
# Building Analytics Dashboards using APIs you already know and love

Philipp Rudiger
Software Engineer
Anaconda Inc.

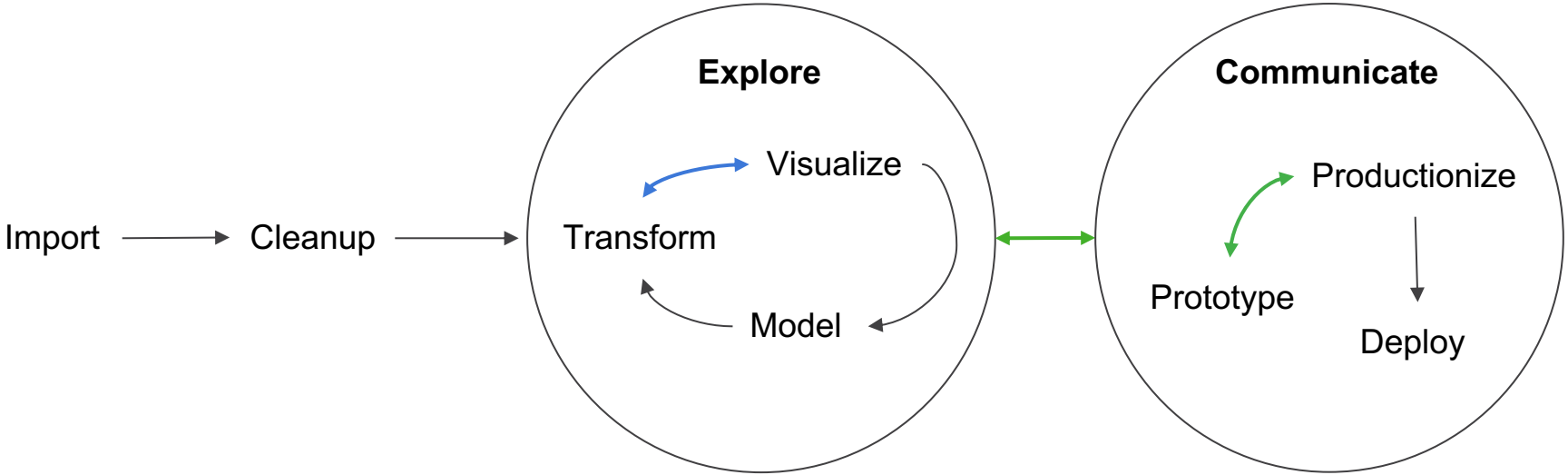
# About Me

Software Engineer at Anaconda Inc.

Develop open-source and client-specific solutions for data management, visualization and analysis.

Author of the open source dashboarding and visualization libraries Panel, hvPlot and GeoViews and one of the core developers of Bokeh and HoloViews.

# Data Science Workflows




# HoloViz

HoloViz provides a set of Python packages that make viz easier, more accurate, and more powerful:

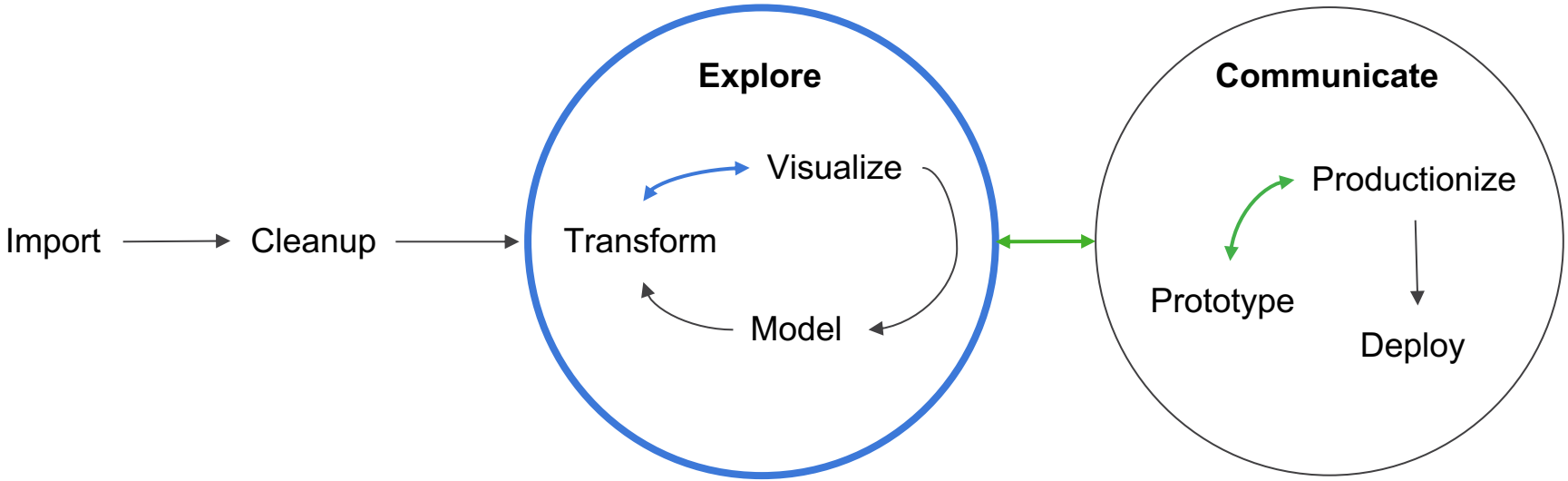
- Panel for making apps and dashboards for your plots from any supported plotting library
- hvPlot to quickly generate interactive plots from your data
- HoloViews to help you make all of your data instantly visualizable
- GeoViews to extend HoloViews for geographic data
- Datashader for rendering even the largest datasets



