GeoViews: From exploratory analysis

to custom GIS dashboards in a few lines of Python code

FOSS4G Boston 2017

Philipp Rudiger and James A. Bednar

Continuum Analytics

Let's say you want to make it easy to explore some dataset. That is, you want to:

- Make a visualization of the data
- Maybe add some custom widgets to see the effects of some variables
- Then deploy the result as a web app.

You can do that in Python, but you typically need to:

- Spend days of effort to get some initial prototype working, e.g. in a Jupyter notebook
- Try to tame the resulting opaque mishmash of domain-specific, widget, and plotting code
- Start over nearly from scratch whenever you need to:
    - Deploy in a standalone server
    - Visualize different aspects of your data
    - Scale up to larger (>100K) datasets

Is there a better way?

In the first part of the talk, we'll introduce some new Python libraries that help make specific steps in th process easier.

In the second part, we'll show how these and other libraries fit together to make it easy to develop custo apps for exploring data.

PyData Ecosystem

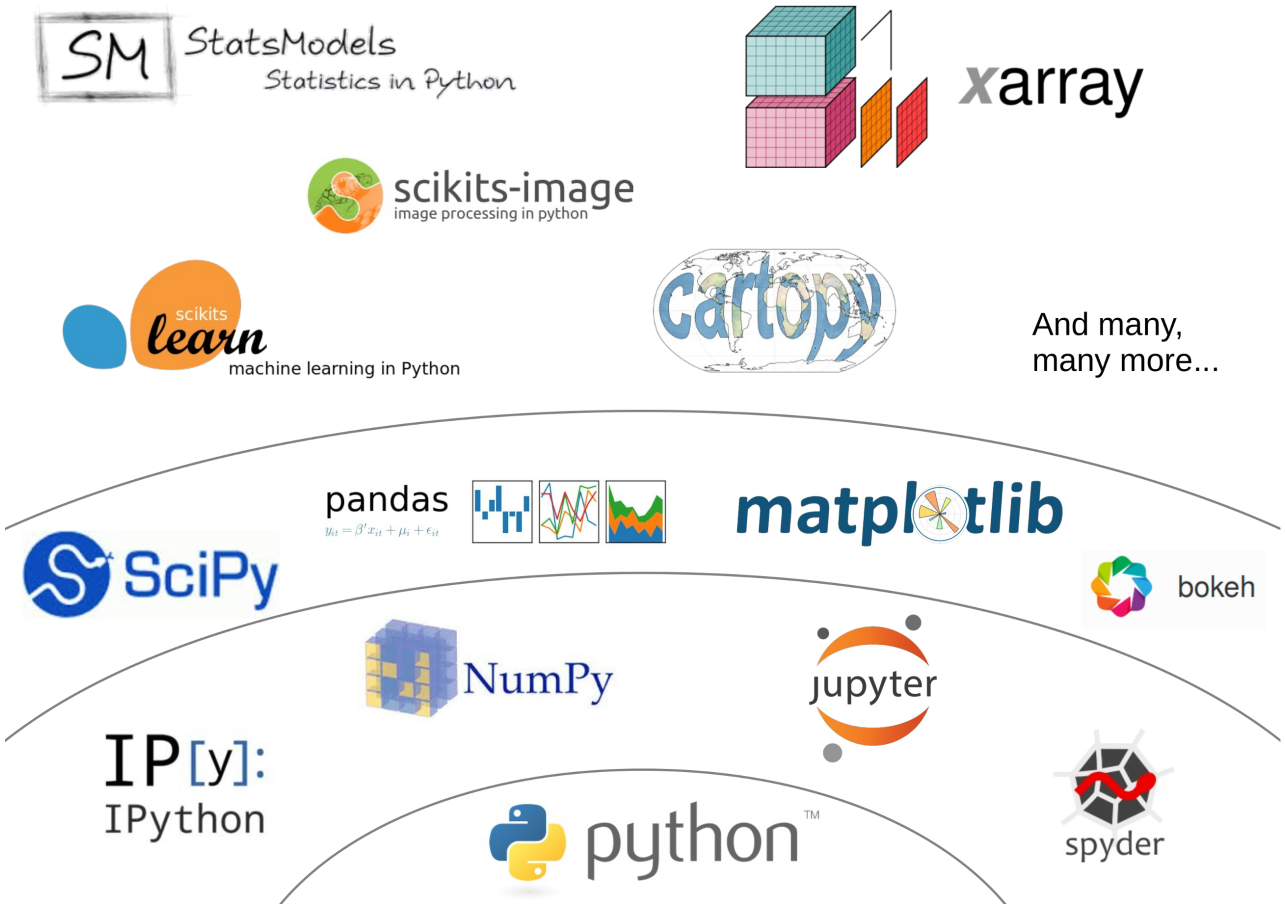
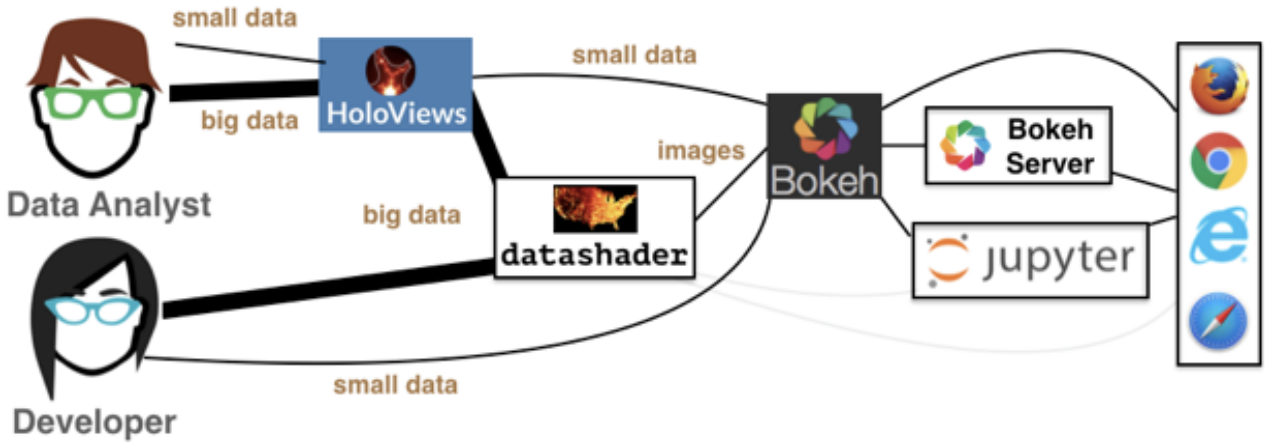


