Lecture 3 - Model choice and inference

See https://github.com/jbrinchmann/MLD2025 as usual

Lectures/Lecture 3 has the PDF for today

No free lunch

There is a theorem (Wolpert 1996) which shows that when you average over all possible data-generating distributions, all classification algorithms have the same error rate when classifying new data.

Thus are all machine learning algorithms equally good?

In theory the answer is yes, in practice it is not!

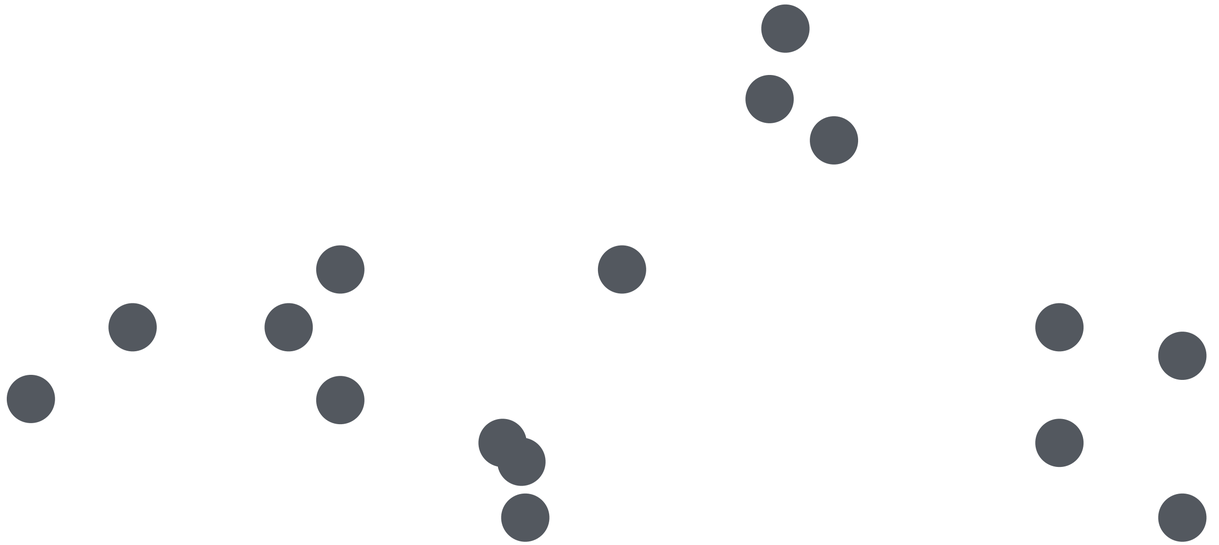
This is because in reality not all possible distributions show up - the world is not sadistic if you wish, thus an algorithm adapted to a particular problem can outperform others in practice.

See e.g. Goldblum et al (2023, https://arxiv.org/abs/2304.05366) for an optimistic view.

Regression -
keeping track of local properties

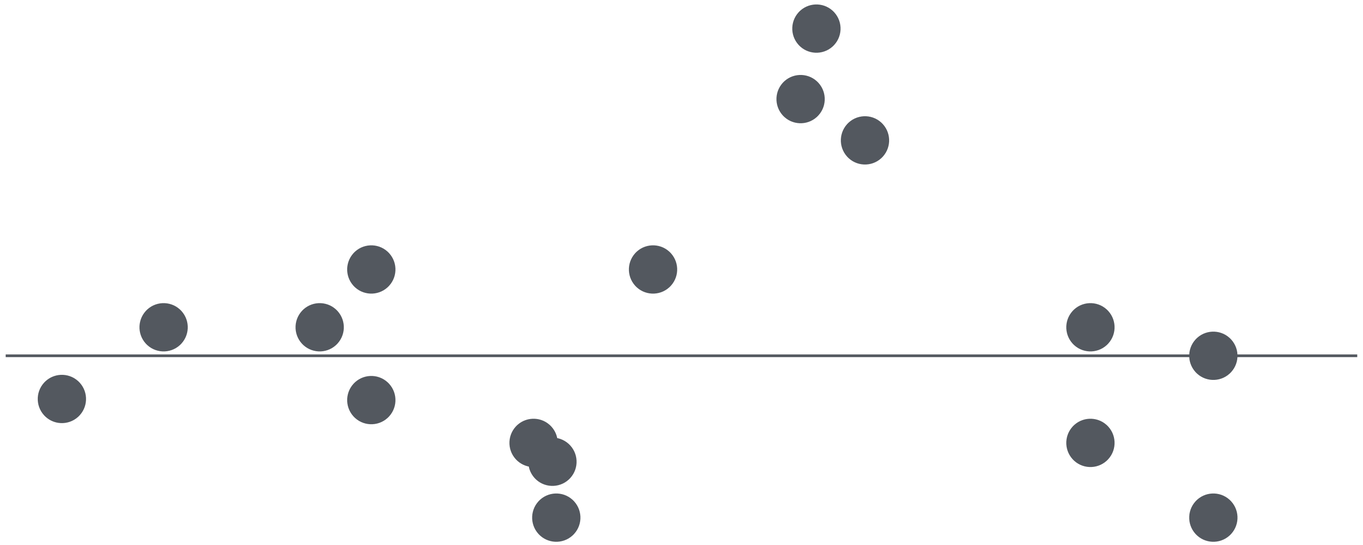
The broad idea

Faced with data where we do not have a clear idea how they are related a simple regression might be unsuitable:



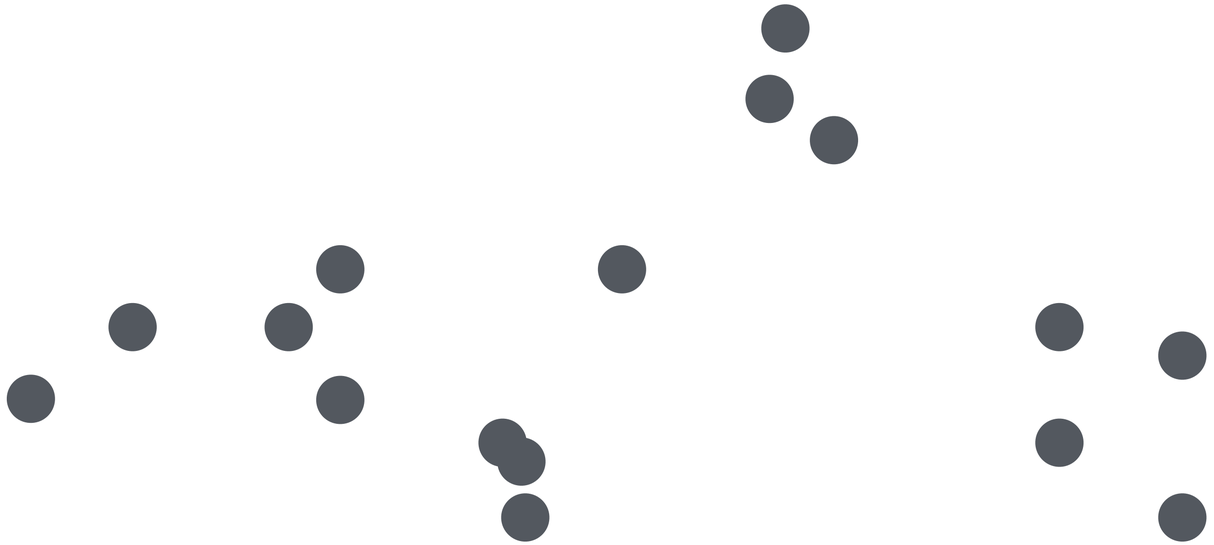
The broad idea

Faced with data where we do not have a clear idea how they are related a simple regression might be unsuitable:



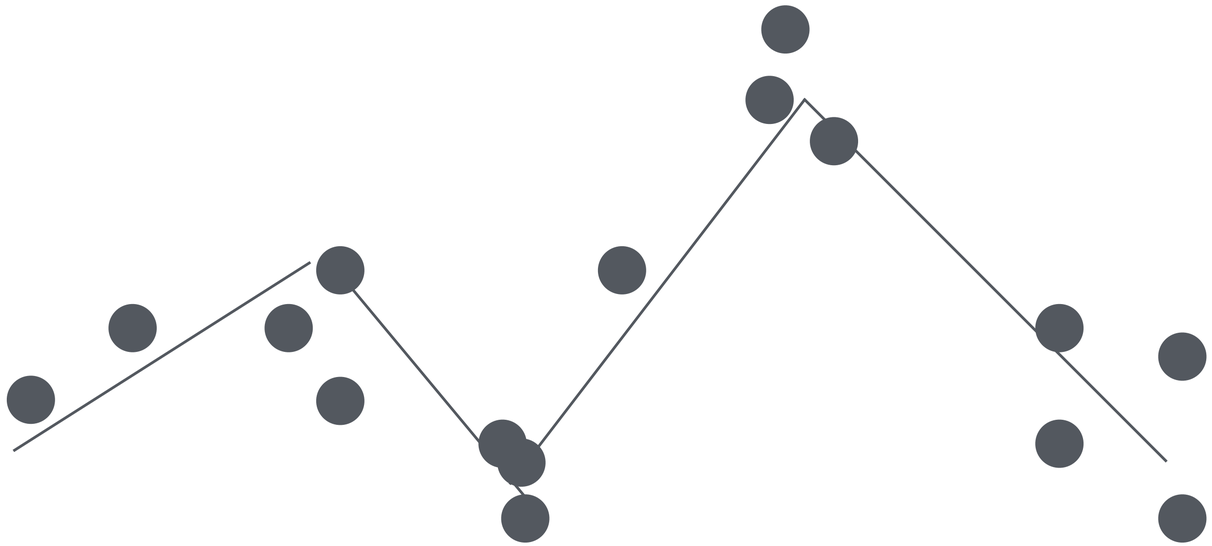
The broad idea

Keeping track of local trends might be a better idea:



The broad idea

Keeping track of local trends might be a better idea:



The main techniques

Nearest neighbour regression

Take the mean of the k nearest points

Kernel regression

Calculate the weighted mean of training points

Locally linear regression

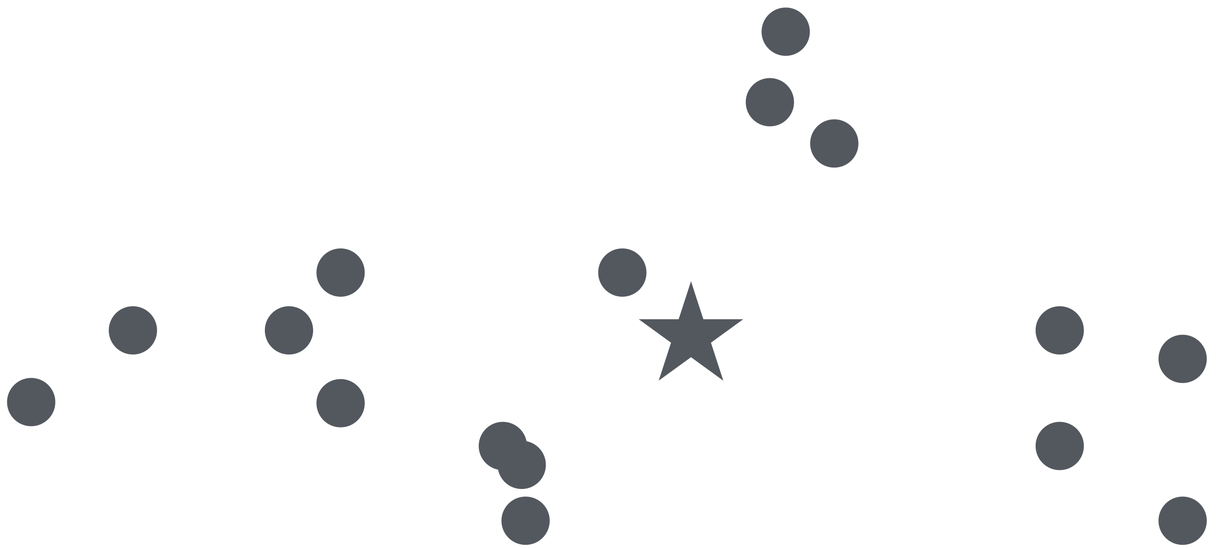
Calculate a weighted linear regression at each point

Gaussian process regression

Drop fixed functions and try to fit in the space of “all” functions

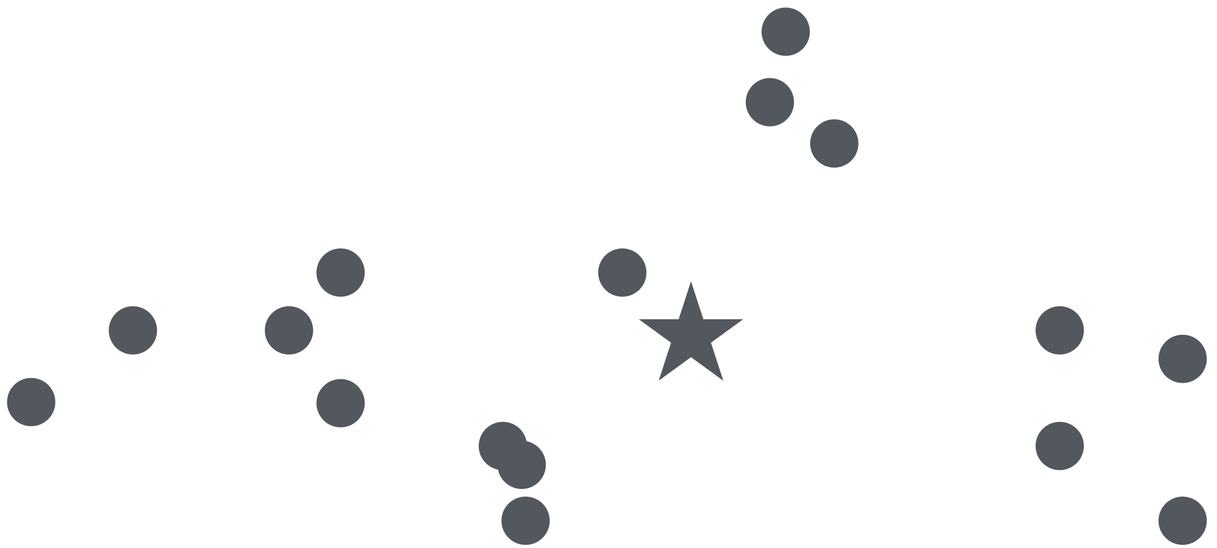
Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$