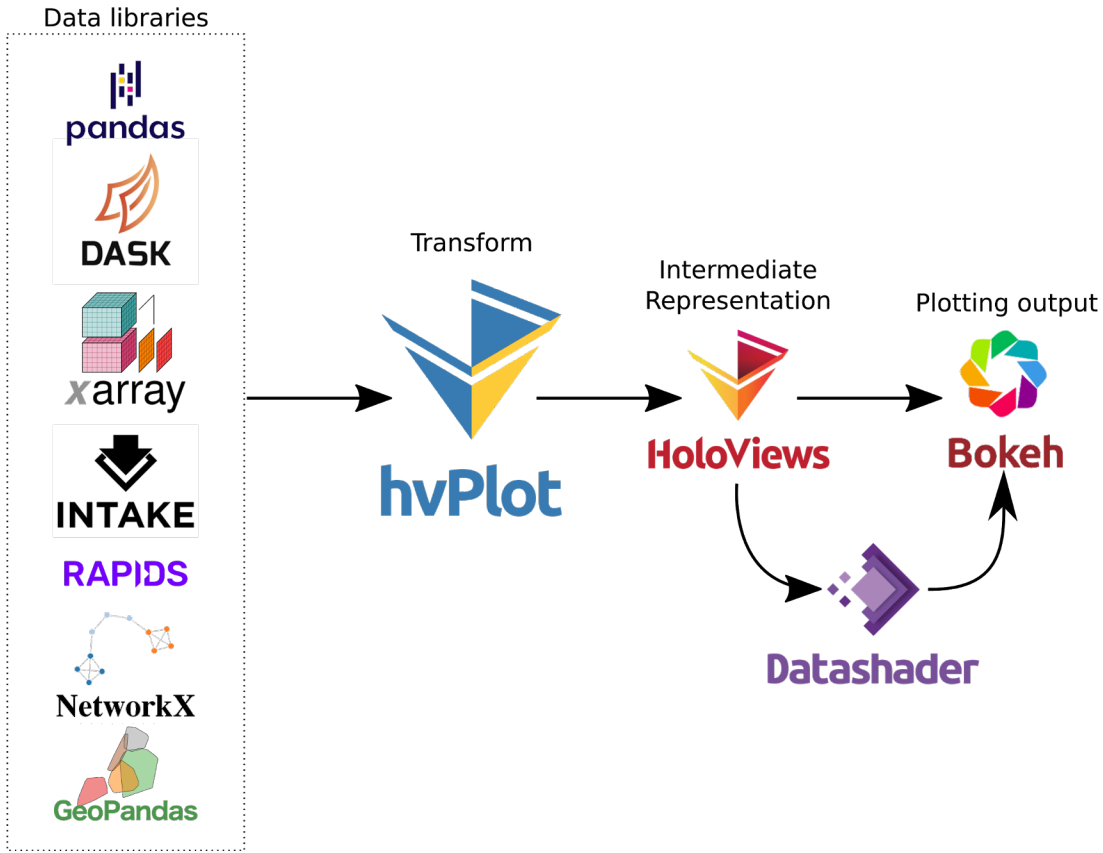




# Data Science Workflows



# Exploratory Visualization with hvPlot



# hvPlot: Tabular Data


- Familiar to users
- Widely supported
- Does not scale well


- Out-of-core and lazy execution
- Scales up and out


- Leverages GPUs for greater performance
- Newer and not yet as API complete

# hvPlot: Tabular Data

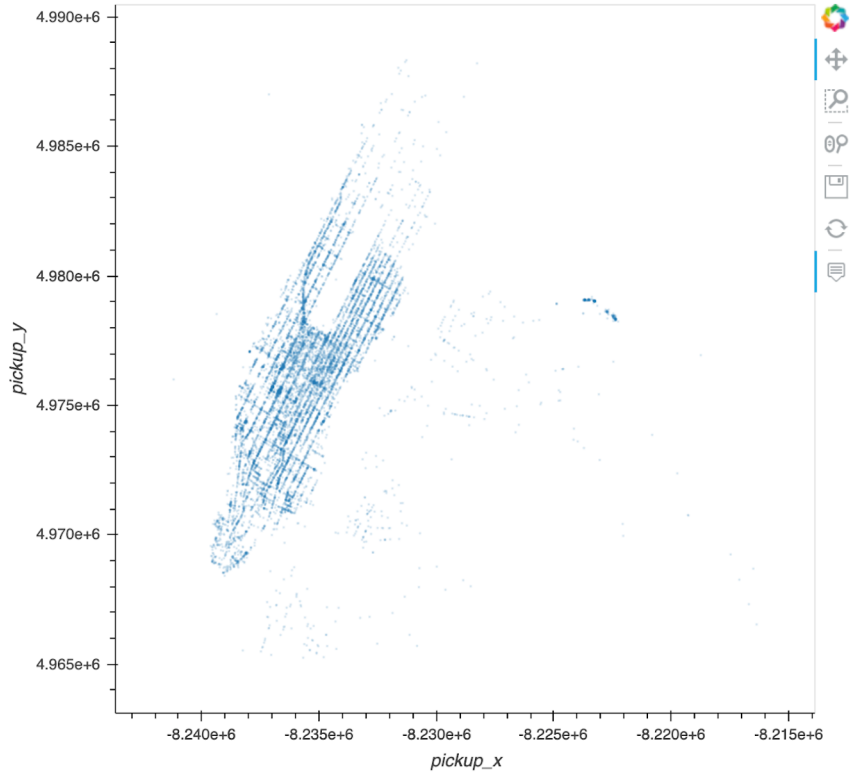
Load Data | Points | Lines | Datashader

```
filename = '/Users/philippjfr/development/datashader/examples/data/nyc_taxi_wide.parq'

data_backend = 'dask'

columns = ['pickup_x', 'pickup_y', 'dropoff_x', 'dropoff_y', 'trip_distance',
           'pickup_hour', 'dropoff_hour', 'fare_amount']

df = dd.read_parquet(filename, columns=columns)

if data_backend == 'dask':
    df = df.persist()
elif data_backend == 'pandas':
    df = df.compute()
elif data_backend == 'cuDF':
    df = df.compute()
    df['pickup_hour'] = df.pickup_hour.astype('i4')
    df['dropoff_hour'] = df.dropoff_hour.astype('i4')
    df = cudf.from_pandas(df)
```

# hvPlot: Tabular Data

Load Data | Points | Lines | Datashader

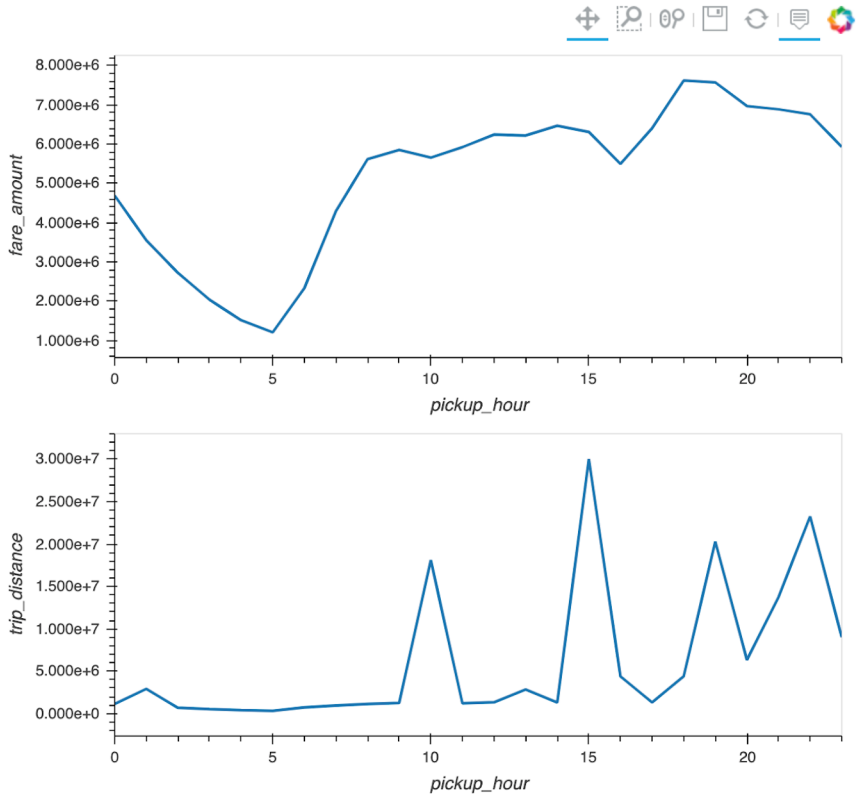
```
1  frac_df = df.sample(frac=0.001)
2
3  frac_df.hvplot.points(
4      'pickup_x', 'pickup_y', alpha=0.1, size=1, data_aspect=1
5  )
6
```



