleException`

Indicates that a user has passed a value that cannot be mapped using this scale.

15.2 Creating Scales

All scale objects are created from the `Toolbox` object using the `new_scale` method, supplying the label and arguments for the desired scale. The scale objects in this chapter are organized by the label used to create them in the `new_scale` method.

15.3 Scale Limits

Unlike functions, scale objects can automatically fit the limits of their input to the data that will be scaled. Before rendering a *chart*, Rayon first registers all the datasets used in a visualization with the scales that will transform them.

Users can override this behavior to specify fixed custom limits using the `input_min` and `input_max` parameters to `new_scale`. These parameters fix the lower and upper bounds of the scale input at the specified values.

If the `fixed` parameter to `new_scale` is set to `False`, `input_min` and `input_max` will be taken as initial minimum and maximum values; if values exceeding these limits are associated with the scale, the limits will grow to include them.

15.4 Range Scales

Most of the scales in this chapter map input data values to a single output value, but this isn't appropriate in all cases. For instance, a *bar* plot draws its bars between two points on the X axis. This means that the *bar* plot requires a range scale like the `catrange` scale, which maps a set of categories into a set of non-overlapping ranges along the X axis.

Range scales can be converted into corresponding single-value scales that emit a point within the range; these single-value scales can, in turn, be converted back to range scales. For instance, a `catrange` scale may be converted into a `categorical` scale using the `CategoricalRangeScale.to_categorical_scale` method.

15.5 Categorical Scales

Categorical scales are used to display data that falls into distinct categories. Examples include zip codes, unique identifiers, or choices in a multiple choice question.

Categorical data is usually unordered; a unique ID of 1 isn't necessarily "less than" a unique ID of 100, just different. When visualizing data, however, one may prefer to show categories in a certain order, either because it will be easier for users to navigate (e.g., alphabetical order), or to keep the order the same across multiple visualizations.

For this reason, categorical scales in Rayon maintain their categories in the order in which they were inserted.

15.5.1 `categorical`

```
toolbox.new_scale('categorical', categories : list, output_min=0, output_max=1 fixed=True, re-  
verse=False)
```

Maps categories to evenly-spaced points on a numeric range. These points are located inside non-overlapping subranges within the output range.

`categories` is a sequence of input categories. The elements in the sequence may be of any type provided that they are unique and can be tested for equality. This parameter may be omitted; in that case, the categories will be determined by Rayon based on the data the scale is required to transform.

output_min and *output_max* define the boundaries of a numeric range. The scale will map categories to points evenly distributed within this range. When used as a scale on a spatial axis in a plot or border, the default values will use all the space allocated for the element being visualized.

alignment indicates where the output point will lie within the allotted subrange. Acceptable values are *beginning*, *center* or *end*, indicating that the point should be at the beginning, midpoint or end of the subrange, respectively.

If *fixed* is `True`, the value of *categories* will be the final value for the scale; no categories may be added or removed. Otherwise, the category list may change if new data is associated with the scale.

If *reverse* is `False`, categories will be ordered as they were added to the scale. Additional categories beyond the initial set will be appended in the order they were inserted. If *reverse* is `True`, this order will be reversed.

get_input_max()

Returns the last category added to the scale. If a sequence of categories was supplied to the scale when it was created and no additional data has been associated with the object, the last category in that sequence is returned. If data has been associated with this scale since it was created, this method returns the last unique category in the most recent set of data currently associated with it.

If the *reverse* parameter was passed to this scale when it was created, the order described above is reversed. The result of calling this method is then equivalent to calling `get_input_min` on a scale that has not been reversed.

If no categories are associated with this scale, this method returns `None`.

get_input_min()

Returns the first category added to the scale. If a sequence of categories was supplied to the scale when it was created, the first category in that sequence is returned. If the scale was created without categories and data has been associated with this scale, this method returns the first category in the oldest set of data currently associated with it.

If the *reverse* parameter was passed to this scale when it was created, the order described above is reversed. The result of calling this method is then equivalent to calling `get_input_max` on a scale that has not been reversed.

If no categories are associated with this scale, this method returns `None`.

get_nice_tick_positions([num_ticks=5, inside=False])

Returns an iterator over all the categories currently associated with this scale. The *num_ticks* and *inside* parameters are ignored.

If the *reverse* parameter was passed to this scale when it was created, this method is equivalent to calling `reversed(get_nice_tick_positions)` on a non-reversed version of this scale.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

to_categorical_range_scale([fixed=True])

Returns a `catrange` scale using the same categories and range bounds data as this scale.

The resulting `catrange` scale will be fixed (i.e., categories cannot be added to it) unless *fixed* is `False`.

15.5.2 catrange

`toolbox.new_scale('catrange', categories : list, output_min=0, output_max=1, fixed=True, reverse=False)`

Maps categories to contiguous, non-overlapping sub-ranges in a larger range. This scale is a ranged version of

the `categorical` scale. (See *Range Scales*)

`categories` is a sequence of input categories. The elements in the sequence may be of any type provided that they are unique and can be tested for equality. This parameter may be omitted; in that case, the categories will be determined by Rayon based on the data the scale is required to transform.

`output_min` and `output_max` define the boundaries of a numeric range. The scale will map categories to contiguous subranges of equal size that cover the range. When used as a scale on a spatial axis in a plot or border, the default values for `output_min` and `output_max` will use all the space allocated for the element being visualized.

If `fixed` is `True`, the value of `categories` will be the final value for the scale; no categories may be added or removed. Otherwise, the category list may change if new data is associated with the scale.

If `reverse` is `False`, categories will be ordered as they were added to the scale. Additional categories beyond the initial set will be appended in the order they were inserted. If `reverse` is `True`, this order will be reversed.

`get_input_max()`

Returns the last category added to the scale. If a sequence of categories was supplied to the scale when it was created and no additional data has been associated with the object, the last category in that sequence is returned. If data has been associated with this scale since it was created, this method returns the last unique category in the most recent set of data currently associated with it.

If the `reverse` parameter was passed to this scale when it was created, the order described above is reversed. The result of calling this method is then equivalent to calling `get_input_min` on a scale that has not been reversed.

If no categories are associated with this scale, this method returns `None`.

`get_input_min()`

Returns the first category added to the scale. If a sequence of categories was supplied to the scale when it was created, the first category in that sequence is returned. If the scale was created without categories and data has been associated with this scale, this method returns the first category in the oldest set of data currently associated with it.

If the `reverse` parameter was passed to this scale when it was created, the order described above is reversed. The result of calling this method is then equivalent to calling `get_input_max` on a scale that has not been reversed.

If no categories are associated with this scale, this method returns `None`.

`get_nice_tick_positions([num_ticks=5, inside=False])`

Returns an iterator over all the categories currently associated with this scale. The `num_ticks` and `inside` parameters are ignored.

If the `reverse` parameter was passed to this scale when it was created, this method is equivalent to calling `reversed(get_nice_tick_positions)` on a non-reversed version of this scale.

`get_output_max()`

Returns the highest value in the output range.

`get_output_min()`

Returns the lowest value in the output range.

`range_size()`

Returns the size of the subranges allocated to each category, in terms of the output range. This value multiplied by the number of categories should equal the size of the output range.

`to_categorical_scale([alignment="beginning", fixed=True])`

Returns a `CategoricalScale` using the same categories and range bounds data as this scale.

`alignment` is where in the range to put the reference point for the categories; see `:meth:CategoricalScale.__init__`.

The resulting `CategoricalScale` will be fixed (i.e., categories cannot be added to it) unless *fixed* is `False`.

15.6 Color Scales

15.6.1 gradient

```
toolbox.new_scale('gradient', input_min : num, input_max : num, output_min="#FFFFFF", out-
                    put_max="#000000", numsteps=255, fixed=True)
```

Maps a numeric range linearly to a range of colors along a gradient.

input_min and *input_max* specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

output_min and *output_max* are color specifications (see [Specifying Colors](#)) representing the beginning and end points of a color gradient. *numsteps* is the number of colors in the gradient, including the beginning and end points.

If *fixed* is `True`, the values of *input_min* and *input_max* will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions([num_ticks=5, inside=False])

Returns an iterator over a sequence of aesthetically pleasing positions in the input range of the scale. These positions will be used for placing tick marks.

num_ticks is the requested number of tick positions. This is only a request; `get_nice_tick_positions` may choose to honor it or not as appropriate to the scale.

If *inside* is `True`, the lowest and highest points in the set of positions will fall inside the minimum and maximum values in the input range. If it is `False`, the lowest and highest points in the set must fall on or outside the minimum and maximum values in the input range.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.6.2 loggrad

```
toolbox.new_scale('loggrad', input_min : num, input_max : num, output_min="#FFFFFF", out-
                    put_max="#000000", numsteps=255, fixed=True)
```

Maps a numeric range to a range of colors along a gradient, using a log-transformed linear scale.

This scale's relationship to the `gradient` scale is analogous to the `log` scale's relationship to the `linear` scale. As with the `log` scale, valid input for this scale is any positive real number; zero and negative numbers are invalid input and will raise a `RayonException` (a `RayonLimitException` if invalid scale limits are specified, or a `RayonRangeException` if the user attempts to convert an invalid value).

If your data is integer data containing positive numbers and zero (such as most count data), consider using the `cloggrad` scale, which is designed for this kind of data.

`input_min` and `input_max` specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

`output_min` and `output_max` are color specifications (see *Specifying Colors*) representing the beginning and end points of a color gradient. `numsteps` is the number of colors in the gradient, including the beginning and end points.

If `fixed` is `True`, the values of `input_min` and `input_max` will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

`get_input_max()`

Returns the highest value in the input range.

`get_input_min()`

Returns the lowest value in the input range.

`get_nice_tick_positions([num_ticks=5, inside=False])`

Returns an iterator over a set of tick positions evenly spaced on the log scale for this range of data. (E.g. 10, 100, 1000,...) The value of `num_ticks` is ignored; `inside` is treated as `True` regardless of its value, meaning the range of data associated with this scale will always fall inside the range emitted by this method.

`get_output_max()`

Returns the highest value in the output range.

`get_output_min()`

Returns the lowest value in the output range.

15.6.3 `cloggrad`

```
toolbox.new_scale('cloggrad', input_min : num, input_max : num, output_min="#FFFFFF", out-
                    put_max="#000000", numsteps=255, fixed=True)
```

Maps a numeric range to a range of colors along a gradient, using a modified log-transformed linear scale suitable for count data.

This scale is analogous to the `clog` scale. Its input range can include zeroes (but not negative numbers), and is suitable for integer data. However, it distorts non-integer data relative to a simple log scale. If the data being scaled contains only positive values and no zeroes, the `loggrad` scale is more appropriate.

`input_min` and `input_max` specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

`output_min` and `output_max` are color specifications (see *Specifying Colors*) representing the beginning and end points of a color gradient. `numsteps` is the number of colors in the gradient, including the beginning and end points.

If `fixed` is `True`, the values of `input_min` and `input_max` will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

`get_input_max()`

Returns the highest value in the input range.

`get_input_min()`

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a set of tick positions evenly spaced on the log scale for this range of data. (E.g. 10, 100, 1000,...) The value of *num_ticks* is ignored; *inside* is treated as `True` regardless of its value, meaning the range of data associated with this scale will always fall inside the range emitted by this method.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7 Spatial Scales

15.7.1 linear

`toolbox.new_scale('linear', [input_min : num, input_max : num, output_min=0, output_max=1, fixed=True])`

Maps values from one numeric range to another with linear scaling.

input_min and *input_max* specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

output_min and *output_max* specify the initial lower and upper bounds of the output range.

If *fixed* is `True`, the values of *input_min* and *input_max* will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a sequence of aesthetically pleasing positions in the input range of the scale. These positions will be used for for placing tick marks.

num_ticks is the requested number of tick positions. This is only a request; `get_nice_tick_positions` may choose to honor it or not as appropriate to the scale.

If *inside* is `True`, the lowest and highest points in the set of positions will fall inside the minimum and maximum values in the input range. If it is `False`, the lowest and highest points in the set must fall on or outside the minimum and maximum values in the input range.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7.2 linbins

`toolbox.new_scale('linbins', bin_size[, input_min : num, input_max : num, output_min=0, output_max=1, fixed=True])`

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a sequence of aesthetically pleasing positions in the input range of the scale. These positions will be used for placing tick marks.

num_ticks is the requested number of tick positions. This is only a request; `get_nice_tick_positions` may choose to honor it or not as appropriate to the scale.

If *inside* is `True`, the lowest and highest points in the set of positions will fall inside the minimum and maximum values in the input range. If it is `False`, the lowest and highest points in the set must fall on or outside the minimum and maximum values in the input range.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7.3 log

`toolbox.new_scale('log', [input_min : num, input_max: num, output_min=0, output_max=1, fixed=True])`

Maps values from one numeric range to another with log-transformed linear scaling.

Valid input for this scale is any positive real number; 0 and negative numbers are invalid input and will raise a `RayonException` (a `RayonLimitException` if invalid scale limits are specified, or a `RayonRangeException` if the user attempts to convert an invalid value).

If your data is integer data containing positive numbers and zero (such as most count data), consider using the `clog` scale, which is designed for this kind of data.

input_min and *input_max* specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

output_min and *output_max* specify the initial lower and upper bounds of the output range.

If *fixed* is `True`, the values of *input_min* and *input_max* will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a set of tick positions evenly spaced on the log scale for this range of data. (E.g. 10, 100, 1000,...) The value of *num_ticks* is ignored; *inside* is treated as `True` regardless of its value, meaning the range of data associated with this scale will always fall inside the range emitted by this method.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7.4 clog

```
toolbox.new_scale('clog', input_min : num, input_max: num, output_min=0, output_max=1,
                  fixed=True)
```

Maps values from one numeric range to another using a modified log-transformed linear scale best suited to count data.

The input range for this scale can include zeroes (but not negative numbers), and is suitable for integer data. However, it distorts non-integer data relative to a simple log scale. If the data being scaled contains only positive values and no zeroes, the `log` scale is more appropriate.

input_min and *input_max* specify the initial lower and upper bounds of the input range. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

output_min and *output_max* specify the initial lower and upper bounds of the output range.

If *fixed* is `True`, the values of *input_min* and *input_max* will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions([num_ticks=5, inside=False])

Returns an iterator over a set of tick positions evenly spaced on the log scale for this range of data. (E.g. 10, 100, 1000,...) The value of *num_ticks* is ignored; *inside* is treated as `True` regardless of its value, meaning the range of data associated with this scale will always fall inside the range emitted by this method.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7.5 time

```
toolbox.new_scale('time', input_min : datetime, input_max : datetime, output_min=0, output_max=1,
                  fixed=True)
```

Maps values from a time range linearly to values in a numeric range.

input_min and *input_max* specify the initial lower and upper bounds of the input range, as `datetime.datetime` objects. If one or both of these parameters is not supplied, Rayon will select a value that will minimally contain the data associated with the scale.

output_min and *output_max* specify the initial lower and upper bounds of the output range.

If *fixed* is `True`, the values of *input_min* and *input_max* will be the final values for the scale; otherwise, Rayon may adjust the input range if new data is associated with the scale.

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a sequence of aesthetically pleasing positions in the input range of the scale. These positions will be used for for placing tick marks.

num_ticks is the requested number of tick positions. This is only a request; `get_nice_tick_positions` may choose to honor it or not as appropriate to the scale.

If *inside* is `True`, the lowest and highest points in the set of positions will fall inside the minimum and maximum values in the input range. If it is `False`, the lowest and highest points in the set must fall on or outside the minimum and maximum values in the input range.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

15.7.6 timebins

`toolbox.new_scale('timebins', bin_size[, input_min : datetime, input_max : datetime, output_min=0, output_max=1, fixed=True])`

get_input_max()

Returns the highest value in the input range.

get_input_min()

Returns the lowest value in the input range.

get_nice_tick_positions ([*num_ticks*=5, *inside*=False])

Returns an iterator over a sequence of aesthetically pleasing positions in the input range of the scale. These positions will be used for for placing tick marks.

num_ticks is the requested number of tick positions. This is only a request; `get_nice_tick_positions` may choose to honor it or not as appropriate to the scale.