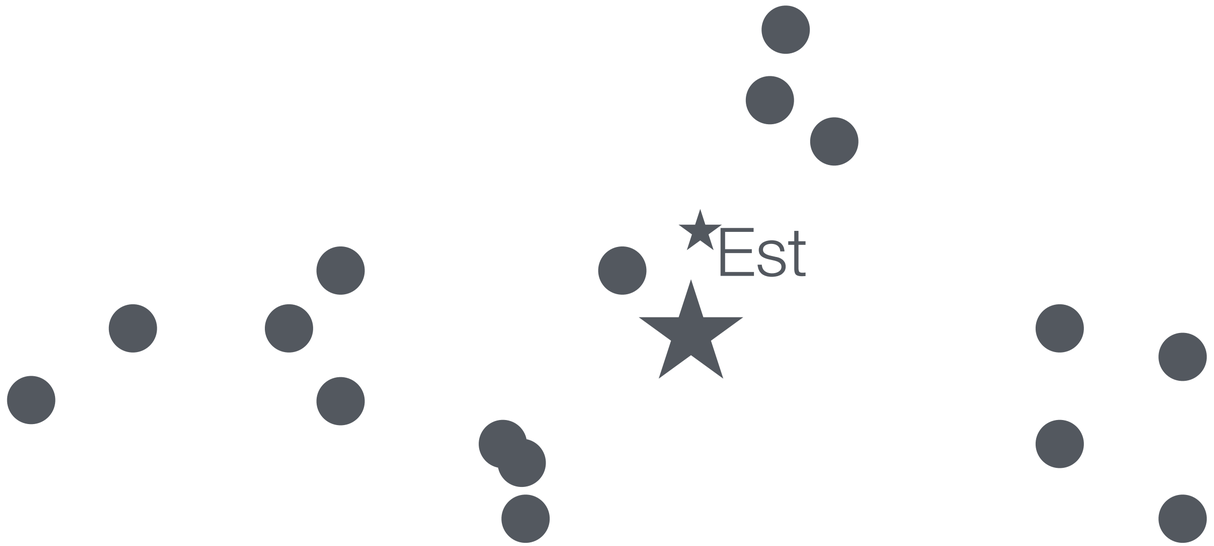
Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$



Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$



Nearest neighbour regression

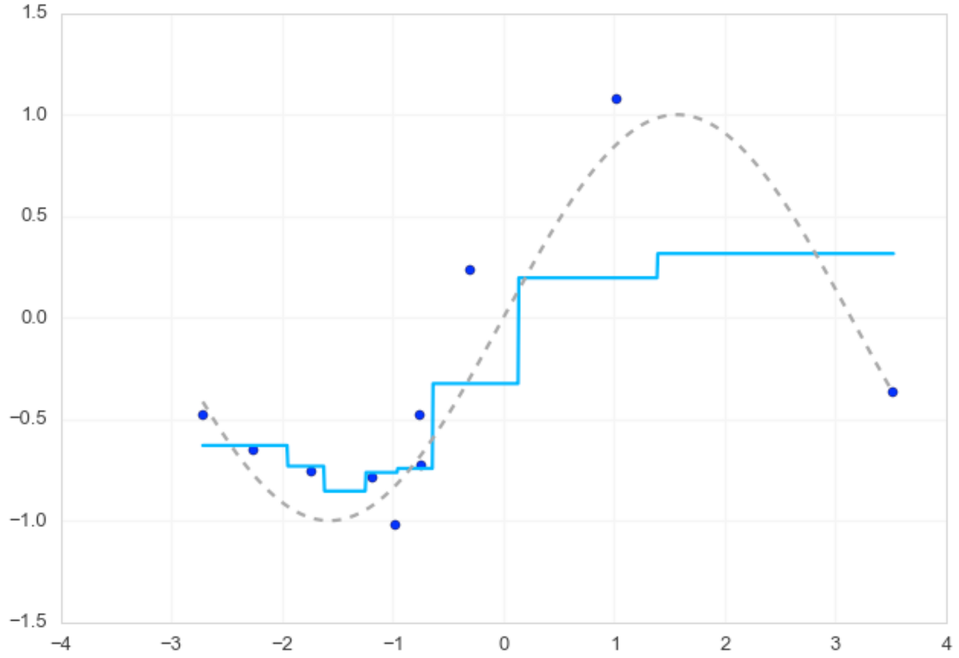
$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$

Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$

```
from sklearn import neighbors

k = 3
knn = neighbors.KNeighborsRegressor(k)
y_est = knn.fit(X, y).predict(Xplot)
```

Nearest neighbour regression




```
from sklearn import neighbors

k = 3
knn = neighbors.KNeighborsRegressor(k)
y_est = knn.fit(X, y).predict(Xplot)
```

Kernel regression

In knn regression we give equal weight to each point. If instead we give a variable weight we get kernel regression

$$\hat{y}(x) = \sum_{i=1}^{N} K_h(x, x_i)\, y_i$$

It is actually not necessary that the x_i are at the same place as y_i, but I will assume that they are. (if they are not you have to be careful with the normalisation of the basis functions)

h is a complexity parameter so needs to be determined by AIC/BIC or cross-validation for instance.

Kernel regression

The most common formulation of kernel regression renormalises the kernel functions to give the Nadaraya-Watson method:

$$\hat{y}(x) = \frac{\sum_{i=1}^{N} K_h\left(x, x_i\right) y_i}{\sum_{i=1}^{N} K_h\left(x, x_i\right)}$$

This comes from

$$\hat{y}(x) = E[Y|X = x] = \int y p(y|x) dy = \int y \frac{p(x,y)}{p(x)} dx$$

and inserting kernel density estimates for $p(x,y)$ and $p(x)$

Kernel regression

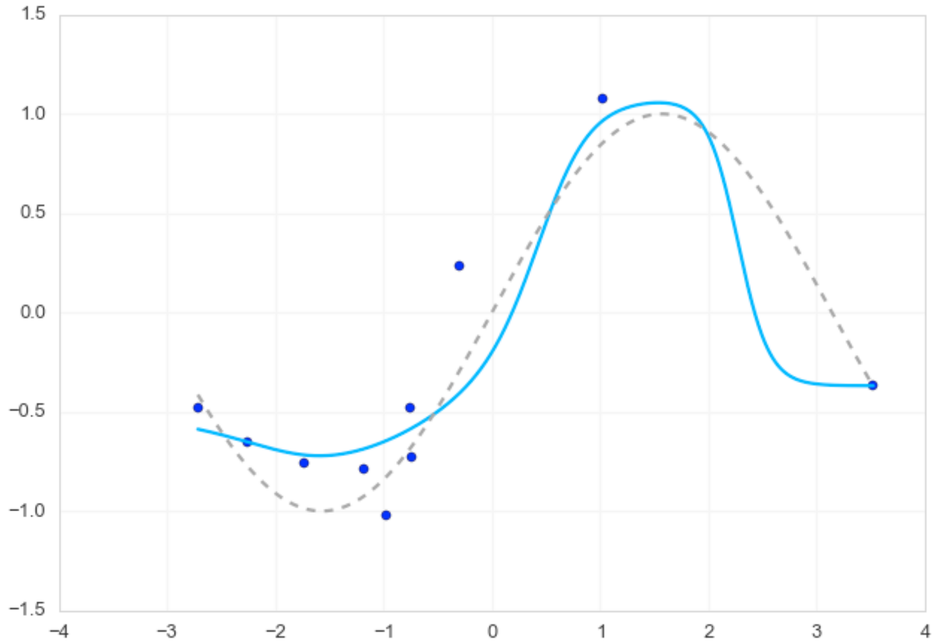
The most common formulation of kernel regression renormalises the kernel functions to give the Nadaraya-Watson method:

$$\hat{y}(x) = \frac{\sum_{i=1}^{N} K_h(x, x_i)\, y_i}{\sum_{i=1}^{N} K_h(x, x_i)}$$

```
from astroML.linear_model import NadarayaWatson


model = NadarayaWatson('gaussian', np.asarray(0.6))
model.fit(X, y)
y_est = model.predict(Xplot)
```

The kernels supported are those in `sklearn.metrics.pairwise`

Kernel regression




```
from astroML.linear_model import NadarayaWatson


model = NadarayaWatson('gaussian', np.asarray(0.6))
model.fit(X, y)
y_est = model.predict(Xplot)
```

Locally linear regression

In knn and kernel regression we effectively work with the zeroth level Taylor expansion - the constant term. The next step is to fit a weighted linear regression:

$$\theta_0(x), \theta_1(x) = \underset{\theta_0,\theta_1}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \theta_0 - \theta_1(x - x_i)\right)^2 K_h(x, x_i)$$

This turns out to be very useful in many situations and is often used as a powerful smoother under the name loess/lowess and a powerful package `locfit` is available in R (see rpy2)

h is a complexity parameter so needs to be determined by AIC/BIC or cross-validation for instance.

Locally linear regression

The weight/kernel is usually take to be the tri-cubic function:

$$w_i = (1 - |t|^3)^3 \; I(|t| \leq 1)$$

with $t = (x - x_i)/h$

Python packages:

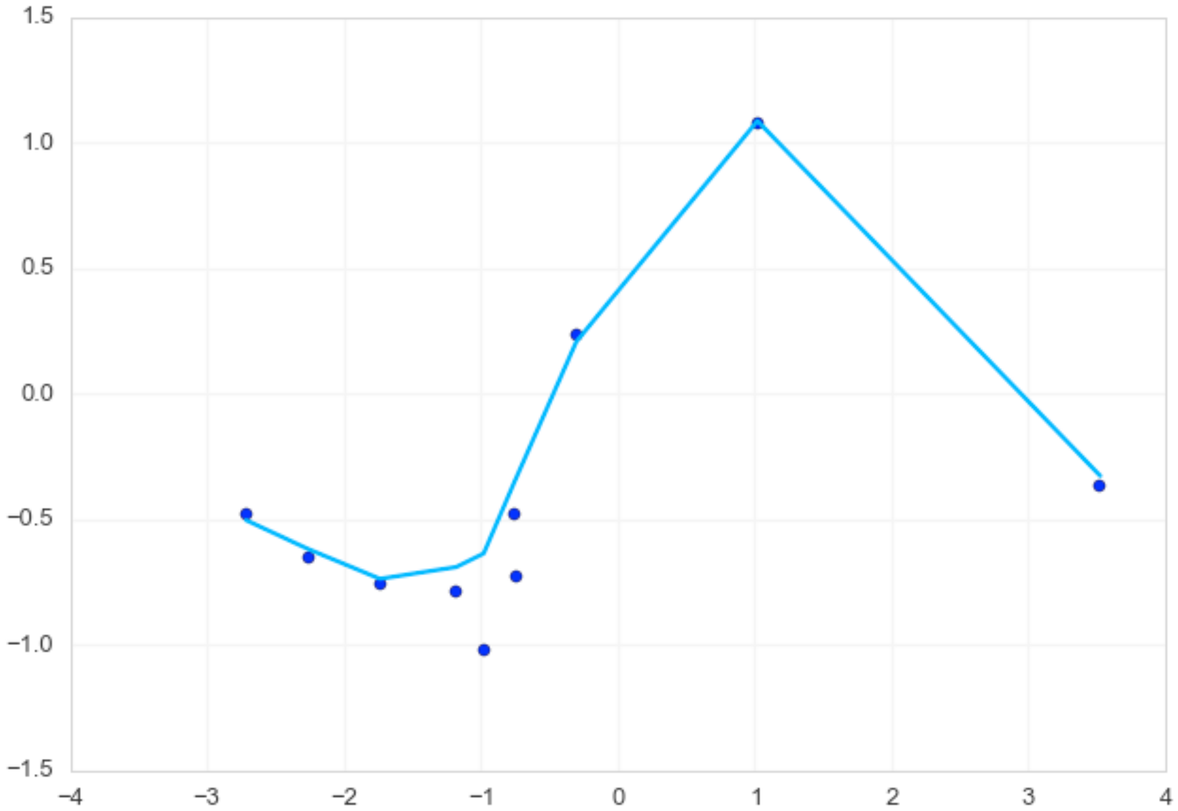
`statsmodels.nonparametric.smoothers_lowess.lowess`

`cylowess`

This is an area where R is better, but cylowess is decent.

Locally linear regression

```
import cylowess
c_lowess = cylowess.lowess

res_c = c_lowess(y,x)
plt.plot(res_c[:, 0], res_c[:, 1])
```



Gaussian process regression

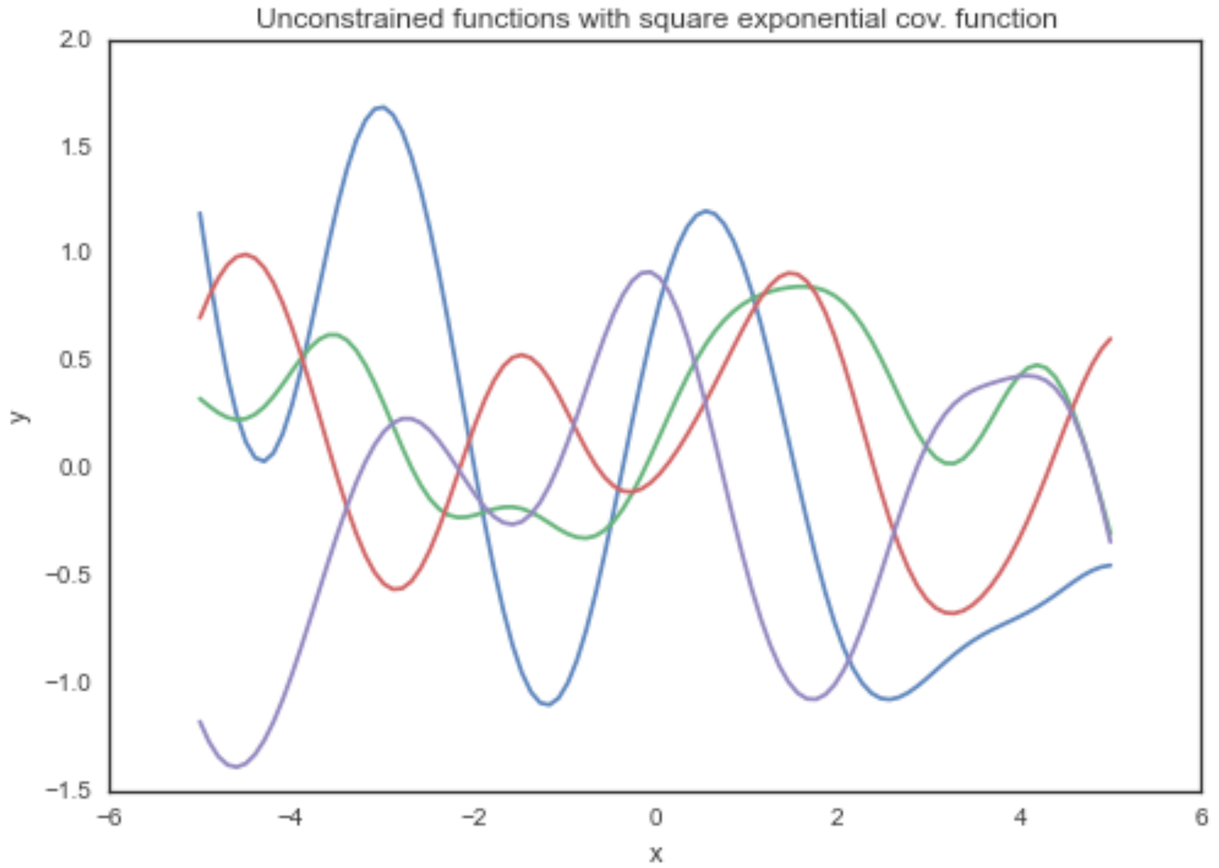
In this case we apply a prior in function space - this prior is specified using a mean & covariance function (since that is all we need for a Gaussian). The most common is:

$$\mathrm{Cov}\,(x, x') = K(x, x') = \exp\left(-\frac{|x - x'|}{2h}\right)$$

If we set the mean to zero, we can then draw random functions because at each x we know what the covariance matrix should be and that is all we need.

