# leather

***Release 0.3.3***

December 01, 2016

Contents

Leather is the Python charting library for those who need charts *now* and don't care if they're perfect.

Leather isn't picky. It's rough. It gets dirty. It looks sexy just hanging on the back of a chair. Leather doesn't need your accessories. Leather is how Snake Plissken would make charts.

Get it?

Important links:

- Documentation: http://leather.rtfd.io
- Repository: https://github.com/wireservice/leather
- Issues: https://github.com/wireservice/leather/issues

CHAPTER 1

# About leather

## 1.1 Why leather?

- A readable and user-friendly API.
- Optimized for exploratory charting.
- Produces scale-independent SVG charts.
- Completely type-agnostic. Chart your data, whatever it is.
- Designed with iPython, Jupyter and atom/hydrogen in mind.
- Pure Python. No C dependencies to compile.
- MIT licensed and free for all purposes.
- Zealously zen.
- Made with love.

CHAPTER 2

# Installation

## 2.1 Users

To use leather install it with pip:

```
pip install leather
```

## 2.2 Developers

If you are a developer that also wants to hack on leather, install it from git:

```
git clone git://github.com/wireservice/leather.git
cd leather
mkvirtualenv leather

# If running Python 3 (strongly recommended for development)
pip install -r requirements-py3.txt

# If running Python 2
pip install -r requirements-py2.txt

python setup.py develop
tox
```

**Note:** To run the leather tests with coverage:

```
nosetests --with-coverage tests
```

## 2.3 Supported platforms

leather supports the following versions of Python:

- Python 2.7
- Python 3.3+
- PyPy

It is tested primarily on OSX, but due to its minimal dependencies it should work on both Linux and Windows.

**Note:** iPython or Jupyter user? Leather works great there too.

CHAPTER 3

# Examples

## 3.1 Data series

### 3.1.1 Simple pairs

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

chart = leather.Chart('Simple pairs')
chart.add_dots(data)
chart.to_svg('examples/charts/simple_pairs.svg')
```

### 3.1.2 Table from csv.reader

Sequence row data, such as is returned by `csv.reader()` can be accessed by specifying the indices of the columns containing the `x` and `y` values.

Note that leather does not automatically convert numerical strings, such as those stored in a CSV. If you want that you'll need to use a smarter table reader, such as agate

```
import csv

import leather

with open('examples/realdata/gii.csv') as f:
    reader = csv.reader(f)
    next(reader)
    data = list(reader)[:10]

    for row in data:
        row[1] = float(row[1]) if row[1] is not None else None

chart = leather.Chart('Data from CSV reader')
```

```
chart.add_bars(data, x=1, y=0)
chart.to_svg('examples/charts/csv_reader.svg')
```

### 3.1.3 Table from csv.DictReader

Dict row data, such as is returned by `csv.DictReader` can be accessed by specifying the indices of the columns containing the `x` and `y` values.

See previous example for note on strings from CSVs.

```
import csv

import leather

with open('examples/realdata/gii.csv') as f:
    reader = csv.DictReader(f)
    data = list(reader)[:10]

    for row in data:
        mmr = row['Maternal mortality ratio']
        row['Maternal mortality ratio'] = float(mmr) if mmr is not None else None

chart = leather.Chart('Data from CSV reader')
chart.add_bars(data, x='Maternal mortality ratio', y='Country')
chart.to_svg('examples/charts/csv_dict_reader.svg')
```

### 3.1.4 Custom data

Completely custom data formats are also supported via accessor functions.

```
import leather

data = [
    { 'x': 0, 'q': { 'y': [3] } },
    { 'x': 4, 'q': { 'y': [5] } },
    { 'x': 7, 'q': { 'y': [9] } },
    { 'x': 8, 'q': { 'y': [4] } }
]

def x(row, index):
    return row['x']

def y(row, index):
    return row['q']['y'][0]

chart = leather.Chart('Line')
chart.add_line(data, x=x, y=y)
chart.to_svg('examples/charts/custom_data.svg')
```

### 3.1.5 Multiple series

Multiple data series can be displayed on a single chart so long as they all use the same type of *Scale*.