# hvPlot: Tabular Data

Load Data | Points | Lines | Datashader

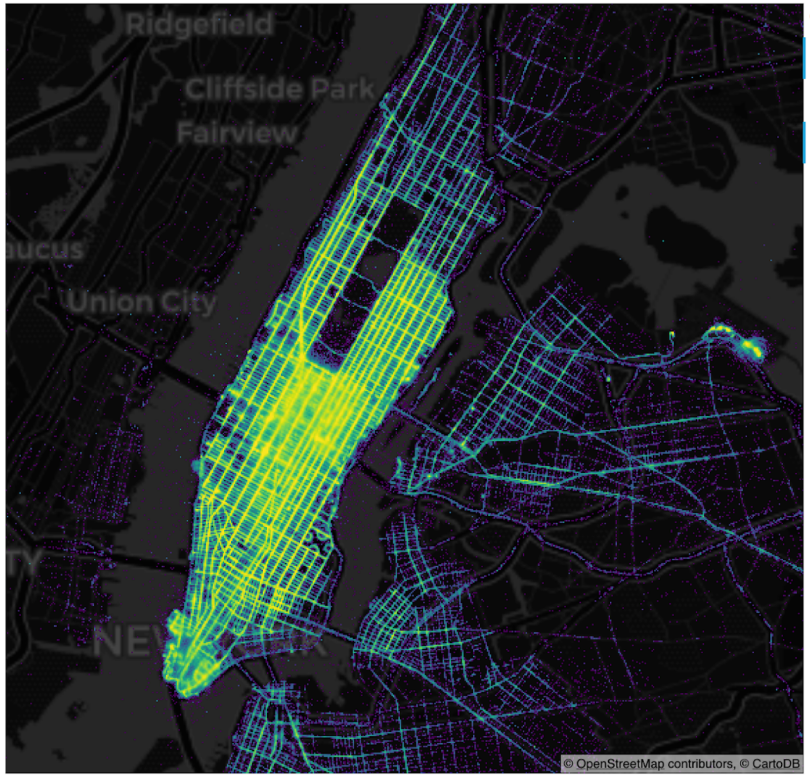
```
1
2  hourly_agg = df.groupby('pickup_hour').sum()
3
4  (hourly_agg.hvplot('pickup_hour', 'fare_amount') +
5   hourly_agg.hvplot('pickup_hour', 'trip_distance')
6  ).cols(1)
7
```



# hvPlot: Tabular Data

Load Data | Points | Lines | Datashader

```
df.hvplot.points(
    'pickup_x', 'pickup_y', datashade=True, tiles='CartoDark',
    cmap='viridis', xaxis=None, yaxis=None
)

```



# hvPlot: Geometries


- Widely used in the community
- Depends on geo-stack
- Slow (although significant work being done to improve this)

Spatialpandas

- Fast
- Built on Pandas extension arrays
- Not feature complete
- Supported by datashader

# hvPlot: Geometries

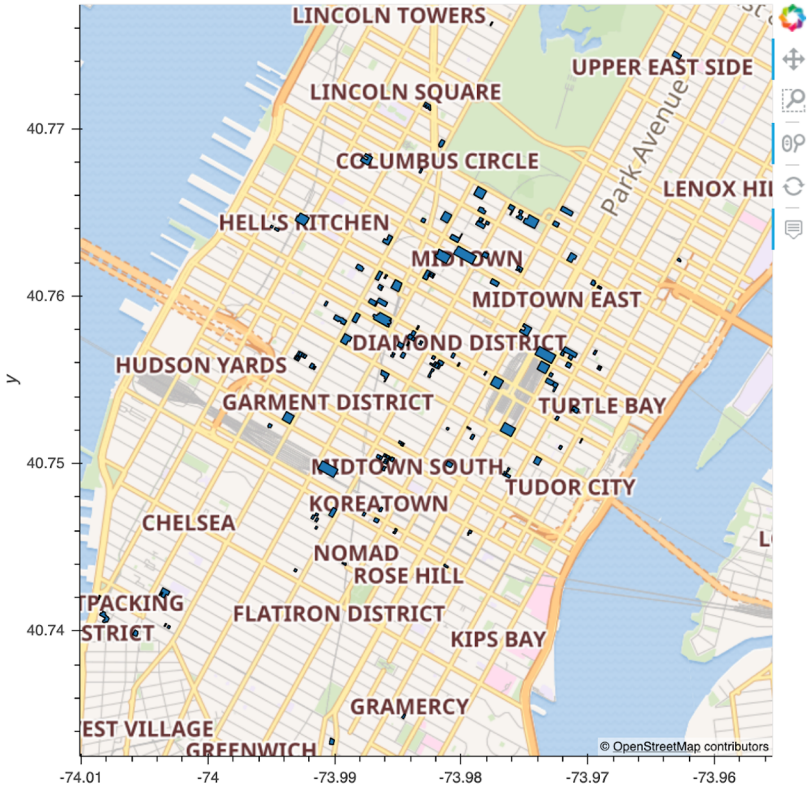
Load Data | Polygons | Datashaded

```
1
2  import geopandas as gpd
3  import spatialpandas as spd
4
5  data_backend = 'spatialpandas'
6
7  buildings = gpd.read_file('/Users/philippjfr/Downloads/NewYork-shp/shape/buildings.shp')
8  if data_backend == 'spatialpandas':
9      buildings = spd.GeoDataFrame(buildings)
10
```

# hvPlot: Geometries

Load Data | Polygons | Datashaded

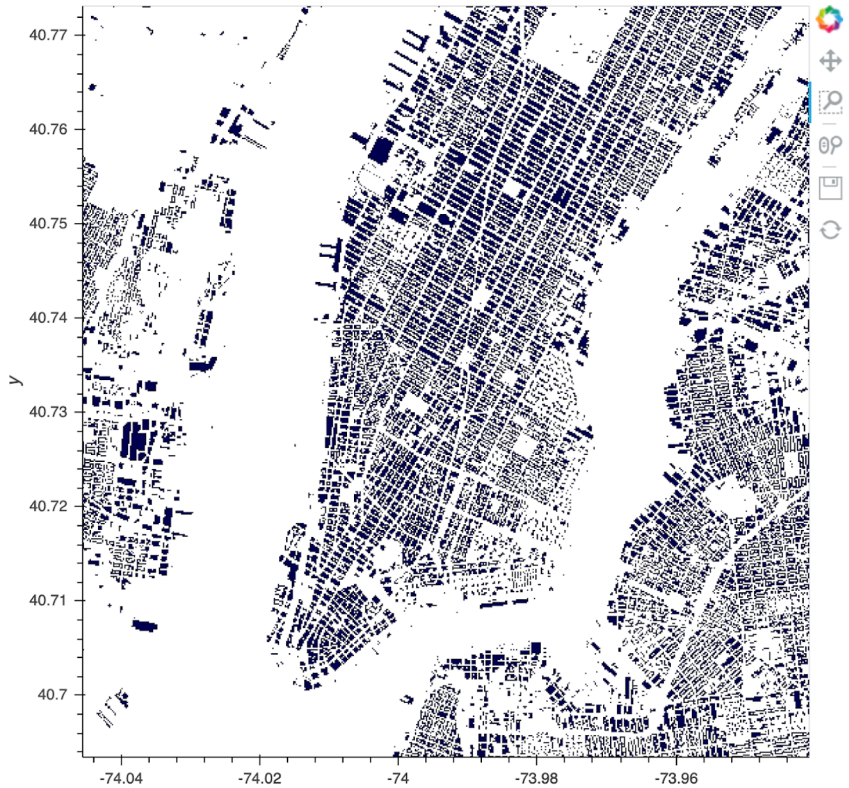
```
1  hotels = buildings[buildings['type'] == 'hotel']
2
3  hotels.hvplot.polygons(geo=True, tiles='Wikipedia')
```



