altair$_{r}ecipes Documentation$

Release __version__ = '0.9.0'

Antonio Piccolboni

Feb 23, 2022

Contents:

CHAPTER 1

Introduction to altair_recipes

1.1 A collection of ready-made statistical graphics for vega.

`vega` is a statistical graphics system for the web, meaning the plots are displayed in a browser. As an added bonus, it adds interactions, again through web technologies: select data point, reveal information on hover etc. Interaction and the web are clearly the future of statistical graphics. Even the successor to the famous `ggplot` for R, `ggvis` is based on `vega`.

`altair` is a python package that produces `vega` graphics. Like `vega`, it adopts an approach to describing statistical graphics known as *grammar of graphics* which underlies other well known packages such as `ggplot` for R. It represents a extremely useful compromise of power and flexibility. Its elements are data, marks (points, lines), encodings (relations between data and marks), scales etc.

Sometimes we want to skip all of that and just produce a boxplot (or heatmap or histogram, the argument is the same) by calling:

```
boxplot(data.iris(), columns="petalLength", group_by="species")
```

because:

- It's a well known type of statistical graphics that everyone can recognize and understand on the fly.
- Creativity is nice, in statistical graphics as in many other endeavors, but dangerous: there are more bad charts out there than good ones. The *grammar of graphics* is no insurance.

- While it's simple to put together a boxplot in `altair`, it isn't trivial: there are rectangles, vertical lines, horizontal lines (whiskers), points (outliers). Each element is related to a different statistics of the data. It's about 30 lines of code and, unless you run them, it's hard to tell you are looking at a boxplot.
- One doesn't always need the control that the grammar of graphics affords. There are times when I need to see a boxplot as quick as possible. Others, for instance preparing a publication, when I need to control every detail.

The boxplot is not the only example. The scatterplot, the quantile-quantile plot, the heatmap are important idioms that are battle tested in data analysis practice. They deserve their own abstraction. Other packages offering an abstraction above the grammar level are:

- `seaborn` and the graphical subset of `pandas`, for example, both provide high level statistical graphics primitives (higher than the grammar of graphics) and they are quite successful (but not web-based).
- `ggplot`, even if named after the Grammar of Graphics, slipped in some more complex charts, pretending they are elements of the grammar, such as `geom_boxplot`, because sometimes even R developers are lazy. But a boxplot is not a *geom* or mark. It's a combination of several ones, certain statistics and so on. I suspect the authors of `altair` know better than mixing the two levels.

`altair_recipes` aims to fill this space above `altair` while making full use of its features. It provides a growing list of "classic" statistical graphics without going down to the grammar level. At the same time it is hoped that, over time, it can become a repository of examples and model best practices for `altair`, a computable form of its gallery.

There is *one more thing*. It's nice to have all these famous chart types available at a stroke of the keyboard, but we still have to decide which type of graphics to use and, in certain cases, the association between variables in the data and channels in the graphics (what becomes coordinate, what becomes color etc.). It still is work and things can still go wrong, sometimes in subtle ways. Enter `autoplot`. `autoplot` inspects the data, selects a suitable graphics and generates it. While no claim is made that the result is optimal, it will make reasonable choices and avoid common pitfalls, like overlapping points in scatterplots. While there are interesting research efforts aimed at characterizing the optimal graphics for a given data set, their goal is more ambitious than just selecting from a repertoire of pre-defined graphics types and they are fairly complex. Therefore, at this time `autoplot` is based on a set of reasonable heuristics derived from decades of experience such as:

- use stripplot and scatterplot to display continuous data, barcharts for discrete data
- use opacity to counter mark overlap, but not with discrete color maps
- switch to summaries (count and averages) when the amount of overlap is too high
- use facets for discrete data

`autoplot` is work in progress and perhaps will always be and feedback is most welcome. A large number of charts generated with it is available at the end of the Examples page and should give a good idea of what it does. In particular, in this first iteration we do not make any attempt to detect if a dataset represents a function or a relation, hence scatterplots are preferred over line plots. Moreover there is no special support for evenly spaced data, such as a time series.

1.2 Features

- Free software: BSD license.
- Fully documented.
- Highly consistent API enforced with autosig
- Near 100% regression test coverage.
- Support for dataframe and vector inputs
- Support for both wide and long dataframe formats.