Gaussian process regression

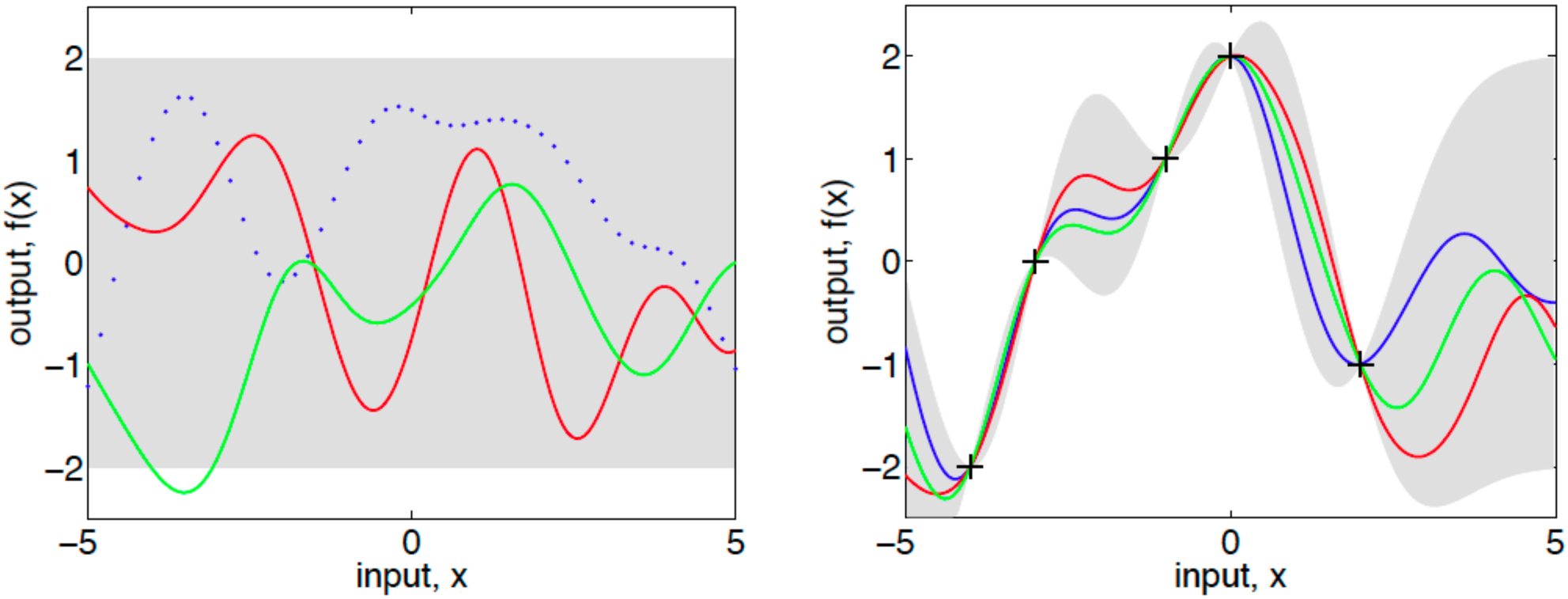
Random functions - h=1



See the Jupyter notebook on the Github site:
MLD2025-06a-Gaussian process regression

Gaussian process regression

We apply constraints by multiplying the prior with the likelihood:



(a), prior

(b), posterior

Taken from Rasmussen & Williams, “Gaussian processes for Machine Learning”, 2006, Figure 2.2
http://www.gaussianprocess.org/

Gaussian Process Regression - features

+ Very flexible
+ Provides covariance estimates on predictions

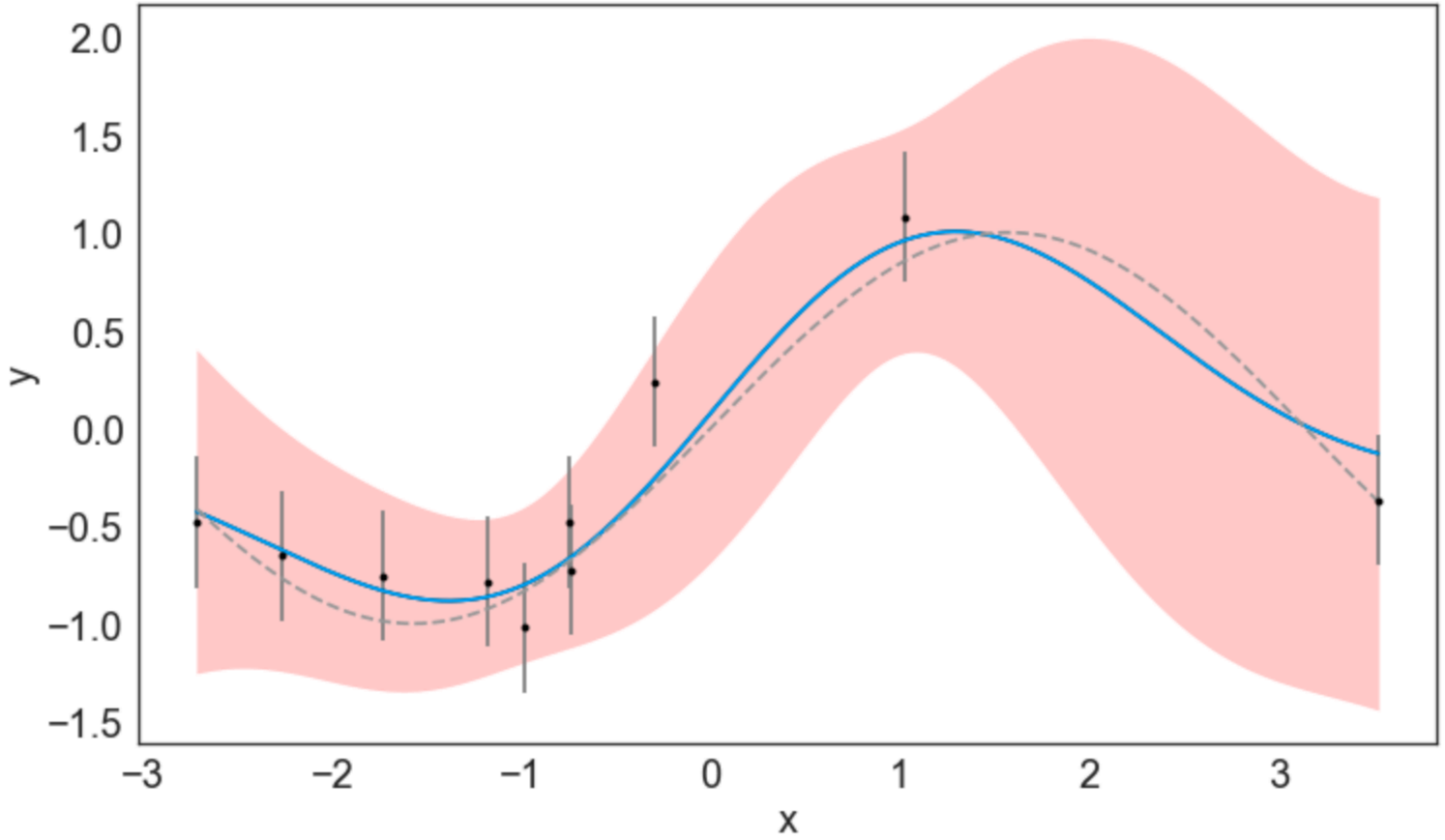
- Fairly slow

Python usage:

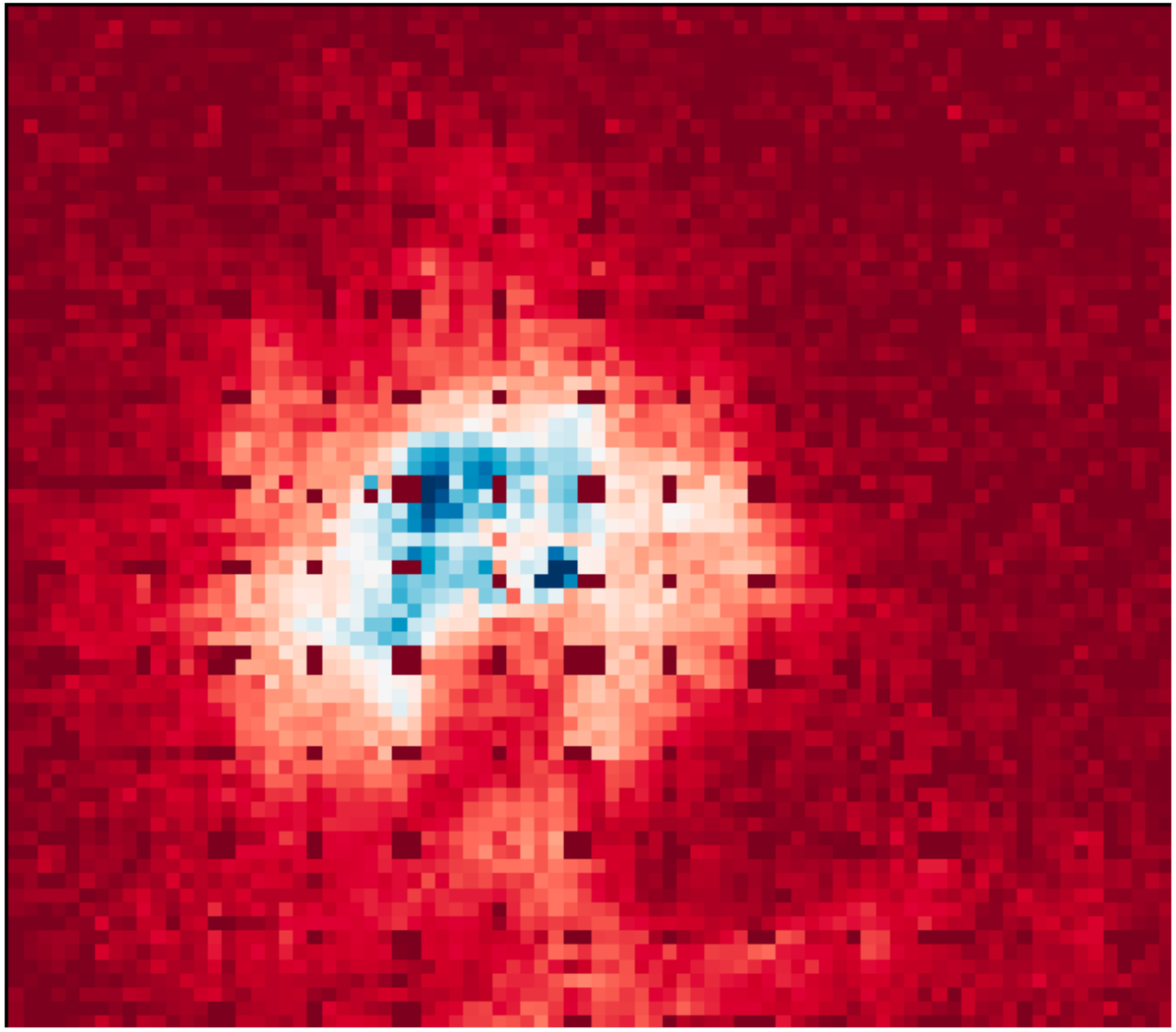
```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel

gp = GaussianProcessRegressor(kernel=RBF(0.1),
                              alpha=(dy/y)**2)
```

Example use (see notebook):

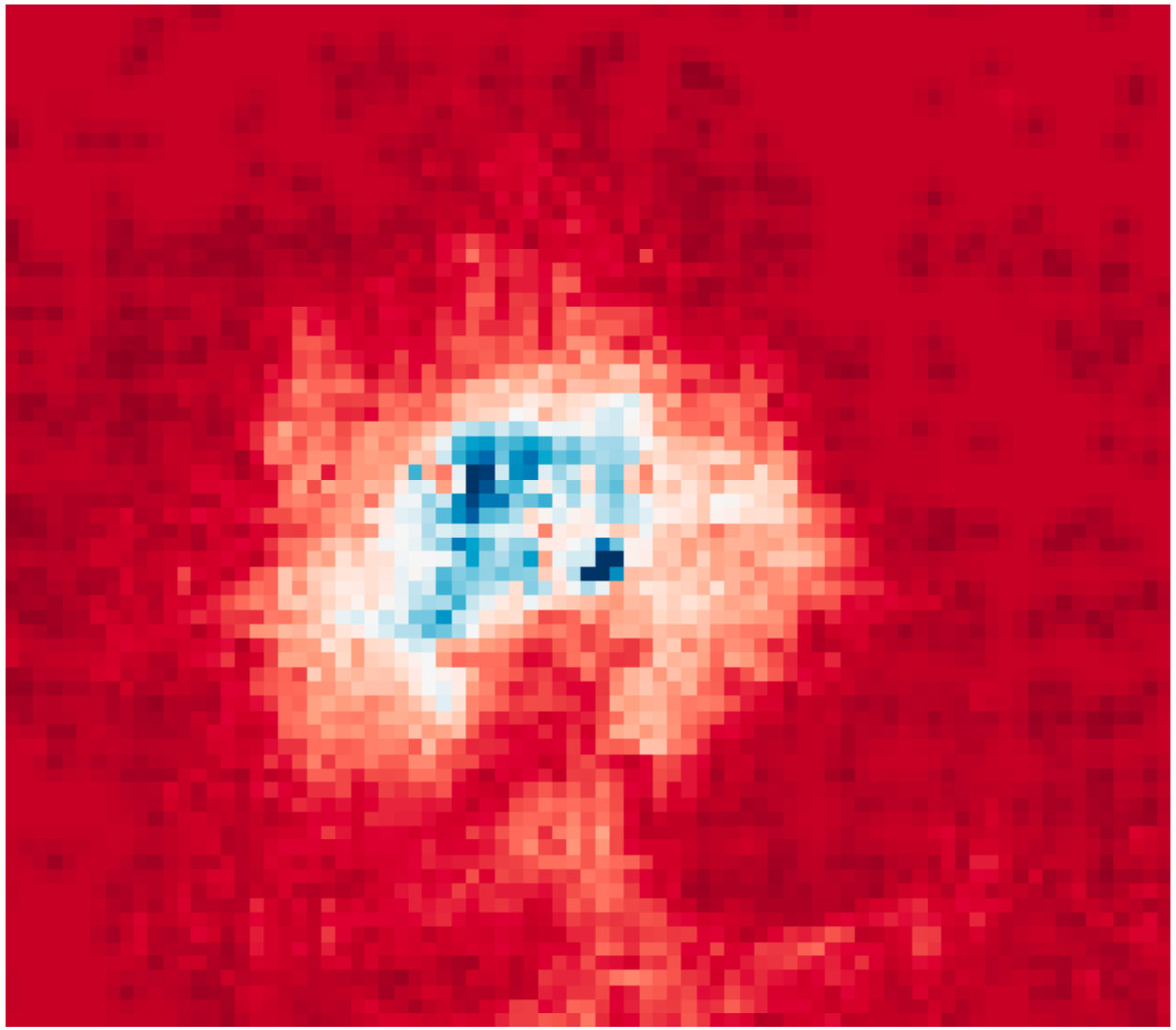
```
g = gp.fit(x[:, np.newaxis], y)
y_pred, sigma = gp.predict(xplot[:, np.newaxis], return_std=True)
plot_a_fit(x, y, dy, xplot, y_pred, sigma, include_true=True)
```



Example for an image



Example for an image



Usefulness in astronomy

- Light-curve modelling: stars (Brewer & Stello 2009), AGNs (Kelly et al 2014), X-ray binaries (Uttley et al 2005).
- Gaussian random field models for cosmic microwave background and large scale structure (e.g. Bond & Efstathiou 1987) are also possible to cast as a Gaussian Process.
- Quasar time-delay modelling (Hojjati et al 2013).
- Spectroscopic calibration (Czekala et al 2017).
- Widely used also as part of a larger model.