# hvPlot: Geometries

Load Data | Polygons | Datashaded

```
1  buildings.hvplot(datashade=True, aggregator='any', responsive=True)
```



# hvPlot: Other datatypes

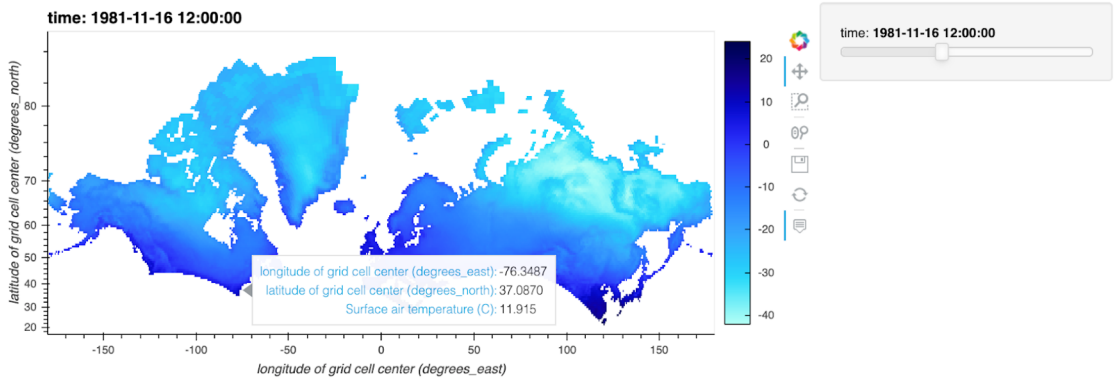

- Labelled multi-dimensional arrays (index with real values)
- Pandas for ND-arrays
- Supports Dask and CuPy arrays


- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms

# hvPlot: Gridded data

xarray | networkx

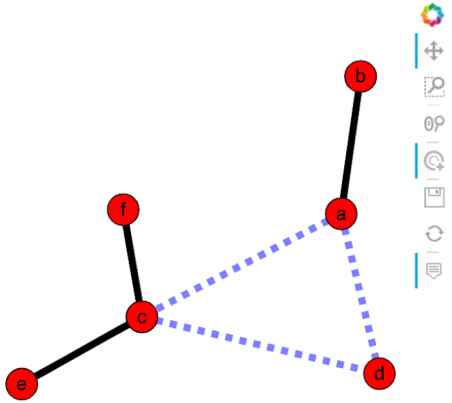
```
1
2  import xarray as xr
3  import hvplot.xarray
4
5  rasm = xr.tutorial.load_dataset('rasm')
6
7  rasm.hvplot.quadmesh('xc', 'yc', project=True, rasterize=True)
8
```



# hvPlot: Network data

xarray | networkx

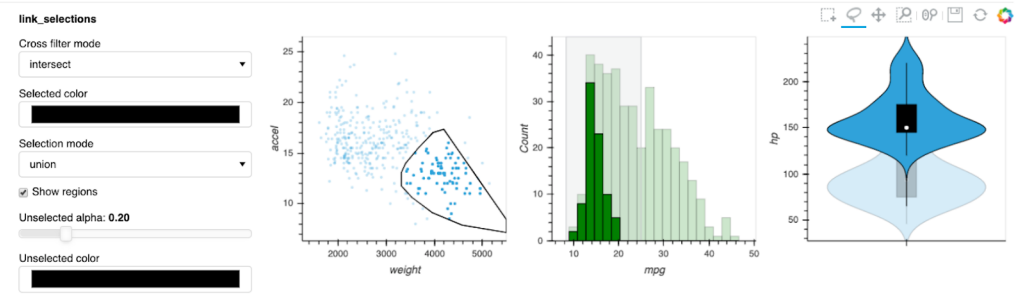
```python

# Author: Aric Hagberg (hagberg@lanl.gov)
# Adapted by Philipp Rudiger <prudiger@anaconda.com>

import networkx as nx
import hvplot.networkx as hvnx

G = nx.Graph()

G.add_edge('a', 'b', weight=0.6)
G.add_edge('a', 'c', weight=0.2)
G.add_edge('c', 'd', weight=0.1)
G.add_edge('c', 'e', weight=0.7)
G.add_edge('c', 'f', weight=0.9)
G.add_edge('a', 'd', weight=0.3)

elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d['weight'] > 0.5]
esmall = [(u, v) for (u, v, d) in G.edges(data=True) if d['weight'] <= 0.5]

pos = nx.spring_layout(G)  # positions for all nodes

# nodes
nodes = hvnx.draw_networkx_nodes(G, pos, node_size=700)

# edges
edges1 = hvnx.draw_networkx_edges(
    G, pos, edgelist=elarge, edge_width=6
)
edges2 = hvnx.draw_networkx_edges(
    G, pos, edgelist=esmall, edge_width=6, alpha=0.5, edge_color='blue', style='dashed'
)
labels = hvnx.draw_networkx_labels(G, pos, font_size=20, font_family='sans-serif')

edges1 * edges2 * nodes * labels