```
import leather

data1    = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

data2 = [
    (2, 4),
    (7, 3),
    (6, 2),
    (5, 9)
]

chart = leather.Chart('Multiple series')
chart.add_dots(data1)
chart.add_dots(data2)
chart.to_svg('examples/charts/multiple_series.svg')
```

## 3.2 Shapes

### 3.2.1 Bars

```
import leather

data = [
    (3, 'Hello'),
    (5, 'How'),
    (9, 'Are'),
    (4, 'You')
]

chart = leather.Chart('Bars')
chart.add_bars(data)
chart.to_svg('examples/charts/bars.svg')
```

### 3.2.2 Columns

```
import leather

data = [
    ('Hello', 3),
    ('How', 5),
    ('Are', 9),
    ('You', 4)
```

```
]

chart = leather.Chart('Columns')
chart.add_columns(data)
chart.to_svg('examples/charts/columns.svg')
```

### 3.2.3 Dots

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

chart = leather.Chart('Dots')
chart.add_dots(data)
chart.to_svg('examples/charts/dots.svg')
```

### 3.2.4 Lines

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

chart = leather.Chart('Line')
chart.add_line(data)
chart.to_svg('examples/charts/lines.svg')
```

### 3.2.5 Mixing shapes

You can mix different shapes for different series on the same chart.

```
import leather

line_data = [
    (0, 1),
    (2, 5),
    (4, 4),
    (8, 3)
]
```

```
dot_data = [
    (1, 3),
    (2, 5),
    (6, 9),
    (10, 4)
]

chart = leather.Chart('Mixed shapes')
chart.add_line(line_data)
chart.add_dots(dot_data)
chart.to_svg('examples/charts/mixed_shapes.svg')
```

## 3.3 Scales

### 3.3.1 Linear

When using numerical data *Linear* scales are created automatically and by default. You may override the domain by adding a scale manually.

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

chart = leather.Chart('Linear')
chart.add_x_scale(0, 20)
chart.add_y_scale(-10, 10)
chart.add_line(data)
chart.to_svg('examples/charts/linear.svg')
```

### 3.3.2 Ordinal

When using text data *Ordinal* scales are created automatically and by default. It is generally not useful to override these defaults.

```
import leather

data = [
    ('Hello', 3),
    ('How', 5,),
    ('Are', 9),
    ('You', 4)
]

chart = leather.Chart('Ordinal')
chart.add_columns(data)
chart.to_svg('examples/charts/ordinal.svg')
```

### 3.3.3 Temporal

When using date/time data *Temporal* scales are created automatically and by default. You may override the domain by adding a scale manually.

```
from datetime import date

import leather

data = [
    (date(2015, 1, 1), 3),
    (date(2015, 3, 1), 5),
    (date(2015, 6, 1), 9),
    (date(2015, 9, 1), 4)
]

chart = leather.Chart('Temporal')
chart.add_x_scale(date(2014, 1, 1), date(2016, 1, 1))
chart.add_line(data)
chart.to_svg('examples/charts/temporal.svg')
```

## 3.4 Axes

### 3.4.1 Changing tick values

You can change the list of ticks that are displayed using *Chart.add_x_axis()* and *Chart.add_y_axis()* methods. This will not adjust automatically adjust the scale, so it is possible to pick tick values that are not displayed.

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

chart = leather.Chart('Line')
chart.add_x_scale(0, 10)
chart.add_x_axis(ticks=[0, 10])
chart.add_line(data)
chart.to_svg('examples/charts/ticks.svg')
```

### 3.4.2 Customizing tick format

You can provide a tick formatter method to change how ticks are displayed using the *Chart.add_x_axis()* and *Chart.add_y_axis()* methods.