Usefulness in astronomy

- Widely used - also as part of a larger model.

Example: Model emulation to create galaxy formation models - Rodrigues, Vernon & Bower (2016, MNRAS, **466**:2, 2418)

Semi-analytic models of galaxy formation. Many parameters, expensive model to calculate.

The 20 parameters considered in Rodrigues et al:

Process modelled	Section	Parameter name [units]	Range		GP14	Scaling
Star formation (quiescent)	§2.2.1	$\nu_{\rm sf}$ [Gyr^{-1}]	0.025	1.0	0.5	lin
		$P_{\rm sf}/k_{\rm B}$ [cm^{-3}K]	1×10^4	5×10^4	1.7×10^4	log
		$\beta_{\rm sf}$	0.65	1.10	0.8	lin
Star formation (bursts)	§2.2.2	$f_{\rm dyn}$	1.0	100.0	10	log
		$\tau_{\rm min,burst}$ [Gyr]	10^{-3}	1	0.05	log
SNe feedback	§2.2.3	$\alpha_{\rm hot}$	1.0	3.7	3.2	lin
		$\beta_{0,\rm burst}$	0.5	40.0	11.16	lin
		$\beta_{0,\rm disc}$	0.5	40.0	11.16	lin
		$\alpha_{\rm reheat}$	0.15	1.5	1.26027	lin
AGN feedback	§2.2.4	$\alpha_{\rm cool}$	0.1	2.0	0.6	log
		$\epsilon_{\rm edd}$	0.004	0.1	0.03979	log
		$f_{\rm smbh}$	0.001	0.01	0.005	lin
Galaxy mergers		$f_{\rm burst}$	0.01	0.5	0.1	log
		$f_{\rm ellip}$	0.01	0.5	0.3	log
Disk stability	§2.2.5	$f_{\rm stab}$	0.61	1.1	0.8	lin
Reionization		$V_{\rm cut}$ [km s^{-1}]	20	60	30	lin
		$z_{\rm cut}$	5	15	10	lin
Metal enrichment		$p_{\rm yield}$	0.02	0.05	0.021	lin
Ram pressure stripping		$\epsilon_{\rm strip}$	0.01	0.99	n/a	lin
		$\alpha_{\rm rp}$	1.0	3.0	n/a	lin

Rodrigues, Vernon & Bower (2016, MNRAS, **466**:2, 2418)

Fitting data to model

The model is prohibitively expensive to run millions/billions of times, so how do you fit this in a Bayesian framework?

Fitting data to model

The model is prohibitively expensive to run millions/billions of times, so how do you fit this in a Bayesian framework?

Write the vector of model outputs as:

$$\mathbf{M}\left(\nu_{0,\mathrm{sf}}, P_{\mathrm{sf}}/k_{\mathrm{B}}, \ldots, \alpha_{\mathrm{rp}}\right) = \mathbf{M}(\theta)$$

Fitting data to model

The model is prohibitively expensive to run millions/billions of times, so how do you fit this in a Bayesian framework?

Write the vector of model outputs as:

$$\mathbf{M}\left(\nu_{0,\mathrm{sf}}, P_{\mathrm{sf}}/k_{\mathrm{B}}, \ldots, \alpha_{\mathrm{rp}}\right) = \mathbf{M}(\theta)$$

Then write this as a regression problem:

$$M_i(\theta) = \sum_j \beta_{ij} g_{ij}(\theta) + u_i(\theta) + \nu_i(\theta)$$

Fitting data to model

The model is prohibitively expensive to run millions/billions of times, so how do you fit this in a Bayesian framework?

Write the vector of model outputs as:

$$\mathbf{M}\left(\nu_{0,\mathrm{sf}}, P_{\mathrm{sf}}/k_{\mathrm{B}}, \ldots, \alpha_{\mathrm{rp}}\right) = \mathbf{M}(\theta)$$

Then write this as a regression problem:

$$M_i(\theta) = \sum_j \beta_{ij} g_{ij}(\theta) + u_i(\theta) + \nu_i(\theta)$$

Polynomials

Fitting data to model

The model is prohibitively expensive to run millions/billions of times, so how do you fit this in a Bayesian framework?

Write the vector of model outputs as:

$$\mathbf{M}\left(\nu_{0,\mathrm{sf}}, P_{\mathrm{sf}}/k_{\mathrm{B}}, \ldots, \alpha_{\mathrm{rp}}\right) = \mathbf{M}(\theta)$$

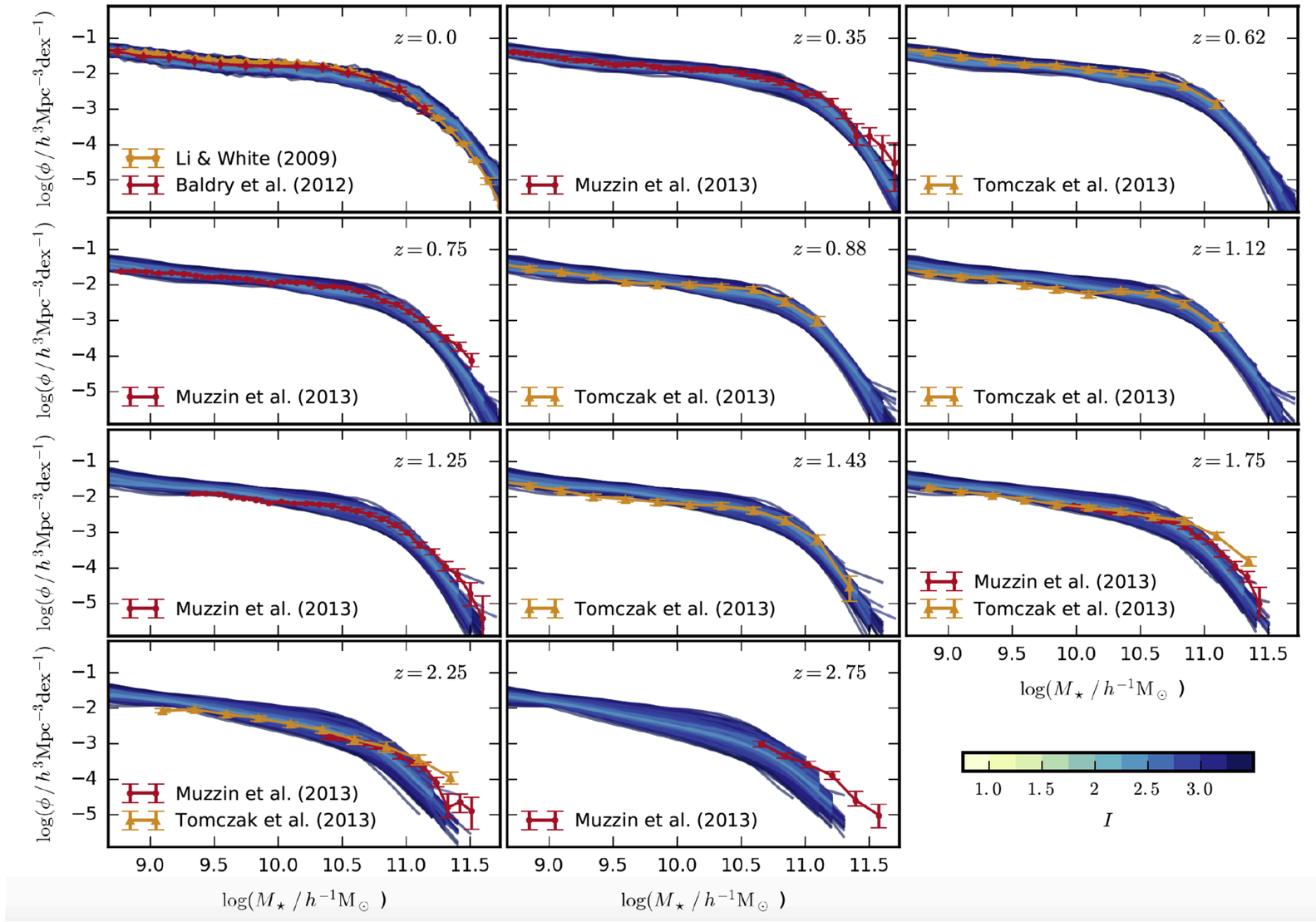
Then write this as a regression problem:

$$M_i(\theta) = \sum_j \beta_{ij} g_{ij}(\theta) + u_i(\theta) + \nu_i(\theta)$$

Polynomials

Gaussian Process

Fitting data to model - galaxy mass functions



Rodrigues, Vernon & Bower (2016, MNRAS, **466**:2, 2418)

Basis function regression

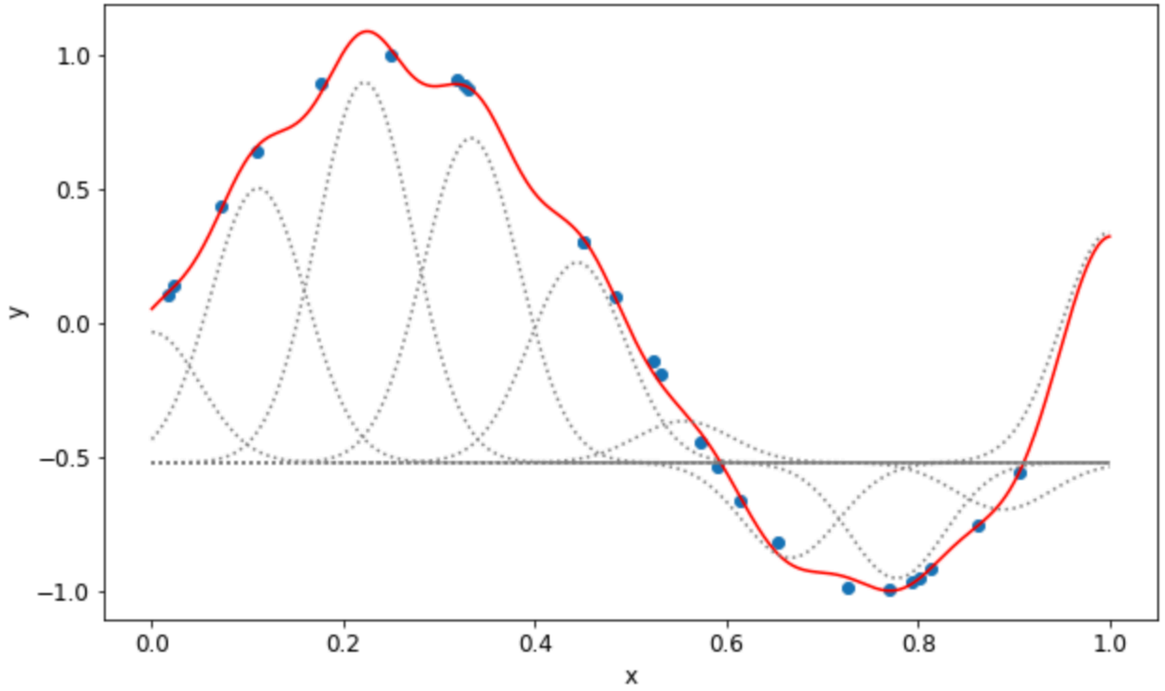
See: MLD2025-06b-Gaussian basis function regression Notebook for details!

Finally, let us return to an earlier topic - basis function regression.

$$f(x) = \sum_i w_i \phi_i(x)$$

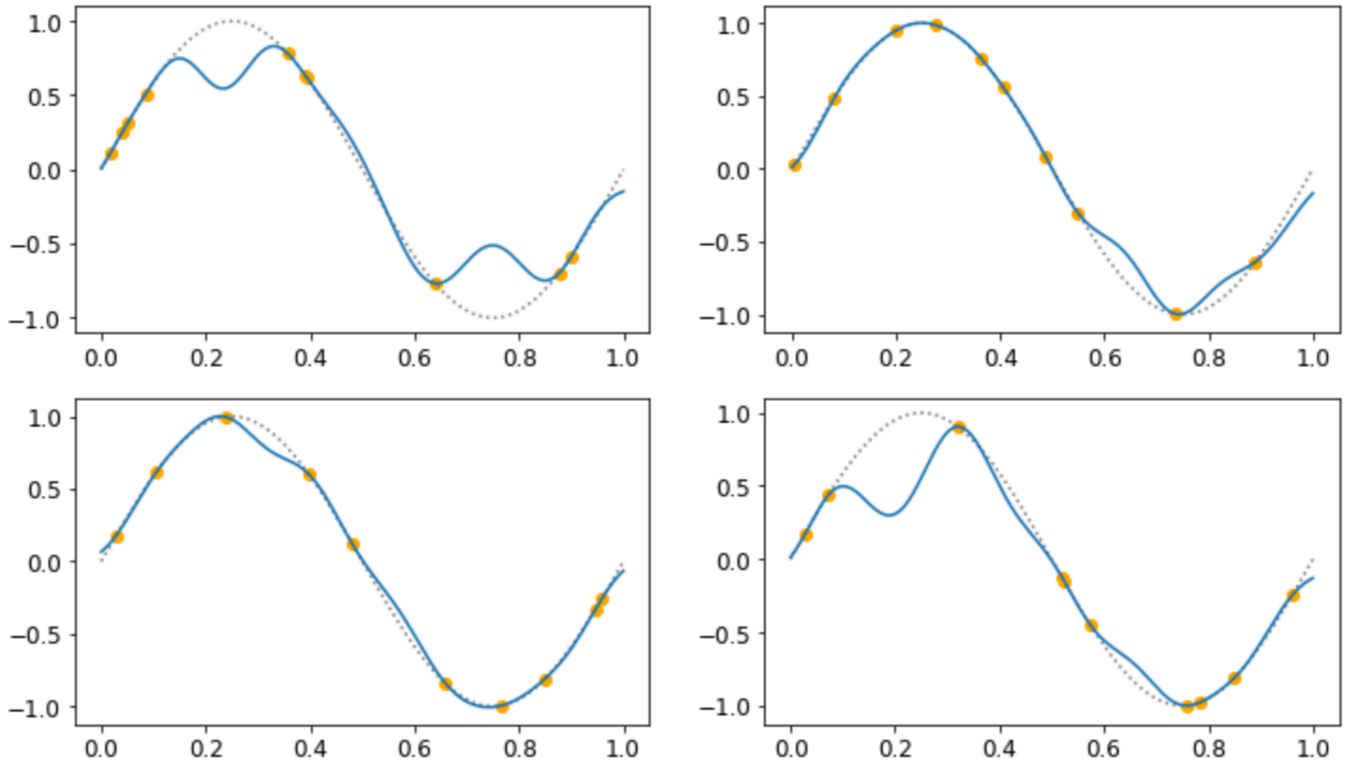
This is linear regression, but transforming the x values.

e.g. Gaussian basis:

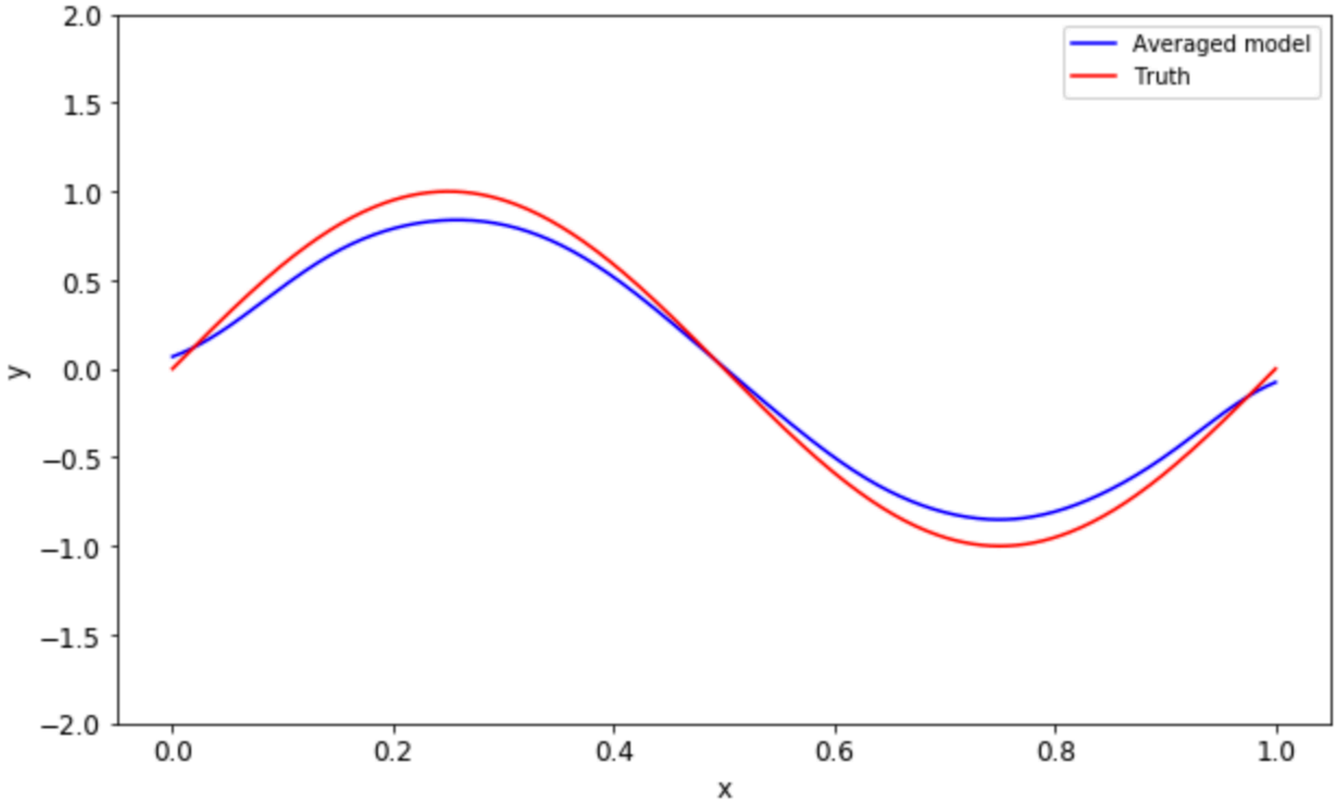


Gaussian-basis function regression

Create 100 datasets with 10 points each, fit these with 10 Gaussians with fixed width. Thus easily affected by overfitting, but low bias.



Combining it all



So averaging the results of the fits gives a better final result, less sensitive to overfitting.

Can this be generalised? We usually do not have N indepedent samples… We'll return to this tomorrow!

Visualising data

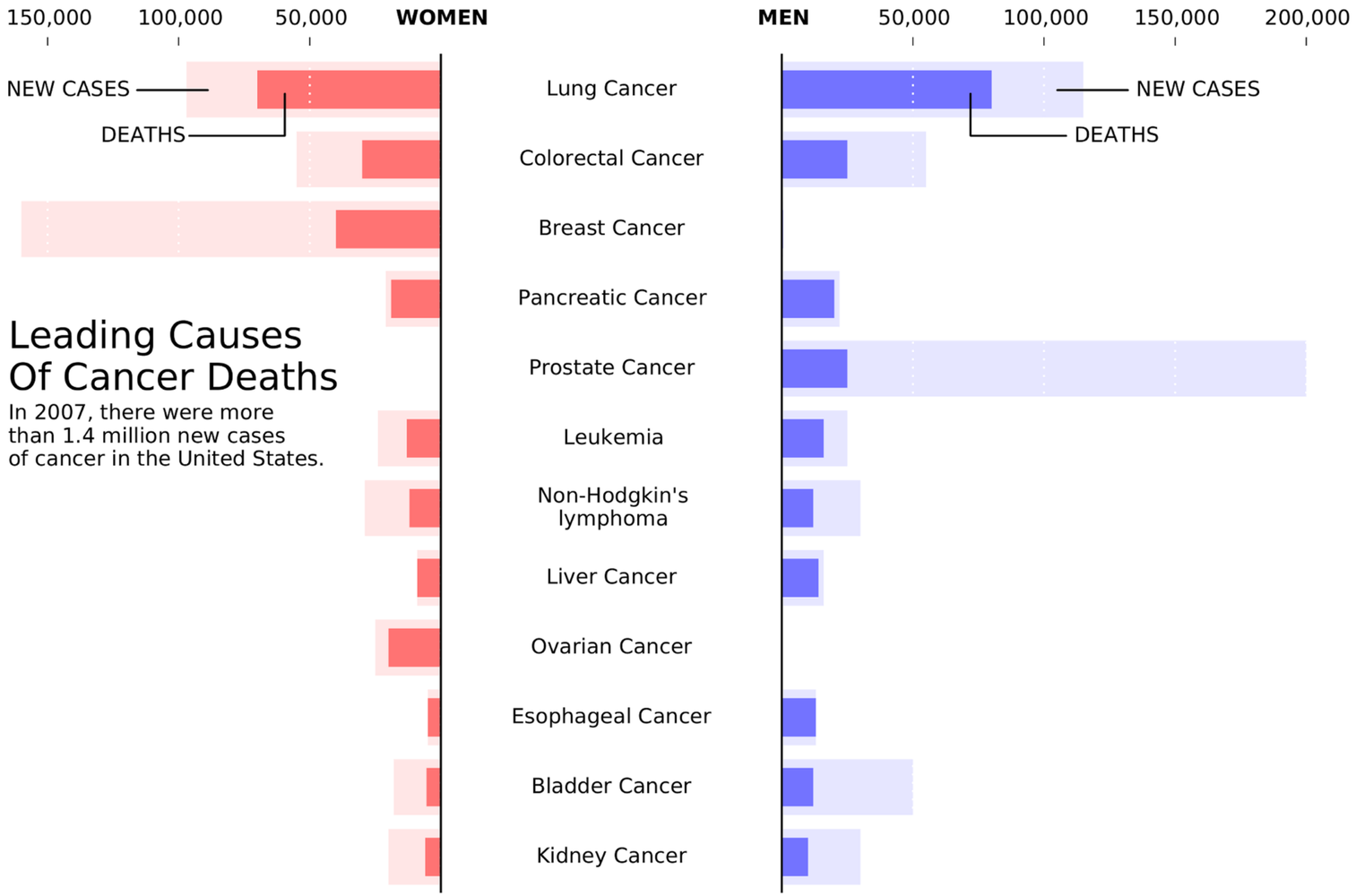
Making good scientific plots

- Know the audience.
- Adapt to the situation.
- Be precise, include all information needed.
- Do not mislead.
- Do not use colour without thinking. Especially bad colour choices.
- Do not overload a figure - or include too little!
- Learn a couple of packages well.

Worth checking out: Rougier NP, Droettboom M, Bourne PE (2014) Ten Simple Rules for Better Figures. PLoS Comput Biol 10(9): e1003833.
http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003833

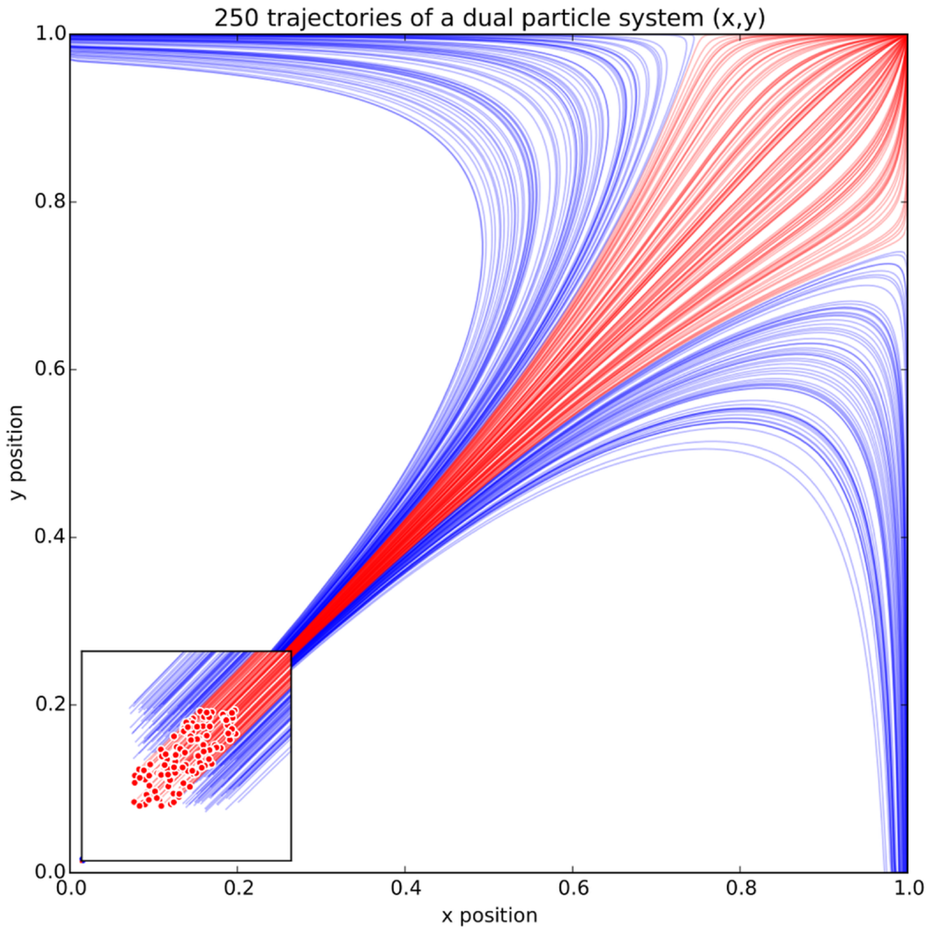
Adapting to the audience

from: Rougier NP, Droettboom M, Bourne PE (2014) - an example of a newspaper article



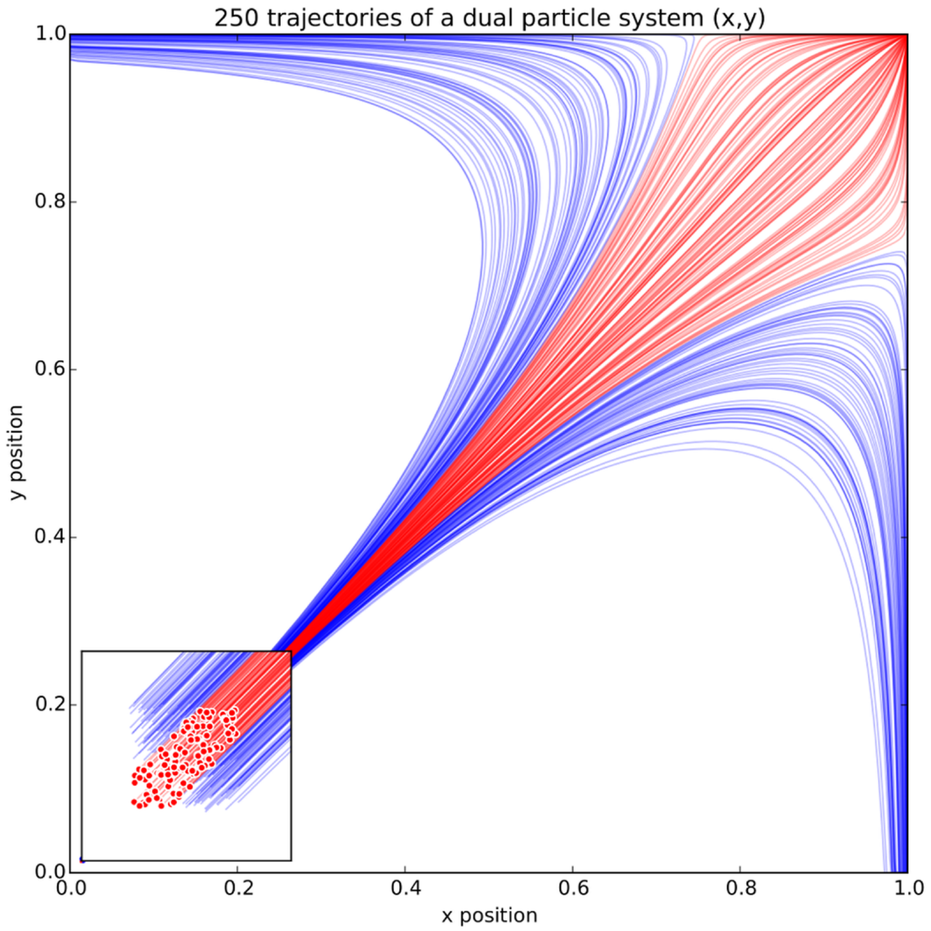
Adapting to the situation

from: Rougier NP, Droettboom M, Bourne PE (2014)



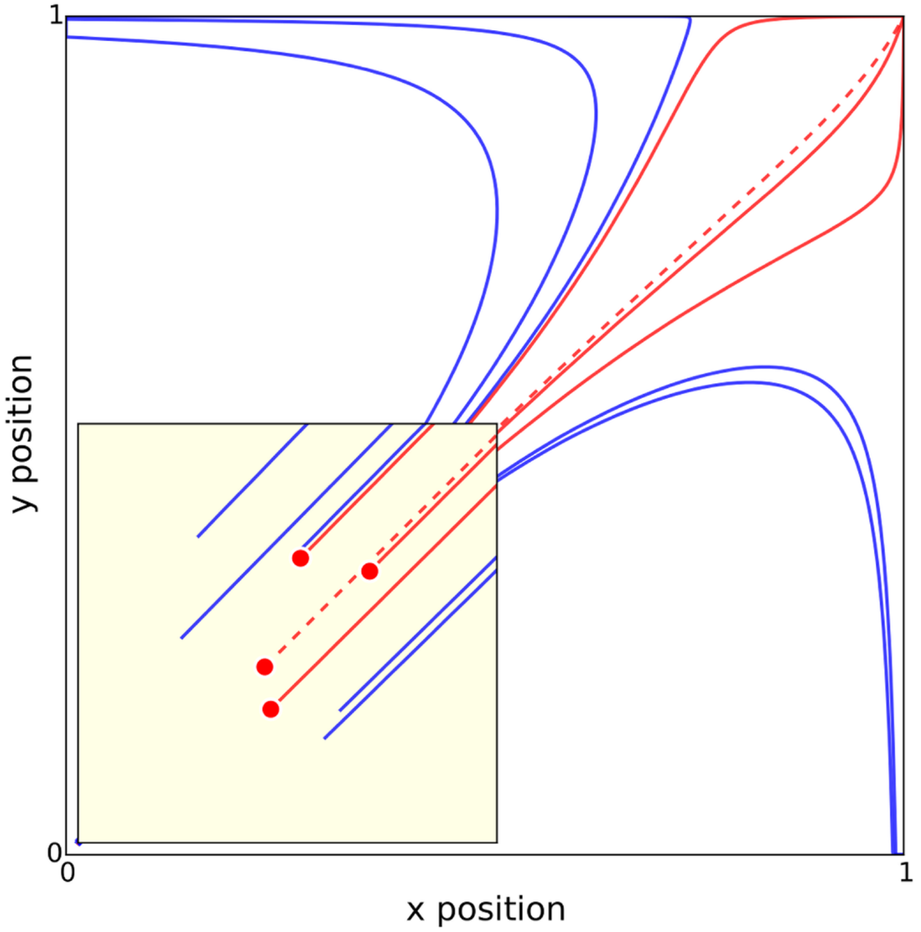
Scientific paper

Adapting to the situation



Scientific paper

from: Rougier NP, Droettboom M, Bourne PE (2014)



Presentation

Why do we create visualisations (plots)?