```



# Linked Brushing

Selection modes | NYC Example

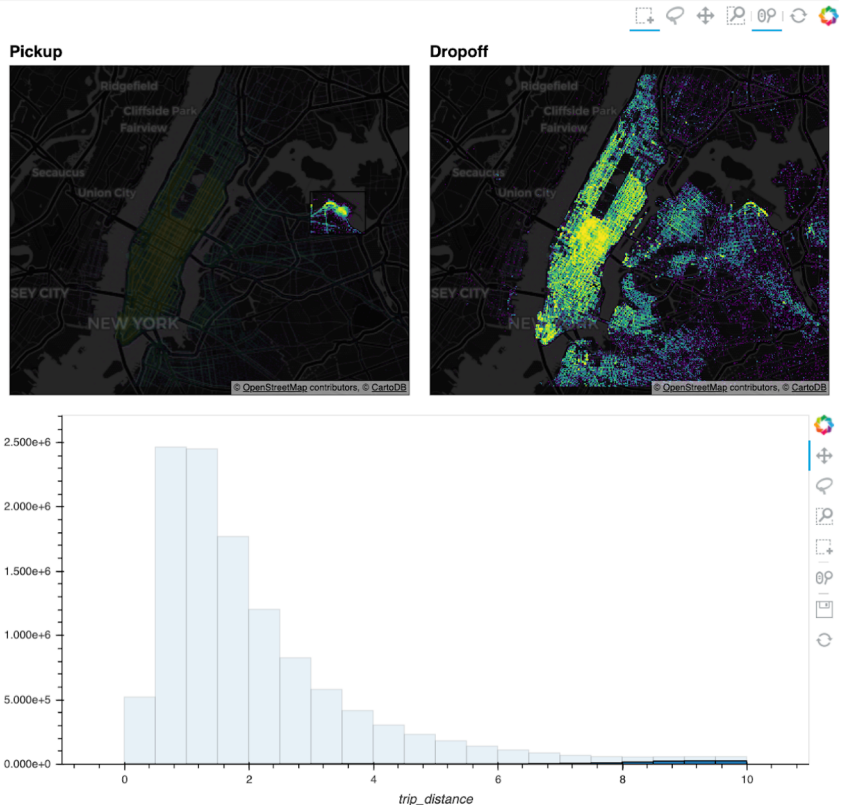
```
1
2  autompg_ds = hv.Dataset(autompg, ['yr', 'name', 'origin'])
3
4  w_accel_scatter = hv.Scatter(autompg_ds, 'weight', 'accel')
5  mpg_hist = histogram(autompg_ds, dimension='mpg', normed=False).opts(color="green")
6  violin = hv.Violin(autompg_ds, [], 'hp')
7
8  mpg_lsp = hv.link_selections.instance()
9
10 params = pn.Param(mpg_lsp, parameters=[
11     'cross_filter_mode', 'selection_mode', 'show_regions',
12     'selected_color', 'unselected_alpha', 'unselected_color'])
13
14 pn.Row(params, mpg_lsp(w_accel_scatter + mpg_hist + violin))
15
```



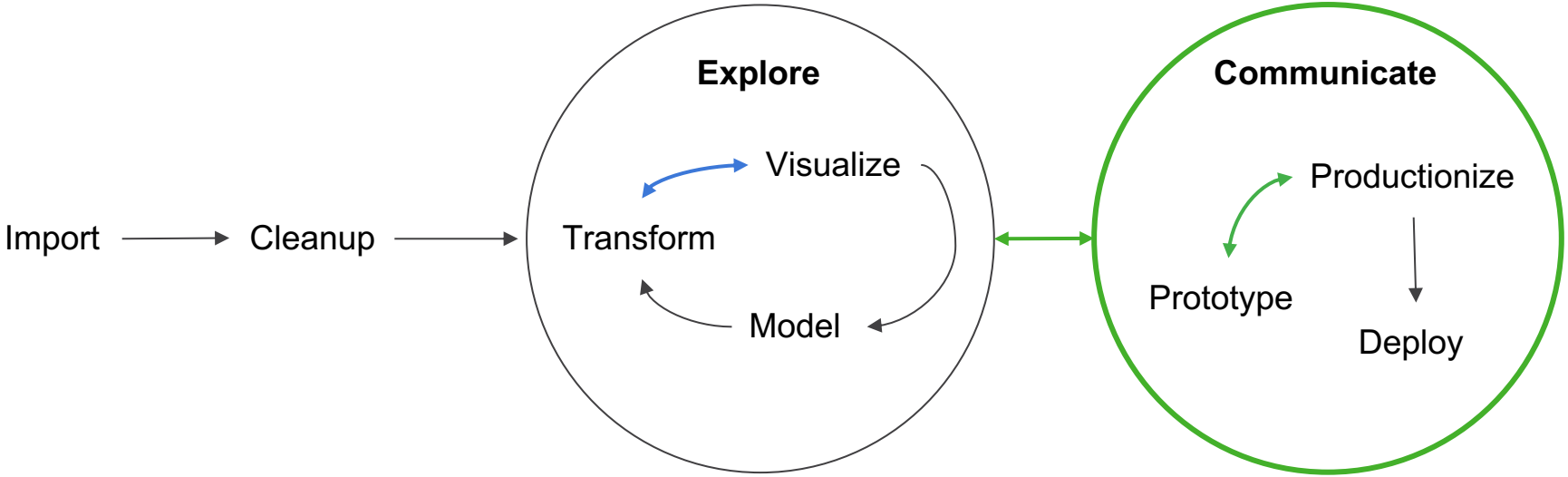
# Linked Brushing

Selection modes | NYC Example

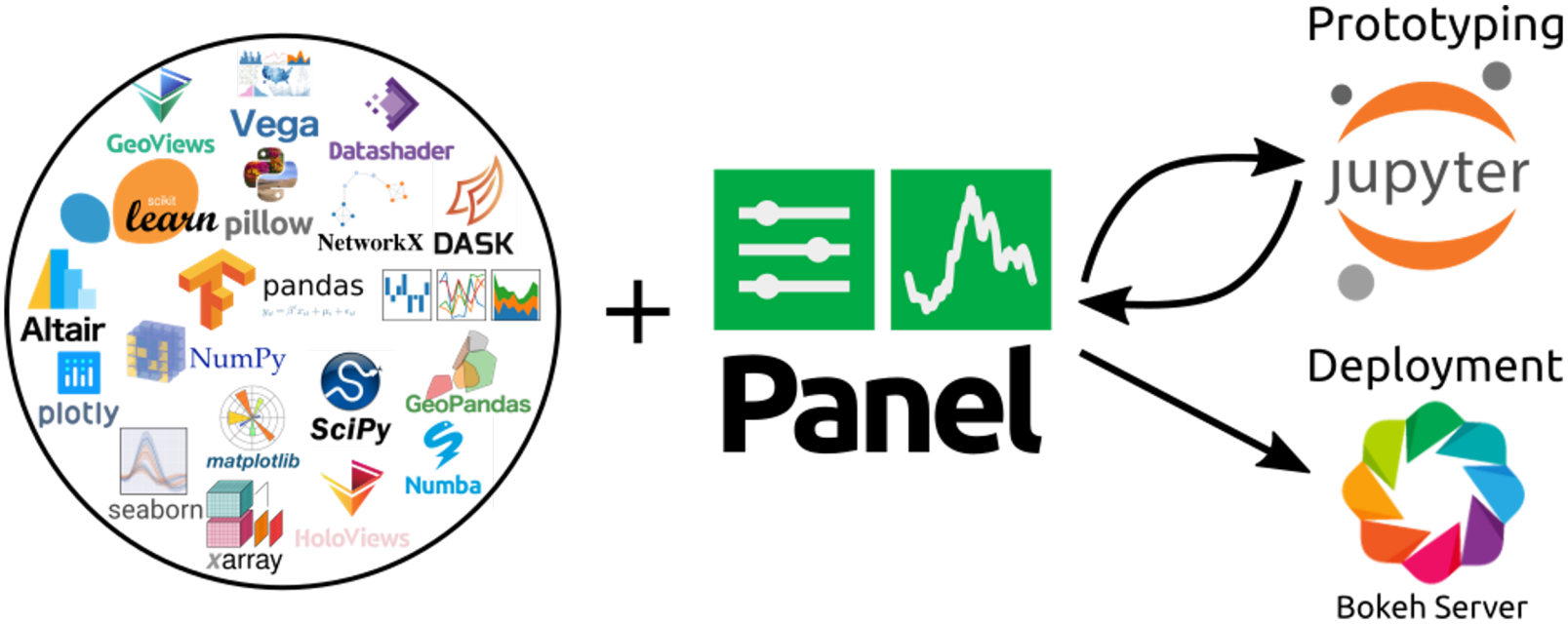
```
pickup = df.hvplot.scatter(
    'pickup_x', 'pickup_y', datashade=True, cmap='viridis',
    xaxis=None, yaxis=None, title='Pickup', responsive=True,
)
dropoff = df.hvplot.scatter(
    'dropoff_x', 'dropoff_y', datashade=True, responsive=True,
    cmap='viridis', xaxis=None, yaxis=None, title='Dropoff'
)
distance = df.hvplot.hist(
    'trip_distance', normed=False, bin_range=(0, 10),
    min_height=300, responsive=True
)

carto = hv.element.tiles.CartoDark()

ls = link_selections.instance()

pn.Column(
    carto * ls(pickup) + carto * ls(dropoff),
    ls(distance),
    sizing_mode='stretch_both'
)
```