- Data can be provided as a dataframe or as a URL pointing to a csv or json file.
- All charts produced are valid `altair` charts, can be modified, combined, saved, served, embedded exactly as one.

1.3 Chart types

- autocorrelation
- barchart
- boxplot
- heatmap
- histogram, in a simple and multi-variable version
- qqplot
- scatterplot in the simple and all-vs-all versions
- smoother, smoothing line with IRQ range shading
- stripplot

See Examples.

CHAPTER 2

Examples

These examples are taken unedited from the test suite. Look at the body of each test to see how `altair_recipes` can be used.

CHAPTER 3

Installation

3.1 Stable release

To install altair_recipes, run this command in your terminal:

```
$ pip install altair_recipes
```

This is the preferred method to install altair_recipes, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

3.2 From sources

The sources for altair_recipes can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/piccolbo/altair_recipes
```

Or download the tarball:

```
$ curl  -OL https://github.com/piccolbo/altair_recipes/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 4

Usage

To use altair_recipes in a project:

```
import altair_recipes
```

CHAPTER 5

altair_recipes

5.1 altair_recipes package

5.1.1 Module contents

Top-level package for altair_recipes.

`altair_recipes.`**`areaplot`**(*data=None*, *x=0*, *y=1*, *color=None*, *stack=<StackType.auto: None>*, *height=600*, *width=800*)

Generate an areaplot.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`x`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **`y`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **`color`** (*str* or *int*) – The column containing the data associated with the color of the mark
- **`stack`** (`StackType`) – One of *StackType.auto* (automatic selection), *StackType.true* (force), *StackType.false* (no stacking) and *StackType.normalize* (for normalized stacked)
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`autocorrelation`**(*data=None*, *column=0*, *max_lag=None*, *height=600*, *width=800*)

Generate an autocorrelation plot.

Parameters

- **data** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **column** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data to be used in the graphics
- **max_lag** (`int`) – Maximum lag to show in the plot, defaults to number of rows in data
- **height** (*int*) – The height of the chart
- **width** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`autoplot`**(*data=None*, *columns=None*, *group_by=None*, *height=600*, *width=800*)

Automatically choose and produce a statistical graphics based on up to three columns of data.

Parameters

- **data** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **columns** (collection of: *int*, *str*, pandas *Series* or a type convertible to it.) – The column or columns to be used in the graphics, defaults to all
- **group_by** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column to be used to group the data when in long form. When group_by is specified columns should contain a single column
- **height** (*int*) – The height of the chart
- **width** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`barchart`**(*data=None*, *x=0*, *y=1*, *color=False*, *height=600*, *width=800*)

Generate a barchart.

Parameters

- **data** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **x** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **y** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **color** (`bool`) – Whether to also use color to encode the same data as the x coordinate
- **height** (*int*) – The height of the chart
- **width** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`boxplot`**(*data=None*, *columns=None*, *group_by=None*, *color=False*, *height=600*, *width=800*)

Generate a boxplot.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`columns`** (collection of: *int*, *str*, pandas *Series* or a type convertible to it.) – The column or columns to be used in the graphics, defaults to all
- **`group_by`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column to be used to group the data when in long form. When group_by is specified columns should contain a single column
- **`color`** (`bool`) – Whether to also use color to encode the same data as the x coordinate
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`layer`**(**layers*, ***kwargs*)

Layer charts: a drop in replacement for altair.layer that does a deepcopy of the layers to avoid side-effects and lifts identical datasets one level down to top level.

`altair_recipes.`**`lineplot`**(*data=None*, *x=0*, *y=1*, *color=None*, *height=600*, *width=800*)

Generate a lineplot.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`x`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **`y`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **`color`** (*str* or *int*) – The column containing the data associated with the color of the mark
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`heatmap`**(*data=None*, *x=0*, *y=1*, *color=2*, *opacity=None*, *aggregate='average'*, *height=600*, *width=800*)

Generate a heatmap.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`x`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **`y`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **`color`** (*str* or *int*) – The column containing the data associated with the color of the mark

