datashader Documentation

Release 0.1.0

datashader contributors

February 10, 2016

Contents

Datashader is a graphics pipeline system for creating meaningful representations of large amounts of data. It breaks the creation of images into 3 steps:

1. Projection

   Each record is projected into zero or more bins, based on a specified glyph.

2. Aggregation

   Reductions are computed for each bin, compressing the potentially large dataset into a much smaller *aggregate*.

3. Transformation

   These aggregates are then further processed to create an image.

Using this very general pipeline, many interesting data visualizations can be created in a performant and scalable way. Datashader contains tools for easily creating these pipelines in a composable manner, using only a few lines of code:

```
>>> import datashader as ds
>>> import datashader.transfer_functions as tf

>>> import pandas as pd
>>> df = pd.read_csv('user_data.csv')

# **Projection & Aggregation Step:**
# Map each record as a point centered by the fields `x_col` and `y_col` to
# a 400x400 grid of bins, computing the mean of `z_col` for all records in
# each bin.
>>> cvs = ds.Canvas(plot_width=400, plot_height=400)
>>> agg = cvs.points(df, 'x_col', 'y_col', ds.mean('z_col'))

# **Transformation Step:**
# Interpolate the resulting means along a logarithmic color palette from
# "lightblue" to "darkblue"
>>> img = tf.interpolate(agg, 'lightblue', 'darkblue', how='log')
```

CHAPTER 1

Examples

The repository contains several runnable examples, which can be found here. Many of the examples are in the form of Jupyter notebooks. Copies of these with all the images and output included can be viewed on Anaconda Cloud here.

1.1 API

1.1.1 Entry Points

Canvas([plot_width, plot_height, x_range, ...])	An abstract canvas representing the space in which to bin.
Pipeline(df, glyph[, agg, transform_fn, ...])	A datashading pipeline callback.

1.1.2 Glyphs

Point(x, y)	A point, with center at `x` and `y`.

1.1.3 Reductions

count([column])	Count elements in each bin.
sum(column)	Sum of all elements in `column`.
min(column)	Minimum value of all elements in `column`.
max(column)	Maximum value of all elements in `column`.
mean(column)	Mean of all elements in `column`.
var(column)	Variance of all elements in `column`.
std(column)	Standard Deviation of all elements in `column`.
count_cat(column)	Count of all elements in `column`, grouped by category.
summary(**kwargs)	A collection of named reductions.

1.1.4 Transfer Functions

interpolate(agg[, low, high, how])	Convert a 2D DataArray to an image.
colorize(agg, color_key[, how, min_alpha])	Color a CategoricalAggregate by field.
stack(*imgs)	Merge a number of images together, overlapping earlier images with later ones.

Continued on next page

Table 1.4 – continued from previous page

merge(*imgs)	Merge a number of images together, averaging the channels

1.1.5 Definitions

class datashader.**Canvas**(*plot_width=600*, *plot_height=600*, *x_range=None*, *y_range=None*, *x_axis_type='linear'*, *y_axis_type='linear'*)

An abstract canvas representing the space in which to bin.

Parameters **plot_width, plot_height** : int, optional

Width and height of the output aggregate in pixels.

x_range, y_range : tuple, optional

A tuple representing the bounds inclusive space `[min, max]` along the axis.

x_axis_type, y_axis_type : str, optional

The type of the axis. Valid options are `'linear'` [default], and `'log'`.

class datashader.**Pipeline**(*df*, *glyph*, *agg=<datashader.reductions.count object>*, *transform_fn=<function identity>*, *color_fn=<function interpolate>*)

A datashading pipeline callback.

Given a declarative specification, creates a callable with the following signature:

```
callback(x_range, y_range, width, height)
```

where `x_range` and `y_range` form the bounding box on the viewport, and `width` and `height` specify the output image dimensions.

Parameters **df** : pandas.DataFrame, dask.DataFrame

glyph : Glyph

The glyph to bin by.

agg : Reduction, optional

The reduction to compute per-pixel. Default is `count()`.

transform_fn : callable, optional

A callable that takes the computed aggregate as an argument, and returns another aggregate. This can be used to do preprocessing before passing to the `color_fn` function.

color_fn : callable, optional

A callable that takes the output of `tranform_fn`, and returns an `Image` object. Default is `interpolate`.

class datashader.glyphs.**Point**(*x*, *y*)

A point, with center at `x` and `y`.

Points map each record to a single bin.

Parameters **x, y** : str

Column names for the x and y coordinates of center of each point.

class datashader.reductions.**count**(*column=None*)

Count elements in each bin.

Parameters **column** : str, optional

If provided, only counts elements in `column` that are not `NaN`. Otherwise, counts every element.

class `datashader.reductions.`**`sum`**(*column*)
Sum of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`min`**(*column*)
Minimum value of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`max`**(*column*)
Maximum value of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`mean`**(*column*)
Mean of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`var`**(*column*)
Variance of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`std`**(*column*)
Standard Deviation of all elements in `column`.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be numeric. `NaN` values in the column are skipped.

class `datashader.reductions.`**`count_cat`**(*column*)
Count of all elements in `column`, grouped by category.

Parameters **column** : str

Name of the column to aggregate over. Column data type must be categorical. Resulting aggregate has a outer dimension axis along the categories present.

class `datashader.reductions.`**`summary`**(***kwargs*)
A collection of named reductions.

Computes all aggregates simultaneously, output is stored as a `xarray.Dataset`.

Examples

A reduction for computing the mean of column "a", and the sum of column "b" for each bin, all in a single pass.

```
>>> import datashader as ds
>>> red = ds.summary(mean_a=ds.mean('a'), sum_b=ds.sum('b'))
```

datashader.transfer_functions.**merge**(**imgs*)
Merge a number of images together, averaging the channels

datashader.transfer_functions.**stack**(**imgs*)
Merge a number of images together, overlapping earlier images with later ones.

datashader.transfer_functions.**interpolate**(*agg*, *low='lightblue'*, *high='darkblue'*, *how='cbrt'*)
Convert a 2D DataArray to an image.

Parameters **agg** : DataArray

low : color name or tuple

The color for the low end of the scale. Can be specified either by name, hexcode, or as a tuple of `(red, green, blue)` values.

high : color name or tuple

The color for the high end of the scale

how : string or callable

The interpolation method to use. Valid strings are 'cbrt' [default], 'log', and 'linear'. Callables take a 2-dimensional array of magnitudes at each pixel, and should return a numeric array of the same shape.

datashader.transfer_functions.**colorize**(*agg*, *color_key*, *how='cbrt'*, *min_alpha=20*)
Color a CategoricalAggregate by field.

Parameters **agg** : DataArray

color_key : dict or iterable

A mapping of fields to colors. Can be either a `dict` mapping from field name to colors, or an iterable of colors in the same order as the record fields.

how : string or callable

The interpolation method to use. Valid strings are 'cbrt' [default], 'log', and 'linear'. Callables take a 2-dimensional array of magnitudes at each pixel, and should return a numeric array of the same shape.

min_alpha : float, optional

The minimum alpha value to use for non-empty pixels, in [0, 255].

d

C

D

I

M

P

S

V