If *inside* is `True`, the lowest and highest points in the set of positions will fall inside the minimum and maximum values in the input range. If it is `False`, the lowest and highest points in the set must fall on or outside the minimum and maximum values in the input range.

get_output_max()

Returns the highest value in the output range.

get_output_min()

Returns the lowest value in the output range.

TEXT

This chapter documents the two classes Rayon uses to put text labels on visualizations. Labelers apply text labels to the drawing surface. Labeled markers marry labelers to *markers*, which can be used as labeled points, tickmarks, and other visual elements.

16.1 Specifying Text Properties

The following parameters may be used by the rendering backend to choose an appropriate font when drawing a label. The rendering backend may not honor these requests. At the time of this writing, only font size is honored in both backends. The wxPython backend honors font family, style and weight, while the Cairo backend does not.

font_size

Font size may be specified as a number (which is taken to be a point size), or one of the following strings specifying a relative size:

- small
- normal
- large
- x-large
- xx-large.

These are identical to the absolute size specifications described in <http://www.w3.org/TR/CSS2/fonts.html#font-size-props>; however, they specify relative, not absolute, sizes.

font_family

Font family is the string `default` (leaving font choice exclusively to the backend) or one of the strings specified in <http://www.w3.org/TR/CSS2/fonts.html#generic-font-families>, specifically:

- serif
- sans-serif
- cursive
- fantasy
- monospace

font_style

Font style is one of the strings specified in <http://www.w3.org/TR/CSS2/fonts.html#font-styling>, specifically:

- `normal`
- `oblique`
- `italic`

font_weight

Font weight is one of the strings or integer weights specified in <http://www.w3.org/TR/CSS2/fonts.html#font-boldness>, specifically:

- `normal`
- `bold`
- `100`
- `200`
- `300`
- `400`
- `500`
- `600`
- `700`
- `800`
- `900`

Integer weights may be specified as integers or strings. The `bolder` and `lighter` keywords are not implemented in Rayon.

halign

The horizontal alignment of the label may be one of the words `left`, `center`, or `right`.

valign

The vertical alignment of the label may be one of the words `top`, `center`, or `bottom`.

color

The color of the label text is specified as described in [Specifying Colors](#).

bgcolor

The color of the label background is specified as described in [Specifying Colors](#).

angle

The angle of the text is measured counterclockwise in degrees from horizontal, left to right. Both horizontal and vertical alignment are computed relative to a bounding box rotated by the amount specified in *angle*.

border (border_left, border_right, border_top, border_bottom)

The border of the text label is a text line specification (see [Specifying Line Styles](#)) describing the line to use to draw border lines around the label text's bounding box. *border_left*, *border_right*, *border_top* and *border_bottom* specify lines that will only be drawn on the left, right, top and bottom edges of the box, respectively. If *border* is specified, the other *border_* arguments will be ignored.

16.2 Labelers

A labeler is a Rayon object that draws text; that text is a label. Labelers are created using the `new_labeler` method.

16.3 Labeled Markers

A labeled marker draws both a text label and a mark at a point. A labeled marker contains both a *marker* and *labeler*, plus information on how to draw them relative to each other.

Labeled markers are created using the `new_labeled_marker` method.

TICK SETS

Tick sets collect information on where to place ticks. *Borders* and *grid lines* use tick sets to mark points of interest in or alongside a plot.

The primary elements of a tick set are:

- A collection of tick positions on where to place ticks, in data units.
- A *scale* used to transform the tick positions into coordinates on the canvas.
- A *labeled marker* used to render the tick mark on the canvas.

Tick sets may be created using the `new_tickset_from_tuples` (which takes arbitrary tick data) and `new_tickset_from_spec` (which computes tick data from a column of data and a string specification of relative positions in the data) methods of the `Toolbox` object. (See the documentation for the `new_tickset_from_spec` method for a description of tick specification syntax.)

17.1 Tick sets and range scales

The `default` tick set is suitable to use with a scale that returns a single value in a range between 0 and 1. If you wish to use a *range scale* with a tick set, use the `ranged` tick set.

17.1.1 default

```
toolbox.new_tickset_from_tuples('default', tick_tuples, scale, labeledmarker)
toolbox.new_tickset_from_spec('default', tick_spec, col, formatter, scale, labeledmarker[, format-
                             ter=None, alignment=.5])
```

Regulates the display of tick marks. The `default` tick set should not be used with a *range scale*. Positions in this tickset are single numbers representing positions on the scale.

Tick sets are iterable objects. The result of iterating over a tick set `tickset` is the equivalent of calling `itertools.izip(tickset.iter_tick_positions(), tickset.iter_tick_labels())`.

`iter_tick_positions()`

Returns an iterator over the positions of ticks in this tick set. Positions will be returned in the same order as the labels in the `iter_tick_labels` method.

`iter_tick_labels()`

Returns an iterator over the tick labels in this tick set. Labels will be returned in the same order as the positions in the `iter_tick_positions` method.

`get_scale()`

Returns the scale associated with this tick set.

17.1.2 ranged

```
toolbox.new_tickset_from_tuples('ranged', tick_tuples, scale, labeledmarker[, alignment=.5])
toolbox.new_tickset_from_spec('ranged', tick_spec, col, formatter, scale, labeledmarker[, alignment=.5])
```

Regulates the display of tick marks within a range. The `default` tick set should not be used with a *range scale*. Positions in this tickset are pairs of numbers representing the beginning and end of the range on the scale.

Ranged tick sets are iterable objects. The result of iterating over a tick set `tickset` is the equivalent of calling `itertools.izip(tickset.iter_tick_positions(), tickset.iter_tick_labels())`.

`iter_tick_positions()`

Returns an iterator over the positions of ticks in this tick set. Positions will be returned in the same order as the labels in the `iter_tick_labels` method.

`iter_tick_labels()`

Returns an iterator over the tick labels in this tick set. Labels will be returned in the same order as the positions in the `iter_tick_positions` method.

`get_scale()`

TOOLBOX

All objects in Rayon are created using the `Toolbox` object. This reduces the number of `import` statements, which, in turn, makes Rayon-related code more portable. The `Toolbox` object also provides a layer of abstraction between the type of visualization rendering (e.g., file output versus GUI) and the actual technologies used to implement that rendering. This permits Rayon to add more and better rendering backends in the future.

18.1 Creating a Toolbox

The `Toolbox.for_file` classmethod creates a `Toolbox` object that can be used to render static images to files. The `Toolbox.for_gui` classmethod creates a `Toolbox` object that can render visualizations to a GUI canvas. (The only supported GUI toolkit at this time is `wxPython`.)

18.2 Using a Toolbox

The `Toolbox` object consists of a number of factory methods. These methods are all prefixed with `new_`. When multiple types or styles of a given object (e.g., plots) are available, the first argument to a factory method is a *nickname*, a string label identifying the requested variant (e.g., `new_plot("scatter", ...)`). Subsequent arguments vary, and are described in the documentation for the various objects.

The following example illustrates the use of the `Toolbox` object:

```
from rayon import toolbox

def build_chart(t):
    d = t.dataset_from_filename("sample_in.txt")
    c = t.new_chart("square")

    p = t.new_plot("scatter",
                  x_data=d.column(0),
                  y_data=d.column(1))
    c.add_plot(p)

    b1 = t.new_border("hlabel", label="Title")
    c.add_bottom_border(b1, height=.05)

    c.set_chart_background("white")
    return c

t = toolbox.Toolbox.for_file()
page = t.new_page_from_filename(
```

```
"sample_out.png", 800, 600)
page.write(build_chart(t))
```

18.3 The `Toolbox` object

class `rayon.toolbox.Toolbox` (*backend*)

A factory object for creating most of the objects available in Rayon. See *Toolbox — Creating objects in Rayon* for more information.

`Toolbox` objects should be created with one of the `Toolbox.for_file` or `Toolbox.for_gui` methods, which will properly associate the `Toolbox` object with a rendering backend.

classmethod `for_file` ()

Creates a `Toolbox` object appropriate for rendering static visualizations to a file.

classmethod `for_gui` (*gui*='wx')

Create a `Toolbox` object with a backend for rendering to a GUI.

gui is a string label identifying the GUI backend to use. At this time, specifying any value except `wx` will raise an exception.

new_border (*nickname*[, ...]) → `border`

Creates a *border* from *nickname*. Additional arguments will be passed to the border subtype referred to by *nickname*.

The *nickname* argument may be one of the following values:

- `hlabel`
- `hline`
- `horizontal`
- `hsplit`
- `none`
- `vertical`
- `vlabel`
- `vline`
- `vsplit`

new_chart (*nickname*[, ...]) → `chart`

Creates a *chart* from *nickname*. Additional arguments will be passed to the chart subtype referred to by *nickname*.

The *nickname* argument may be one of the following values:

- `square`
- `tiled`
- `tiled_adv`

new_column_from_constant (*constant*, *length*) → `column`

Creates a `Column` object from a constant value and a length.

constant will be returned for all elements in this column.

length is the length of the column.

new_column_from_data (*data* : iterable) → column

Creates a new `Column` object from a set of data.

raw_data is an iterable (list, tuple, iterator, etc.) containing single values of the same data type.

new_dataset_from_filename (*fname* [, *colnames*, *typemap*, *delimiter*]) → Dataset

Creates a `Dataset` from a file. The format of the file is described *On-disk format*.

If supplied, *colnames* is a sequence of strings containing column names for the data. There must be as many names in the sequence as there are columns in the data. If not specified, column names from the file will be used; if the file contains no column names, the `Dataset` will be created without column names.

If supplied, *typemap* is a list of functions that take a string representation of an object and return the object. There must be as many functions in the sequence as there are columns in the data. If not specified, strings composed entirely of numbers (optionally with a single period) will be converted to numbers, and everything else will be left as strings.

If supplied, *delimiter* is the character used in the file to delimit fields in a record. The default delimiter is the pipe (|) character. The delimiter may also be specified in the file format; in that case, this argument is ignored.

If *first_line_is_colnames* is `True` and no other column names are supplied in the file (i.e., with a header or a specially-commented first line), assume the first line of the input is a delimited list of column names. Use this when parsing input from external programs (such as the SiLK tools) which pass their column names as the first line in a file but do not indicate them in any other way.

new_dataset_from_stream (*stream* [, *colnames*, *typemap*, *delimiter*]) → Dataset

Creates a `Dataset` from data read from an input stream. The format of the streamed data is described in *On-disk format*.

If supplied, *colnames* is a sequence of strings containing column names for the data. There must be as many names in the sequence as there are columns in the data. If not specified, column names from the file will be used; if the file contains no column names, the `Dataset` will be created without column names.

If supplied, *typemap* is a list of functions that take a string representation of an object and return the object. There must be as many functions in the sequence as there are columns in the data. If not specified, strings composed entirely of numbers (optionally with a single period) will be converted to numbers, and everything else will be left as strings.

If supplied, *delimiter* is the character used in the file to delimit fields in a record. The default delimiter is the pipe (|) character. The delimiter may also be specified in the file format; in that case, this argument is ignored.

If *first_line_is_colnames* is `True` and no other column names are supplied in the file (i.e., with a header or a specially-commented first line), assume the first line of the input is a delimited list of column names. Use this when parsing input from external programs (such as the SiLK tools) which pass their column names as the first line in a file but do not indicate them in any other way.

new_dataset_from_columns (*columns* [, *colnames*, *header*, *sort_key*]) → Dataset

Creates a `Dataset` from an iterable (list, tuple, iterator, etc.) of columns.

columns is an iterable containing columns of data. The columns may be lists, tuples, other iterables, or `Column` objects. All elements in *columns* must be of equal length.

If specified, *colnames* is an iterable of strings. Each item represents the name of the corresponding column in the dataset. The number of elements in *colnames* should be equal to the number of items in *columns*.

If specified, *header* is a dictionary whose keys and values are both strings containing mostly arbitrary metadata about the dataset. Some header names, however, are reserved; see *Special Headers*.

If specified, *sortkey* is a function that takes a *row* and returns a comparable. (See the `sort` method for a more precise definition.) This function will be used to determine the current sort order of the dataset. If

not supplied, the current sort order will be the order of the items as they appear in *columns*.

new_dataset_from_rows (*rows*_[, colnames, header, sort_key]) → Dataset

Creates a [Dataset](#) from an iterable (list, tuple, iterator, etc.) of rows.

rows is an iterable containing rows of data. The rows may be lists, tuples, other iterables or dictionaries. All elements in *rows* must be of equal length. If *rows* contains dictionaries, they must additionally all have the same keys.

If specified, *colnames* is an iterable of strings. Each item represents the name of the corresponding column in the dataset. The number of elements in *colnames* should be equal to the number of elements in any record in *rows*.

If *rows* contains dictionaries and *colnames* is supplied, every element in *colnames* must be a key in the dictionary contents of *rows*. If *colnames* is not supplied, the order of columns in the dataset will be the alphanumeric sort order of the keys of the dictionaries in *rows*.

If specified, *header* is a dictionary whose keys and values are both strings containing mostly arbitrary metadata about the dataset. Some header names, however, are reserved; see [Special Headers](#).

If specified, *sortkey* is a function that takes a [Row](#) object and returns a comparable. (See the `sort` method for a more precise definition.) This function will be used to determine the current sort order of the dataset. If not supplied, the current sort order will be insert order.

new_gridlines (*nickname*_[, ...]) → gridlines

Creates a gridlines from *nickname*. The *nickname* argument may be one of the following values:

- `horizontal`
- `vertical`

Additional arguments beyond *nickname* are scale or data specifications as they would be given to plots. (See [Creating grid lines](#) and [Creating Plots](#).)

new_gridlines_from_border (*nickname*, *border*_[, tickset_idx=0, ...]) → gridlines

Creates grid lines from *nickname*. The ticksets in *border* will be used to generate the positions for the lines.

The *nickname* argument may be one of the following values:

- `horizontal`
- `vertical`

border is a [tickable border](#) containing at least one tick set.

If supplied, *tickset_idx* is the index of the tickset to use to generate grid lines in *border*.

Additional arguments are scale or data specifications as they would be given to plots. (See [Creating grid lines](#) and [Creating Plots](#).)

new_gridlines_from_tick_tuples (*nickname*, *scale*, *tick_tuples*_[, ...]) → gridlines

Creates grid lines from *nickname*. The tick positions in *tick_tuples* will be used to generate the positions for the lines

The *nickname* argument may be one of the following values:

- `HorizontalGridLines`
- `VerticalGridLines`

tick_tuples will be used to generate the data for the `pos` axis of the grid lines.

scale will be used as the scale for the `pos` axis of the grid lines.

Additional arguments are scale or data specifications as they would be given to plots. (See [Creating grid lines](#) and [Creating Plots](#).)

new_labeler (*[txt, font_size="normal", font_family="default", font_style="normal", font_weight="normal", halign="left", valign="center", color="#000000FF", bgcolor, angle, border : border spec, border_top : border spec, border_bottom : border spec, border_left : border spec, border_right : border spec]*) → labeler

Creates a new *labeler*.

If *txt* is supplied, the labeler will be static – it will only ever render *txt*. Static labelers are convenient to use for simple labels that will be drawn only once, such as chart titles and captions. If *txt* is not supplied, the labeler can be drawn many times with different text. In this case, the labeler is used as a template of text properties.

The remaining arguments are described in detail in [Specifying Text Properties](#).

new_labeled_marker (*marker, labeler[, label_position="n", label_spacing=5]*) → labeled marker

Creates a new *labeled marker*.

marker is a *marker* that will be used to generate the marked part of the labeled marker.

labeler is a *labeler* that will be used to generate the labeled part of the labeled marker.

If supplied, *label_position* describes the position of the label relative to the marker. Valid values are n (above), s (below), e (to the left) and w (to the right).

If supplied, *label_spacing* is the amount of space between the label and the marker, in device units (points or pixels).

new_line (*nickname, color, width[, fixup=True]*) → line

Creates Rayon line from *nickname*.

The *nickname* argument may be one of the following values:

- dashed
- dotdash
- dotted
- none
- solid

color is the color of the line.

width is the width of the line in pixels/points.

