Meaningful Histogramming with Physt

Pydata Berlin, 8th July 2018

Jan Pipek, Showmax

Who am I?

Who am I?

before: (particle / medical) physicist

Who am I?

before: (particle / medical) physicist

now: data scientist @ Showmax, Prague

Who am I?

before: (particle / medical) physicist

now: data scientist @ Showmax, Prague

janpipek

Histograms?


Bar chart

❌ bar chart (2 variables)


Bar chart 2

❌ bar chart (categorical variables)


KDE

KDE (similar)


Real histogram

Yes

Real histogram

Yes

Why histograms?

Why histograms?

- powerful for exploratory analysis

Why histograms?

- powerful for exploratory analysis
- precise and compact

Why histograms?

- powerful for exploratory analysis
- precise and compact
- appealing presentation

But…

numpy.histogram

```
import numpy as np
histogram = np.histogram(heights)
```

numpy.histogram

```
import numpy as np
histogram = np.histogram(heights)
```

```
(array([  4,  22,  96, 228, 272,          # Frequencies
        226, 104,  38,   9,   1]),


 array([132.1841, 141.0516, 149.9191,     # Edges
        158.7866, 167.6541, 176.5216,
        185.3891, 194.2566, 203.1241,
        211.9916, 220.8591])
```

numpy.histogram

```
import numpy as np
histogram = np.histogram(heights)
```

```
(array([  4,  22,  96, 228, 272,          # Frequencies
        226, 104,  38,   9,   1]),


 array([132.1841, 141.0516, 149.9191,     # Edges
        158.7866, 167.6541, 176.5216,
        185.3891, 194.2566, 203.1241,
        211.9916, 220.8591])
```

☹ Tuple of arrays?

matplotlib.pyplot.hist

```
import matplotlib.pyplot as plt
histogram = plt.hist(heights)
```

matplotlib.pyplot.hist

```
import matplotlib.pyplot as plt
histogram = plt.hist(heights)
```

matplotlib.pyplot.hist

```
import matplotlib.pyplot as plt
histogram = plt.hist(heights)
```

matplotlib.pyplot.hist

```
import matplotlib.pyplot as plt
histogram = plt.hist(heights)
```

```
(array([  4.,  22.,  96., 228., 272.,
        226., 104.,  38.,   9.,   1.]),

 array([132.1841, ..., 220.8591]),
 <a list of 10 Patch objects>)
```

matplotlib.pyplot.hist

```
import matplotlib.pyplot as plt
histogram = plt.hist(heights)
```

```
(array([  4.,  22.,  96., 228., 272.,
        226., 104.,  38.,   9.,   1.]),

 array([132.1841, ..., 220.8591]),
 <a list of 10 Patch objects>)
```

☹ Tuple of three arrays!

Histograms deserve more!

Histograms

- should be represented as objects
- should have nice bin edges
- should be dynamic
- should be easy to plot


physt

physt

physt

* March 2016

physt

* March 2016

† not yet

physt

* March 2016 † not yet

https://github.com/janpipek/physt

```
pip install physt
```

1) Object representation

2) Nice bins

3) Read-only?

4) Easy plottin′

physt.h1

```
from physt import h1
h1(data)
```

physt.h1

```
from physt import h1
h1(data)
```

```
Histogram1D(bins=(10,), total=1000, dtype=int64)
```

physt.h1

```
from physt import h1
h1(data)
```

```
Histogram1D(bins=(10,), total=1000, dtype=int64)
```

Class!

```
histogram.frequencies
```

```
histogram.frequencies
```

```
array([  4,  22,  96, 228, 272, 226,
       104,  38,   9,   1])
```

```
histogram.bins
```

```
histogram.bins
```

```
array([[-3.24126734, -2.53186746],
       [-2.53186746, -1.82246757],
       [-1.82246757, -1.11306769],
       [-1.11306769, -0.40366781],
       ...
       [ 1.72453184,  2.43393172],
       [ 2.43393172,  3.14333161],
       [ 3.14333161,  3.85273149]])
```

```
histogram.numpy_bins
```

```
histogram.numpy_bins
```

```
array([-3.24126734, -2.53186746, -1.82246757,
       -1.11306769, -0.40366781,  0.30573208,
        1.01513196,  1.72453184,  2.43393172,
        3.14333161,  3.85273149])
```

Meta data

```
histogram.name = "Heights"
histogram.title = "How tall are people in Fakeland?"
histogram.axis_name = "Height [cm]"
```

Meta data

```
histogram.name = "Heights"
histogram.title = "How tall are people in Fakeland?"
histogram.axis_name = "Height [cm]"
```

```
Histogram1D('Heights', bins=(10,), total=1000, dtype=int6

# histogram.meta_data
{
    'name': 'Heights',
    'axis_names': ('Height [cm]',),
    'title': 'How tall are people in Fakeland?'
}
```

```
{
    'name': 'Heights',
    'axis_names': ('Height [cm]',),
    'title': 'How tall are people in Fakeland?'
}
```