Image by Jake VanderPlas

Bokeh Ecosystem

- Bokeh interactive plotting in the browser
- Supports Jupyter notebook for interactive plotting
- Includes Bokeh Server to deploy as standalone apps
- Works well for medium data sizes (up to 100K points)
- Seamlessly integrates with new Datashader library for larger datasets
- HoloViews: New high-level interface coordinating Datashader and Bokeh
- GeoViews: New geography-specific extensions for HoloViews

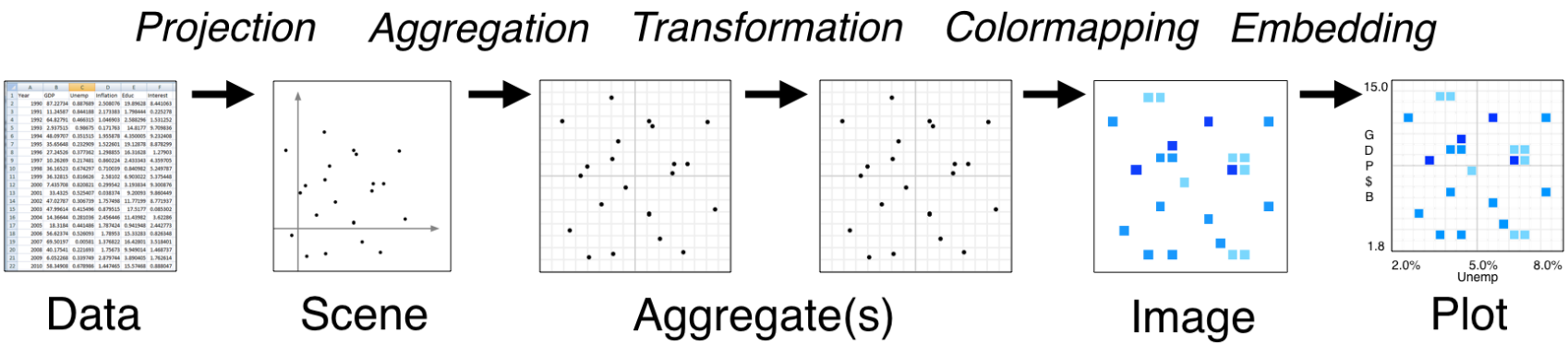


HoloViews/GeoViews

- HoloViews:
    - Declarative objects for instantly visualizable data
    - Supports different plotting extensions including matplotlib and Bokeh
    - Makes it simple to lay out and overlay different types of plots
    - Simplifies exploring multidimensional parameter spaces
    - Accepts data from pandas, xarray (NetCDF), shapefiles, geopandas, etc.
- GeoViews:
    - GIS extension for HoloViews based on Cartopy for geographic projections
    - Wide range of declarative primitives
    - Support for:
        - Tile sources
        - Geographic features
        - Projections
        - Points, Shapes, Rasters, etc.

Dealing with large data: Datashader

- Data larger than ~100k points cannot easily be rendered in the browser
- Aggregating data into fixed size image allows us to explore huge datasets
- Thanks to Numba and Dask we can scale to a billion datapoints on a laptop and many more on a cluster
- Will show examples shortly



Summary so far – Key libraries:

1. **XArray** to load GIS data into flexible objects.
2. **HoloViews** to wrap these objects with metadata that makes them visualizable
3. **GeoViews** to provide GIS-specific functionality like projections to HoloViews
4. **Bokeh** to create browser-based interactive visualizations from HoloViews objects
5. **Datashader** to render large datasets into feasible representations
6. **Dask** and **Numba** behind the scenes to make it all very fast

So, given these libraries, how can they fit together to solve problems?

Let's look at a full data science workflow:

1. Load a dataset to explore
2. Explore the data interactively in a notebook, customizing and tweaking it
3. Identify the important variables and create widgets to let the user adjust them
4. Connect the widgets with the visualization for live updates
5. Deploy the result as a standalone app/dashboard to deliver insight

Step 1: Get some data

- Here we'll use a subset of the often-studied NYC Taxi dataset
- About 12 million points of GPS locations from taxis
- Stored in the efficient Parquet format for easy access
- Loaded into a Dask dataframe for multi-core
  (and if needed, out of core or distributed) computation

In [35]:
```
1 %time df = dd.read_parquet('./data/nyc_taxi.parq/').persist()
2 print(len(df))
3 df.head(2)
```

```
CPU times: user 1.01 s, sys: 505 ms, total: 1.52 s
Wall time: 412 ms
11842094
```

Out[35]:

index	passenger_count	pickup_x	pickup_y	dropoff_x	dropoff_y
0	1	-8236963.0	4975552.5	-8234835.5	4975627.0
1	1	-8237826.0	4971752.5	-8237020.5	4976875.0