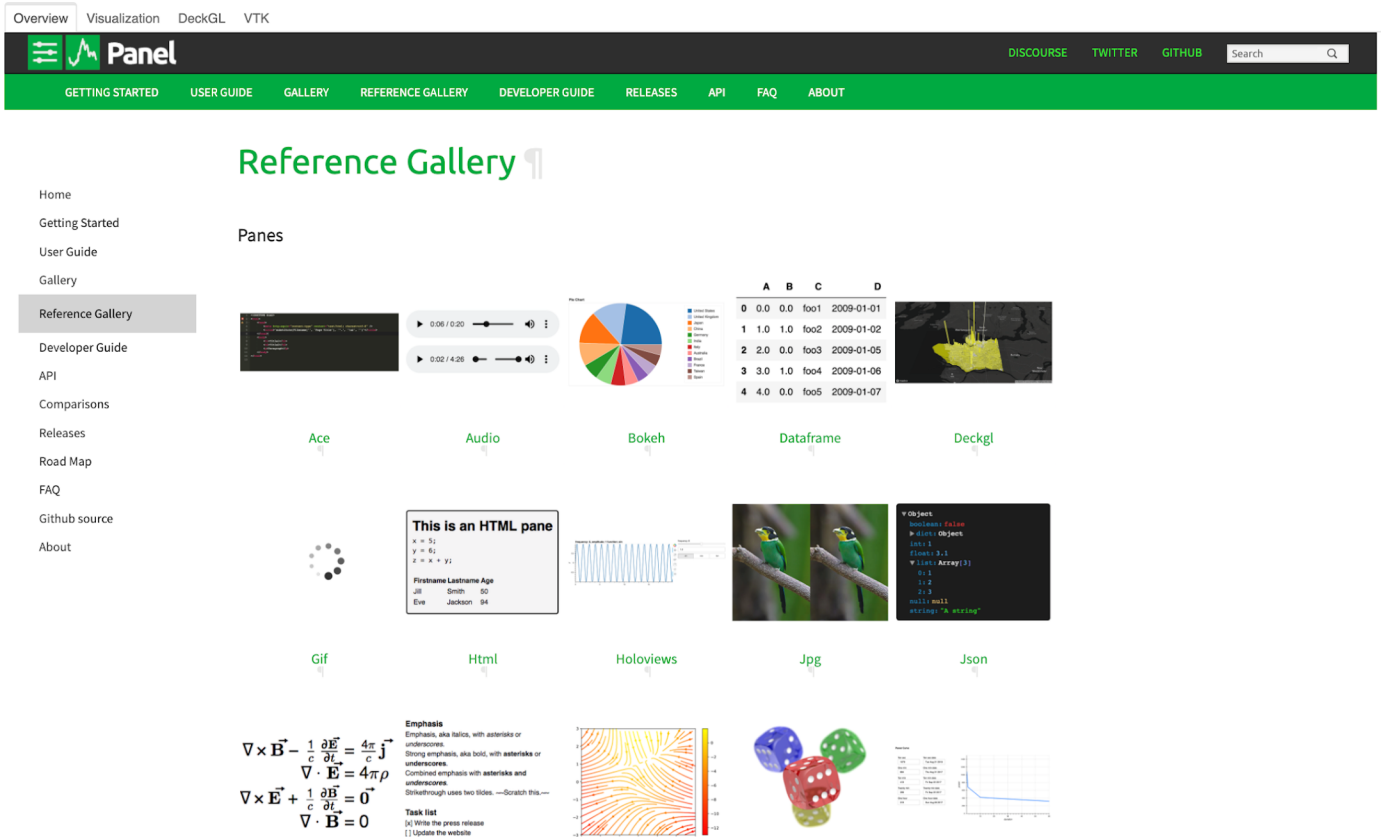
# Data Science Workflows



# Panel: Dashboards made easy



# Panel: Overview



# Panel: Visualization Libraries

Overview | Visualization | DeckGL | VTK

```python
class Gapminder(param.Parameterized):
    year = param.ObjectSelector(default=1952, objects=list(dataset.year.unique()))
    show_legend = param.Boolean(default=True)

    title = 'Life expectancy vs. GDP, %s'
    xlabel = 'GDP per capita (2000 dollars)'
    ylabel = 'Life expectancy (years)'
    ylim = (20, 90)
    xlim = (200, 15000)

    def get_data(self):
        df = dataset[(dataset.year==self.year) & (dataset.gdpPercap < 10000)].copy()
        df['size'] = np.sqrt(df['pop']*2.666051223553066e-05)
        return df

    def mpl_view(self):
        data = self.get_data()
        title = "Matplotlib: " + (self.title % self.year)

        plot = plt.figure(figsize=(7, 6))
        ax = plot.add_subplot(111)
        ax.set_xscale("log", nonposx='clip')
        ax.set_title(title)
        ax.set_xlabel(self.xlabel)
        ax.set_ylabel(self.ylabel)
        ax.set_ylim(self.ylim)
        ax.set_xlim(self.xlim)

        for continent, df in data.groupby('continent'):
            ax.scatter(df.gdpPercap, y=df.lifeExp, s=df['size']*5,
                       edgecolor='black', label=continent)

        if self.show_legend:
            ax.legend(loc=4)

        plt.close(plot)
        return plot

    def plotly_view(self):
        data = self.get_data()
        title = 'Plotly: ' + (self.title % self.year)

        traces = []
        for continent, df in data.groupby('continent'):
            marker=dict(symbol='circle', sizemode='area', sizeref=0.1, size=df['size'], line=dict(width=2))
            traces.append(go.Scatter(x=df.gdpPercap, y=df.lifeExp, mode='markers', marker=marker, name=continent))
```