```
import leather

data = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

def tick_formatter(value, index, tick_count):
    return '%i (%i/%i)' % (value, index, tick_count)

chart = leather.Chart('Line')
chart.add_x_axis(tick_formatter=tick_formatter)
chart.add_line(data)
chart.to_svg('examples/charts/tick_format.svg')
```

## 3.5 Styling

### 3.5.1 Changing theme values

Chart styles are set using a dead simple *theme* system. Leather is meant for making quick and dirty charts. It is neither expected nor recommended for user's to customize these styles.

```
import leather

line_data = [
    (0, 1),
    (2, 5),
    (4, 4),
    (8, 3)
]

dot_data = [
    (1, 3),
    (2, 5),
    (6, 9),
    (10, 4)
]

leather.theme.title_font_family = 'Comic Sans MS'
leather.theme.legend_font_family = 'Comic Sans MS'
leather.theme.tick_font_family = 'Comic Sans MS'
leather.theme.default_series_colors = ['#ff0000', '#00ff00']

chart = leather.Chart('Theme')
chart.add_line(line_data)
chart.add_dots(dot_data)
chart.to_svg('examples/charts/theme.svg')
```

### 3.5.2 Changing series colors

More practically, individual default *Series* colors can be overriden when they are created.

```
import leather

data = [
    ('Hello', 3),
    ('How', 5),
    ('Are', 9),
    ('You', 4)
]

chart = leather.Chart('Series color')
chart.add_columns(data, fill_color='#000000')
chart.to_svg('examples/charts/series_color.svg')
```

### 3.5.3 Styling data based on value

Style attributes of individual data points can be set by value using a *style_function()*.

```
import random

import leather


dot_data = [(random.randint(0, 250), random.randint(0, 250)) for i in range(100)]

def colorizer(d):
    return 'rgb(%i, %i, %i)' % (d.x, d.y, 150)

chart = leather.Chart('Colorized dots')
chart.add_dots(dot_data, fill_color=colorizer)
chart.to_svg('examples/charts/colorized_dots.svg')
```

## 3.6 Chart grids

### 3.6.1 With mixed scales

You can add charts of completely different types to a single graphic by using *Grid*.

```
import leather

data1 = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

data2 = [
```

```
    (3, 4),
    (5, 6),
    (7, 10),
    (8, 2)
]

chart1 = leather.Chart('Dots')
chart1.add_dots(data1)

chart2 = leather.Chart('Lines')
chart2.add_line(data2)

grid = leather.Grid()
grid.add_one(chart1)
grid.add_one(chart2)
grid.to_svg('examples/charts/grid.svg')
```

### 3.6.2 With consistent scales

A grid of charts can automatically be synchronized to a consistent view using *Lattice*.

```
import leather

data1 = [
    (0, 3),
    (4, 5),
    (7, 9),
    (8, 4)
]

data2 = [
    (3, 4),
    (3, 4),
    (5, 6),
    (7, 10),
    (8, 2)
]

data3 = [
    (2, 4),
    (3, 5),
    (6, 2),
    (8, 3),
    (10, 5)
]