Why do we create visualisations (plots)?

To help our argumentation.

Why do we create visualisations (plots)?

To help our argumentation.

To summarise a lot of data

Why do we create visualisations (plots)?

To help our argumentation.

To summarise a lot of data

To discover new relationships

Why do we create visualisations (plots)?

To help our argumentation.

To summarise a lot of data

To discover new relationships

To relate different pieces of information

Why do we create visualisations (plots)?

To help our argumentation.

To summarise a lot of data

To discover new relationships

To relate different pieces of information

To develop intuition

Graphical displays should:

- Show the data.
- Induce the viewer to think about the substance.
- Avoid distorting what the data have to say.
- Present many numbers in a small space.
- Make large data sets coherent.
- Encourage the eye to compare different pieces of data.
- Reveal data at several layers of detail.
- Be relevant.
- **Show units, show scales!**

Based on: Tufte: “The Visual Display of Quantitative Information”

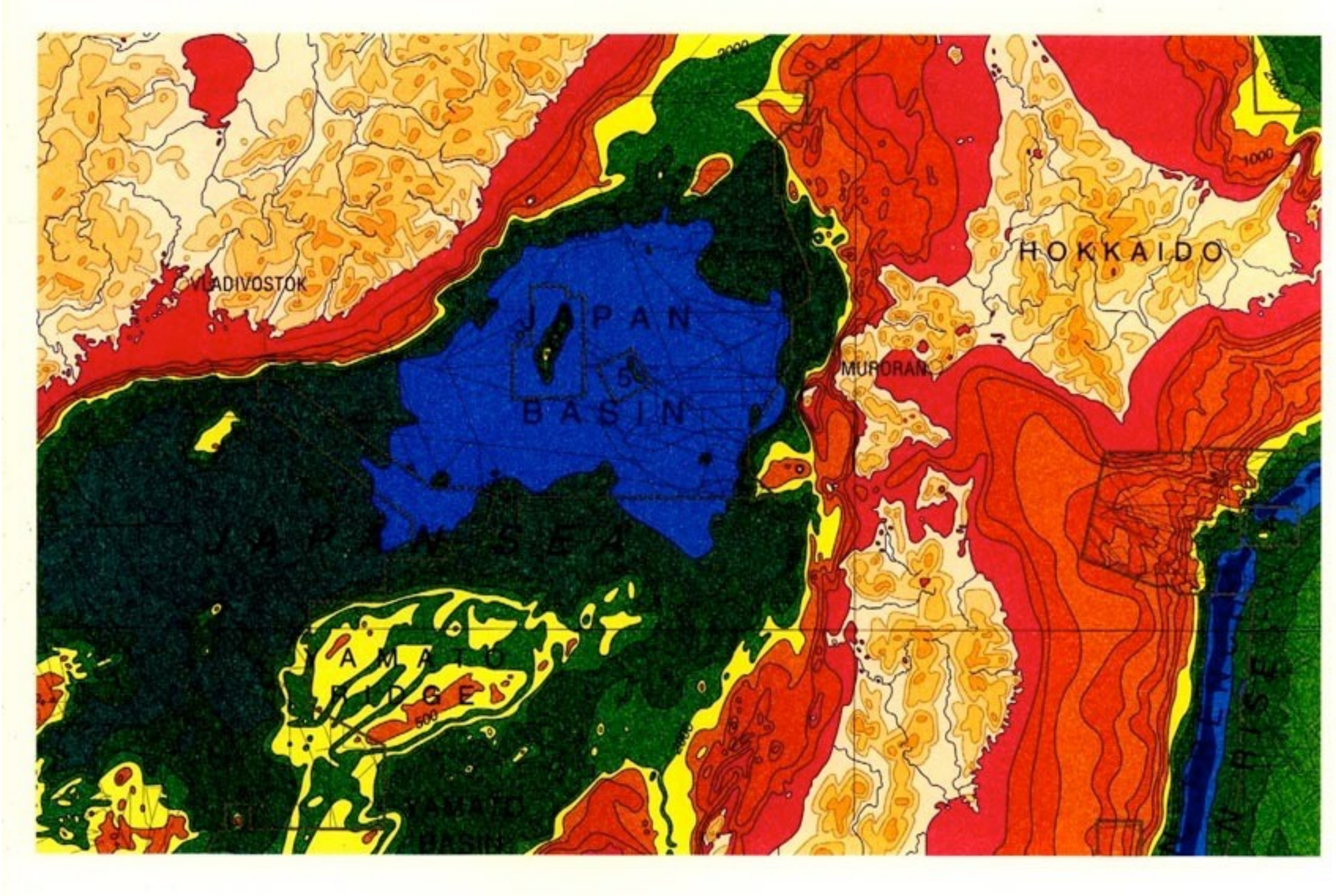
Colour choice

The “Smallest effective difference”

From Edward Tufte (1997), “Visual explanations”

Colour choice

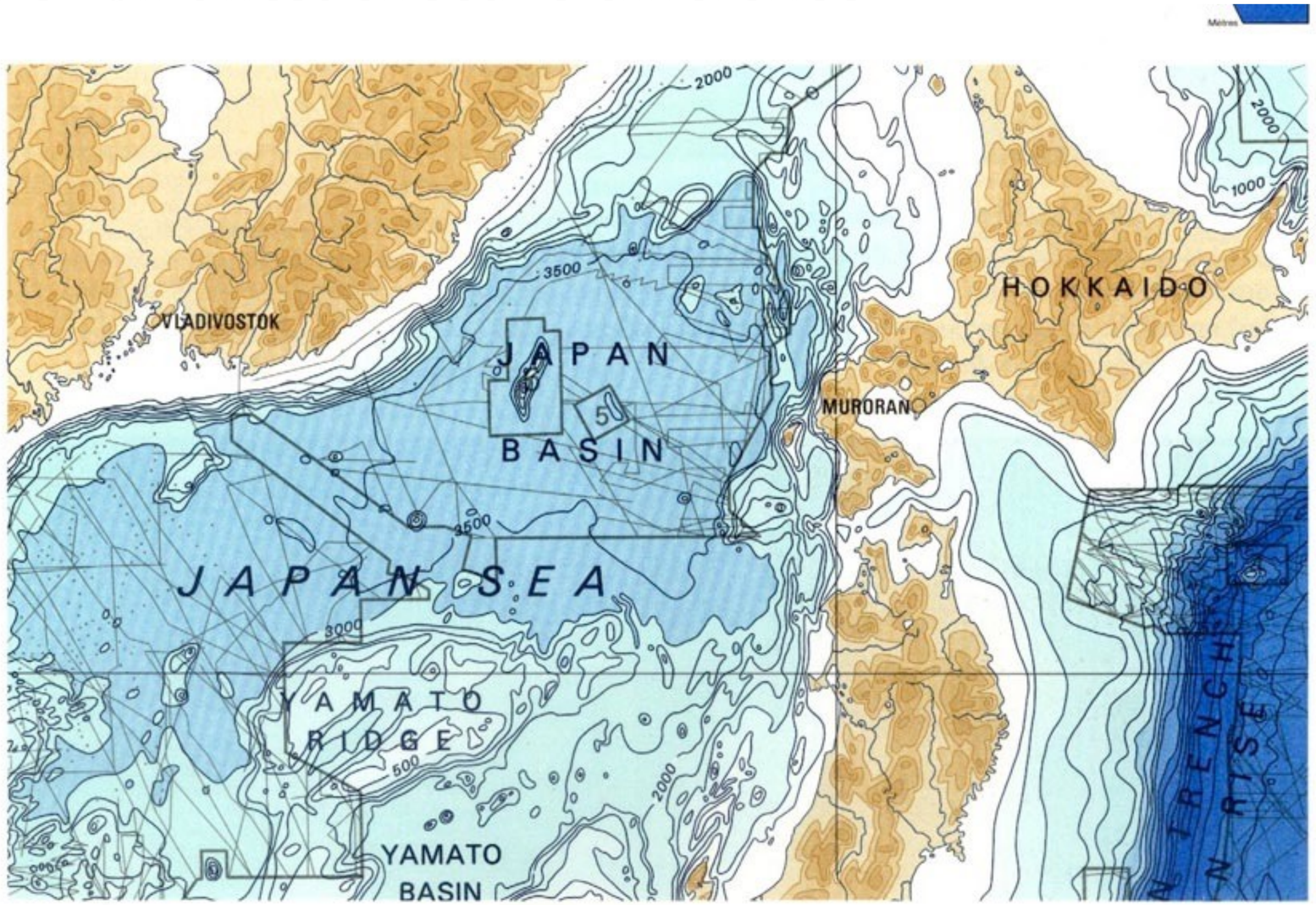
The “Smallest effective difference”



From Edward Tufte (1997), “Visual explanations”

Colour choice

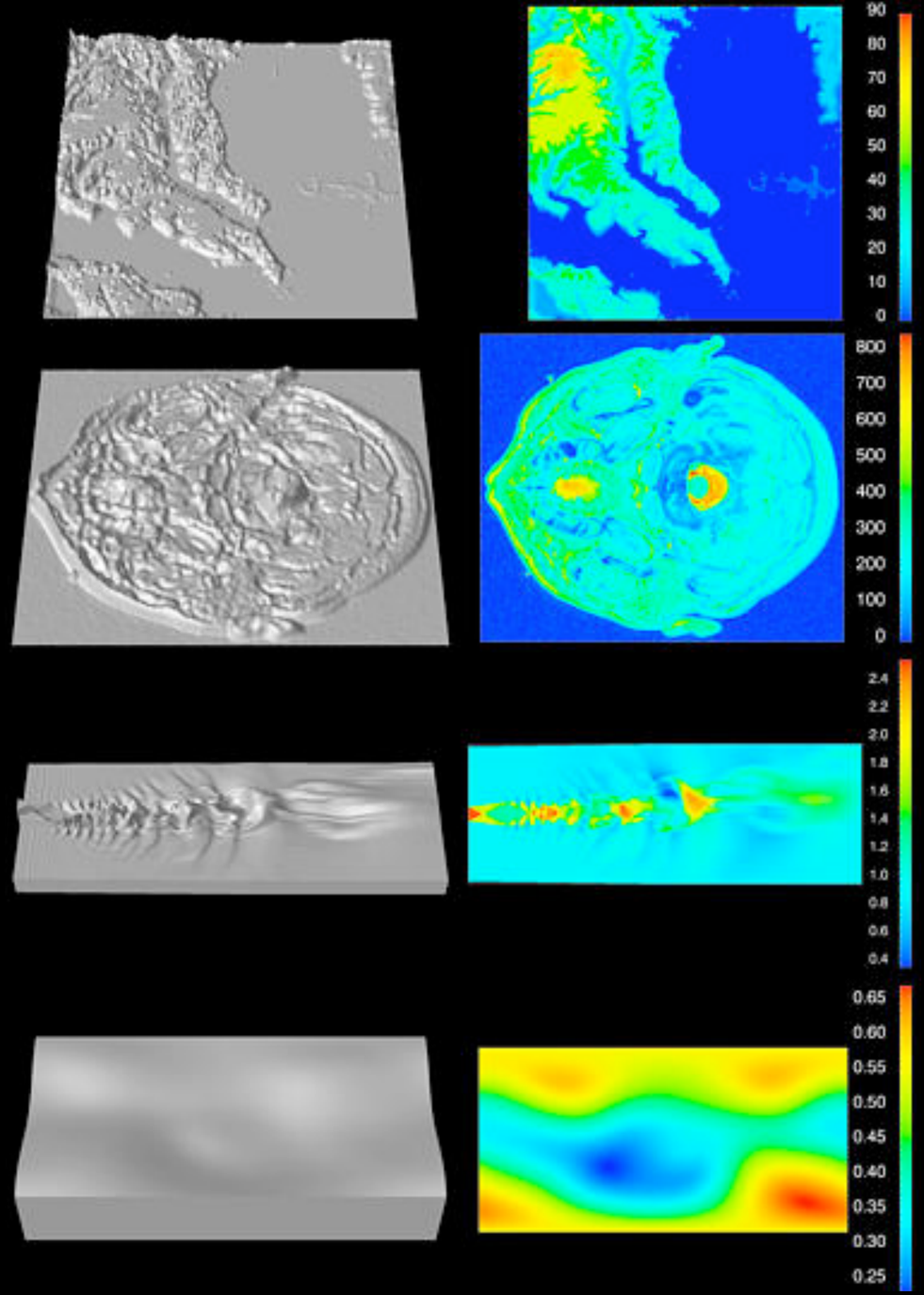
The “Smallest effective difference”



From Edward Tufte (1997), “Visual explanations”

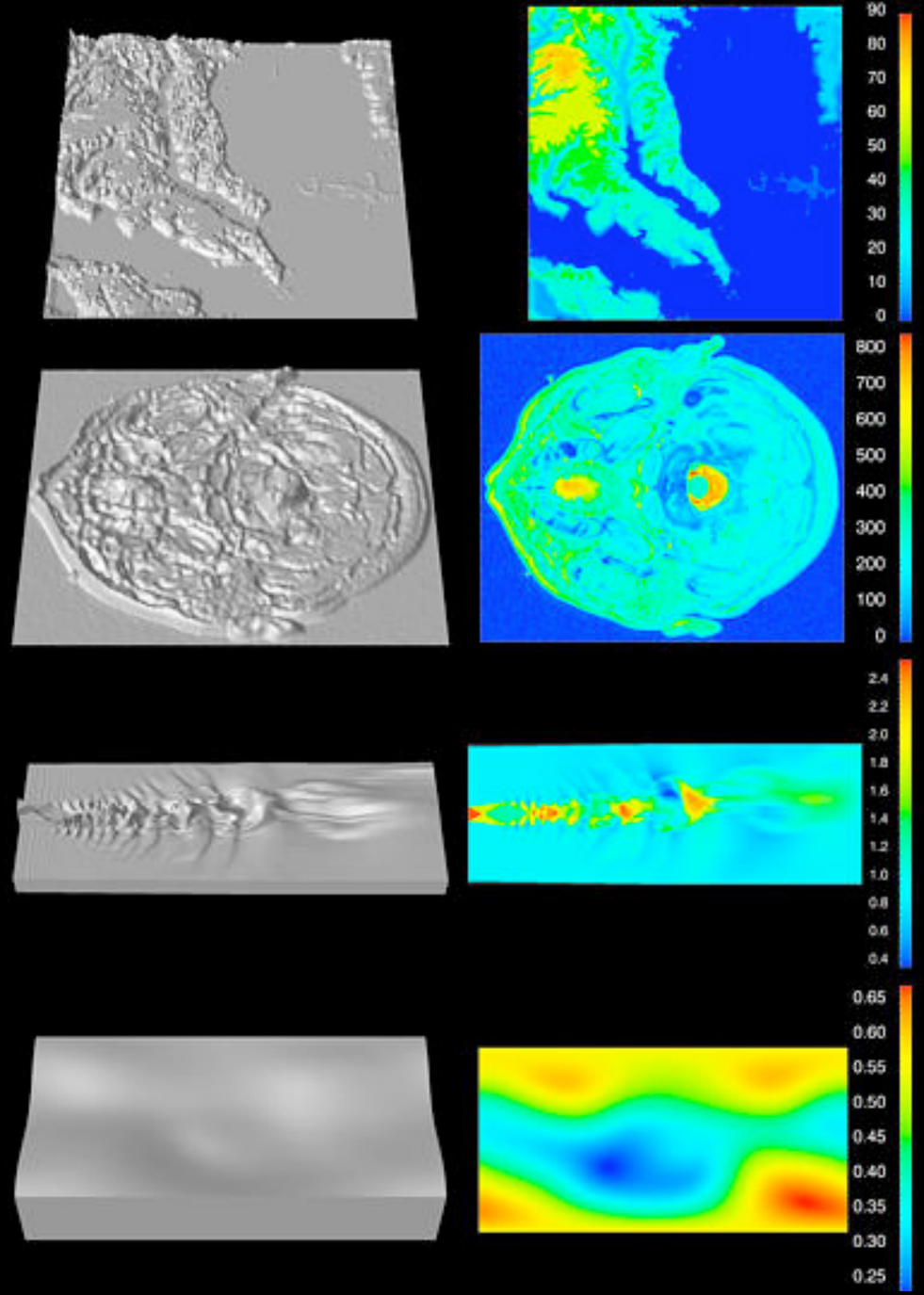
The issue of colour & 2D plots

There is a tendency for astronomers to choose from a small set of colour schemes for their images and plots. Sometimes this leads you to create artificial trends where there are none.