Serialization

```
histogram.to_json("histogram.json", indent=2)
```

Serialization

```
histogram.to_json("histogram.json", indent=2)
```

```
{
  "histogram_type": "Histogram1D",
  "binnings": [
    {
      "binning_type": "NumpyBinning",
      "numpy_bins": [
        -3.2412673400690726, ...,  -1.822467573924314,
      ]
    }
  ],
  "frequencies": [ 4, 22, 96, 228, 272, 226, 104, 38, 9, 1 ],
  "dtype": "int64",
  ...
  "physt_compatible": "0.3.20"
}
```

Mathematical entities?

□ + □ = ?

```
histogram_total = histogram1 + histogram2
```

□ * □ = ?

```
histogram_total = histogram1 * histogram2
```

```
histogram_total = histogram1 * histogram2
```

```
RuntimeError:
    Histograms may be multiplied only by a constant.
```

Supported operators

- addition (same or similar bins)
- subtraction (with warning)
- multiplication & division (by scalar)

Transformations

- normalization (total = 1)
- densities (scaled by bin width)
- cumulative frequencies (~CDF)

Multiple dimensions

```
heights = np.random.normal(172.7, 12.5, size=1000)

weights = (heights - 100 +
    np.random.normal(0, 12.5, size=1000))

iqs = np.random.normal(100, 15, 1000)
```

h2()

```
h2(heights, weights, axis_names=["height", "weight"])
```

h2()

```
h2(heights, weights, axis_names=["height", "weight"])
```

```
Histogram2D(bins=(10, 10), total=1000, dtype=int64)
```

2D heatmap

h3()

```
df = pd.DataFrame({
    "height": heights,
    "weight": weights,
    "iq": iqs
})
H3 = h3(df)    #  From pandas DataFrame
```

h3()

```
df = pd.DataFrame({
    "height": heights,
    "weight": weights,
    "iq": iqs
})
H3 = h3(df)    #  From pandas DataFrame
```

```
HistogramND(bins=(10, 10, 10), total=1000, dtype=int64)
```

h3()

```
df = pd.DataFrame({
    "height": heights,
    "weight": weights,
    "iq": iqs
})
H3 = h3(df)    #  From pandas DataFrame
```

```
HistogramND(bins=(10, 10, 10), total=1000, dtype=int64)
```

☹ No visualization*

As many dimensions as you want

As many dimensions as you want

 and have memory for

Projections

into lower dimensions

```
H3.projection("height", "weight")
```

```
H3.projection("height", "weight")
```

```
Histogram2D(bins=(10, 10), total=1000, dtype=int64)
```

```
H3.projection("height", "weight")
```

```
Histogram2D(bins=(10, 10), total=1000, dtype=int64)
```

Indexing

almost like numpy

```
H3[2]
```

```
H3[2]
```

```
Histogram2D(bins=(10, 10), total=96, dtype=int64)
```

```
H3[2]
```

```
Histogram2D(bins=(10, 10), total=96, dtype=int64)
```

```
H3[4, 5]
```

```
H3[4, 5]
```

```
Histogram1D(bins=(10,), total=49, dtype=int64)
```

```
H3[4, 5]
```

```
Histogram1D(bins=(10,), total=49, dtype=int64)
```

```
H3[2:5, :, 4:6]
```

```
H3[2:5, :, 4:6]
```

```
HistogramND(bins=(3, 10, 2), total=281, dtype=int64)
```

```
H3[4, 5, 6]
```

```
H3[4, 5, 6]
```

```
  (
    (167.6541, 176.5216), # Bin in "height"
    (82.7018, 94.7072),   # Bin in "weight"
    (117.2190, 127.6376)  # Bin in "iq"
  ),
  5                       # Frequency
```

1) Object representation

2) Nice bins, please!

3) Read-only?

4) Easy plottin'

```
h1(heights)                                    # Let numpy decide
```

```
histogram.binning
```

```
histogram.binning
```

```
NumpyBinning(
    array([132.1841, 141.0516, 149.9191, ...
           203.1241, 211.9916, 220.8591])
)
```

```
h1(heights, "fixed_width", 2.5)      # 2.5 cm each bin
```

```
h1(heights, "fixed_width", 2.5)      # 2.5 cm each bin
```


```
FixedWidthBinning(bin_width=2.5, bin_count=37, min=130.0)
```

```
h1(heights, "human", 15)     # => 19 bins, 5 cm each
```

```
h1(heights, "human", 15)      # => 19 bins, 5 cm each
```


```
FixedWidthBinning(bin_width=5.0, bin_count=19, min=130.0)
```

```
h1(heights, "human")          # => 10 bins, 10 cm each
```

```
h1(heights, "human")          # => 10 bins, 10 cm each
```


```
FixedWidthBinning(bin_width=5.0, bin_count=10, min=130.0)
```

```
h1(children)
```

```
h1(children, "integer")
```

```
h1(children, "integer")
```


```
FixedWidthBinning(bin_width=1.0, bin_count=7, min=-0.5)
```