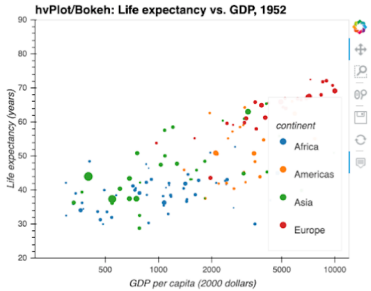
Panel

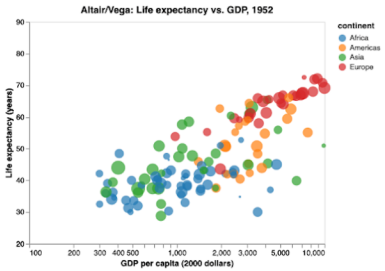
**Plotting library comparison**

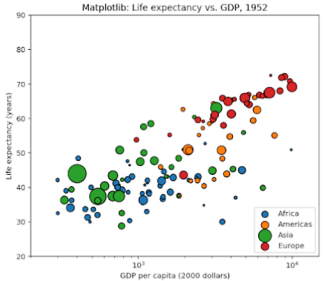
The Panel library from PyViz lets you make widget-controlled apps and dashboards from a wide variety of plotting libraries and data types. Here you can try out five different plotting libraries controlled by a couple of widgets, for Hans Rosling's gapminder example.