If *fixup* is `True`, Rayon will attempt to “fix up” perfectly horizontal and vertical lines to avoid anti-aliasing. This behavior is disabled if *fixup* is `False`. (See [Lines and antialiasing](#).)

new_line_from_spec (*spec[, fixup=True]*) → line

Creates a Rayon line from *spec*, as described in [Specifying Line Styles](#).

spec is a string line specification.

If *fixup* is `True`, Rayon will attempt to “fix up” perfectly horizontal and vertical lines to avoid anti-aliasing. This behavior is disabled if *fixup* is `False`. (See [Lines and antialiasing](#).)

new_marker (*nickname[, ...]*) → marker

Creates a *marker* from *nickname*. Additional arguments will be passed to the marker subtype referred to by *nickname*.

The *nickname* argument may be one of the following values:

- darrow

- circle
- cross
- dot
- hline
- none
- vline

new_page_from_buffer (*buff*) → page

Create a *page* that can be used to render a *chart* into *buff*. *buff* should be an in-memory buffer object from the system associated with the rendering backend, representing a 2-dimensional pixel surface.

The rendering backend must support rendering to memory.

new_page_from_filename (*filename*, *width*, *height*) → page

Create a *page* that can be used to render a *chart* to *filename* with dimensions *width* and *height*. The type of file will be selected based on file extension. Currently supported file formats are SVG, PDF and PNG.

new_plot (*nickname*[, ...]) → plot

Creates a *plot* from *nickname*. Additional arguments will be passed to the plot subtype referred to by *nickname*.

The *nickname* argument may be one of the following values:

- bar
- field
- filledline
- hbar
- labeledscatter
- line
- scatter

new_scale (*nickname*[, ...]) → scale

Creates a *scale* from *nickname*. Additional arguments will be passed to the scale subtype referred to by *nickname*.

The *nickname* argument may be one of the following values:

- categorical
- catrange
- clog
- cloggrad
- gradient
- linear
- linbins
- log
- loggrad
- timebins

new_tickset_from_tuples (*nickname*[, *tick_tuples*, *scale*, *labeledmarker*, *alignment*=.5]) →
 tickset

Creates a tick set from a set of tick tuples representing where and how ticks should be created.

The *nickname* argument may be one of the following values:

- default
- ranged

tick_tuples is an iterable of 2-tuples (*value*, *label*). *value* should be a valid input to *scale*. *label* is a string representation of *value*. If *label* is None, Rayon will convert *value* to a string using `str` and use that as the label.

scale is a Rayon [scale](#) or a function that can serve as a scale. The output of *scale* should be a number between 0 and 1 if *nickname* is `default`. If *nickname* is `ranged`, *scale* should output a tuple (*min*, *max*) describing the minimum and maximum bounds of a range between 0 and 1. (See [Range Scales](#))

labeledmarker is a [labeled marker](#) that will be used to draw the tick mark.

If *nickname* is `ranged` and if supplied, *alignment* is a number between 0 and 1 specifying the alignment of the tick mark within the allotted range. A value of 0 for *alignment* will put the tick mark at the leftmost point in the range; a value of 1 will put it at the rightmost point.

new_tickset_from_spec (*nickname*[, *tick_spec*, *col*, *formatter*, *scale*, *marker*, *alignment*=.5]) →
 tickset

Creates a tick set from a string specification and a set of sortable data.

The *nickname* argument may be one of the following values:

- default
- ranged

tick_spec is a string of tick specifications, separated by commas. A tick specification may be one of the following:

Spec	Definition
at (<i>N</i> , <i>M</i>)	The literal value <i>N</i> , <i>M</i> , etc.. ('Put a tick at <i>N</i> ')
every (<i>N</i>)	The first point in the data, and every subsequent point at interval <i>N</i> ('Put a tick every <i>N</i> (e.g., every 10 minutes)')
ge (<i>N</i>)	All points in the data greater than or equal to <i>N</i>
gt (<i>N</i>)	All points in the data greater than <i>N</i>
le (<i>N</i>)	All points in the data less than or equal to <i>N</i>
lt (<i>N</i>)	All points in the data less than <i>N</i>
max	The largest value in the data
med	The median value in the data (a synonym for <code>p(50)</code>)
min	The smallest value in the data
n (<i>N</i>)	Place <i>N</i> ticks evenly along the scale, at 'nice' places
p (<i>N</i>)	The value for percentile <i>N</i>
smax	The largest value on the scale
smin	The smallest value on the scale

col is a column of data. (Either an iterable or a [Column](#) object.) This data will be used with the specifications to determine the tick positions.

scale is the scale to use to place the tickmark. (See [Scales](#).) It will also be used, if necessary, to compute the `smax` and `smin` tick specifications. The output of *scale* should be a number between 0 and 1 if *nickname* is `default`. If *nickname* is `ranged`, *scale* should output a tuple (*min*, *max*) describing the minimum and maximum bounds of a range between 0 and 1. (See [Range Scales](#)).

labeledmarker is a [labeled marker](#) that will be used to draw the tick marks.

If supplied, *formatter* is either a printf-style format string or a function that takes a value in *col* and returns a string that will be used to label the tick mark with that value. If *formatter* is `None`, Rayon supply a formatter that converts the input value to a string using `str` and use that as the label.

If *nickname* is `ranged` and if supplied, *alignment* is a number between 0 and 1 specifying the alignment of the tick mark within the allotted range. A value of 0 for *alignment* will put the tick mark at the leftmost point in the range; a value of 1 will put it at the rightmost point.

RYHILBERT

19.1 SYNOPSIS

```
ryhilbert [options]
```

19.2 DESCRIPTION

`ryhilbert` plots values on a Hilbert curve. A Hilbert curve preserves locality, meaning a range of values (`ryhilbert` can map numbers or IP address space) can be mapped into 2-dimensional space in such a way that values which are close to one another numerically are also close spatially. Because a Hilbert curve maps a single line, only one value is needed to compute a two-dimensional value (versus the two values required to plot a point in a scatterplot, for example).

Hilbert curves always take up a square; therefore, the range of numbers plotted is always a power of two.

`ryhilbert` input should be a column of numbers (or IP addresses) representing position (called the *indexes* or *index input*) and a column of numbers representing values to plot at that position. The index data should all be greater than or equal to 0. The values can be any decimal number.

Value data will be plotted either in a single color for any observation at that index, or as a color gradient indicating the quantity at that index.

19.3 CONFIGURATION

Every option available at the command line may be specified in a configuration file. For more information on the format of the configuration file, see [ryrc\(5\)](#).

19.4 REQUIRED ARGUMENTS

- input-path <file>** Required. A file containing the input data. If `--input-path` is a hyphen (-), input will be read from standard input. The data should be in the format described in [rydataformat\(5\)](#).
- output-path <file>** Required. A file containing the output visualization. If the file does not exist, it will be created; if it does exist it will be overwritten. The extension of the file determines the output file format. Understood extensions are `.png` (PNG), `.svg` (SVG), `.ps` (PostScript) and `.pdf` (PDF).

19.5 INPUT ASSOCIATION

`ryhilbert` can requires one of two kinds of input:

- A single-column of indexes. (E.g., the output of `rwsetcat(1)`.) In this case, `--binary-plot` is a required option.
- Two columns, one of which contains index data, and one of which contains value data.

Additional columns may be present in the data, but will be ignored.

The following options associate data with dimensions in the visualization.

- `--index-input <colspec>` A column specification (see `ryspects(5)`) of the data column to be used for position.
- `--val-input <colspec>` A column specification (see `ryspects(5)`) of the data column to be used for value input.

19.6 SCALING DATA

Index data is mapped 1:1 to discrete values on the Hilbert curve, and so is not scaled further. (The range of plottable values may be adjusted, however.) Value data is normally scaled linearly, but may be logarithmically scaled.

- `--log-colors` Flag. Compute color values on a logarithmic scale.
- `--index-max` The largest index value to be plotted on the Hilbert curve. This can be numeric (e.g. 255) or an IPv4 address in dotted-quad notation (e.g., 255.255.255.255, the largest allowable value). The value will be rounded up to the nearest power of 2.
- `--cidr-netmask <num>` A numeric value indicating the “resolution” of the visualization. Points on the curve will represent bins of size $2^{(32 - x)}$, where x is the value of `--cidr-netmask`. This notation was chosen to work will with IPv4 data, but can be used with both IP and numeric indexes.

19.7 FILTERING DATA

It may sometimes be desirable to ignore some of the input data. To facilitate this, `ryhilbert` defines four levels in the data:

- The *floor* is the “lowest plotted point.”
- The *ceiling* (abbreviated *ceil*) is the “highest plotted point.”
- Points below the floor are said to be in the *basement*. Basement points may optionally be visualized in a different color.
- Conversely, points above the ceiling are said to be in the *attic*. Like basement points, attic points may optionally be visualized in a different

These values apply only to the value data. Data is not filtered based on index.

- `--floor <num>` A numeric value. Data below this point in the input will not be visualized.
- `--floor-pct <pct>` A percentile value between 1 and 100. Data below this percentile in the input will not be visualized.
- `--ceil <num>` A numeric value. Data above this point in the input will not be visualized.

--ceil-pct <pct> A percentile value between 1 and 100. Data above this percentile in the input will not be visualized.

Colors for the floor, ceiling, basement and attic may be selected using options in the *DISPLAY* section.

19.8 DISPLAY

The following options control how the data is displayed and what *decorations* are applied to the visualization.

19.8.1 Size

The size of the Hilbert curve is a function of the resolution, as given by `--cidr-netmask`. After producing the visualization at that resolution, it may then be scaled to an arbitrary width and height using the following options. (This may introduce some distortions. At a minimum, it is recommended that width and height be equal, to preserve the aspect ratio of the original visualization.)

--width <num> The width of the image, as a number of points or pixels.

--height <num> The width of the image, as a number of points or pixels.

19.8.2 Selecting Colors

--num-colors <num> The number of color “steps” to be used in the color gradient between the floor and the ceiling.

--floor-color <color> A specification of the color to use for the “lowest” plotted value. (See *ryspects(5)*.)

--ceil-color <color> A specification of the color to use for the “highest” plotted value. (See *ryspects(5)*.)

--basement-color <color> A specification of the color to use for all points below the “floor” value. (See *ryspects(5)*.)

--attic-color <color> A specification of the color to use for all points above the “ceiling” value. (See *ryspects(5)*.)

--binary-plot Flag. Instead of plotting quantity on a color gradient, color each position on the curve containing any data. `--ceil-color` will be used as the color.

--background-color <color> A specification of the color to use for the background of the visualization. (See *ryspects(5)*.)

19.8.3 Overlays

A user may overlay a PNG image highlighting regions of the curve, to provide the user with context when reading this visualization. `ryhilbert` ships with an overlay subdividing the 2^{32} IPv4 addresses by the entities to which they were assigned by the Internet Assigned Numbers Authority.

--overlay-file <file> A PNG image to use as the overlay. The dimensions of the image should be square, and should match the resolution of the image (given using `--cidr-netmask`), such that both width and height of the image are $2^{(x/2)}$ pixels on a side, where x is the resolution given by `--cidr-netmask`.

--no-overlay Flag. Don’t use an overlay image.

19.9 GENERAL OPTIONS

--quiet Flag. Produce no output on standard output or standard error. Otherwise, certain warnings may be emitted for values `ryhilbert` deems questionable.

19.10 EXAMPLES

Create a PNG visualization using the default options:

```
ryhilbert --input-path foo.txt --output-path bar.png
```

Generate a binary plot from a SiLK IP set using the SiLK `rwsetcat(1)` tool:

```
rwsetcat foo.set | \  
ryhilbert --input-path - --output-path bar.png --binary-plot
```

Generate a plot from a SiLK bag using the SiLK `rwbagcat(1)` tool. Lower values in light blue, higher values in dark blue, with values over the 90th percentile in red:

```
rwbagcat foo.bag | \  
ryhilbert --input-path - --output-path bar.png \  
  --floor-color 000088 \  
  --ceil-color  0000ff \  
  --ceil-pct 90 \  
  --attic-color ff0000
```

19.11 SEE ALSO

rytools(5)

RYCATEGORIES

20.1 SYNOPSIS

```
rycategories [options]
```

20.2 DESCRIPTION

`rycategories` reads data from a file or standard input and writes a plot of the data to a PNG, SVG, PDF or PostScript file. `rycategories` renders categorical data—data that is aggregated by a set of discrete, unordered categories.

Categorical data can be presented in a number of ways. Currently, `rycategories` only supports presentation as a bar plot. Other presentation styles are planned for the future.

20.2.1 Ordering

Categories in categorical data should not, in theory, have any order relationship to other categories. However, in practice when visualizing categorical data it is sometimes useful to order categories. For example, it may be easier for a viewer to locate categories of interest if the categories are ordered alphabetically, or in such a way that “families” of categories can be grouped together.

In recognition of this, `rycategories` will treat the order of data in the input file as a form of sort order, and will use this order to lay out categories if no other order is specified (e.g., with `--reverse-cats`). If rows with different keys are interleaved, the order used will be the relative order of the first rows to contain the keys.

20.3 CONFIGURATION

Every option available at the command line may be specified in a configuration file. For more information on the format of the configuration file, see *ryrc(5)*.

20.4 REQUIRED ARGUMENTS

--input-path <file> Required. A file containing the input data. If `--input-path` is a hyphen (-), input will be read from standard input. The data should be in the format described in *rydataformat(5)*.

--output-path <file> Required. A file containing the output visualization. If the file does not exist, it will be created; if it does exist it will be overwritten. The extension of the file determines the output file format. Understood extensions are `.png` (PNG), `.svg` (SVG), `.ps` (PostScript) and `.pdf` (PDF).

20.5 INPUT ASSOCIATION

The following options associate data with dimensions in the visualization.

--cat-input <colspec> A column specification (see *ryspects(5)*) of the data column to be used for categorical input. This will determine the number and placement of bars on the plot.

--val-input <colspec>

A column specification (see *ryspects(5)*) of the data column to be used for categorical input. This will determine the bars' height. (Or width, if `--horizontal` is supplied.)

20.6 SCALING DATA

The following options control how data is scaled to the dimensions of the visualization.

--val-scale <scale> The type of scale to use on the value axis, which determines bar height (width if `--horizontal` is supplied). See *ryspects(5)* for options. By default, a linear scale is used.

--val-scale-min <num> A numeric value representing the lowest end of the value axis scale. This is not the same thing as the origin of the bar; set that using `--bar-origin`.

--bar-origin <num> A numeric value representing the point on the axis where the bars originate. When the value data is all positive, this is always the bottom of the bar; when the value data is all negative, this is always the top of the bar.

This value defaults to 1 if `--val-scale` is `log`, otherwise 0.

20.7 DISPLAY

The following options control how the data is displayed and what *decorations* (titles, captions, tick marks) are applied to the visualization.

--width <num> The width of the image, as a number of points or pixels.

--height <num> The width of the image, as a number of points or pixels.

--padding <num> A number of pixels or points of padding to applied uniformly to each side of the image. A `--padding` value of 10, for instance, will apply ten pixels/points of padding to each of the top, bottom, left and right edges, reducing the drawable width and height by 20 pixels/points.

--pad-top <num> A number of pixels or points of padding to add to the top edge of the image.

--pad-bottom <num> A number of pixels or points of padding to add to the top edge of the image.

--pad-left <num> A number of pixels or points of padding to add to the top edge of the image.

--pad-right <num> A number of pixels or points of padding to add to the top edge of the image.

--title <string> Title of the visualization, printed on the top in a large typeface.

--caption <string> Caption of the visualization, printed on the bottom in a smaller typeface.

20.7.1 Horizontal Bar Plots

--horizontal Flag. Render the data using horizontal bars arranged from top to bottom. (Default behavior is vertical bars arranged left to right.)

20.7.2 Category Layout

--reverse-cats Flag. Reverse the order of the bars from their position in the dataset.

20.7.3 Bars

--bar-width <num> A number between 0 and 1, representing the proportion of available space each bar will use. For example, `--bar-width .4` directs `rycategories` to use 40% of the allocated space for each bar. `--bar-width 1` means use all allocated space, leaving no space between bars. Empty space will always be allocated evenly to either side of the bar.

If the `--horizontal` option is supplied, `--bar-width` refers to the vertical dimension of the bar, rather than the horizontal dimension.