- **`opacity`** (*str*) –
- **`column containing the data that determines opacity of the mark`** (*`The`*) –
- **`aggregate`** (*str*) – The aggregation function to set the color of each mark, see https://altair-viz.github.io/user_guide/encoding.html#encoding-aggregates for available options
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`histogram`**(*data=None*, *column=0*, *height=600*, *width=800*)

Generate a histogram.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`column`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data to be used in the graphics
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`layered_histogram`**(*data=None*, *columns=None*, *group_by=None*, *height=600*, *width=800*)

Generate multiple overlapping histograms.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`columns`** (collection of: *int*, *str*, pandas *Series* or a type convertible to it.) – The column or columns to be used in the graphics, defaults to all
- **`group_by`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column to be used to group the data when in long form. When group_by is specified columns should contain a single column
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`multiscatterplot`**(*data=None*, *columns=None*, *group_by=None*, *color=None*, *opacity=1*, *tooltip=None*, *height=600*, *width=800*)

Generate many scatterplots.

Based on several columns, pairwise.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`columns`** (collection of: *int*, *str*, pandas *Series* or a type convertible to it.) – The column or columns to be used in the graphics, defaults to all
- **`group_by`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column to be used to group the data when in long form. When group_by is specified columns should contain a single column
- **`color`** (*str* or *int*) – The column containing the data associated with the color of the mark
- **`opacity`** (*float*) – A constant value for the opacity of the mark
- **`tooltip`** (*str* or *int*) – The column containing the data associated with the tooltip text
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`qqplot`**(*data=None*, *x=0*, *y=1*, *height=600*, *width=800*)

Generate a quantile-quantile plot.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`x`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **`y`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`scatterplot`**(*data=None*, *x=0*, *y=1*, *color=None*, *opacity=1*, *tooltip=None*, *height=600*, *width=800*)

Generate a scatterplot.

Parameters

- **`data`** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **`x`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **`y`** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **`color`** (*str* or *int*) – The column containing the data associated with the color of the mark
- **`opacity`** (*float*) – A constant value for the opacity of the mark
- **`tooltip`** (*str* or *int*) – The column containing the data associated with the tooltip text

- **height** (*int*) – The height of the chart
- **width** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

`altair_recipes.`**`smoother`**(*data=None*, *x=0*, *y=1*, *window=None*, *interquartile_area=True*, *height=600*, *width=800*)

Generate a smooth line plot with optional IRQ shading area.

Parameters

- **data** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **x** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the horizontal dimension
- **y** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column containing the data associated with the vertical dimension
- **window** (`int`) – The size of the smoothing window
- **interquartile_area** (`interquartile_area: bool`) – Whether to plot the IRQ as an area
- **height** (*int*) – The height of the chart
- **width** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

class `altair_recipes.`**`StackType`**

Bases: `enum.Enum`

An enumeration.

auto = None

false = False

normalize = 'normalize'

true = True

`altair_recipes.`**`stripplot`**(*data=None*, *columns=None*, *group_by=None*, *color=None*, *opacity=1*, *height=600*, *width=800*)

Generate a stripplot.

Parameters

- **data** (*altair.Data* or *pandas.DataFrame* or csv or json file URL) – The data from which the statistical graphics is being generated
- **columns** (collection of: *int*, *str*, pandas *Series* or a type convertible to it.) – The column or columns to be used in the graphics, defaults to all
- **group_by** (*int*, *str*, pandas *Series* or a type convertible to it.) – The column to be used to group the data when in long form. When group_by is specified columns should contain a single column
- **color** (*str* or *int*) – The column containing the data associated with the color of the mark
- **opacity** (`float`) – The value of the constant opacity of the mark (use to counter overlap)

- **`height`** (*int*) – The height of the chart
- **`width`** (*int*) – The width of the chart

Returns An altair Chart.

Return type type altair.Chart or altair.LayerChart

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/piccolbo/altair_recipes/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Propose Features

The types of new features we can think of are of two types. First is more flexibility for charts that altair_recipes can produce already, e.g. the recent addition of height and width controls; second is entirely new types of chars. As to the first, we are trying to balance two aims: keeping it simple and making it powerful enough to cover common visualization needs. This isn't very precise, but we will try to make it more so over time. Controlling the width seemed a necessity. Changing a color palette, maybe not so much (it can also be controlled with `altair`'s `configure_*` methods). As to entirely new types of chart, we'd like to include any charts that are in widespread use in data analysis practice, which may have a scientific article or a wikipedia entry devoted to them or other supporting evidence of statistical relevance. Chart types that have been used once or are implemented in a single library, like the *jointplot*, are not good candidates. To propose a new feature, please open a new issue with description, rationale, an example and, ideally, sample implementation in `altair` or `vega-lite`.