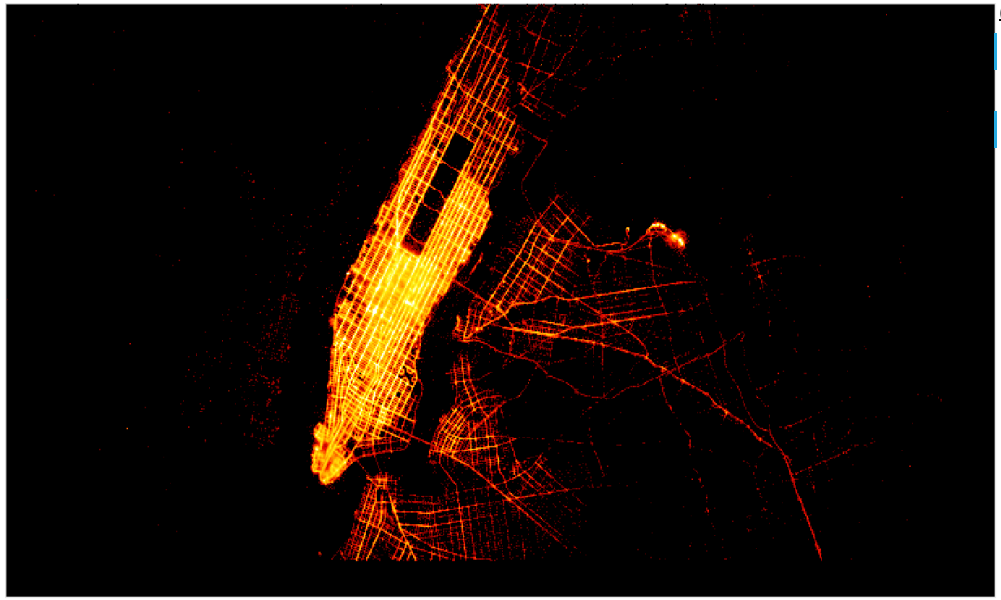
Step 2: Prototype a plot in a notebook

- A text-based representation isn't very useful for big datasets like this, so we need to build a plot
- Because we don't want to start a software project, we use HoloViews
- Because we want to overlay data on maps, we use GeoViews with HoloViews
- Because we want our plots to be interactive, we use Bokeh as the plotting library
- Because our data is much too big for Bokeh directly, we'll use Datashader to rasterize it first

In [36]:

```
points = gv.Points(df, kdims=['pickup_x', 'pickup_y'], vdims=['passenger_count'], crs=ccrs.GOOGLE_MERCATOR)
options = dict(width=800,height=475,xaxis=None,yaxis=None,bgcolor='black')
taxi_trips = datashade(points, x_sampling=1, y_sampling=1, cmap=cm['fire']).opts(plot=options)
taxi_trips
```

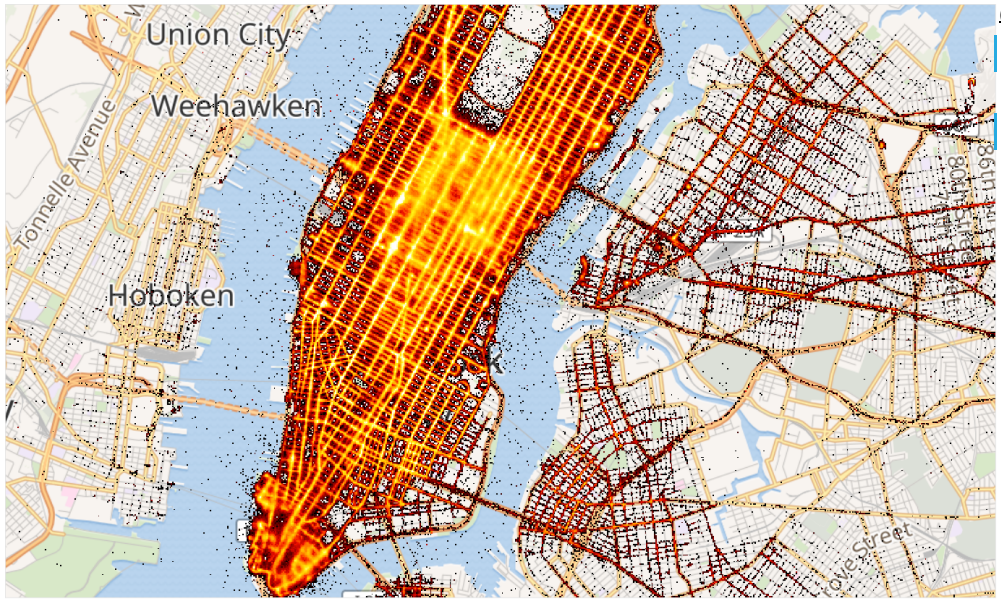
Out[36]:



(http://bokeh.pydata.org/)

In [30]:
```
taxi_trips = datashade(points, x_sampling=1, y_sampling=1, cmap=cm['fire']).opts(plot=options)
gv.WMTS(tiles['Wikipedia']) * taxi_trips
```

Out[30]:



(http://bokeh.pydata.org/)

Step 3: Declare your Parameters

Now that we've prototyped a nice plot in just a few lines of code, we want it to be widely shareable, with controls for safe and easy exploration.