--bar-fill-color <color> A specification of the interior color of the bar. (See [*ryspecs\(5\)*](#).)

--bar-border-color <color> A specification of the color of the border of the bar. (See [*ryspecs\(5\)*](#).)

--bar-border-line-style <style> The style of line to draw as the bar border. (See [*ryspecs\(5\)*](#).)

--bar-border-width <size> The width of the bar border, in pixels or points.

20.7.4 Displaying Tickmarks

--val-ticks <tickspec> A specification of where to place tick marks on the value axis. (See [*ryspecs\(5\)*](#).)

--no-cat-ticks Flag. Do not place tick marks on each category.

--cat-tick-size <size> The length of the category tickmarks, in pixels or points.

--val-tick-size <size> The length of the value tickmarks, in pixels or points.

--cat-label-angle <angle> A number from 0 to 360, indicating the angle that the category tickmark labels should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.

--val-label-angle <angle> A number from 0 to 360, indicating the angle that the value tickmark labels should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.

--cat-label-halign <halign> One of the values “left”, “center” or “right,” indicating whether the category tickmark label should be aligned to the left, center or right of the text, respectively.

- cat-label-valign <valign>** One of the values “top”, “center” or “bottom,” indicating whether the category tickmark label should be aligned to the top, center or bottom of the text, respectively.
- val-label-halign <halign>** One of the values “left”, “center” or “right,” indicating whether the value tickmark label should be aligned to the left, center or right of the text, respectively.
- val-label-valign <valign>** One of the values “top”, “center” or “bottom,” indicating whether the value tickmark label should be aligned to the top, center or bottom of the text, respectively.
- cat-label-spacing <num>** Space between the category tickmark and its label, as a number of points or pixels.
- val-label-spacing <num>** Space between the value tickmark and its label, as a number of points or pixels.

20.7.5 Gridlines

- vgrid** Flag. Draw vertical grid lines.
- vgrid-color <color>** A specification of the color of the vertical grid lines. (See *ryspects(5)*.)
- vgrid-style <stylespec>** The style of line to draw for the vertical grid lines.
- vgrid-width <num>** A number indicating the width of the vertical grid lines, in points or pixels.
- vgrid-lines-at <tickspec>** A specification of where to place the vertical grid lines, relative to the horizontal axis. (See *ryspects(5)*.)
- hgrid** Flag. Draw horizontal grid lines.
- hgrid-color <color>** A specification of the color of the horizontal grid lines. (See *ryspects(5)*.)
- hgrid-style <stylespec>** The style of line to draw for the horizontal grid lines.
- hgrid-width <num>** A number indicating the width of the horizontal grid lines, in points or pixels.
- hgrid-lines-at <tickspec>** A specification of where to place the horizontal grid lines, relative to the vertical axis. (See *ryspects(5)*.)

20.7.6 Borders and Border Labels

- bottom-border-line-style <style>** The style of the line to draw (if any) along the bottom axis of the frame to plot. (See *ryspects(5)* for valid line style values.)
- left-border-line-style <style>** The style of the line to draw (if any) along the left axis of the frame to plot. (See *ryspects(5)* for valid line style values.)

20.7.7 Layout, Decoration and Annotation

- chart-bgcolor <color>** A specification of the color of the background of the entire subchart. (See *ryspects(5)*.)
- plot-bgcolor <color>** A specification of the color of the background of just the plot or plots. (See *ryspects(5)*.)

20.8 EXAMPLES

Create PDF visualization using the default options:

```
rycategories --input-path foo.txt --output-path bar.pdf
```

Using the output from the SiLK [rwuniq\(1\)](#) tool, plot the byte volume of traffic in the file `input.rw`, grouped by source IP:

```
rwuniq --fields=sip --bytes \  
  --no-titles input.rw | \  
rycategories --input-path - \  
  --output-path bar.png
```

20.9 SEE ALSO

rytools(5)

RYPIECHART

21.1 SYNOPSIS

`rypiechart [options]`

21.2 DESCRIPTION

`rypiechart` is a tool for generating pie charts. Pie charts are a ubiquitous way to visualize proportional data.

21.2.1 A Caution

Before proceeding, this note from the Wikipedia page on pie charts (as of October 2012) is appropriate:

Statisticians generally regard pie charts as a poor method of displaying information, and they are uncommon in scientific literature. One reason is that it is more difficult for comparisons to be made between the size of items in a chart when area is used instead of length and when different items are shown as different shapes.

Further, in research performed at AT&T Bell Laboratories, it was shown that comparison by angle was less accurate than comparison by length....Most subjects have difficulty ordering the slices in [a] pie chart by size; when [a] bar chart is used the comparison is much easier. Similarly, comparisons between data sets are easier using the bar chart. However, if the goal is to compare a given category (a slice of the pie) with the total (the whole pie) in a single chart and the multiple is close to 25 or 50 percent, then a pie chart can often be more effective than a bar graph.

[T]he research of Spence and Lewandowsky did not find pie charts to be inferior. Participants were able to estimate values with pie charts just as well as with other presentation forms.

21.2.2 Alternatives to Pie Charts

The `rytools` suite provides the `rycategories` tool for visualizing data using a bar plot.

21.3 CONFIGURATION

Every option available at the command line may be specified in a configuration file. For more information on the format of the configuration file, see *ryrc(5)*.

21.4 REQUIRED ARGUMENTS

- input-path <file>** Required. A file containing the input data. If **--input-path** is a hyphen (-), input will be read from standard input. The data should be in the format described in *rydataformat(5)*.
- output-path <file>** Required. A file containing the output visualization. If the file does not exist, it will be created; if it does exist it will be overwritten. The extension of the file determines the output file format. Understood extensions are `.png` (PNG), `.svg` (SVG), `.ps` (PostScript) and `.pdf` (PDF).

21.5 INPUT ASSOCIATION

- cat-input <colspec>** A column specification (see *ryspecs(5)*) of the data column containing the *category* of the observation. This will be used to label the observation in the chart.
- val-input <colspec>**
- A column specification (see *ryspecs(5)*) of the data column containing the observation values. This will determine the observation's share of the pie.
- color-input <colspec>** A column specification (see *ryspecs(5)*) of the data column containing colors associated with this observation. This will determine the color of the share of the pie corresponding to this observation.

21.6 DISPLAY

- width <num>** The width of the image, as a number of points or pixels.
- height <num>** The width of the image, as a number of points or pixels.
- title <string>** Title of the visualization, printed on the top of the visualization.
- background-color <color>** A specification of the color of the background of the visualization. (See *ryspecs(5)*.)

21.7 EXAMPLES

Visualize data in `foo.txt` using the defaults:

```
rypiechart --input-path foo.txt --output-path bar.png
```

21.8 SEE ALSO

rytools(5)

RYSCATTERPLOT

22.1 SYNOPSIS

```
rscatterplot [options]
```

22.2 DESCRIPTION

`rscatterplot` reads data from a file or standard input and writes a scatterplot of the data to a PNG, SVG, PDF or PostScript file. The user may specify which columns of data to plot, parts of the plotted data columns to ignore, how to style the visualization, and whether to plot a trend line in addition to the data.

22.3 CONFIGURATION

Every option available at the command line may be specified in a configuration file. For more information on the format of the configuration file, see *ryrc(5)*.

22.4 REQUIRED ARGUMENTS

- input-path <file>** Required. A file containing the input data. If `--input-path` is a hyphen (-), input will be read from standard input. The data should be in the format described in *rydataformat(5)*.
- output-path <file>** Required. A file containing the output visualization. If the file does not exist, it will be created; if it does exist it will be overwritten. The extension of the file determines the output file format. Understood extensions are `.png` (PNG), `.svg` (SVG), `.ps` (PostScript) and `.pdf` (PDF).

22.5 MARKERS

The following options control the presentation of markers on the scatterplot. A marker's spatial position, size and color can all be determined from the input data. In addition, marker size and color can be set to a static value if the user chooses not to map data onto this axis.

- marker <marker>** A specification of the shape of the marker. (See *ryspecs(5)*.)

- marker-color <color>** If the marker color will not vary with data, a specification of the color of the marker. (See [ryspecs\(5\)](#).)
- marker-color-input <colspec>** If the marker color will vary with data, the name or index of the column in the input data to use for marker color.
- marker-color-scale-min <color>** A specification of the color to be used for the lowest value in `--marker-color-input`. (See [ryspecs\(5\)](#).)
- marker-color-scale-max** A specification of the color to be used for the highest value in `--marker-color-input`. (See [ryspecs\(5\)](#).)
- marker-size <num>** If the marker size will not vary with data, a number specifying the size of the marker, in points or pixels. (This size value will be interpreted differently for different marker sizes, with the intent of producing a mark that fits in a square no greater than `--marker-size-input` pixels/points on a side.)
- marker-size-input <colspec>** If the marker size will vary with data, the name or index of the column in the input data to use for marker size. (See [ryspecs\(5\)](#).)
- marker-size-scale-min <num>** A specification of the size to be used for the lowest value in `--marker-size-input`.
- marker-size-scale-max <num>** A specification of the size to be used for the highest value in `--marker-size-input`.

22.6 INPUT ASSOCIATION

The following options associate data with dimensions in the visualization.

- x-input <colspec>** A column specification (see [ryspecs\(5\)](#)) of the data column to be plotted on the X axis.
- y-input** A column specification (see [ryspecs\(5\)](#)) of the data column to be plotted on the Y axis.

22.7 SCALING DATA

The following options control how data is scaled to the dimensions of the visualization.

- x-scale <scale>** The type of scale to use on the X axis. (See [ryspecs\(5\)](#).) By default, a linear scale is used.
- y-scale <scale>** The type of scale to use on the y axis. (See [ryspecs\(5\)](#).) By default, a linear scale is used.

By default, `ryscatterplot` will start both the X and Y axes at zero if the input data for that axis is all positive, and with end axes at the highest points of their respective input data. The following options allow the user to manually set the high and low points of the X and Y scales.

- x-scale-dont-zero** Flag. Start the X axis scale at the lowest point in the X data, instead of zero.
- y-scale-dont-zero** Flag. Start the Y axis scale at the lowest point in the X data, instead of zero.
- x-scale-min <num>** A numeric value representing the lowest end of the X axis scale.
- y-scale-min <num>** A numeric value representing the lowest end of the Y axis scale.
- x-scale-max <num>** A numeric value representing the highest end of the X axis scale.
- y-scale-max <num>** A numeric value representing the highest end of the Y axis scale.

22.8 FILTERING DATA

It may sometimes be desirable to ignore some of the input data. The following options specify which input data should be ignored.

- x-floor <num>** A numeric value. Data below this point in the input will not be visualized.
- y-floor <num>** A numeric value. Data below this point in the input will not be visualized.
- x-floor-pct <pct>** A percentile value between 1 and 100. Values below this percentile in the data will not be visualized.
- y-floor-pct <pct>** A percentile value between 1 and 100. Values below this percentile in the data will not be visualized.
- x-ceiling <num>** A numeric value. Data above this point in the input will not be visualized.
- y-ceiling <num>** A numeric value. Data above this point in the input will not be visualized.
- x-ceiling-pct <pct>** A percentile value between 1 and 100. Values above this percentile in the data will not be visualized.
- y-ceiling-pct <pct>** A percentile value between 1 and 100. Values above this percentile in the data will not be visualized.

22.9 STATISTICS

`rscatterplot` can fit a trend line to the data. The following options control the trending method used and the line properties.

- trend-line <trendtype>** Add a trend line showing central tendency over time; the argument determines the algorithm used to compute the central tendency. Valid values are `kernel` (kernel smoothing using a [Nadaraya-Watson](#) estimator), `ols` (linear regression using the [ordinary least squares](#) method), and `moving_avg` (calculation of averages over a sliding time window).
- trend-line-color <color>** A specification of the color of the trend line. (See [ryspects\(5\)](#).)
- trend-line-width <num>** The width of the trend line, in pixels or points.
- trend-untrimmed-data** Flag. Normally, the data used to compute the trend is the input data after it has been filtered through `--x-floor`, `--x-ceiling`, `--y-floor`, `--y-ceiling`, `--x-floor-pct`, `--y-floor-pct`, `--x-ceiling-pct`, or `--y-ceiling-pct`. If this flag is supplied, the raw data will be used before it was trimmed by these options.

22.10 DISPLAY

The following options control how the data is displayed and what *decorations* (titles, captions, tick marks) are applied to the visualization.

- width <num>** The width of the image, as a number of points or pixels.
- height <num>** The width of the image, as a number of points or pixels.
- padding <num>** A number of pixels or points of padding to applied uniformly to each side of the image. A `--padding` value of 10, for instance, will apply ten pixels/points of padding to each of the top, bottom, left and right edges, reducing the drawable width and height by 20 pixels/points.
- pad-top <num>** A number of pixels or points of padding to add to the top edge of the image.

- pad-bottom <num>** A number of pixels or points of padding to add to the top edge of the image.
- pad-left <num>** A number of pixels or points of padding to add to the top edge of the image.
- pad-right <num>** A number of pixels or points of padding to add to the top edge of the image.
- title <string>** Title of the visualization, printed on the top in a large typeface.
- caption <string>** Caption of the visualization, printed on the bottom in a smaller typeface.

22.10.1 Borders and Border Labels

- bottom-border-line-style <style>** The style of the line to draw (if any) along the bottom axis of the frame to plot. (See *ryspects(5)* for valid line style values.)
- left-border-line-style <style>** The style of the line to draw (if any) along the left axis of the frame to plot. (See *ryspects(5)* for valid line style values.)
- bottom-label <string>** A string label which will be printed below the scatterplot.
- left-label <string>** A string label which will be printed to the left of the scatterplot.
- x-label-angle <angle>** A number from 0 to 360, indicating the angle that the left label should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.
- y-label-angle <angle>** A number from 0 to 360, indicating the angle that the bottom label should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.
- x-label-halign <halign>** One of the values “left”, “center” or “right,” indicating whether the left label should be aligned to the left, center or right of the text, respectively.
- x-label-valign <valign>** One of the values “top”, “center” or “bottom,” indicating whether the left label should be aligned to the top, center or bottom of the text, respectively.
- y-label-halign <halign>** One of the values “left”, “center” or “right,” indicating whether the bottom label should be aligned to the left, center or right of the text, respectively.
- y-label-valign <valign>** One of the values “top”, “center” or “bottom,” indicating whether the bottom label should be aligned to the top, center or bottom of the text, respectively.
- x-label-spacing <num>** Space between the bottom border and its label, as a number of points or pixels.
- y-label-spacing <num>** Space between the left border and its label, as a number of points or pixels.

22.10.2 Layout, Decoration and Annotation

- chart-bgcolor <color>** A specification of the color of the background of the entire subchart. (See *ryspects(5)*.)
- plot-bgcolor <color>** A specification of the color of the background of just the plot or plots. (See *ryspects(5)*.)

22.10.3 Displaying Tickmarks

--x-ticks <tickspec> A specification of where to place tick marks on the X axis. (See *ryspects(5)*.)

--y-ticks <tickspec> A specification of where to place tick marks on the Y axis. (See *ryspects(5)*.)

22.11 GRIDDED SCATTERPLOTS

`ryscatterplot` can generate grids of scatterplot data as well as single plots. This can be useful for rapidly comparing sets of similar data to each other.

In order to be plotted in a grid scatterplot, a dataset must contain an additional column called a *key column*. This column effectively partitions the dataset into smaller datasets; each of the smaller datasets shares the same label in the key column.

The following switches control gridded display of scatterplots.

--grid-plot Flag. Enable gridded scatterplot display.

--grid-key-input <colname> A column name or index. Each unique value in the column will form a cell in the scatterplot grid.

--grid-label <label> Specifies a label to be given to each subplot. The label may contain the wildcard pattern `%s`, which will be replaced with the label in the grid key column for that subplot.

22.12 DEPRECATED OPTIONS

The following options will work, but have been deprecated. They will be removed in Rayon 2.x. Where applicable, alternatives are provided.