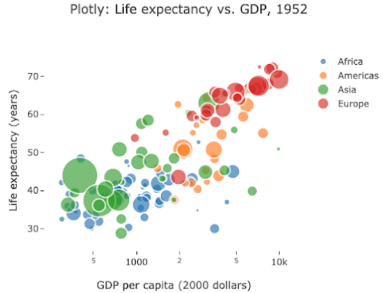
Year: **1952**

Show legend

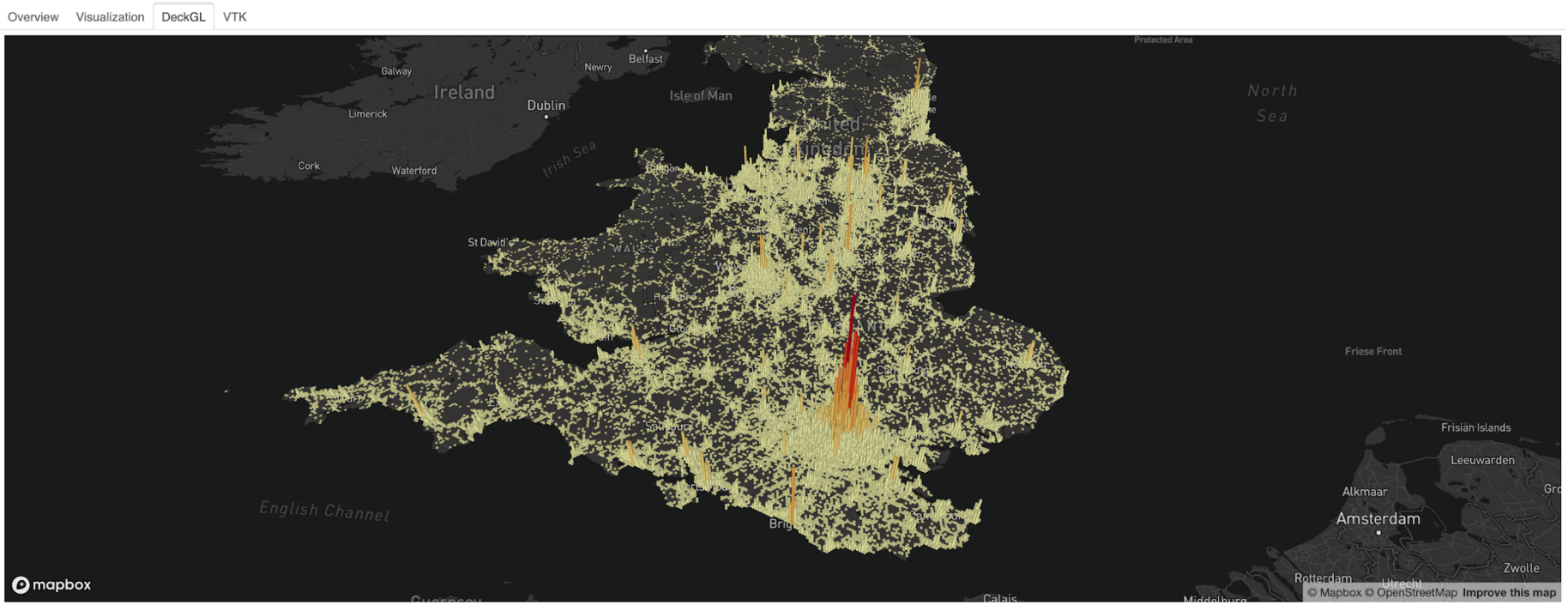


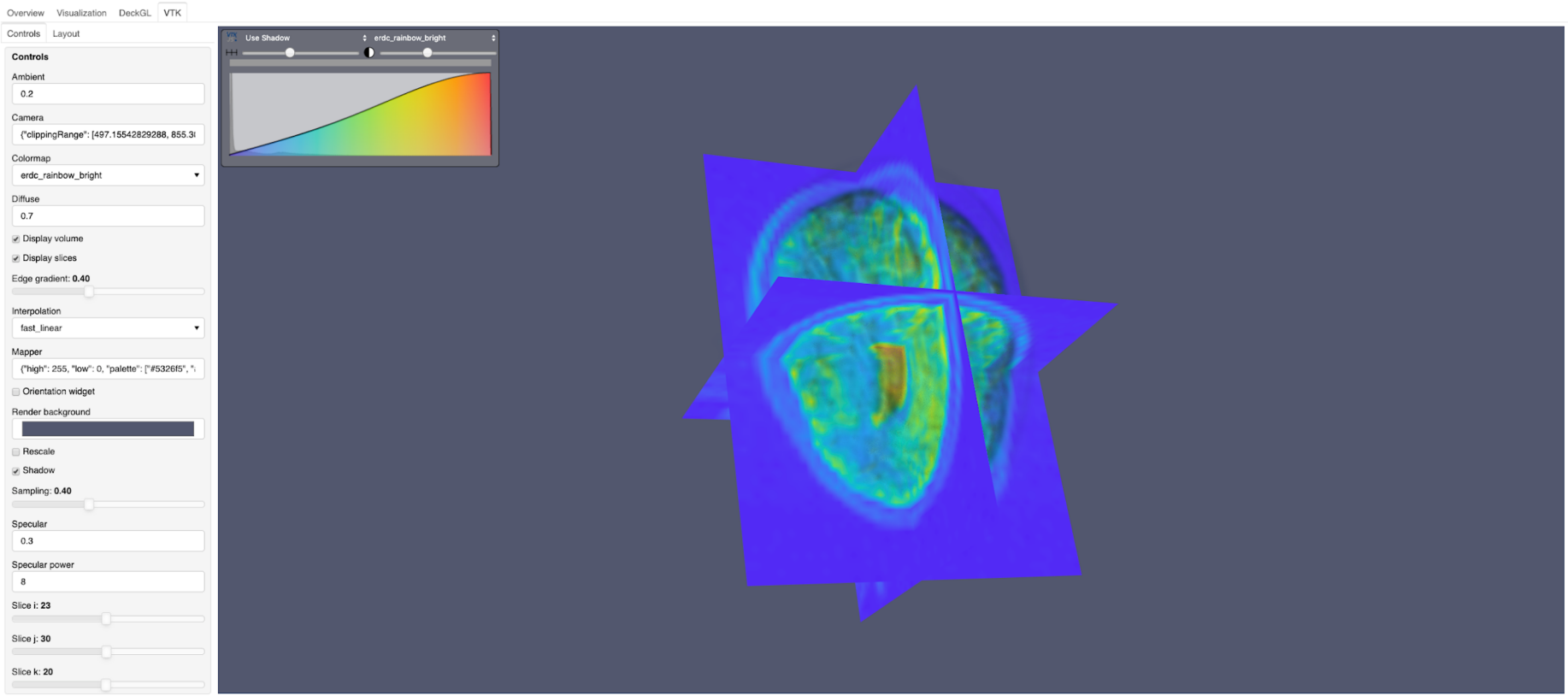



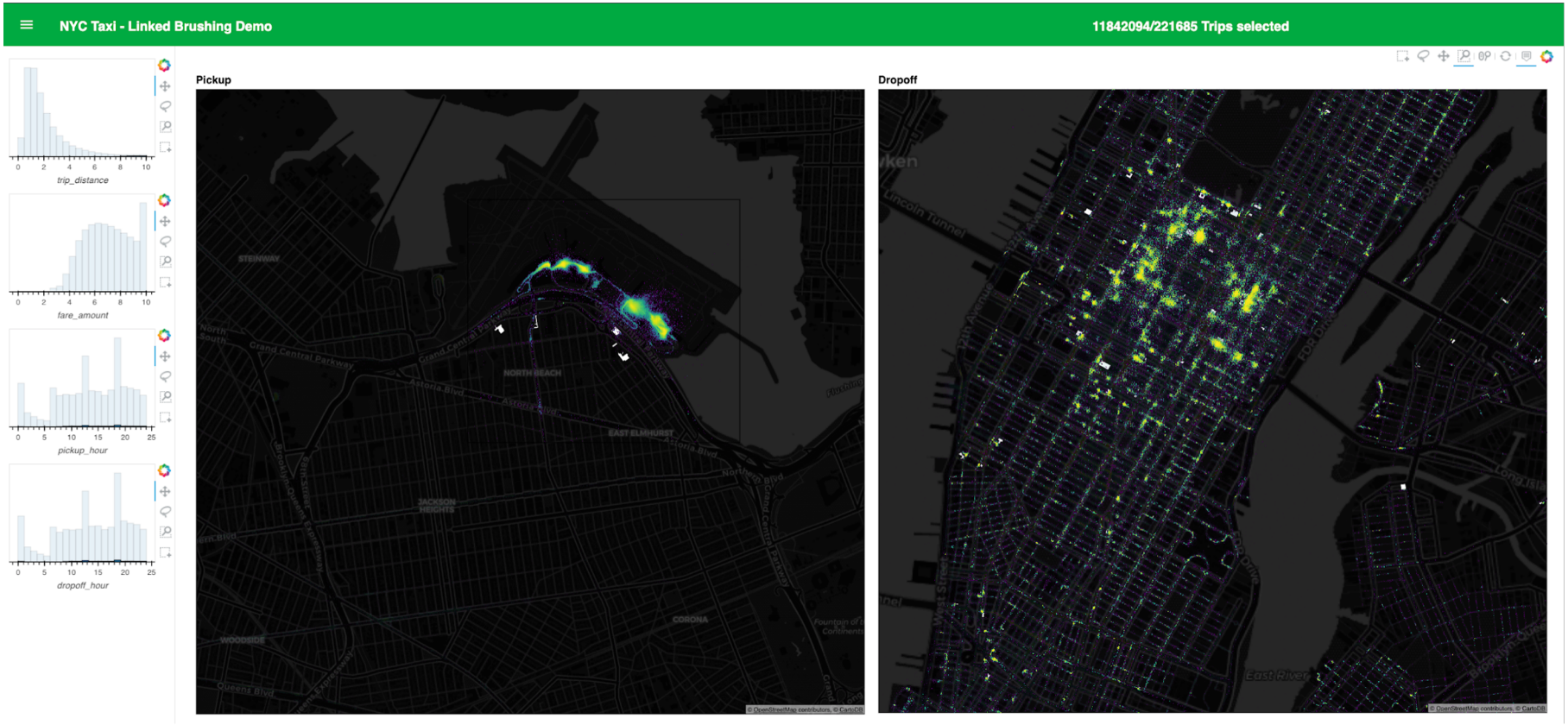
# Panel: Visualization Libraries



# Panel: Visualization Libraries



# Panel: Linked selection dashboard



# Panel: Linked selection dashboard

Code | Dashboard

```
ls = hv.link_selections.instance()

pn.config.sizing_mode = 'stretch_both'

opts = dict(datashade=True, cmap='viridis', xaxis=None, yaxis=None,
            responsive=True, shared_axes=False)

carto = hv.element.tiles.CartoDark()
hotel_polys = hotels.hvplot.polygons(data_aspect=1, geo=True, responsive=True, color='white')

pickup = carto * hotel_polys * ls(df.hvplot.scatter('pickup_x', 'pickup_y', title='Pickup', **opts)
dropoff = carto * hotel_polys * ls(df.hvplot.scatter('dropoff_x', 'dropoff_y', title='Dropoff',  **opts)

hist_ranges = {'trip_distance': (0, 10), 'fare_amount': (0, 10), 'pickup_hour': (0, 24), 'dropoff_hour': (0, 24)}
hists = pn.Column()
for value, bin_range in hist_ranges.items():
    hist = df.hvplot.hist(value, normed=False, bin_range=bin_range, yaxis=None, height=200, responsive=True)
    hists.append(ls(hist))

ds = hv.Dataset(df)
df_N = len(df)

@pn.depends(ls.param.selection_expr)
def count(expr):
    N = df_N if expr is None else len(df[expr.apply(ds)])
    return '## %d/%d Trips selected' % (df_N, N)

# Templating
tmpl = MaterialTemplate()

tmpl.header.append(pn.Row("## NYC Taxi - Linked Brushing Demo", pn.layout.HSpacer(), count))
tmpl.sidebar.append(hists)
tmpl.main.append(pn.Column((pickup + dropoff).opts(shared_axes=False), sizing_mode='stretch_both'))
```

# Summary: hvPlot + Panel

hvPlot

Panel

- Use the data libraries you are used to and generate plots with familiar *.plot* APIs:
  - Get interactive Bokeh plots
  - Explore large datasets with Datashader
  - Explore parameter spaces using Panel widgets
- Use HoloViews linked brushing support (basically for free) to gain insights into complex datasets
- Build dashboards using your favorite plotting libraries using Panel while taking advantage of:
  - Visually polished default templates
  - Scalable deployment
  - Wide and expanding range of visual components

## Visit holoviz.org

hvplot.holoviz.org
panel.holoviz.org
awesome-panel.org

*Thank you.*