Bad Habits

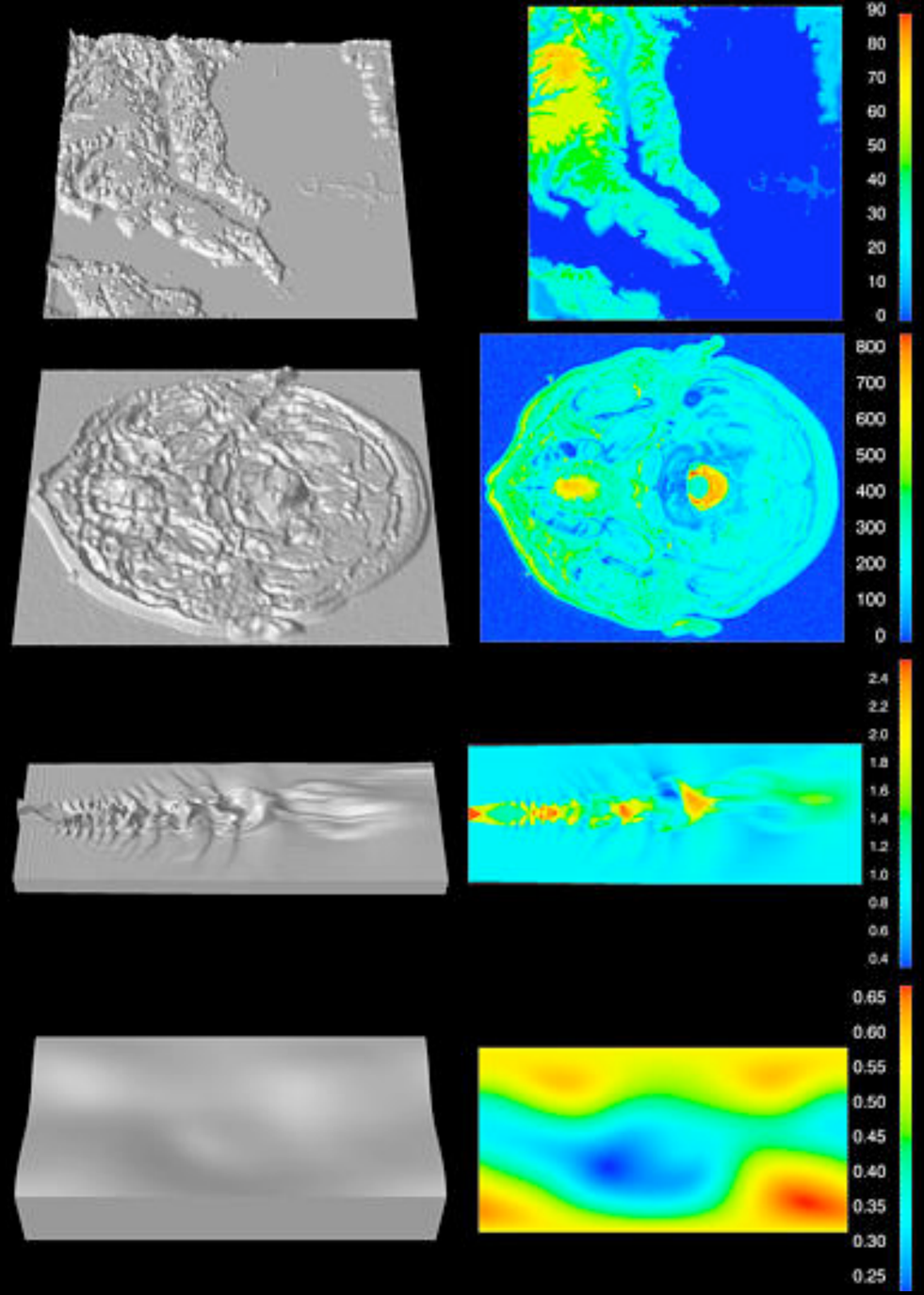
For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM



Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

Bad Habits

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM

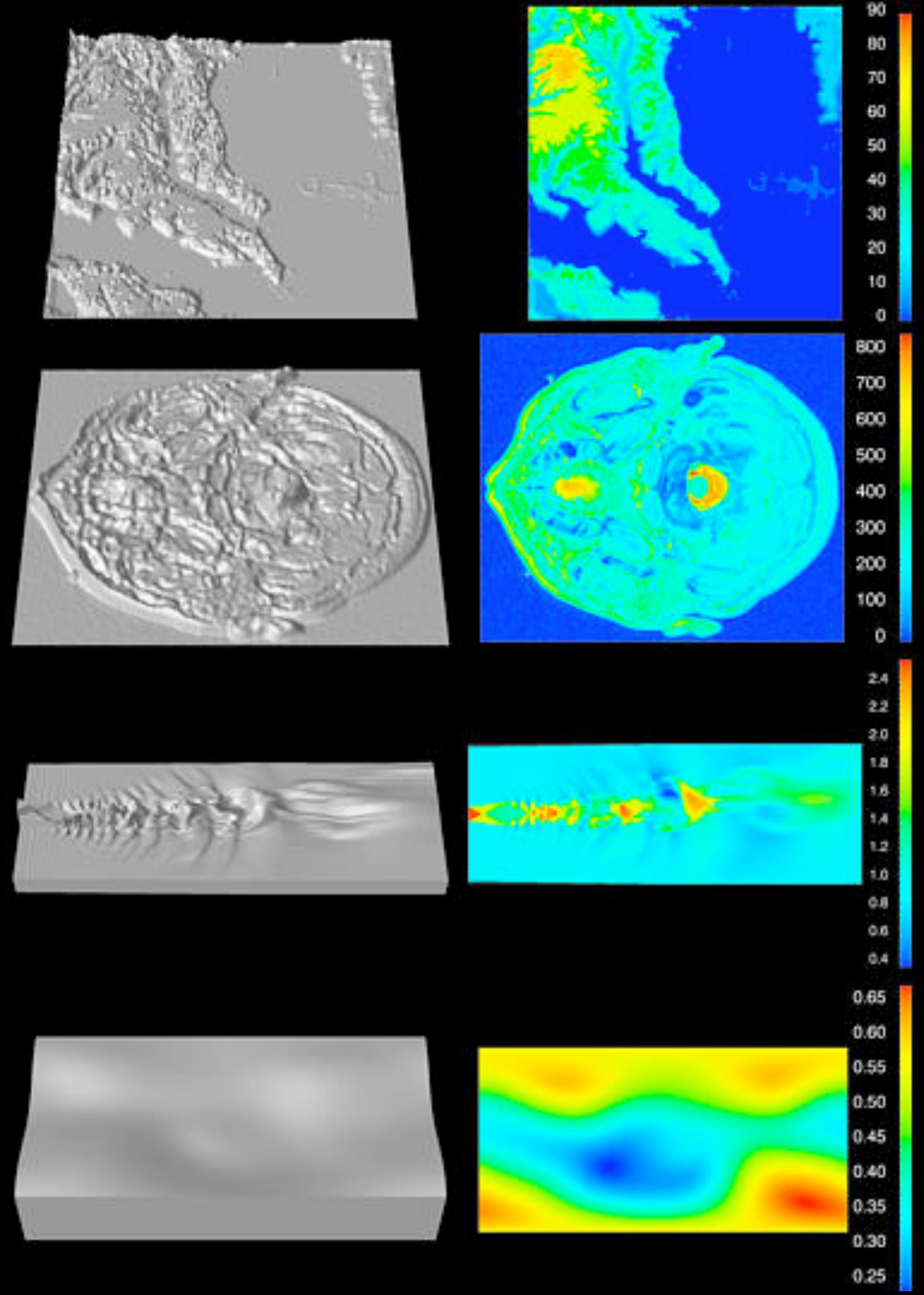


Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

An slice of an MRI scan of a human brain. Washing out of detail and artificial structure.

Bad Habits

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM



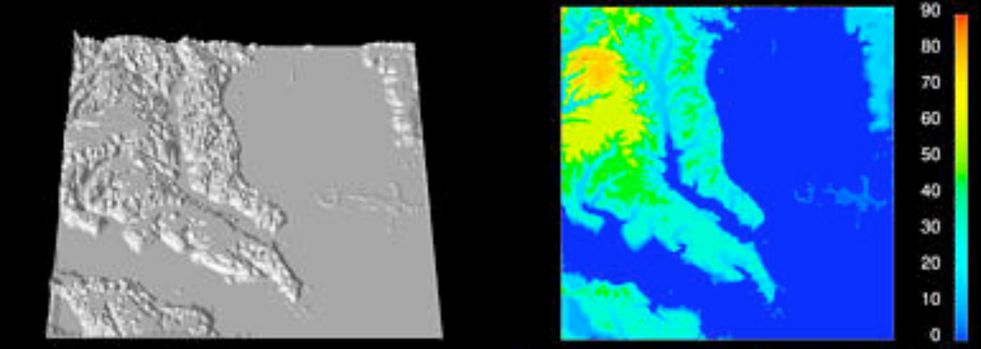
Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

An slice of an MRI scan of a human brain. Washing out of detail and artificial structure.

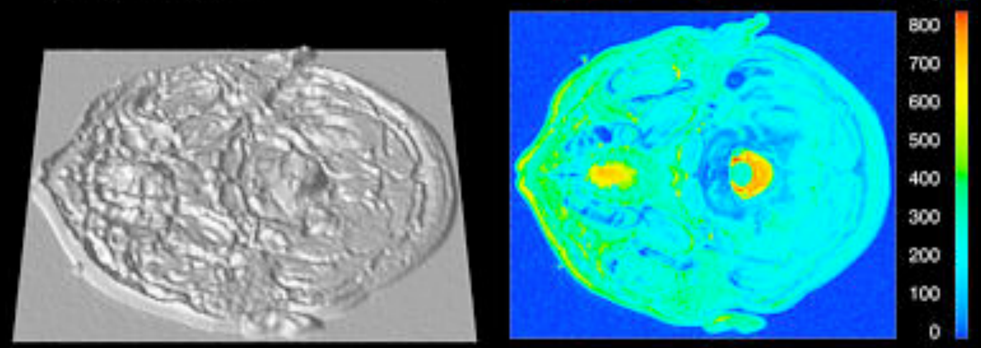
Bad Habits

Turbulent flow from a jet engine.

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM

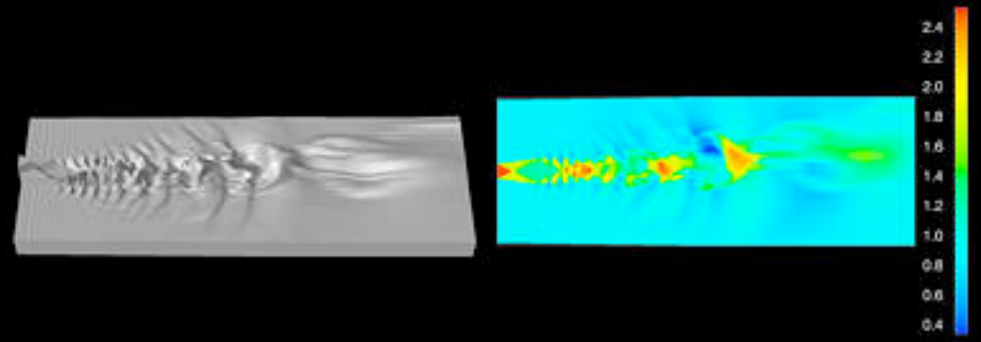


Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

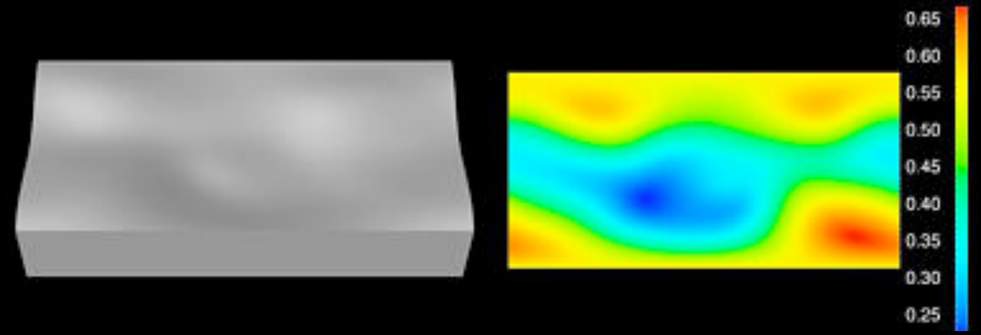


An slice of an MRI scan of a human brain. Washing out of detail and artificial structure.