Deprecated option	Alternative
--xborder	--bottom-border-line-style
--yborder	--left-border-line-style
--xticks	--x-ticks
--yticks	--y-ticks
--xscale	--x-scale
--xscale-max	--x-scale-max
--xscale-min	--x-scale-min
--yscale	--y-scale
--yscale-max	--y-scale-max
--yscale-min	--y-scale-min
--xfloor	--x-floor
--xfloor-pct	--x-floor-pct
--xceiling	--x-ceiling
--xceiling-pct	--x-ceiling-pct
--yfloor	--y-floor
--yfloor-pct	--y-floor-pct
--yceiling	--y-ceiling
--yceiling-pct	--y-ceiling-pct
--xlabel	--bottom-label
--ylabel	--left-label
--background-color	--chart-bgcolor
--grid	--grid-plot

22.13 EXAMPLES

Create a visualization in PDF format using the default options:

```
ryscatterplot --input-path foo.txt --output-path bar.pdf
```

Using data from the SiLK `rwuniq(1)` tool, plot the number of bytes in for all netflows grouped by source IP address against the number of records. Output will go to the PNG file `bar.png`:

```
rwuniq --fields=sip --bytes --flows --no-titles | \  
ryscatterplot --x-input=1 --y-input=2 --output-path=bar.png
```

Visualize a grid of scatterplots from data in `foo.txt`:

```
ryscatterplot --input-path=foo.txt \  
  --output-path=bar.png \  
  --grid \  
  --grid-key-input=2 \  
  --grid-label="Server %s"
```

22.14 SEE ALSO

rytools(5)

RYTIMESERIES

23.1 SYNOPSIS

```
rytimeseries [options]
```

23.2 DESCRIPTION

`rytimeseries` is a command-line tool for generating visualizations from time-series data.

Some features of `rytimeseries` include:

- Specification of different visualization styles.
- Support for many ways of extracting time series from input data.
- Visualization of single or multiple time series.
- Automatic (re-)binning of input data, if desired.
- Visualization of data's central tendency and variability.
- Annotations of highest points in data.

23.3 TERMINOLOGY

This section describes the visual layout of an `rytimeseries` visualization, and defines terms that will be used later in this document to refer to parts of the visualization.

One can think of an `rytimeseries` visualization as a list of smaller visualizations, laid out vertically. We call these visualizations *subcharts*. Each subchart can plot up to 2 data series, or *sides*. On the *top side*, the horizontal axis represents time and runs from left to right; the vertical axis represents value, with larger values above smaller ones. On a *1-sided* subchart, the top side takes up the full vertical space. A *2-sided* subchart has both a top and *bottom side*. On the bottom side, the horizontal axis again represents time and runs left to right, and the vertical axis represents value, but the axis is inverted, such that larger values are below smaller ones. Each side takes 50% of the vertical space, with the lowest point of their value scale occupying the center.

When data only exists for one subchart, that chart takes up all available horizontal and vertical space. When the data suggests multiple subcharts, an equal amount of space to the left of each subchart is set aside for a *chart label* to identify each subchart.

23.4 CONFIGURATION

Every option available at the command line may be specified in a configuration file. For more information on the format of the configuration file, see *ryrc(5)*.

23.5 READING DATA

Time series data can be embedded in tabular data in many different ways. *rytimeseries* supports the extraction of this data in several ways using the `--top-column`, `--bottom-column`, `--top-filter` and `--bottom-filter` options. (See *Input*.) The `--top-column` and `--bottom-column` select the columns in the input from which the data for the top and bottom sides of the subcharts will come. The `--top-filter` and `--bottom-filter` options express which rows in those columns will be displayed on the top and bottom sides as tests. The tests can be done on any item of data in the row, not just the data that will be displayed in the top or bottom side.

23.5.1 Filter Expressions

The `--top-filter` and `--bottom-filter` options take filter expressions as their values. A filter expression has the following form:

```
colspec oper value
```

Where *colspec*, *oper* and *value* are defined as:

colspec A column name or index, enclosed in parentheses. For a three-column dataset with column names *foo*, *bar* and *baz*, valid *colspecs* would be `[0]`, `[1]`, `[2]`, `[foo]`, `[bar]`, and `[baz]`.

oper Either `==`, indicating that the value in *colspec* should be equal to *value*, or `!=`, indicating that the value in *colspec* should NOT be equal to *value*.

value A value to compare to the value at each row in *colspec*.

The following are valid filter expressions for a 4-column dataset with column names *foo*, *bar*, *baz*, and *quux*:

`[0]==1` Selects all rows for which the value in the first column is 1.

`[foo]==1` Identical to the preceding example.

`[3]!=gargle` Selects all rows for which the value in the fourth column is NOT the string “gargle.”

`[quux]!=1` Identical to the preceding example.

Filter expressions will be coerced into the data type of the column. So a value of “1.2.3.4” will match the string “1.2.3.4” or the IP address 1.2.3.4, depending on the type of the column.

23.6 VISUALIZATION STYLES

rytimeseries can visualize data in one of four styles: `dots`, `lines`, `filled_lines`, and `bars`.

Style	Result
<code>dots</code>	A scatterplot. Individual observations are plotted as marks (usually dots).
<code>lines</code>	A line plot. Lines are drawn from one observation to the next.
<code>filled_lines</code>	Similar to <code>lines</code> , but the area underneath the line is shaded (“filled”).
<code>bars</code>	A bar plot. Each bar represents a time range. Bar height signifies the sum of all observations in that range.

The `--style` argument controls which visual style to use. If the user specifies no style, `rytimeseries` will use the `dots` style.

23.7 ARGUMENTS AND OPTIONS

The following arguments and options control `rytimeseries` behavior.

23.7.1 Required Arguments

--input-path <file> Required. A file containing the input data. If `--input-path` is a hyphen (-), input will be read from standard input. The data should be in the format described in *rydataformat*.

--output-path <file> Required. A file containing the output visualization. If the file does not exist, it will be created; if it does exist it will be overwritten. The extension of the file determines the output file format. Understood extensions are `.png` (PNG), `.svg` (SVG), `.ps` (PostScript) and `.pdf` (PDF).

23.7.2 Input

--first-line-colnames If this option is specified, `rytimeseries` will extract column names from the first line of input, ignoring it as data. Use this when reading input from tools such as SiLK, where this form is the standard. You don't need to specify this flag if the column name information is in the format described in *rydataformat*, but doing so is harmless.

--group-by <colname> A column name or index. Each unique value in the column will form a subchart in the visualization.

--show A list of unique values in the `--group-by` column. Only subcharts for these values will be drawn.

--labels <list> A list of values. If `--show` is supplied, this list should have the same number of items. Otherwise, it should have the same number of unique values as there are in the `--group-by` column. (If `--group-by` is not supplied, this option is ignored.) These names will be substituted, in order, for values in `--group-by` for the purposes of labeling subcharts.

--time-column <colname> A column name or index of a column containing datetime values. The data in this column will be used for time values for all sides and subcharts.

--top-column <colname> A column name or index of a column containing numeric values. The data in this column will be used to populate the top side of the subcharts.

--top-filter <filter-expr> A filter expression (see *Filter Expressions*) that will be used to select data from `top-column` to display on the top side.

--bottom-column <colname> A column name or index of a column containing numeric values. The data in this column will be used to populate the bottom side of the subcharts.

--bottom-filter <filter-expr> A filter expression (see *Filter Expressions*) that will be used to select data from `top-column` to display on the bottom side.

--start-time <time> A datetime, in ISO-8601 format. Input data before to this time will be ignored.

--end-time <time> A datetime, in ISO-8601 format. Input data after this time will be ignored.

- value-min <num>** A numeric value. Data points less than this value (after binning, if performed) will be ignored.
- value-max <num>** A numeric value. Data points greater than this value (after binning, if performed) will be ignored (or plotted as outliers; see [Displaying Outliers](#))
- value-min-pct <int>** An integer between 0 and 100, representing a percentile. For the data to be plotted, data points in percentiles less than this will be ignored (or plotted as outliers; see [Displaying Outliers](#))
- value-max-pct <int>** An integer between 0 and 100, representing a percentile. For the data to be plotted, data points in percentiles greater than this will be ignored (or plotted as outliers; see [Displaying Outliers](#))
- fix-scale-min <num>** A numeric value signifying the fixed lower bound of the value scale. (If not supplied, the lower bound will be the lowest-valued data point if there is negative data, or 0 if all the data has positive value.) If multiple plots are generated, they will ALL use this minimum scale. Points below this threshold are not plotted, regardless of the value of `--value-min`. (`--value-min` may still be used to make `rytimeseries` ignore values between `--value-min` and `--fix-scale-min`.)
- fix-scale-max <num>** A numeric value signifying the fixed upper bound of the value scale. (If not supplied, the upper bound will be the highest-valued data point.) If multiple plots are generated, they will ALL use this maximum scale. Points above this threshold are not plotted, regardless of the value of `--value-max`. (`--value-max` may still be used to make `rytimeseries` ignore values between `--value-max` and `--fix-scale-max`.)

23.7.3 Sorting

- presorted-input** Flag. By default, `rytimeseries` will sort incoming data on the value column. Passing this switch indicates that the input is already sorted by time. This option can speed up processing, but will cause errors if the data is not actually sorted.

23.7.4 Scaling

- value-scale <scaletype>** Type of scale to use for values. Valid values for this option are `linear` (linear scale), `log` (log scale), or `clog` (a “counting log” scale that distorts values between 0 and 1, but accepts zero as a value). By default, a linear scale is used.

23.7.5 Binning

Some visualizations (such as barplots) make more sense when the input data is binned. `rytimeseries` supports the binning (or re-binning) of input data. (Note, however, that it may be more efficient to do this prior to calling `rytimeseries`. For instance, when dealing with [SiLK](#) data, it will be faster to run your output through `rwcount` rather than using the binning in `rytimeseries`.)

- bin-size** A `timedelta`, in ISO-8601 format. If the data is to be binned, this specifies the size of the bins.
- prebinned-input** Indicates that the input is already in *bin-size* bins. This option can speed up processing, but will cause errors if the data is not actually correctly binned.

23.8 STATISTICS

In addition to displaying the raw data, `rytimeseries` can visualize basic statistics on the central tendency and variation of the data.

- trend-line <trendtype>** Add a trend line showing central tendency over time; the argument determines the algorithm used to compute the central tendency. Valid values are `kernel` (kernel smoothing using a [Nadaraya-Watson](#) estimator), `ols` (linear regression using the [ordinary least squares](#) method), and `moving_avg` (calculation of averages over a sliding time window).
- trend-line-color <color>** A specification of the color of the trend line. (See [ryspects\(5\)](#).)
- trend-line-width <num>** The width of the trend line, in pixels or points.
- variation-field <fieldtype>** Add a color field showing the variation of the data over time; the argument determines the algorithm used to compute variation. Currently, the only valid value is `stdev` (moving standard deviation, expressed relative to the moving average).
- variation-field-color <color>** A specification of the color of the variation field. (See [ryspects\(5\)](#).)
- variation-line-color <color>** A specification of the color of the line surrounding the variation field. (See [ryspects\(5\)](#).)
- variation-line-width <num>** The width of the line surrounding the variation field, in pixels or points.
- variation-line-style <style>** The style of line to draw around the variation field. (See [ryspects\(5\)](#).)

23.9 DISPLAY

The following options control how the data is displayed and what *decorations* (titles, captions, tick marks) are applied to the visualization.

- style** Required. Select the visualization style. Valid values are `dots`, `lines`, `filled_lines` and `bars`. Default is `dots`. (See [VISUALIZATION STYLES](#).)
- value-units <string>** Units in which the value is measured.
- top-label <string>** Label to be printed on the top half of visualizations, presumably representing the way the data on top is different from the data on the bottom.
- bottom-label <string>** Label to be printed on the bottom half of visualizations, presumably representing the way the data on bottom is different from the data on the top.
- title <string>** Title of the visualization, printed on the top in a large typeface.
- caption <string>** Caption of the visualization, printed on the bottom in a smaller typeface.
- draw-as-multiple** Flag. Draw a single timeseries in the style used for drawing multiple timeseries. This is useful when “stitching” a multiple-series visualization together using individual images. (As an HTML page, for example.)
- no-timeline** Flag. Do not draw labeled time axis on bottom of visualization. This may be used with `--draw-as-multiple` to generate composite visualizations using individual images.
- group-label-size <sizespec>** A textual size specification of the size of the tick mark label. (See [ryspects\(5\)](#).) For example, to specify a tick label size of 12 pixels, use a value of `12px`.

23.9.1 Displaying Outliers

Instead of ignoring the data trimmed with `--value-max`, the user may instead wish to plot it at the edge of the data area as outliers. The following options control this behavior.

- `--plot-high-outliers` Flag. Display values above those of `--value-max` as outliers. Outliers will be plotted at the appropriate position on the time axis, and at the very top of the value axis.
- `--outlier-marker-color <color>` A specification of the color of the outlier marker. (See *ryspects(5)*.)
- `--outlier-marker-size <num>` Size of the outlier marker, as a number of points or pixels.
- `--outlier-marker-shape <shape>` Shape of the outlier marker.

23.9.2 Width and Height

The following options control the dimensions of the output image `rytimeseries` generates.

By default, the number of subcharts in a visualization will determine the height of the image; if desired, the user can specify a static height regardless of the number of subcharts.

- `--width <num>` The width of the image, as a number of points or pixels. By default, `rytimeseries` will use a width of 800 pixels.
- `--height <num>` The height of the image, as a number of points or pixels. By default, `rytimeseries` will determine height dynamically. See `--height-per-subchart`.)
- `--height-per-subchart <num>` The height of the image, as a number of points or pixels per subchart. (e.g., `--height-per-subchart 100` would yield a 100-pixel height for one subchart, 200 pixels for two, 1000 pixels for ten, etc.) By default, `rytimeseries` will allocate 450 pixels per subchart.

23.9.3 Visualization Style Options

The following options are used for specific values of the `--style`. If the `--style` value does not apply to these options, they will be ignored.

`dots`

- `--marker-color <color>` A specification of the color of the observation marker. (See *ryspects(5)*.)
- `--marker-size <num>` Size of the observation marker, as a number of points or pixels.
- `--marker-shape <shape>` Shape of the observation marker.

`lines`

- `--line-color <color>` A specification of the color of the line. (See *ryspects(5)*.)
- `--line-width <num>` Width of the line, as a number of points or pixels.
- `--line-style <style>` The style of the line. (See *ryspects(5)* for valid line style values.)

filled_lines

- line-color** <color> A specification of the color of the line. (See *ryspects(5)*.)
- line-width** <num> Width of the line, as a number of points or pixels.
- line-style** <style> The style of the line. (See *ryspects(5)*.)
- top-field-color** <color> A specification of the color of the filled area below the line. (See *ryspects(5)*.)
- bottom-field-color** <color> A specification of the color of the filled area above the line. (See *ryspects(5)*.)

bars

- bar-width** <proportion> A number between 0 and 1, indicating the width of the bars, as a proportion of the available width. For example, a value of 1 indicates that the bar should occupy all available width (so there is no empty space between bars). A value of .5 will occupy half the available width.
- bar-border-width** <num> Width of the border of the bar, as a number of points or pixels.
- bar-fill-color** <color> A specification of the color of the filled area inside the bar. (See *ryspects(5)*.)
- bar-border-color** <color> A specification of the color of the border of the bar. (See *ryspects(5)*.)

23.9.4 Tick Mark Display Options

These options control how tick marks look, and where `rytimeseries` places them on the value (Y) axis.

- value-ticks** <tickspec> A specification of where to place tickmarks on the value (vertical) axis. (See *ryspects(5)*.)
- value-tick-size** <num> Length of the tick mark, as a number of points or pixels.
- value-tick-label-format** <format> A format string or labeling style specifying how `rytimeseries` should derive the label from its position on the value scale.

The valid label styles are `autofloat`, `binary` or `metric`.

Style	Result
<code>autofloat</code>	Display the value with a number of decimal places chosen to compromise readability and accuracy.
<code>binary</code>	Format the value using SI binary prefix notation. (E.g., 1024 bytes == 1 kibibyte, or 1 KiB.) This style is appropriate for quantities that are meaningful as powers of two.
<code>metric</code>	Format the value using the SI metric prefix notation. (E.g., 1000 bytes == 1 kilobyte, or 1 KB). This style is appropriate for quantities that are meaningful as powers of ten.