Huge range

Country	Population
…	…
Malaysia	31,187,265
Maldives	427,756
Mali	17,994,837
Malta	437,418
…	…

```
h1(population)
```

```
h1(population)
```


```
NumpyBinning(array([1.1097e+04, 1.3787+08, ..., 1.3786+09
```

```
h1(population, "exponential")
```

```
h1(population, "exponential")
```


```
ExponentialBinning(array([1.109e+04, 4.085e+04, ..., 1.37
```

```
h1(population, "exponential", 12, range=(10000, 1e10))
```

```
h1(population, "exponential", 12, range=(10000, 1e10))
```


```
ExponentialBinning(array([1.000e+04, 3.162e+04, ..., 1.00
```

1) Object representation

2) Nice bins

3) Read-only?

4) Easy plottin’

1) Object representation

2) Nice bins

3) Read-only? No.

4) Easy plottin’

```
histogram = h1(heights, "fixed_width", 5)
```

I move to Fakeland...

```
histogram << 190
```

A group of 200 basketball players moves in...

```
newcomers = np.random.normal(220, 15, 200)
histogram.fill_n(newcomers)
```

```
histogram.adaptive = True
# or histogram = h1(heights, "fixed_width", 5, adaptive=T
histogram.fill_n(newcomers)
```

```
histogram1 = h1(heights, "fixed_width", 5, adaptive=True)
histogram2 = h1(newcomers, "fixed_width", 5, adaptive=Tru
histogram + histogram2
```

1) Object representation

2) Nice bins

3) Read-only?

4) Easy plottin'

1) Object representation

2) Nice bins

3) Read-only?

4) Easy plottin’

not tied to a specific plotting library!

Plotting backends

- matplotlib
- vega*
- plotly*
- folium (for geo data)

Plotting backends

- matplotlib
- vega*
- plotly*
- folium (for geo data)

...add yours!

```
histogram.plot.bar()
```

```
histogram.plot.step()
```

```
histogram.plot.scatter()
```

```
histogram.plot.line()
```

```
H.plot(show_values=True)
```

```
H.plot(show_values=True, color="#c0ffc0")
```

```
H.plot(show_values=True, color="#c0ffc0", lw=1)
```

```
H.plot(color="#c0ffc0", lw=1, errors=True)
```

```
H.plot(color="#c0ffc0", lw=1, errors=True, show_stats=Tru
```

```
histogram2.plot()
```

```
histogram2.plot(lw=0)
```

```
histogram2.plot(lw=0, show_values=True)
```

```
histogram2.plot(lw=0, show_values=True, cmap="rainbow")
```

```
histogram2.plot(lw=0, ..., show_zeros=False)
```

```
histogram2.plot(lw=0, ..., cmap_normalize="log")
```

Real-World Examples

Temperature during the day

Source: BrnoHacks 2017

Fluence of neutrons

Source: ELIMED project

Prague International Marathon

Source: RunCzech

Prague International Marathon

Source: RunCzech

Histograms

Histograms

- ✔ can be represented as objects

Histograms

- ✔ can be represented as objects
- ✔ can have nice bin edges

Histograms

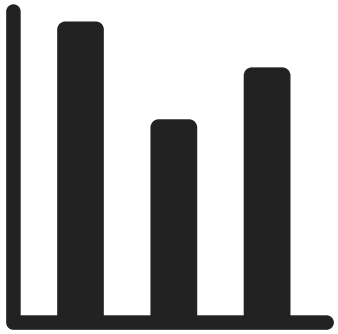
- ✔ can be represented as objects
- ✔ can have nice bin edges
- ✔ can be dynamic

Histograms

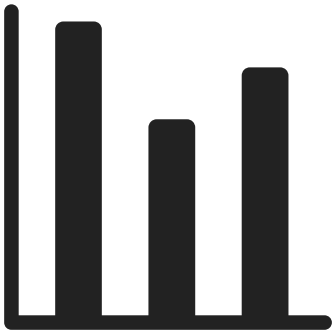
- ✔ can be represented as objects
- ✔ can have nice bin edges
- ✔ can be dynamic
- ✔ can be easy to plot

'\U0001f4ca'

'\U0001f4ca'



'\U0001f4ca'

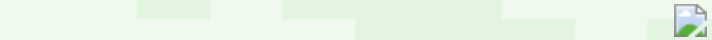


Bar chart!

janpipek/physt

janpipek/pydata2018-berlin

jan.pipek@gmail.com



Open positions in Prague