lattice = leather.Lattice()
lattice.add_many([data1, data2, data3], titles=['First', 'Second', 'Third'])
lattice.to_svg('examples/charts/lattice.svg')
```

CHAPTER 4

# API

## 4.1 Chart

| | |
|---|---|
| *leather.Chart* | Container for all chart types. |

### 4.1.1 Adding data

| | |
|---|---|
| *leather.Chart.add_series* | Add a data *Series* to the chart. |
| *leather.Chart.add_bars* | Create and add a *Series* rendered with *Bars*. |
| *leather.Chart.add_columns* | Create and add a *Series* rendered with *Columns*. |
| *leather.Chart.add_dots* | Create and add a *Series* rendered with *Dots*. |
| *leather.Chart.add_line* | Create and add a *Series* rendered with *Line*. |

### 4.1.2 Customizing

| | |
|---|---|
| *leather.Chart.set_x_scale* | Set the X *Scale* for this chart. |
| *leather.Chart.set_y_scale* | See *Chart.set_x_scale()*. |
| *leather.Chart.set_x_axis* | Set an *Axis* class for this chart. |
| *leather.Chart.set_y_axis* | See *Chart.set_x_axis()*. |

### 4.1.3 Rendering

| | |
|---|---|
| *leather.Chart.to_svg* | Render this chart to an SVG document. |
| *leather.Chart.to_svg_group* | Render this chart to an SVG group element. |

### 4.1.4 Detailed list

**class** leather.**Chart**(*title=None*)

Bases: object

Container for all chart types.

**Parameters** **title** – An optional title that will be rendered at the top of the chart.

**add_bars**(*data*, *x=None*, *y=None*, *name=None*, *fill_color=None*)
Create and add a *Series* rendered with *Bars*.

Note that when creating bars in this way the order of the series data will be reversed so that the first item in the series is displayed as the top-most bar in the graphic. If you don't want this to happen use *Chart.add_series()* instead.

**add_columns**(*data*, *x=None*, *y=None*, *name=None*, *fill_color=None*)
Create and add a *Series* rendered with *Columns*.

**add_dots**(*data*, *x=None*, *y=None*, *name=None*, *fill_color=None*, *radius=None*)
Create and add a *Series* rendered with *Dots*.

**add_line**(*data*, *x=None*, *y=None*, *name=None*, *stroke_color=None*, *width=None*)
Create and add a *Series* rendered with *Line*.

**add_series**(*series*, *shape*)
Add a data *Series* to the chart. The data types of the new series must be consistent with any series that have already been added.

There are several shortcuts for adding different types of data series. See *Chart.add_bars()*, *Chart.add_columns()*, *Chart.add_dots()*, and *Chart.add_line()*.

**add_x_axis**(*ticks=None*, *tick_formatter=None*, *name=None*)
Create and add an X *Axis*.

If you want to set a custom axis class use *Chart.set_x_axis()* instead.

**add_x_scale**(*domain_min*, *domain_max*)
Create and add a *Scale*.

If the provided domain values are `date` or `datetime` then a *Temporal* scale will be created, otherwise it will *Linear*.

If you want to set a custom scale class use *Chart.set_x_scale()* instead.

**add_y_axis**(*ticks=None*, *tick_formatter=None*, *name=None*)
See *Chart.add_x_axis()*.

**add_y_scale**(*domain_min*, *domain_max*)
See *Chart.add_x_scale()*.

**set_x_axis**(*axis*)
Set an *Axis* class for this chart.

**set_x_scale**(*scale*)
Set the X *Scale* for this chart.

**set_y_axis**(*axis*)
See *Chart.set_x_axis()*.

**set_y_scale**(*scale*)
See *Chart.set_x_scale()*.

**to_svg**(*path=None*, *width=None*, *height=None*)
Render this chart to an SVG document.

The `width` and `height` are specified in SVG's "unitless" units, however, it is usually convenient to specify them as though they were pixels.

**Parameters**

- **path** – Filepath or file-like object to write to. If omitted then the SVG will be returned as a string. If running within IPython, then this will return a SVG object to be displayed.

- **width** – The output width, in SVG user units. Defaults to *theme.default_chart_width*.
- **height** – The output height, in SVG user units. Defaults to *theme.default_chart_height*.

**to_svg_group**(*width=None*, *height=None*)
Render this chart to an SVG group element.

This can then be placed inside an `<svg>` tag to make a complete SVG graphic.

See *Chart.to_svg()* for arguments.

## 4.2 Grid

**class** leather.**Grid**
Bases: object

A container for a set of *Chart* instances that are rendered in a grid layout.

**add_many**(*charts*)
Add a sequence of charts to this grid.