(Note that bytes are commonly counted using both the binary and metric styles.)

If the value is not one of these literals, it is presumed to be a format string, following the rules of [Python 2.0 string formatting](#). A dictionary is used as the input to the format string, containing these keys:

```
.. list-table::

* - Key
  - Description

* - ``value``
  - The value the tick mark represents

* - ``autofloat_value``
  - The value the tick mark represents, converted as with ``autofloat``.

* - ``units``
  - The value of the ``value-units`` option.
```

--value-tick-label-size <sizespec> A textual size specification of the size of the tick mark label. (See *ryspecs(5)*.) For example, to specify a tick label size of 12 pixels, use a value of 12px.

--value-tick-label-spacing <num> Space between a tick mark and its label, as a number of points or pixels.

--value-tick-label-halign <halign> One of the values “left”, “center” or “right,” indicating whether the tick label should be aligned to the left, center or right of the text, respectively.

--value-tick-label-valign <valign> One of the values “top”, “center” or “bottom,” indicating whether the tick label should be aligned to the top, center or bottom of the text, respectively.

--value-tick-label-angle <angle> A number from 0 to 360, indicating the angle that the tick label should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.

--time-major-ticks (auto|none) A specification of where to place major tick marks on the time (horizontal) axis. At this time, the only two options are `auto` (place tickmarks automatically) and `none` (don’t place any tick marks).

--time-major-tick-size <num> Length of the major tick mark, as a number of points or pixels.

--time-major-tick-label-size <sizespec> A textual size specification of the size of the major tick mark label. (See *ryspecs(5)*.) For example, to specify a tick label size of 12 pixels, use a time of 12px.

--time-major-tick-label-spacing <num> Space between a major tick mark and its label, as a number of points or pixels.

--time-minor-ticks (auto|none) A specification of where to place minor tick marks on the time (horizontal) axis. At this time, the only two options are `auto` (place tickmarks automatically) and `none` (don’t place any tick marks).

--time-minor-tick-size <num> Length of the minor tick mark, as a number of points or pixels.

--time-minor-tick-label-size <sizespec> A textual size specification of the size of the minor tick mark label. (See *ryspecs(5)*.) For example, to specify a tick label size of 12 pixels, use a time of 12px.

--time-minor-tick-label-spacing <num> Space between a minor tick mark and its label, as a number of points or pixels.

--time-tick-label-halign <halign> One of the values “left”, “center” or “right,” indicating whether the tick label should be aligned to the left, center or right of the text, respectively.

- time-tick-label-valign <valign>** One of the values “top”, “center” or “bottom,” indicating whether the tick label should be aligned to the top, center or bottom of the text, respectively.
- time-tick-label-angle <angle>** A number from 0 to 360, indicating the angle that the tick label should be rotated. A value of 0 (or 360) indicates that the label should be drawn perfectly horizontally, drawn left to right. Increasing values from 0 will rotate the text counterclockwise—a value of 90 results in text drawn vertically, bottom to top, 180 is text drawn horizontally but upside-down, right to left.

23.9.5 Annotation Display Options

- annotate-max** Flag. Place an annotation at the first instance of the maximum value plotted in the visualization. Annotated points are called out with a circular marker and labeled with the value of the annotated observation.
- annotation-marker-color <color>** A specification of the color of the marker calling out the annotated point. (See *ryspects(5)*.)
- annotation-marker-size <num>** Number of points/pixels describing the radius of the annotation callout marker.
- annotation-label-size <sizespec>** A textual size specification of the size of the tick mark label. (See *ryspects(5)*.) For example, to specify an annotation label size of 12 pixels, use a time of 12px.
- annotation-label-color <color>** A specification of the color of the annotation label. (See *ryspects(5)*.)
- annotation-label-background-color <color>** A specification of the color of the background of the annotation label. (See *ryspects(5)*.)
- annotation-label-spacing <num>** A number indicating the space between the annotation marker and its label, as points or pixels.

23.9.6 Border Options

- vertical-border-line-style <stylespec>** The style of line to draw around on the vertical border of the visualization. (See *ryspects(5)*.) For visualizations with a top and bottom component, this option will affect the vertical border for both components.
- horizontal-border-line-style <stylespec>** The style of line to draw around on the horizontal border of the visualization. (See *ryspects(5)*.)

23.9.7 Backgrounds

- chart-background-color <color>** A specification of the color of the background of the entire subchart. (See *ryspects(5)*.)
- plot-background-color <color>** A specification of the color of the background of just the plots. (See *ryspects(5)*.)

23.9.8 Gridlines

- vgrid** Flag. Draw vertical grid lines.

- vgrid-color <color>** A specification of the color of the vertical grid lines. (See *ryspects(5)*.)
- vgrid-style <stylespec>** The style of line to draw for the vertical grid lines.
- vgrid-width <num>** A number indicating the width of the vertical grid lines, in points or pixels.
- vgrid-lines-at <tickspec>** A specification of where to place the vertical grid lines, relative to the horizontal axis. (See *ryspects(5)*.)
- hgrid** Flag. Draw horizontal grid lines.
- hgrid-color <color>** A specification of the color of the horizontal grid lines. (See *ryspects(5)*.)
- hgrid-style <stylespec>** The style of line to draw for the horizontal grid lines.
- hgrid-width <num>** A number indicating the width of the horizontal grid lines, in points or pixels.
- hgrid-lines-at <tickspec>** A specification of where to place the horizontal grid lines, relative to the vertical axis. (See *ryspects(5)*.)

23.9.9 Padding

- padding <num>** A number of pixels/points representing the amount of padding to be applied to the left, right, top and bottom edges of the visualization. (For example, `--padding 4` will add 4 pixels/points of padding each to every edge, so total vertical padding is 8 pixels, and total horizontal padding is also 8 pixels.) This option may not be used with any of the other padding options.
- pad-left <num>** A number of pixels/points representing the amount of padding to be applied to the left edge of the visualization. This option may not be used with `--padding`.
- pad-right <num>** A number of pixels/points representing the amount of padding to be applied to the right edge of the visualization. This option may not be used with `--padding`.
- pad-top <num>** A number of pixels/points representing the amount of padding to be applied to the top edge of the visualization. This option may not be used with `--padding`.
- pad-bottom <num>** A number of pixels/points representing the amount of padding to be applied to the bottom edge of the visualization. This option may not be used with `--padding`.

23.10 EXAMPLES

The following examples all assume that the first column is the time column. The time column may be set using the `--time-column` option.

23.10.1 Default Values

Visualize a single, one-sided series of data in the file `in.txt` to the file `out.png`, using the defaults. (First column is time, second column is data):

```
rytimeseries --input-path=in.txt --output-path=out.png
```

This is equivalent to:

```
rytimeseries --input-path=in.txt --output-path=out.png \  
--time-column=0 --top-column=1
```

23.10.2 Two Columns, Top and Bottom

Consider the following input as `in.txt`:

```
2000-04-01 00:00:00+00:00|982.74|516.37
2000-04-01 01:00:00+00:00|1033.26|541.63
2000-04-01 02:00:00+00:00|1049.99|550.00
2000-04-01 03:00:00+00:00|1031.77|540.88
2000-04-01 04:00:00+00:00|979.87|514.93
2000-04-01 05:00:00+00:00|897.91|473.95
# ...
```

Visualize the input with data from column 1 (the first column is column 0) on the top side, and data from column 2 on the bottom side:

```
rytimeseries --input-path in.txt --output-path out.png \
              --top-column=1 --bottom-column=2
```

23.10.3 One Column, Top and Bottom

Consider the following input as `in.txt`:

```
2000-04-01 00:00:00+00:00|a|982.74
2000-04-01 01:00:00+00:00|b|1033.26
2000-04-01 02:00:00+00:00|a|1049.99
# ...
```

Visualize the input with data on the top if column 1 is a and on the bottom if column 1 is b:

```
rytimeseries --input-path in.txt --output-path out.png \
              --top-column=2 --top-filter="[1]==a" \
              --bottom-column=2 --bottom-filter="[1]==b"
```

Rows where column 1 is neither a nor b will be ignored.

23.10.4 Grouping Into Multiple Subcharts

Consider the following input as `in.txt`:

```
2000-04-01 00:00:00+00:00|moe |a|982.74
2000-04-01 00:00:00+00:00|moe |b|483.79
2000-04-01 01:00:00+00:00|larry|a|1033.26
2000-04-01 01:00:00+00:00|larry|b|243.31
2000-04-01 02:00:00+00:00|curly|a|1049.99
2000-04-01 02:00:00+00:00|curly|b|492.01
# ...
```

Visualize all values from column 3 as a subchart for each of the values of column 1 (moe, larry and curly):

```
rytimeseries --input-path in.txt --output-path out.png \
              --group-by=1 \
              --top-column=3
```

Put rows where column 2 is a on top and rows where column 2 is b on the bottom

```
rytimeseries --input-path in.txt --output-path out.png --group-by=1 --top-column=3 --top-  
filter="[2]==a" --bottom-column=3 --bottom-filter="[2]==b"
```

23.11 SEE ALSO

rytools(5)

RYTOOLS

24.1 DESCRIPTION

The rytools is a set of command-line utilities distributed with [Rayon](#). They provide basic visualization capabilities at the command line, for users who either prefer to spend most of their time in command-line environments or produce (semi-)automated reporting on Unix systems.

The rytools are part of a suite of analysis tools created by the [CERT Coordination Center](#) to enable computer network situational awareness. Other such tools include [SiLK](#) and [YAF](#).

Full documentation for Rayon is available from the *Rayon documentation* website. There are also several manual pages outlining the use of the rytools.

24.2 TOOLS

The following tools comprise the rytools suite:

- ryscatterplot** Scatterplot visualization.
- rycategories** Visualize categorical data (principally with barplots)
- rytimeseries** Visualize time-series data
- ryhilbert** Plot data on a Hilbert curve
- rypiechart** Visualize data in pie charts

24.3 SEE ALSO

ryrc(5), ryspecs(5), rydataformat(5), ryscatterplot(1), rycategories(1), rytimeseries(1), ryhilbert(1), rypiechart(1)

RYDATAFORMAT

25.1 DESCRIPTION

The rytools take the Rayon data format as input. The Rayon data format is primarily delimited text, with extensions to permit the description and proper typing of input data.

25.2 USING SILK OUTPUT WITH RAYON

The text format of most SiLK tools' output is a subset of the Rayon input format, provided the `--no-columns` option is provided to the tool. This data will contain no type hinting or column names. To retain the SiLK column names, it is sufficient to prepend the SiLK output with a single octothorpe (`#`). Rayon will generally infer the correct types from SiLK input data; if SiLK plugins altering display are used, it may rarely be necessary to prepend the SiLK output with a header line describing the data. (See *SPECIAL HEADERS*.)

25.3 BASIC FORMAT

Rayon's default delimiter is the pipe character (`|`). The delimiting character must not appear in the input. An example of valid input is:

```
foo|1|2  
bar|3|4
```

Whitespace around delimiters will be removed, so the following is equivalent to the above:

```
foo| 1| 2  
bar| 3| 4
```

Whitespace lines are also ignored, so this is also equivalent to the above:

```
foo|1|2  
  
bar|3|4
```

The delimiter may be changed using the `Delimiter` header. (See *SPECIAL HEADERS*.)

25.4 COMMENTS

Lines beginning with an octothorpe (#) are comments, and are ignored (with the exceptions outlined in *SPECIAL HEADERS* and *COLUMN NAMES*). Therefore, this is also equivalent to the above:

```
foo| 1| 2
# this line will be ignored
bar| 3| 4
```

The Rayon data format does not support infix or postfix comments. In the following, the text `# this is not` will be interpreted as content:

```
# this is a comment
foo | a | a
bar | b | b # this is not
```

25.5 HEADERS

Headers are special comments at the top of the file (or beginning of the stream) that start with exactly two octothorpes (##). Headers are used to store metadata about a dataset.

A header contains a name, followed by a colon, followed by a value. There may be whitespace between the colon and either the name or value. Here is an example of a header:

```
## Description: This is an example dataset
foo| 1| 2
bar| 3| 4
```

This data set contains a header named `Description`. The header has the value `This is an example dataset`.

Header names may contain the upper- and lower-case letters, numbers, underscore (`_`) and hyphen (`-`). Header values may contain any of the ASCII character set between `0x20` and `0x7e`, inclusive. Certain header names are reserved, specifically those in *SPECIAL HEADERS* and names beginning with `Rayon-`; notwithstanding these restrictions, users may create arbitrary headers as they see fit. Header case will be preserved, but header lookups are case-insensitive.

The first line of data will terminate the processing of headers; any subsequent lines beginning with any number of octothorpes will be treated as a comment.

25.6 SPECIAL HEADERS

Some headers have special meaning when parsed. For instance, the `Delimiter` header may be used to change the delimiting character of the file:

```
## Delimiter: ,
foo,1,2
bar,3,4
```

The following header names have special meanings:

Title The title of the dataset

Delimiter A single character to be used as the delimiting character between items in a row.

Typemap A set of type names used to convert data in the file from text to a native data type.

man-rydata-column-names A list of column names, delimited by the same character as the data. (See *COL-UMN NAMES*)

25.7 COLUMN NAMES

Column names may be specified with the Column-Names header:

```
## Column-Names: label|value1|value2
foo|1|2
bar|3|4
```

This input will generate a dataset with the column names “label”, “value1” and “value2”, respectively.

As a convenience, there is an alternate syntax for specifying column names. The last comment line before the first data row may optionally specify the names of the columns in the dataset. If the last comment line before the first data row is delimited with the delimiting character and contains as many elements as the first data line of the file, its contents will be used as the names of the dataset columns. The following is equivalent to the previous example:

```
# label| value1| value2
foo|1|2
bar|3|4
```

As with headers, whitespace between the comment character and the column name designation will be ignored, but multiple comment characters will probably give unwanted results. Thus, the following is legal:

```
#label| value1| value2
foo|1|2
bar|3|4
```

The following is also legal; the dataset ignores whitespace surrounding column names:

```
#label|value1|value2
foo|1|2
bar|3|4
```


RYRC

26.1 DESCRIPTION

The various command line tools that come with Rayon (called the *rytools*) get their configuration, in order, from the following sources:

1. The system-wide configuration file `/etc/ryrc`.
2. The user configuration file `~/.ryrc`.
3. A per-invocation configuration file, passed in via the `--config-file` option.
4. A file specified by the `--config-file` options.
5. Options passed in on the command line.

If an option is defined twice, the last defined value will be used, unless the source specifies that previous configuration should be ignored. (See *IGNORING CONFIGURATION*.)

26.2 CONFIGURATION FILE FORMAT

The configuration file uses “.ini file”-style syntax documented in Python’s [ConfigParser](#) documentation.

Options to apply to all tools should go in the `[ALL]` section of the configuration file. Options for a specific tool should go in a section named for the tool. (E.g., options for `rscatterplot` go in the section `[rscatterplot]`.) This allows a single file to contain configuration for many *rytools* commands.

Configuration option names and values are identical to the names and values of command-line options or the tool. Flags (options passed on the command line without values) may be enabled in a configuration file by supplying any value except `false` or `0`. (For consistency, users may wish to use the values `true` or `1`.)

26.3 IGNORING CONFIGURATION

If the user passes the `--ignore-config` option on the command-line or includes `ignore_config` in a configuration file, previous configuration information will be ignored.