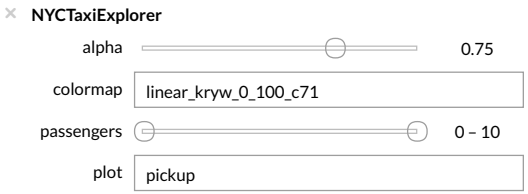
So the next step: declare what the intended user can change, with:

- type and range checking
- documentation strings
- default values

The Param library makes it simple to declare Python attributes having these features
(and more, such as dynamic values and inheritance).

In [42]:
```
class NYCTaxiExplorer(hv.streams.Stream):
    alpha       = param.Magnitude(default=0.75, doc="Alpha value for the map opacity")
    plot        = param.ObjectSelector(default="pickup", objects=["pickup","dropoff"])
    colormap    = param.ObjectSelector(default=cm["fire"], objects=cm.values())
    passengers  = param.Range(default=(0, 10), bounds=(0, 10), doc="""
        Filter for taxi trips by number of passengers""")
```

In [45]:
```
paramnb.Widgets(NYCTaxiExplorer)
```



Step 4: Link your Parameters to your data

Because the Parameters defined earlier are *about* a plot, it makes sense to combine the parameter and plotting declarations into a single object:

In [46]:

```
class NYCTaxiExplorer(hv.streams.Stream):
    alpha       = param.Magnitude(default=0.75, doc="Alpha value for the map opacity")
    colormap    = param.ObjectSelector(default=cm["fire"], objects=cm.values())
    plot        = param.ObjectSelector(default="pickup",   objects=["pickup","dropoff"])
    passengers  = param.Range(default=(0, 10), bounds=(0, 10))

    def make_view(self, x_range=None, y_range=None, **kwargs):
        map_tiles = gv.WMTS(tiles['Wikipedia']).opts(style=dict(alpha=self.alpha), plot=options)

        points = hv.Points(df, kdims=[self.plot+'_x', self.plot+'_y'], vdims=['passenger_count'])
        selected = points.select(passenger_count=self.passengers)
        taxi_trips = datashade(selected, x_sampling=1, y_sampling=1, cmap=self.colormap,
                               dynamic=False, x_range=x_range, y_range=y_range,
                               width=800, height=475)
        return map_tiles * taxi_trips
```

In [47]:

```
explorer = NYCTaxiExplorer()
paramnb.Widgets(explorer, callback=explorer.event)
hv.DynamicMap(explorer.make_view, streams=[explorer, RangeXY()])
```

NYCTaxiExplorer70193

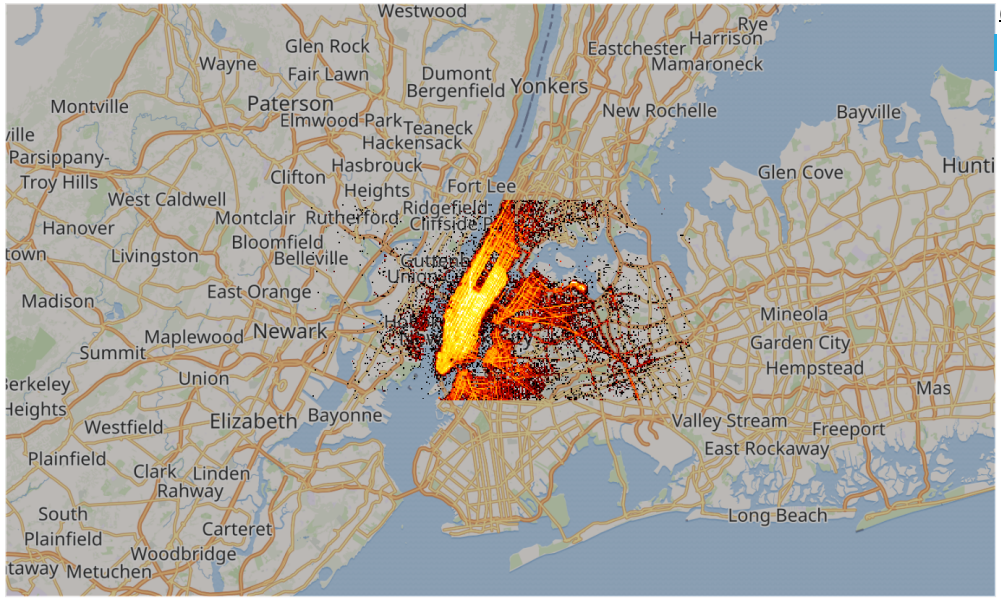
alpha 0.75

colormap linear_kryw_0_100_c71

passengers 0 – 10

plot pickup

Out[47]:



(http://bokeh.pydata.org/)

Step 5: Deploy your dashboard

Now *you* can see the visualization, but it requires a live Python process, which makes it difficult to share with people who don't use Python.