**add_one**(*chart*)
Add a *Chart* to the grid.

**to_svg**(*path=None*, *width=None*, *height=None*)
Render the grid to an SVG.

The `width` and `height` arguments refer to the size of the entire grid. The size of individual charts will be inferred automatically.

See *Chart.to_svg()* for arguments.

## 4.3 Lattice

**class** leather.**Lattice**(*shape=None*)
Bases: object

A grid of charts with synchronized shapes, scales, and axes.

Lattice only supports graphing a single series of data.

**Parameters** **shape** – An instance of *Shape* to use to render all series. Defaults to *Line* if not specified.

**add_many**(*data*, *x=None*, *y=None*, *titles=None*)
Same as *Lattice.add_one()* except `data` is a list of data series to be added simultaneously.

See *Lattice.add_one()* for other arguments.

Note that `titles` is a sequence of titles that must be the same length as `data`.

**add_one**(*data*, *x=None*, *y=None*, *title=None*)
Add a data series to this lattice.

**Parameters**

- **data** – A sequence of data suitable for constructing a *Series*, or a sequence of such objects.

- **x** – See *Series*.
- **y** – See *Series*.
- **title** – A title to render above this chart.

**add_x_axis**(*ticks=None*, *tick_formatter=None*, *name=None*)
Create and add an X *Axis*.

If you want to set a custom axis class use *Lattice.set_x_axis()* instead.

**add_x_scale**(*domain_min*, *domain_max*)
Create and add a *Scale*.

If the provided domain values are `date` or `datetime` then a *Temporal* scale will be created, otherwise it will *Linear*.

If you want to set a custom scale class use *Lattice.set_x_scale()* instead.

**add_y_axis**(*ticks=None*, *tick_formatter=None*, *name=None*)
See *Lattice.add_x_axis()*.

**add_y_scale**(*domain_min*, *domain_max*)
See *Lattice.add_x_scale()*.

**set_x_axis**(*axis*)
Set an *Axis* class for this lattice.

**set_x_scale**(*scale*)
Set the X *Scale* for this lattice.

**set_y_axis**(*axis*)
See *Lattice.set_x_axis()*.

**set_y_scale**(*scale*)
See *Lattice.set_x_scale()*.

**to_svg**(*path=None*, *width=None*, *height=None*)
Render the lattice to an SVG.

See *Grid* for additional documentation.

## 4.4 Scales

**class** `leather.`**Scale**
Bases: `object`

Base class for various kinds of scale objects.

**contains**(*v*)
Return `True` if a given value is contained within this scale's displayed domain.

**format_tick**(*value*, *i*, *count*)
Format ticks for display.

This method is used as a default which will be ignored if the user provides a custom tick formatter to the axis.

**classmethod infer**(*layers*, *dimension*, *data_type*)
Infer's an appropriate default scale for a given sequence of *Series*.

**Parameters**

- **chart_series** – A sequence of *Series* instances
- **dimension** – The dimension, `X` or `Y` of the data to infer for.
- **data_type** – The type of data contained in the series dimension.

**project**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to a target range.

**project_interval**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to an interval in the target range. This is used for places *Bars* and *Columns*.

**ticks**()
Generate a series of ticks for this scale.

**class** leather.**Linear**(*domain_min*, *domain_max*)
Bases: leather.scales.base.Scale

A scale that linearly maps values from a domain to a range.

**Parameters**

- **domain_min** – The minimum value of the input domain.
- **domain_max** – The maximum value of the input domain.

**contains**(*v*)
Return `True` if a given value is contained within this scale's domain.

**project**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to a target range.

**project_interval**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to an interval in the target range. This is used for places *Bars* and *Columns*.

**ticks**()
Generate a series of ticks for this scale.

**class** leather.**Ordinal**(*domain*)
Bases: leather.scales.base.Scale

A scale that maps individual values (e.g. strings) to a range.

**contains**(*v*)
Return `True` if a given value is contained within this scale's displayed domain.

**project**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to a target range.

**project_interval**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to an interval in the target range. This is used for places *Bars* and *Columns*.

**ticks**()
Generate a series of ticks for this scale.