26.4 EXAMPLES

Invoke `rscatterplot` with some options:

```
ryscatterplot \  
  --title="the title" \  
  --width=800 \  
  --height=200 \  
  --marker-size=5 \  
  --x-scale-dont-zero \  
  --input-path="indata.txt" \  
  --output-path="test.png"
```

The same options may be placed in the configuration file `one-off.cfg`, with the following format:

```
[ryscatterplot]  
title="the title"  
width=800  
height=200  
marker_size=5  
x-scale-dont-zero=true  
input_path=indata.txt  
output_path=test.png
```

`ryscatterplot` may then be invoked with just a reference to the configuration file:

```
ryscatterplot --config-file=one-off.cfg
```

If the file is renamed to `~/ryrc`, the `--config-file` option is unnecessary:

```
ryscatterplot
```

However, options may still be overridden on the command line. This example uses `otherindata.txt` for input:

```
ryscatterplot --input-path=otherindata.txt
```

To use the width and height values in multiple tools, put them in the `[ALL]` section of the configuration file. If this is `~/ryrc`:

```
[ALL]  
width=800  
height=200  
  
[ryscatterplot]  
title="the title"  
marker_size=5  
input_path=indata.txt  
output_path=test.png
```

...other commands such as `rytimeseries` will use the `--width` and `--height` options specified in the configuration file, unless they are overridden on the command-line:

```
# output will be 800 x 200  
rytimeseries --input-path=foo.txt --output=test2.png
```

Among other things, the system-wide value for `--marker-size` is 10 in this `/etc/ryrc` file:

```
[ryscatterplot]  
marker_size=10  
# ...other configuration...
```

This (and all other customization in `/etc/ryrc`) can be changed to the command defaults per-user by putting `--ignore-config` in the `~/ryrc` file:

```
[ryscatterplot]
ignore_config=1
# marker_size set back to default
```

Alternately, `--ignore-config` could be passed in on the command line to ensure that no unexpected configuration values are set:

```
ryscatterplot --ignore-config ...
```

To disable the behavior of `--x-scale-dont-zero`, omit it from the configuration file, or alternately, change the value to `false` or `0`. (This can be used to temporarily disable features, or make it clear that the feature is meant to be disabled.):

```
[ryscatterplot]
title="the title"
width=800
height=200
marker_size=5
x-scale-dont-zero=false
input_path=indata.txt
output_path=test.png
```


RYSPECS

27.1 DESCRIPTION

The set of command-line tools that come with it (called the rytools) use a number of common conventions to specify colors, line styles, font sizes, and other configuration parameters.

The conventions used in the rytools differs slightly from those used when programming with Rayon in Python. For a detailed treatment of the latter, see *Conventions Used in Rayon* in the Rayon Reference Manual.

27.2 COMMAND INVOCATION

For a detailed description of rytool configuration, see *ryrc*.

All rytools support the following options.

- help** Flag. Print a brief tool description and synopsis of available options and exit.
- input-path <filename>** The name of a file to use as input. If *filename* is a hyphen (-), standard input will be used.
- output-path <filename>** Required. The name of a file to use as output. The type of file will be inferred from the file extension. Supported extensions (and file formats) are *.pdf* (Portable Document Format), *.png* (Portable Network Graphics), *.svg* (Scalable Vector Graphics), and *.ps* (PostScript).
- width <num>** The width of the image, as a number of pixels or points. (See *SIZE AND DISTANCE*.)
- height <num>** The height of the image, as a number of pixels or points. (See *SIZE AND DISTANCE*.)

27.3 COLUMN SPECIFICATION

Many command options require the specification of a column of data from the command's input. Columns may be specified as a numeric index, where the leftmost column lexicographically in the input is column 0. If the input data has column names (see *rydataformat*), the names may be used to specify columns as well.

27.4 SIZE AND DISTANCE

Two different kinds of rytools conventions exist to specify size or distance. Some options simply take a number of points or pixels:

```
rytimeseries --annotation-label-spacing=4 ...
```

Other options take a size specification: a number followed by one of the suffixes “px” or “pt”, representing the number of pixels or points to use for the size/distance. The unit suffixes are interchangeable; the unit used will be the device unit of the output (generally pixels for raster output like PNG and points for vector output like PDF or SVG.)

27.4.1 Relative Sizes

Arguments that take size specifications for font sizes may also accept relative sizes. The exact sizes will be computed with the intent of maintaining the size of the element relative to the dimensions of the image.

Acceptable size specifications are:

- `small`
- `normal`
- `large`
- `x-large`
- `xx-large`.

27.5 SCALES

Many rytools can plot data apply different scales to data before visualization. In general, the following scales are available.

linear Plot the data on a linear scale.

log Plot the data on a log scale.

clog Add the constant value 1 to the data, then plot on a log scale. This introduces some error to the scale (particularly for small non-integer values), but, unlike `log`, permits the input to contain zeroes.

27.6 TICK MARKS

Several rytools use *tick mark specifications* to express where tick marks should be placed. A tick mark specification is a comma-separated list of *tick specifiers*, which consist of a specifier type and (optionally) a set of arguments to customize the time, again separated by commas, for example the following:

```
ryscatterplot ... --x-ticks at(1,2),min,gt(1000)
```

Would place ticks on the X axis at the values 1 and 2, the minimum value on the scale, and all points on the scale whose value is greater than 1000.

The possible position specifiers are:

at (pos, . . .) Place a tick mark at each of the specified positions on the scale.

every (mod) Place a tick mark at every position on the scale evenly divisible by *mod*. For example, on a scale running from 0-10, `every (3)` will place tick marks at 3, 6 and 9.

If *mod* is a time delta specification (see [DATES AND TIMES](#)) and the column contains datetime data, tick marks will be placed at time points divisible by that interval. For instance, on a scale running from 2001-01-01T01:23:45 to 2001-01-01T01:25:03, `every (30s)` will place tick marks at 01:24:00 and 01:24:30.

ge (num) Place a tick mark for each data point whose value is greater than or equal to *num*.

gt (num) Place a tick mark for each data point whose value is greater than *num*.

le (num) Place a tick mark for each data point whose value is less than or equal to *num*.

lt (num) Place a tick mark for each data point whose value is less than *num*.

max Place a tick mark at the largest point in the data.

med Place a tick mark at the median of the data.

min Place a tick mark at the smallest point in the data.

n (num) Place *n* automatically-selected “nice” tick marks on the scale. The tick selection algorithm will try to select exactly *n* tick marks, but may choose more or less if it cannot find *n* suitable tick marks.

none Place no tick marks on the scale. (Use to override defaults.)

p (num) Place a tick mark at the position marking the *n* th percentile of the data.

smax Place a tick mark at the largest point on the scale.

smin Place a tick mark at the smallest point on the scale.

27.7 MARKER SHAPE

The following marker shape styles are available:

arrow Draw an v-shaped arrow head at the point

dot Draw a filled dot at the point

circle Draw an unfilled circle at the point

vline Draw a vertical line at the point

hline Draw a horizontal line at the point

cross Draw a cross at the point

none Draw nothing at the point

27.8 LINE STYLES

A number of line-drawing styles are available in the rytools. For options specifying line styles, the following values are valid:

- `solid`
- `dotted`
- `dashed`
- `dotdash`

- none

27.9 COLORS

For specifying colors, Rayon follows the format described in <http://www.w3.org/TR/CSS2/syndata.html#color-units> and <http://www.w3.org/TR/css3-color>, with three two exceptions:

- When specifying colors in hexadecimal notation, the leading octothorpe (#) character may be omitted.
- Alpha channels are supported in hexadecimal notation as well as in functional notation.
- The `hsl(...)` and `hsl(...)` specification forms are not supported.

For example, here are all the ways to specify a particular shade of blue to the rytools (text following a # character is a comment):

Method	Value
Name	blue
RGB	00f
RGBA	00ff
RRGGBB	0000ff
RRGGBBAA	0000ffff
RGB (functional)	rgb(0, 0, 255)
RGB (functional, percentage)	rgb(0%, 0%, 100%)
RGBA (functional)	rgb(0, 0, 255, 1.0)
RGBA (functional, percentage)	rgb(0%, 0%, 100%, 1.0)

27.9.1 Specifying Colors By Name

The color names Rayon understands are identical to those in the CSS2 and CSS3 standards, reproduced here for convenience.

Name	rgba(...) value
aliceblue	rgba(240,248,255,255)
antiquewhite	rgba(250,235,215,255)
aqua	rgba(0,255,255,255)
aquamarine	rgba(127,255,212,255)
azure	rgba(240,255,255,255)
beige	rgba(245,245,220,255)
bisque	rgba(255,228,196,255)
black	rgba(0,0,0,255)
blanchedalmond	rgba(255,235,205,255)
blue	rgba(0,0,255,255)
blueviolet	rgba(138,43,226,255)
brown	rgba(165,42,42,255)
burlywood	rgba(222,184,135,255)
cadetblue	rgba(95,158,160,255)
chartreuse	rgba(127,255,0,255)
chocolate	rgba(210,105,30,255)
coral	rgba(255,127,80,255)
cornflowerblue	rgba(100,149,237,255)
cornsilk	rgba(255,248,220,255)
crimson	rgba(220,20,60,255)
Continued on next page	

Table 27.1 – continued from previous page

cyan	rgba(0,255,255,255)
darkblue	rgba(0,0,139,255)
darkcyan	rgba(0,139,139,255)
darkgoldenrod	rgba(184,134,11,255)
darkgray	rgba(169,169,169,255)
darkgreen	rgba(0,100,0,255)
darkgrey	rgba(169,169,169,255)
darkkhaki	rgba(189,183,107,255)
darkmagenta	rgba(139,0,139,255)
darkolivegreen	rgba(85,107,47,255)
darkorange	rgba(255,140,0,255)
darkorchid	rgba(153,50,204,255)
darkred	rgba(139,0,0,255)
darksalmon	rgba(233,150,122,255)
darkseagreen	rgba(143,188,143,255)
darkslateblue	rgba(72,61,139,255)
darkslategray	rgba(47,79,79,255)
darkslategrey	rgba(47,79,79,255)
darkturquoise	rgba(0,206,209,255)
darkviolet	rgba(148,0,211,255)
deeppink	rgba(255,20,147,255)
deepskyblue	rgba(0,191,255,255)
dimgray	rgba(105,105,105,255)
dimgrey	rgba(105,105,105,255)
dodgerblue	rgba(30,144,255,255)
firebrick	rgba(178,34,34,255)
floralwhite	rgba(255,250,240,255)
forestgreen	rgba(34,139,34,255)
fuchsia	rgba(255,0,255,255)
gainsboro	rgba(220,220,220,255)
ghostwhite	rgba(248,248,255,255)
gold	rgba(255,215,0,255)
goldenrod	rgba(218,165,32,255)
gray	rgba(128,128,128,255)
green	rgba(0,128,0,255)
greenyellow	rgba(173,255,47,255)
grey	rgba(128,128,128,255)
honeydew	rgba(240,255,240,255)
hotpink	rgba(255,105,180,255)
indianred	rgba(205,92,92,255)
indigo	rgba(75,0,130,255)
ivory	rgba(255,255,240,255)
khaki	rgba(240,230,140,255)
lavender	rgba(230,230,250,255)
lavenderblush	rgba(255,240,245,255)
lawngreen	rgba(124,252,0,255)
lemonchiffon	rgba(255,250,205,255)
lightblue	rgba(173,216,230,255)
lightcoral	rgba(240,128,128,255)
lightcyan	rgba(224,255,255,255)
lightgoldenrodyellow	rgba(250,250,210,255)
lightgray	rgba(211,211,211,255)
Continued on next page	

Table 27.1 – continued from previous page

lightgreen	rgba(144,238,144,255)
lightgrey	rgba(211,211,211,255)
lightpink	rgba(255,182,193,255)
lightsalmon	rgba(255,160,122,255)
lightseagreen	rgba(32,178,170,255)
lightskyblue	rgba(135,206,250,255)
lightslategray	rgba(119,136,153,255)
lightslategrey	rgba(119,136,153,255)
lightsteelblue	rgba(176,196,222,255)
lightyellow	rgba(255,255,224,255)
lime	rgba(0,255,0,255)
limegreen	rgba(50,205,50,255)
linen	rgba(250,240,230,255)
magenta	rgba(255,0,255,255)
maroon	rgba(128,0,0,255)
mediumaquamarine	rgba(102,205,170,255)
mediumblue	rgba(0,0,205,255)
mediumorchid	rgba(186,85,211,255)
mediumpurple	rgba(147,112,219,255)
mediumseagreen	rgba(60,179,113,255)
mediumslateblue	rgba(123,104,238,255)
mediumspringgreen	rgba(0,250,154,255)
mediumturquoise	rgba(72,209,204,255)
mediumvioletred	rgba(199,21,133,255)
midnightblue	rgba(25,25,112,255)
mintcream	rgba(245,255,250,255)
mistyrose	rgba(255,228,225,255)
moccasin	rgba(255,228,181,255)
navajowhite	rgba(255,222,173,255)
navy	rgba(0,0,128,255)
oldlace	rgba(253,245,230,255)
olive	rgba(128,128,0,255)
olivedrab	rgba(107,142,35,255)
orange	rgba(255,165,0,255)
orangered	rgba(255,69,0,255)
orchid	rgba(218,112,214,255)
palegoldenrod	rgba(238,232,170,255)
palegreen	rgba(152,251,152,255)
paleturquoise	rgba(175,238,238,255)
palevioletred	rgba(219,112,147,255)
papayawhip	rgba(255,239,213,255)
peachpuff	rgba(255,218,185,255)
peru	rgba(205,133,63,255)
pink	rgba(255,192,203,255)
plum	rgba(221,160,221,255)
powderblue	rgba(176,224,230,255)
purple	rgba(128,0,128,255)
red	rgba(255,0,0,255)
rosybrown	rgba(188,143,143,255)
royalblue	rgba(65,105,225,255)
saddlebrown	rgba(139,69,19,255)
salmon	rgba(250,128,114,255)
Continued on next page	

Table 27.1 – continued from previous page

sandybrown	rgba(244,164,96,255)
seagreen	rgba(46,139,87,255)
seashell	rgba(255,245,238,255)
sienna	rgba(160,82,45,255)
silver	rgba(192,192,192,255)
skyblue	rgba(135,206,235,255)
slateblue	rgba(106,90,205,255)
slategray	rgba(112,128,144,255)
slategrey	rgba(112,128,144,255)
snow	rgba(255,250,250,255)
springgreen	rgba(0,255,127,255)
steelblue	rgba(70,130,180,255)
tan	rgba(210,180,140,255)
teal	rgba(0,128,128,255)
thistle	rgba(216,191,216,255)
tomato	rgba(255,99,71,255)
turquoise	rgba(64,224,208,255)
violet	rgba(238,130,238,255)
wheat	rgba(245,222,179,255)
white	rgba(255,255,255,255)
whitesmoke	rgba(245,245,245,255)
yellow	rgba(255,255,0,255)
yellowgreen	rgba(154,205,50,255)

27.10 DATES AND TIMES

To specify dates, times and relative time periods (“time deltas”), use the ISO-8601 standard representation. <http://www.w3.org/TR/NOTE-datetime> describes this standard.