So, let's set up a web server that people can visit to explore the data themselves:

- If you used **ParamBokeh**, deploy with **Bokeh Server**:
    - Write the above code to a file `nyc_parambokeh.py`,
      switching to server mode when calling `Widgets`, which will return a bokeh `Document`
    - `bokeh serve nyc_parambokeh.py`
    - Send people a link to `http://<your-machine>:5006` (http://localhost:5006)

Full listing of the code required:

```
import holoviews as hv, geoviews as gv, param, parambokeh, dask.dataframe as dd

from colorcet import cm
from bokeh.models import WMTSTileSource
from holoviews.operation.datashader import datashade
from holoviews.streams import RangeXY, PlotSize

hv.extension('bokeh')

df = dd.read_parquet('./data/nyc_taxi.parq/').persist()
url='https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{Z}/{Y}/{X}.jpg'
tiles = gv.WMTS(WMTSTileSource(url=url))
tile_options = dict(width=1000,height=525,xaxis=None,yaxis=None,bgcolor='black',show_grid=False)

passenger_counts = (0, int(df.passenger_count.max().compute()+1))

class NYCTaxiExplorer(hv.streams.Stream):
    alpha      = param.Magnitude(default=0.75, doc="Alpha value for the map opacity")
    colormap   = param.ObjectSelector(default=cm["fire"], objects=[cm[k] for k in cm.keys() if not '_' in k])
    plot       = param.ObjectSelector(default="pickup",   objects=["pickup","dropoff"])
    passengers = param.Range(default=passenger_counts, bounds=passenger_counts)
    output     = parambokeh.view.Plot()

    def make_view(self, x_range, y_range, **kwargs):
        map_tiles = tiles(style=dict(alpha=self.alpha), plot=tile_options)
        points = hv.Points(df, kdims=[self.plot+'_x', self.plot+'_y'], vdims=['passenger_count'])
        if self.passengers != passenger_counts: points = points.select(passenger_count=self.passengers)
        taxi_trips = datashade(points, x_sampling=1, y_sampling=1, cmap=self.colormap,
                               dynamic=False, x_range=x_range, y_range=y_range, width=1000, height=525)
        return map_tiles * taxi_trips

selector = NYCTaxiExplorer(name="NYC Taxi Trips")
selector.output = hv.DynamicMap(selector.make_view, streams=[selector, RangeXY(), PlotSize()])

doc = parambokeh.Widgets(selector, view_position='right', callback=selector.event, mode='server')
```

Branching out

Much more ambitious apps are possible with surprisingly little additional code or effort:

- Adding additional linked or separate subplots of any type; see holoviews.org (http://holoviews.o
- Declaring code that runs for clicking or selecting *within* the Bokeh plot; see "streams" at holoviews.org (http://holoviews.org)
- Using multiple sets of widgets of many different types; see ParamNB (https://github.com/ioam/paramnb) and ParamBokeh (https://github.com/ioam/parambokeh)
- Using datasets too big for any one machine, with Dask.Distributed (https://distributed.readthedocs.io)

Future work

- Bokeh Server is mature and well supported, but does not currently support drag-and-drop layou
- Jupyter Dashboards Server does support drag-and-drop layout, but not currently maintained an requires older ipywidgets version
- ParamBokeh still needs some polishing and work to make it ready for widespread use; ParamNB more mature so far
- Both ParamNB and ParamBokeh need to provide more flexible widget layouts, page layouts
- Let us know what you would like to see out of these tools!

Join us on our Gitter channel or file issues!

Acknowledgements

- GeoViews development was partially funded by work with the UK MetOffice
- Thanks to Continuum Analytics for supporting FOSS
- Thanks to contributors

- Come find us at **holoviews.org** and **geoviews.org**