**class** leather.**Temporal**(*domain_min*, *domain_max*)
Bases: leather.scales.base.Scale

A scale that linearly maps date/datetime values from a domain to a range.

**Parameters**

- **domain_min** – The minimum date/datetime of the input domain.

- **domain_max** – The maximum date/datetime of the input domain.

**contains**(*v*)
Return `True` if a given value is contained within this scale's domain.

**format_tick**(*value*, *i*, *count*)
Format ticks for display.

This method is used as a default which will be ignored if the user provides a custom tick formatter to the axis.

**project**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to a target range.

**project_interval**(*value*, *range_min*, *range_max*)
Project a value in this scale's domain to an interval in the target range. This is used for places *Bars* and *Columns*.

**ticks**()
Generate a series of ticks for this scale.

## 4.5 Axis

**class** leather.**Axis**(*ticks=None*, *tick_formatter=None*, *name=None*)
Bases: object

A horizontal or vertical chart axis.

**Parameters**

- **ticks** – Instead of inferring tick values from the data, use exactly this sequence of ticks values. These will still be passed to the `tick_formatter`.
- **tick_formatter** – An optional `tick_format_function()`.

**estimate_label_margin**(*scale*, *orient*)
Estimate the space needed for the tick labels.

**to_svg**(*width*, *height*, *scale*, *orient*)
Render this axis to SVG elements.

## 4.6 Series

**class** leather.**Series**(*data*, *x=None*, *y=None*, *name=None*)
Bases: object

A series of data and its associated metadata.

Series object does not modify the data it is passed.

**Parameters**

- **data** – A sequence (rows) of sequences (columns), a.k.a. `csv.reader()` format. If the `x` and `y` are not specified then the first column is used as the X values and the second column is used for Y.

  Or, a sequence of (rows) of dicts (columns), a.k.a. `csv.DictReader` format. If this format is used then `x` and `y` arguments must specify the columns to be charted.

  Or, a custom data format, in which case `x` and `y` must specify *key_function()*.

- **x** – If using sequence row data, then this may be either an integer index identifying the X column, or a *key_function()*.

  If using dict row data, then this may be either a key name identifying the X column, or a *key_function()*.

  If using a custom data format, then this must be a *key_function()*.‘

- **y** – See x.
- **name** – An optional name to be used in labeling this series. This will be used as the chart title if rendered in a *Lattice*.

**data**()
Return data for this series.

**data_type**(*dimension*)
Return the data type for a dimension of this series.

**max**(*dimension*)
Compute the minimum value of a given dimension.

**min**(*dimension*)
Compute the minimum value of a given dimension.

**values**(*dimension*)
Get a flattened list of values for a given dimension of the data.

leather.**key_function**(*row*, *index*)
This example shows how to define a function to extract X and Y values from custom data.

**Parameters**

- **row** – The function will be called with the row data, in whatever format it was provided to the *Series*.
- **index** – The row index in the series data will also be provided.

**Returns** The function must return a chartable value.

## 4.7 Shapes

**class** leather.**Shape**
Bases: object

Base class for shapes that can be used to render data *Series*.

**legend_to_svg**(*series*, *palette*)
Render the legend entries for these shapes.

**to_svg**(*width*, *height*, *x_scale*, *y_scale*, *series*, *palette*)
Render this shape to an SVG.

**validate_series**(*series*)
Verify this shape can be used to render a given series.

**class** leather.**Bars**(*fill_color=None*)
Bases: leather.shapes.base.Shape

Render a series of data as bars.

**Parameters** **fill_color** – The color to fill the bars. You may also specify a *style_function()*.

**to_svg**(*width*, *height*, *x_scale*, *y_scale*, *series*, *palette*)
Render bars to SVG elements.

**validate_series**(*series*)
Verify this shape can be used to render a given series.

**class** leather.**Columns**(*fill_color=None*)
Bases: leather.shapes.base.Shape

Render a series of data as columns.