6.1.4 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it. A new type of chart will require a new test.

6.1.5 Write Documentation

altair_recipes could always use more documentation, whether as part of the official altair_recipes docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.6 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/piccolbo/altair_recipes/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-only project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here’s how to set up *altair_recipes* for local development.

1. Fork the *altair_recipes* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/altair_recipes.git
```

3. Install your local copy into a virtualenv. This is how you set up your fork for local development:

```
$ curl -sSL https://raw.githubusercontent.com/sdispater/poetry/master/get-poetry.
↪py | python #if needed, or other method to install poetry
$ cd altair_recipes/
$ poetry install
```

4. Create a branch for local development:

```
$ git checkout -b <branch-name>
```

   Where <branch-name> can be as simple as `issue-<issue-number>` but should always end with `-<issue-number>`. Now you can make your changes locally.

5. When you’re done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 altair_recipes tests
$ make test
$ tox # in the works
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin <branch-name>
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests. Coverage should never decrease (check with make coverage)
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add new chat types to the list in README.rst.
3. The pull request should work for Python 3.5 and 3.6, or as listed in file travis.yml. Check https://travis-ci.org/piccolbo/altair_recipes/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_altair_recipes
```

Tests should be decorated with `@viz-reg-test` and produce an altair chart. This will save the json output for regression testing and produce an html file for visual inspection.

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
```

We use semantic versioning. Then:

```
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass (not implemented yet, use `make release`)

CHAPTER 7

Credits

7.1 Development Lead

- Antonio Piccolboni <antonio@piccolboni.info>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.9.0 (2020-06-11)

- Fixed color in boxplot
- Upgrade to altair 4. Mandatory. Let me know if you need compatibility with 3.x.x

8.2 0.8.0 (2019-10-16)

- Added lineplots and areaplots #11 and #12

8.3 0.7.1 (2019-10-07)

- Accepts vector data in addition to dataframe, as in:

```
import altair_recipes as ar
from numpy.random import normal
ar.scatterplot(x=normal(size=100), y=normal(size=100))
```

8.4 0.6.5 (2019-10-01)

- Make ipython dep optional (for pweave support). Use piccolbo's pweave fork (upstream doesn't pass its own tests) for doc generation. Adapt to breaking changes in autosig (a dependency).

8.5 0.6.4 (2019-09-18)

- Switched to poetry for package management

8.6 0.6.0 (2019-01-25)

- **Fine tuned API:**
  - no faceting but all returned charts are facet-able
  - Color made a bool option when separate color dim can't work
  - Eliminated some special cases from autoplot for very small datasets
  - Some refactor in boxpolot and autoplot to shrink, clarify code

8.7 0.5.0 (2019-01-17)

- Autoplot for automatic statistical graphics
- Stripplots and barcharts

8.8 0.4.0 (2018-09-25)

- Custom height and width for all charts

8.9 0.3.2 (2018-09-21)

- Dealt with breaking changes from autosig, but code is simpler and paves the way for some new features

8.10 0.3.1 (2018-09-20)

- Addressing a documentation mishap

8.11 0.3.0 (2018-09-20)

- Better readme and a raft of examples
- Some test flakiness addressed

8.12 0.2.4 (2018-08-29)

- One more issue with col resolution
- Switch to using docstring support in autosig

8.13 0.2.3 (2018-08-29)

- Some issues with processing of *columns* and *group_by* args
- Fixed travis-ci build (3.6 only, 3.5 looks like a minor RNG issue)

8.14 0.2.2 (2018-08-28)

- Switch to a simpler, flatter API a la qplot
- Added two types of heatmaps
- Extensive use of autosig features for API consistency and reduced boilerplate
- Fixed build to follow requests model (pip for users, pipenv for devs)

8.15 0.1.2 (2018-08-14)

- Fixed a number of loose ends particularly wrt docs

8.16 0.1.0 (2018-08-06)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index