27.11 SEE ALSO

rytools(5)

PYTHON MODULE INDEX

r

`rayon.toolbox`, [11](#)

INDEX

A

`add_bottom_border()` (rayon.charts.SimpleTiledChart method), 45
`add_bottom_border()` (rayon.charts.SquareChart method), 52
`add_bottom_border()` (rayon.charts.TiledChart method), 49
`add_bottom_title()` (rayon.charts.SimpleTiledChart method), 45
`add_bottom_title()` (rayon.charts.SquareChart method), 52
`add_bottom_title()` (rayon.charts.TiledChart method), 49
`add_chart()` (rayon.charts.SimpleTiledChart method), 45
`add_chart()` (rayon.charts.SquareChart method), 52
`add_chart()` (rayon.charts.TiledChart method), 49
`add_front_gridlines()` (rayon.charts.SquareChart method), 52
`add_left_border()` (rayon.charts.SimpleTiledChart method), 45
`add_left_border()` (rayon.charts.SquareChart method), 53
`add_left_border()` (rayon.charts.TiledChart method), 49
`add_left_title()` (rayon.charts.SimpleTiledChart method), 45
`add_left_title()` (rayon.charts.SquareChart method), 53
`add_left_title()` (rayon.charts.TiledChart method), 49
`add_ne_corner()` (rayon.charts.SimpleTiledChart method), 46
`add_ne_corner()` (rayon.charts.SquareChart method), 53
`add_ne_corner()` (rayon.charts.TiledChart method), 49
`add_nw_corner()` (rayon.charts.SimpleTiledChart method), 46
`add_nw_corner()` (rayon.charts.SquareChart method), 53
`add_nw_corner()` (rayon.charts.TiledChart method), 49
`add_plot()` (rayon.charts.SimpleTiledChart method), 46
`add_plot()` (rayon.charts.SquareChart method), 53
`add_plot()` (rayon.charts.TiledChart method), 49
`add_rear_gridlines()` (rayon.charts.SquareChart method), 53
`add_right_border()` (rayon.charts.SimpleTiledChart method), 46
`add_right_border()` (rayon.charts.SquareChart method), 53
`add_right_title()` (rayon.charts.SimpleTiledChart method), 46
`add_right_title()` (rayon.charts.SquareChart method), 53
`add_right_title()` (rayon.charts.TiledChart method), 50
`add_se_corner()` (rayon.charts.SimpleTiledChart method), 46
`add_se_corner()` (rayon.charts.SquareChart method), 54
`add_se_corner()` (rayon.charts.TiledChart method), 50
`add_sw_corner()` (rayon.charts.SimpleTiledChart method), 46
`add_sw_corner()` (rayon.charts.SquareChart method), 54
`add_sw_corner()` (rayon.charts.TiledChart method), 50
`add_tickset()` (rayon.borders.HorizontalBorder method), 40
`add_tickset()` (rayon.borders.HorizontalLineBorder method), 39
`add_tickset()` (rayon.borders.VerticalBorder method), 40
`add_tickset()` (rayon.borders.VerticalLineBorder method), 40
`add_top_border()` (rayon.charts.SimpleTiledChart method), 46
`add_top_border()` (rayon.charts.SquareChart method), 54
`add_top_border()` (rayon.charts.TiledChart method), 50
`add_top_title()` (rayon.charts.SimpleTiledChart method), 46
`add_top_title()` (rayon.charts.SquareChart method), 54
`add_top_title()` (rayon.charts.TiledChart method), 50
`append_column()` (rayon.data.Dataset method), 57

C

`clear_padding()` (rayon.charts.SimpleTiledChart method), 47
`clear_padding()` (rayon.charts.SquareChart method), 54
`clear_padding()` (rayon.charts.TiledChart method), 50

D

`del_header()` (rayon.data.Dataset method), 58

F

`filter_both()` (rayon.data.Dataset method), 58

`filter_fail()` (rayon.data.Dataset method), [59](#)
`filter_pass()` (rayon.data.Dataset method), [59](#)
`flatten()` (rayon.data.Dataset method), [60](#)
`for_file()` (rayon.toolbox.Toolbox class method), [110](#)
`for_gui()` (rayon.toolbox.Toolbox class method), [110](#)

G

`get_chart()` (rayon.charts.SimpleTiledChart method), [47](#)
`get_chart()` (rayon.charts.SquareChart method), [54](#)
`get_chart()` (rayon.charts.TiledChart method), [50](#)
`get_charts()` (rayon.charts.SimpleTiledChart method), [47](#)
`get_charts()` (rayon.charts.SquareChart method), [54](#)
`get_charts()` (rayon.charts.TiledChart method), [51](#)
`get_column()` (rayon.data.Dataset method), [60](#)
`get_column_index_from_name()` (rayon.data.Dataset method), [60](#)
`get_column_name_from_index()` (rayon.data.Dataset method), [60](#)
`get_column_names()` (rayon.data.Dataset method), [60](#)
`get_event_source()` (rayon.charts.SimpleTiledChart method), [47](#)
`get_event_source()` (rayon.charts.SquareChart method), [54](#)
`get_event_source()` (rayon.charts.TiledChart method), [51](#)
`get_header()` (rayon.data.Dataset method), [60](#)
`get_headers()` (rayon.data.Dataset method), [60](#)
`get_input_max()` (rayon.scales.CategoricalRangeScale method), [96](#)
`get_input_max()` (rayon.scales.CategoricalScale method), [95](#)
`get_input_max()` (rayon.scales.CountingLogGradientScale method), [98](#)
`get_input_max()` (rayon.scales.CountingLogScale method), [101](#)
`get_input_max()` (rayon.scales.LinearBinnedScale method), [99](#)
`get_input_max()` (rayon.scales.LinearGradientScale method), [97](#)
`get_input_max()` (rayon.scales.LinearScale method), [99](#)
`get_input_max()` (rayon.scales.LogGradientScale method), [98](#)
`get_input_max()` (rayon.scales.LogScale method), [100](#)
`get_input_max()` (rayon.scales.TimeBinnedScale method), [102](#)
`get_input_max()` (rayon.scales.TimeScale method), [101](#)
`get_input_min()` (rayon.scales.CategoricalRangeScale method), [96](#)
`get_input_min()` (rayon.scales.CategoricalScale method), [95](#)
`get_input_min()` (rayon.scales.CountingLogGradientScale method), [98](#)
`get_input_min()` (rayon.scales.CountingLogScale method), [101](#)
`get_input_min()` (rayon.scales.LinearBinnedScale method), [100](#)
`get_input_min()` (rayon.scales.LinearGradientScale method), [97](#)
`get_input_min()` (rayon.scales.LinearScale method), [99](#)
`get_input_min()` (rayon.scales.LogGradientScale method), [98](#)
`get_input_min()` (rayon.scales.LogScale method), [100](#)
`get_input_min()` (rayon.scales.TimeBinnedScale method), [102](#)
`get_input_min()` (rayon.scales.TimeScale method), [101](#)
`get_nice_tick_positions()` (rayon.scales.CategoricalRangeScale method), [96](#)
`get_nice_tick_positions()` (rayon.scales.CategoricalScale method), [95](#)
`get_nice_tick_positions()` (rayon.scales.CountingLogGradientScale method), [98](#)
`get_nice_tick_positions()` (rayon.scales.CountingLogScale method), [101](#)
`get_nice_tick_positions()` (rayon.scales.LinearBinnedScale method), [100](#)
`get_nice_tick_positions()` (rayon.scales.LinearGradientScale method), [97](#)
`get_nice_tick_positions()` (rayon.scales.LinearScale method), [99](#)
`get_nice_tick_positions()` (rayon.scales.LogGradientScale method), [98](#)
`get_nice_tick_positions()` (rayon.scales.LogScale method), [100](#)
`get_nice_tick_positions()` (rayon.scales.TimeBinnedScale method), [102](#)
`get_nice_tick_positions()` (rayon.scales.TimeScale method), [101](#)
`get_num_columns()` (rayon.data.Dataset method), [60](#)
`get_num_rows()` (rayon.data.Dataset method), [60](#)
`get_output_max()` (rayon.scales.CategoricalRangeScale method), [96](#)
`get_output_max()` (rayon.scales.CategoricalScale method), [95](#)
`get_output_max()` (rayon.scales.CountingLogGradientScale method), [99](#)
`get_output_max()` (rayon.scales.CountingLogScale method), [101](#)
`get_output_max()` (rayon.scales.LinearBinnedScale method), [100](#)
`get_output_max()` (rayon.scales.LinearGradientScale method), [97](#)
`get_output_max()` (rayon.scales.LinearScale method), [99](#)
`get_output_max()` (rayon.scales.LogGradientScale method), [98](#)

method), 98
 get_output_max() (rayon.scales.LogScale method), 100
 get_output_max() (rayon.scales.TimeBinnedScale method), 102
 get_output_max() (rayon.scales.TimeScale method), 102
 get_output_min() (rayon.scales.CategoricalRangeScale method), 96
 get_output_min() (rayon.scales.CategoricalScale method), 95
 get_output_min() (rayon.scales.CountingLogGradientScale method), 99
 get_output_min() (rayon.scales.CountingLogScale method), 101
 get_output_min() (rayon.scales.LinearBinnedScale method), 100
 get_output_min() (rayon.scales.LinearGradientScale method), 97
 get_output_min() (rayon.scales.LinearScale method), 99
 get_output_min() (rayon.scales.LogGradientScale method), 98
 get_output_min() (rayon.scales.LogScale method), 100
 get_output_min() (rayon.scales.TimeBinnedScale method), 102
 get_output_min() (rayon.scales.TimeScale method), 102
 get_plot() (rayon.charts.SimpleTiledChart method), 47
 get_plot() (rayon.charts.SquareChart method), 55
 get_plot() (rayon.charts.TiledChart method), 51
 get_plot_element() (rayon.charts.SimpleTiledChart method), 47
 get_plot_element() (rayon.charts.SquareChart method), 55
 get_plot_element() (rayon.charts.TiledChart method), 51
 get_plot_elements() (rayon.charts.SimpleTiledChart method), 47
 get_plot_elements() (rayon.charts.SquareChart method), 55
 get_plot_elements() (rayon.charts.TiledChart method), 51
 get_plots() (rayon.charts.SimpleTiledChart method), 47
 get_plots() (rayon.charts.SquareChart method), 55
 get_plots() (rayon.charts.TiledChart method), 51
 get_row() (rayon.data.Dataset method), 61
 get_scale() (rayon.ticks.RangeTickSet method), 108
 get_scale() (rayon.ticks.TickSet method), 107
 get_tickset() (rayon.borders.HorizontalBorder method), 40
 get_tickset() (rayon.borders.HorizontalLineBorder method), 39
 get_tickset() (rayon.borders.VerticalBorder method), 41
 get_tickset() (rayon.borders.VerticalLineBorder method), 40

I

iter_columns() (rayon.data.Dataset method), 61
 iter_ignore_sorted() (rayon.data.Dataset method), 61

iter_tick_labels() (rayon.ticks.RangeTickSet method), 108
 iter_tick_labels() (rayon.ticks.TickSet method), 107
 iter_tick_positions() (rayon.ticks.RangeTickSet method), 108
 iter_tick_positions() (rayon.ticks.TickSet method), 107
 iter_ticksets() (rayon.borders.HorizontalBorder method), 40
 iter_ticksets() (rayon.borders.HorizontalLineBorder method), 39
 iter_ticksets() (rayon.borders.VerticalBorder method), 41
 iter_ticksets() (rayon.borders.VerticalLineBorder method), 40

M

max() (rayon.data.Column method), 65
 mean() (rayon.data.Column method), 65
 meld() (rayon.data.Dataset method), 61
 min() (rayon.data.Column method), 65

N

new_border('hlabel'), 41
 new_border('hline'), 39
 new_border('horizontal'), 40
 new_border('hsplit'), 42
 new_border('none'), 39
 new_border('vertical'), 40
 new_border('vlabel'), 41
 new_border('vline'), 40
 new_border('vsplit'), 42
 new_border() (rayon.toolbox.Toolbox method), 110
 new_chart('square'), 52
 new_chart('tiled'), 45
 new_chart('tiled_adv'), 48
 new_chart() (rayon.toolbox.Toolbox method), 110
 new_column_from_constant() (rayon.toolbox.Toolbox method), 110
 new_column_from_data('aw_data : iterable'), 65
 new_column_from_data() (rayon.toolbox.Toolbox method), 110
 new_dataset_from_columns() (rayon.toolbox.Toolbox method), 111
 new_dataset_from_filename() (rayon.toolbox.Toolbox method), 111
 new_dataset_from_rows() (rayon.toolbox.Toolbox method), 112
 new_dataset_from_stream('tree'), 57
 new_dataset_from_stream() (rayon.toolbox.Toolbox method), 111
 new_gridlines('horizontal'), 69
 new_gridlines('vertical'), 70
 new_gridlines() (rayon.toolbox.Toolbox method), 112
 new_gridlines_from_border() (rayon.toolbox.Toolbox method), 112

`new_gridlines_from_tick_tuples()`
(`rayon.toolbox.Toolbox` method), 112

`new_labeled_marker()` (`rayon.toolbox.Toolbox` method), 113

`new_labeler()` (`rayon.toolbox.Toolbox` method), 113

`new_line('dashed')`, 71

`new_line('dotted')`, 72

`new_line('dotted')`, 72

`new_line('null')`, 72

`new_line('solid')`, 72

`new_line()` (`rayon.toolbox.Toolbox` method), 113

`new_line_from_spec()` (`rayon.toolbox.Toolbox` method), 113

`new_marker('circle')`, 74

`new_marker('cross')`, 74

`new_marker('darrow')`, 73

`new_marker('dot')`, 74

`new_marker('hline')`, 75

`new_marker('larrow')`, 74

`new_marker('none')`, 75

`new_marker('rarrow')`, 74

`new_marker('uarrow')`, 73

`new_marker('vline')`, 75

`new_marker()` (`rayon.toolbox.Toolbox` method), 113

`new_page_from_buffer('uf')`, 77

`new_page_from_buffer()` (`rayon.toolbox.Toolbox` method), 114

`new_page_from_filename()` (`rayon.toolbox.Toolbox` method), 114

`new_plot('bar')`, 80

`new_plot('field')`, 82

`new_plot('filledline')`, 84

`new_plot('hbar')`, 86

`new_plot('labeledscatter')`, 87

`new_plot('line')`, 89

`new_plot('scatter')`, 91

`new_plot()` (`rayon.toolbox.Toolbox` method), 114

`new_scale('categorical')`, 94

`new_scale('catrange')`, 95

`new_scale('clog')`, 101

`new_scale('cloggrad')`, 98

`new_scale('gradient')`, 97

`new_scale('linbins')`, 99

`new_scale('linear')`, 99

`new_scale('log')`, 100

`new_scale('loggrad')`, 97

`new_scale('time')`, 101

`new_scale('timebins')`, 102

`new_scale()` (`rayon.toolbox.Toolbox` method), 114

`new_tickset_from_spec()` (`rayon.toolbox.Toolbox` method), 115

`new_tickset_from_tuples('default')`, 107

`new_tickset_from_tuples('ranged')`, 108

`new_tickset_from_tuples()` (`rayon.toolbox.Toolbox` method), 114

P

`partition()` (`rayon.data.Dataset` method), 61

`percentile()` (`rayon.data.Column` method), 65

R

`range_size()` (`rayon.scales.CategoricalRangeScale` method), 96

`rayon.toolbox` (module), 11

`RayonLimitException` (class in `rayon.scales`), 93

`RayonRangeException` (class in `rayon.scales`), 93

`RayonScaleException` (class in `rayon.scales`), 93

S

`sample_stdev()` (`rayon.data.Column` method), 65

`sample_variance()` (`rayon.data.Column` method), 65

`set_chart_background()` (`rayon.charts.SimpleTiledChart` method), 47

`set_chart_background()` (`rayon.charts.SquareChart` method), 55

`set_chart_background()` (`rayon.charts.TiledChart` method), 51

`set_chart_foreground()` (`rayon.charts.SimpleTiledChart` method), 47

`set_chart_foreground()` (`rayon.charts.SquareChart` method), 55

`set_chart_foreground()` (`rayon.charts.TiledChart` method), 51

`set_column_names()` (`rayon.data.Dataset` method), 63

`set_header()` (`rayon.data.Dataset` method), 63

`set_padding()` (`rayon.charts.SimpleTiledChart` method), 48

`set_padding()` (`rayon.charts.SquareChart` method), 55

`set_padding()` (`rayon.charts.TiledChart` method), 51

`set_parent()` (`rayon.charts.SimpleTiledChart` method), 48

`set_parent()` (`rayon.charts.SquareChart` method), 55

`set_parent()` (`rayon.charts.TiledChart` method), 52

`set_plot_background()` (`rayon.charts.SimpleTiledChart` method), 48

`set_plot_background()` (`rayon.charts.SquareChart` method), 55

`set_plot_background()` (`rayon.charts.TiledChart` method), 52

`set_plot_foreground()` (`rayon.charts.SimpleTiledChart` method), 48

`set_plot_foreground()` (`rayon.charts.SquareChart` method), 55

`set_plot_foreground()` (`rayon.charts.TiledChart` method), 52

`sort()` (`rayon.data.Dataset` method), 63

`stdev()` (`rayon.data.Column` method), 65

T

`to_categorical_range_scale()`
(`rayon.scales.CategoricalScale` method),
[95](#)
`to_categorical_scale()` (`rayon.scales.CategoricalRangeScale`
method), [96](#)
`to_file()` (`rayon.data.Dataset` method), [64](#)
`to_stream()` (`rayon.data.Dataset` method), [64](#)
`to_string()` (`rayon.data.Dataset` method), [64](#)
`Toolbox` (class in `rayon.toolbox`), [110](#)

U

`uniq()` (`rayon.data.Column` method), [65](#)

V

`variance()` (`rayon.data.Column` method), [65](#)

W

`write()` (`rayon.backend.Page` method), [77](#)