**Parameters** **fill_color** – The color to fill the columns. You may also specify a *style_function()*.

**to_svg**(*width*, *height*, *x_scale*, *y_scale*, *series*, *palette*)
Render columns to SVG elements.

**validate_series**(*series*)
Verify this shape can be used to render a given series.

**class** leather.**Dots**(*fill_color=None*, *radius=None*)
Bases: leather.shapes.base.Shape

Render a series of data as dots.

**Parameters**

- **fill_color** – The color to fill the dots. You may also specify a *style_function()*. If not specified, default chart colors will be used.
- **radius** – The radius of the rendered dots. Defaults to *theme.default_dot_radius*. You may also specify a *style_function()*.

**legend_to_svg**(*series*, *palette*)
Render the legend entries for these shapes.

**to_svg**(*width*, *height*, *x_scale*, *y_scale*, *series*, *palette*)
Render dots to SVG elements.

**validate_series**(*series*)
Verify this shape can be used to render a given series.

**class** leather.**Line**(*stroke_color=None*, *width=None*)
Bases: leather.shapes.base.Shape

Render a series of data as a line.

**Parameters**

- **stroke_color** – The color to stroke the lines. If not provided, default chart colors will be used.
- **width** – The width of the lines. Defaults to *theme.default_line_width*.

**to_svg**(*width*, *height*, *x_scale*, *y_scale*, *series*, *palette*)
Render lines to SVG elements.

**validate_series**(*series*)
Verify this shape can be used to render a given series.

leather.**style_function**(*datum*)
This example shows how to define a function to specify style values for individual data points.

**Parameters** **datum** – A Datum instance for the data row.

## 4.8 Theme

This module contains all style configuration for rendering charts. Setting any of these variables will change how charts are rendered.

leather.theme.**axis_title_color = '#666'**
Axis title text color

leather.theme.**axis_title_font_char_height = 14**
Approximate glyph height of the axis title font

leather.theme.**axis_title_font_char_width = 8**
Approximate glyph width of the axis title font

leather.theme.**axis_title_font_family = 'Monaco'**
Axis title font

leather.theme.**axis_title_font_size = 14**
Axis title font size

leather.theme.**axis_title_gap = 16**
Gap between axis title and rest of chart

leather.theme.**background_color = '#f9f9f9'**
Chart background color

leather.theme.**default_chart_height = 600**
Default chart height

leather.theme.**default_chart_width = 800**
Default chart width

leather.theme.**default_dot_radius = 3**
Default *Dots* radius

leather.theme.**default_line_width = 2**
Default *Line* width

leather.theme.**default_series_colors = ['#e41a1c', '#377eb8', '#4daf4a', '#984ea3', '#ff7f00']**
Default sequence of *Shape* colors

leather.theme.**label_color = '#9c9c9c'**
Color of tick label text

leather.theme.**legend_bubble_offset = 4**
Offset from the top of the glyph

leather.theme.**legend_bubble_size = 10**
Size of the bubble next to an legend item

leather.theme.**legend_color = '#666'**
Chart legend text color

leather.theme.**legend_font_char_height = 14**
Approximate glyph height of the legend font

leather.theme.**legend_font_char_width = 8**
Approximate glyph width of the legend font

leather.theme.**legend_font_family = 'Monaco'**
Chart legend font

leather.theme.**legend_font_size = 14**
Chart legend font size

leather.theme.**legend_gap = 4**
Gap between legend and rest of chart

leather.theme.**margin = 0.05**
Chart margin as a percent of chart width

leather.theme.**tick_color = '#eee'**
Color of tick marks

leather.theme.**tick_font_char_height = 14**
Approximate glyph height of the tick label font

leather.theme.**tick_font_char_width = 8**
Approximate glyph width of the tick label font

leather.theme.**tick_font_family = 'Monaco'**
Tick label font

leather.theme.**tick_font_size = 14**
Tick label font size

leather.theme.**tick_size = 4**
Length of a tick mark

leather.theme.**tick_width = 1**
Width of a tick mark

leather.theme.**title_color = '#333'**
Chart title text color

leather.theme.**title_font_char_height = 16**
Approximate glyph height of the title font

leather.theme.**title_font_char_width = 9**
Approximate glyph width of the title font

leather.theme.**title_font_family = 'Monaco'**
Chart title font

leather.theme.**title_font_size = 16**
Chart title font size

leather.theme.**title_gap = 4**
Gap between title and rest of chart

leather.theme.**zero_color = '#a8a8a8'**
Color of the zero tick mark

CHAPTER 5

# Changelog

## 5.1 0.3.3 - November 30, 2016

- Fix examples that used invalid column data. (#81)
- lxml is no longer imported by default. (#83)
- Ordinal scales can now display data from multiple series with different values. (#76)
- Better error handling for data types supported by different shapes.

## 5.2 0.3.2 - November 11, 2016

- Fix trove classifiers.

## 5.3 0.3.1 - November 11, 2016

- Fix unicode rendering issue in Python2.7 and PyPy. (#74)

## 5.4 0.3.0 - November 11, 2016

- Add examples for many more use-cases. (#11)
- Fixed bars so that data are displayed top-down when using `Chart.add_bars()`. (#72)
- Changed default colors. (#51)
- Fixed a rare file handling bug when saving SVG files.
- Leather will now issue a warning if you attempt to render a chart with data exceeding the scale domain. (#42)
- Linear scales will now default to the domain `[0, 1]` if no values are provided. (#66)
- Axis no longer takes a number of ticks as an argument. Instead pass a list of custom tick values.
- Scales `tick` methods no longer take a number of ticks as an argument. (They should self-optimize.)
- Scales that cross `0` will now always have a tick at `0`. (#54)
- Implemented auto-ticking. (#23)

- *style_function()* now takes a Datum instances, rather than a list of arguments.
- Renamed the Lines class to *Line* to be more accurate.
- Implemented CategorySeries.
- Implemented a more elegant pattern for colorizing series.
- Refactored *Series* so *Shape* is no longer a parameter.
- Tick values can now be overridden with the tick_values argument. (#56)
- Added methods to customize scales and axes for *Lattice* charts. (#17)
- Expanded unit tests for *Scale* subclasses.
- Zero lines now render above other tick marks. (#31)
- Fixed rendering of Bar and Column shapes for negative values. (#52)
- Refactored the *Lattice* API.

## 5.5 0.2.0

- Initial prototype

## 5.6 0.1.0

- Never released

CHAPTER 6

# Release process

This is the release process for leather:

1. Verify all unit tests pass with fresh environments: `tox -r`.
2. Check test coverage: `nosetests --with-coverage tests`.
3. Ensure any new modules have been added to setup.py's `packages` list.
4. Ensure any new public interfaces have been added to the documentation.
5. Make sure the example scripts still work: `./examples.sh`.
6. Ensure `CHANGELOG.rst` is up to date. Add the release date and summary.
7. Create a release tag: `git tag -a x.y.z -m "x.y.z release."`
8. Push tags upstream: `git push --tags`
9. If this is a major release, merge `master` into `stable`: `git checkout stable; git merge master; git push`
10. Upload to PyPI: `python setup.py sdist bdist_wheel upload`.
11. Flag the release to build on RTFD.
12. Update the "default version" on RTFD to the latest.
13. Rev to latest version: `docs/conf.py`, `setup.py` and `CHANGELOG.rst` need updates.
14. Commit revision: `git commit -am "Update to version x.y.z for development."`.

CHAPTER 7

# License

The MIT License

Copyright (c) 2016 Christopher Groskopf and contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 8

# Show me docs

- About - why you should use leather
- Install - how to install for users and developers
- Examples - code + output examples of every feature of leather
- API - technical documentation for the leather API
- Changelog - a record of every change made for each release

CHAPTER 9

# Show me code

```
import random

import leather


dot_data = [(random.randint(0, 250), random.randint(0, 250)) for i in range(100)]

def colorizer(d):
    return 'rgb(%i, %i, %i)' % (d.x, d.y, 150)

chart = leather.Chart('Colorized dots')
chart.add_dots(dot_data, fill_color=colorizer)
chart.to_svg('examples/charts/colorized_dots.svg')
```

CHAPTER 10

# Join us

- Release process - the process for maintainers to publish new releases
- License - a copy of the MIT open source license covering leather

CHAPTER 11

# Who we are

The following individuals have contributed code, documentation, or expertise to leather:

- Christopher Groskopf

CHAPTER 12

# Indices and tables

- genindex
- modindex
- search

## l

## M



## O



## P



## S



## T



## V



## Z