Bad Habits

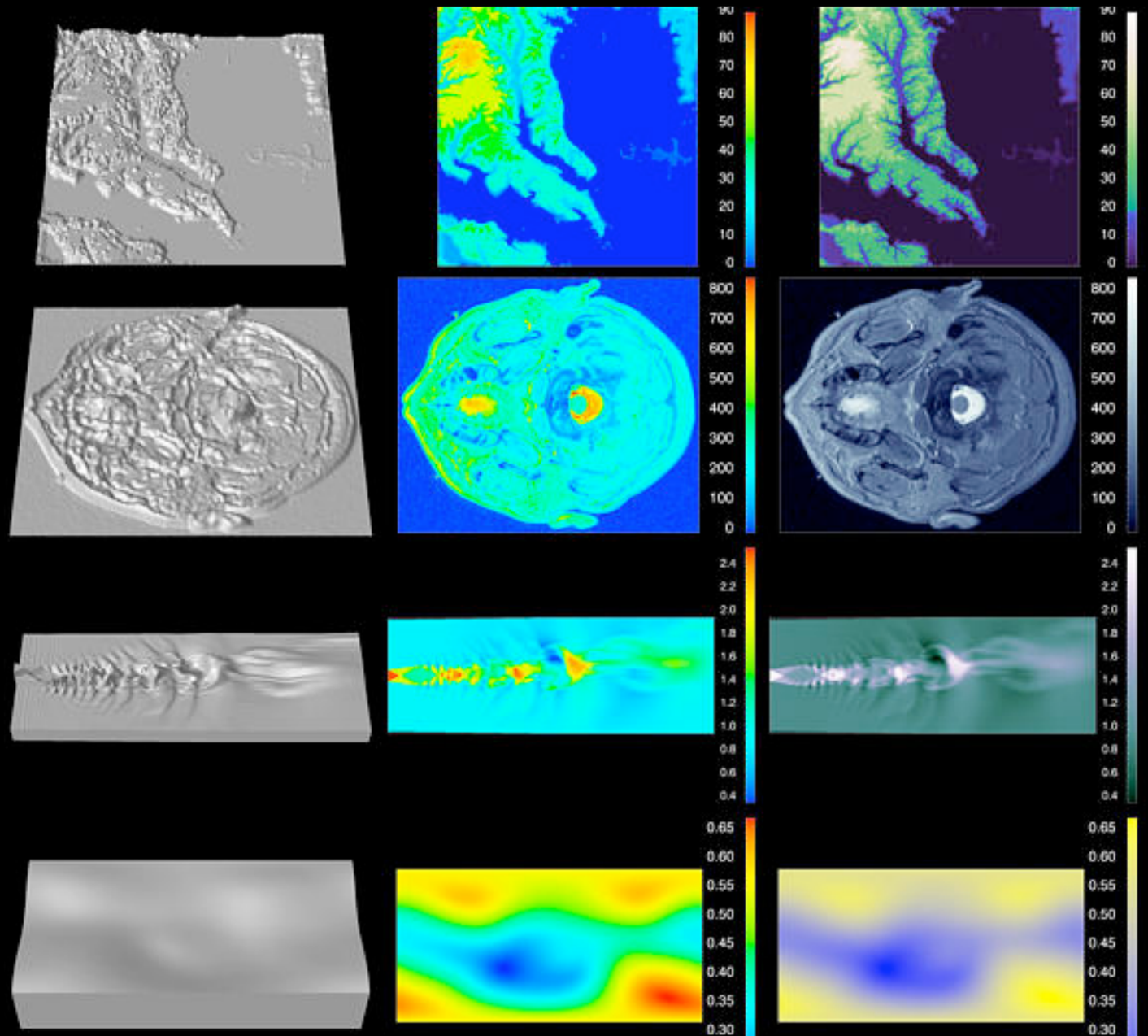


Turbulent flow from a jet engine.



Earth's magnetic field in a Cartesian projection - note the smooth structure.

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM

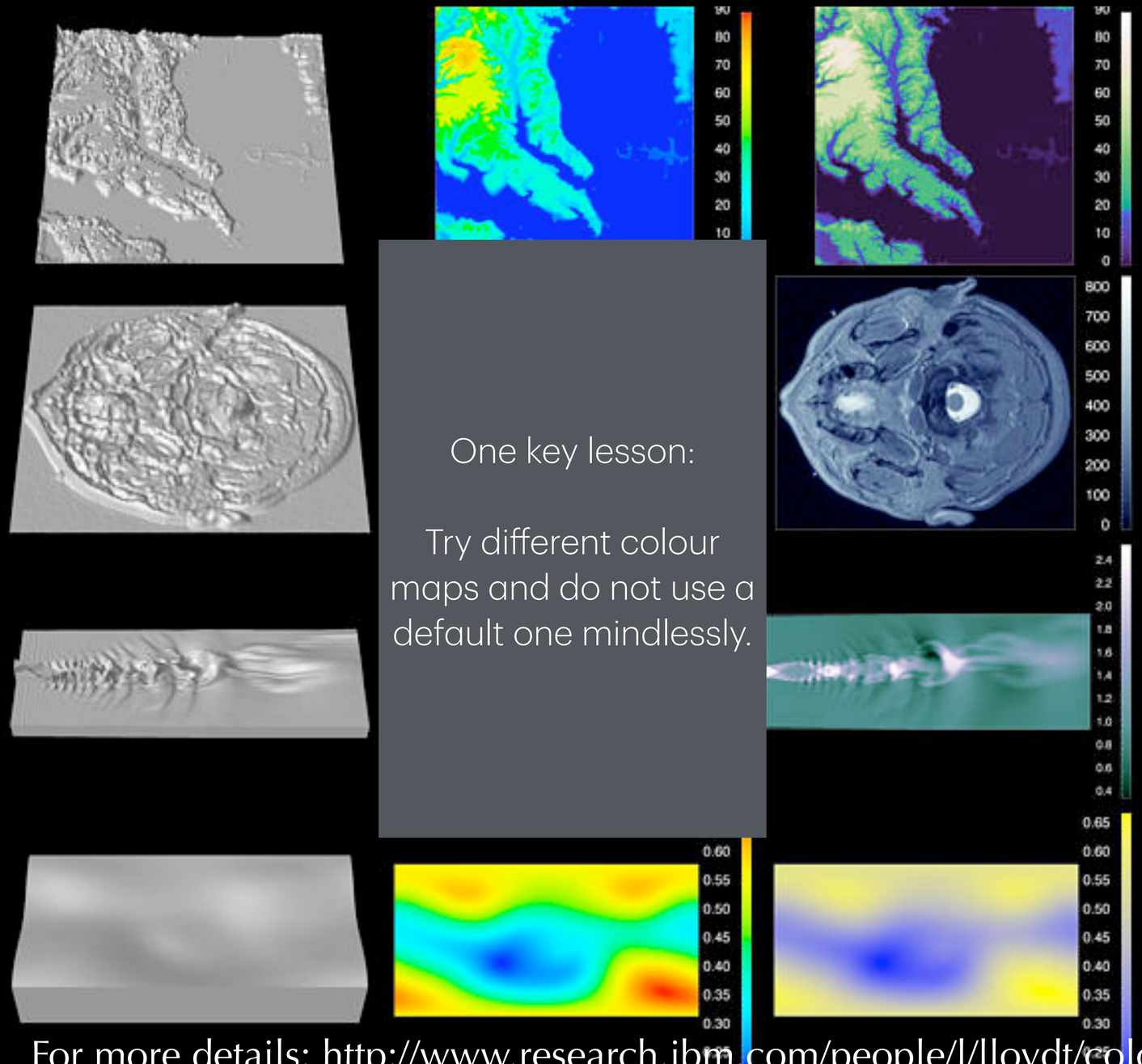


A natural zero

High frequency information - best with variation of luminance

Low frequency information - colour variation (saturation is good)

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM



A natural zero

One key lesson:

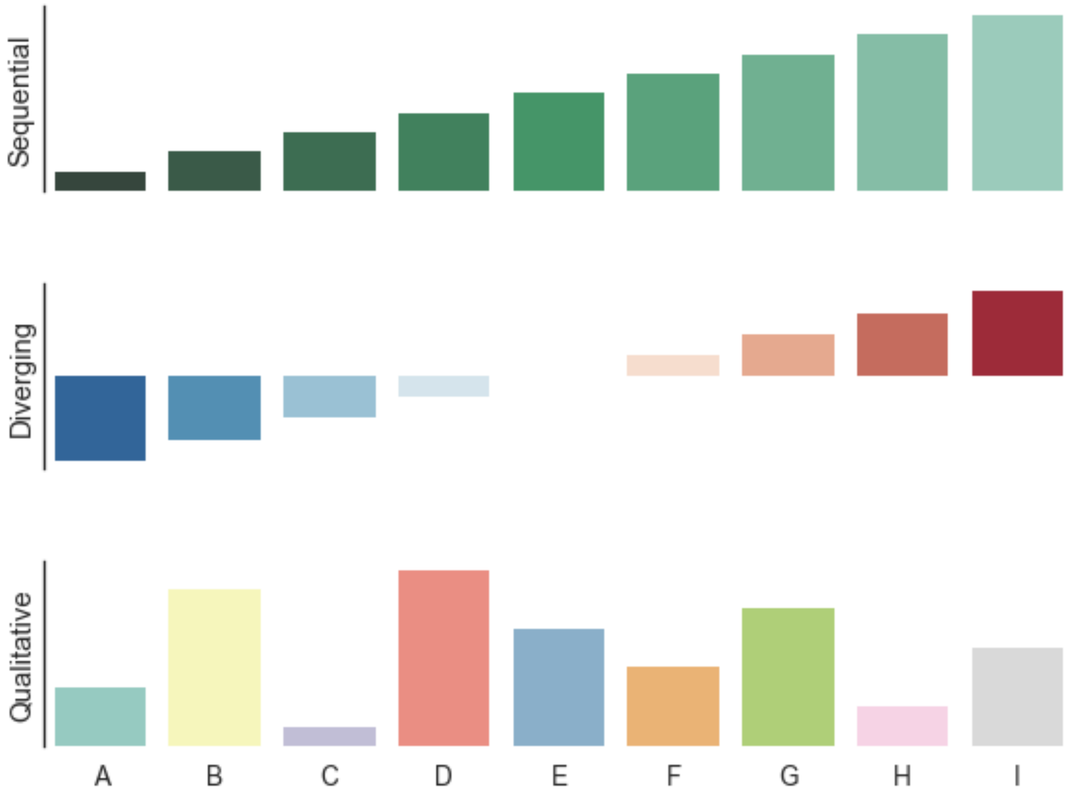
Try different colour maps and do not use a default one mindlessly.

High frequency information - best with variation of luminance

Low frequency information - colour variation (saturation is good)

For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM

Colour should provide information



From the seaborn gallery [but meant to illustrate colour choices]

Colour choice - colour vision deficiency

This affects ~10% of the population so worth keeping in mind when you make plots for sharing with others.

Simplest: Do not use red & green to create contrast.

Better: use safe colour palettes (viridis/cividis in matplotlib):

In Seaborn:

```
import seaborn as sns
sns.set_palette('colorblind')
```

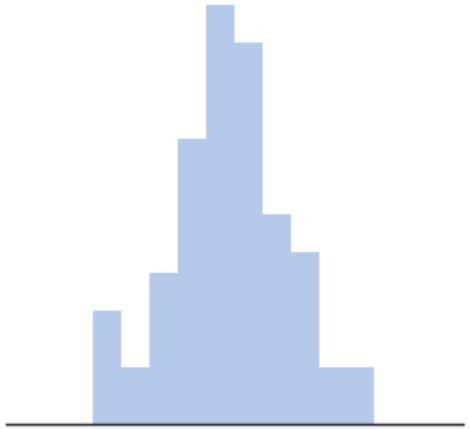
Paul Tol has a very useful page:

https://personal.sron.nl/~pault/

Checking: Photoshop/Illustrator have options, in Python, use Jörg Dietrich's Daltonize package: https://github.com/joergdietrich/daltonize

Various ways to show 1D distributions

```
import matplotlib.pyplot as plt

import seaborn as sns
```



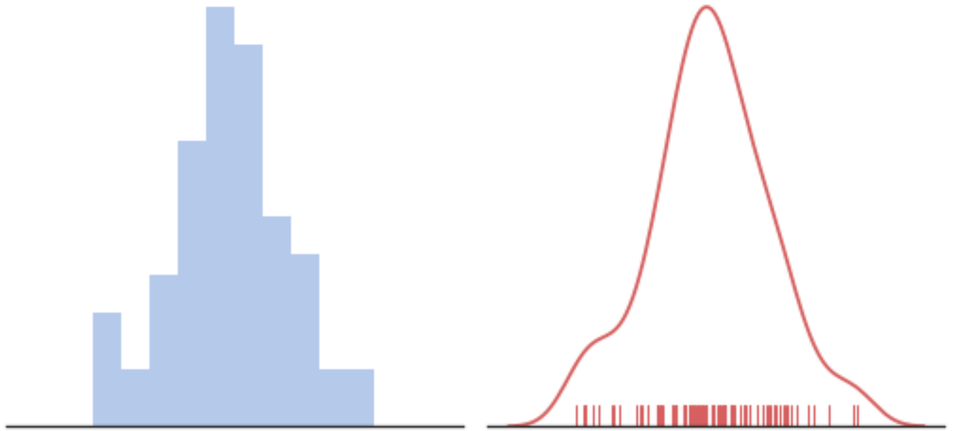
```
sns.distplot(x, kde=False)

or

plt.hist(x)
```

Histograms are convenient but bin the data and are not differentiable - we'll look at kernel density plots later. Rug plots complement histograms and kernel density plots by showing the data.

http://seaborn.pydata.org/examples/distplot_options.html

Various ways to show 1D distributions

```
import matplotlib.pyplot as plt

import seaborn as sns
```



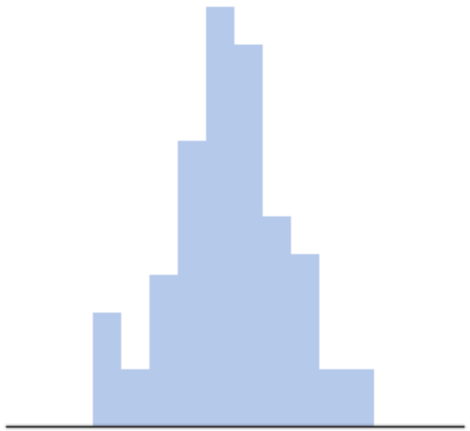
```
sns.distplot(x, kde=True,
rug=True)
```

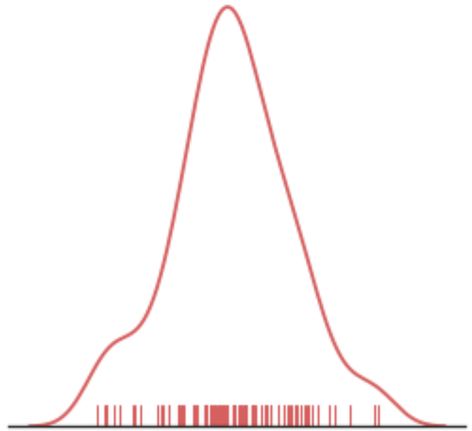
Histograms are convenient but bin the data and are not differentiable - we'll look at kernel density plots later. Rug plots complement histograms and kernel density plots by showing the data.

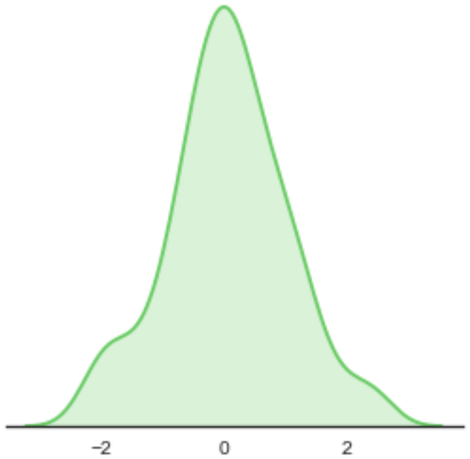
http://seaborn.pydata.org/examples/distplot_options.html

Various ways to show 1D distributions

```
import matplotlib.pyplot as plt

import seaborn as sns
```







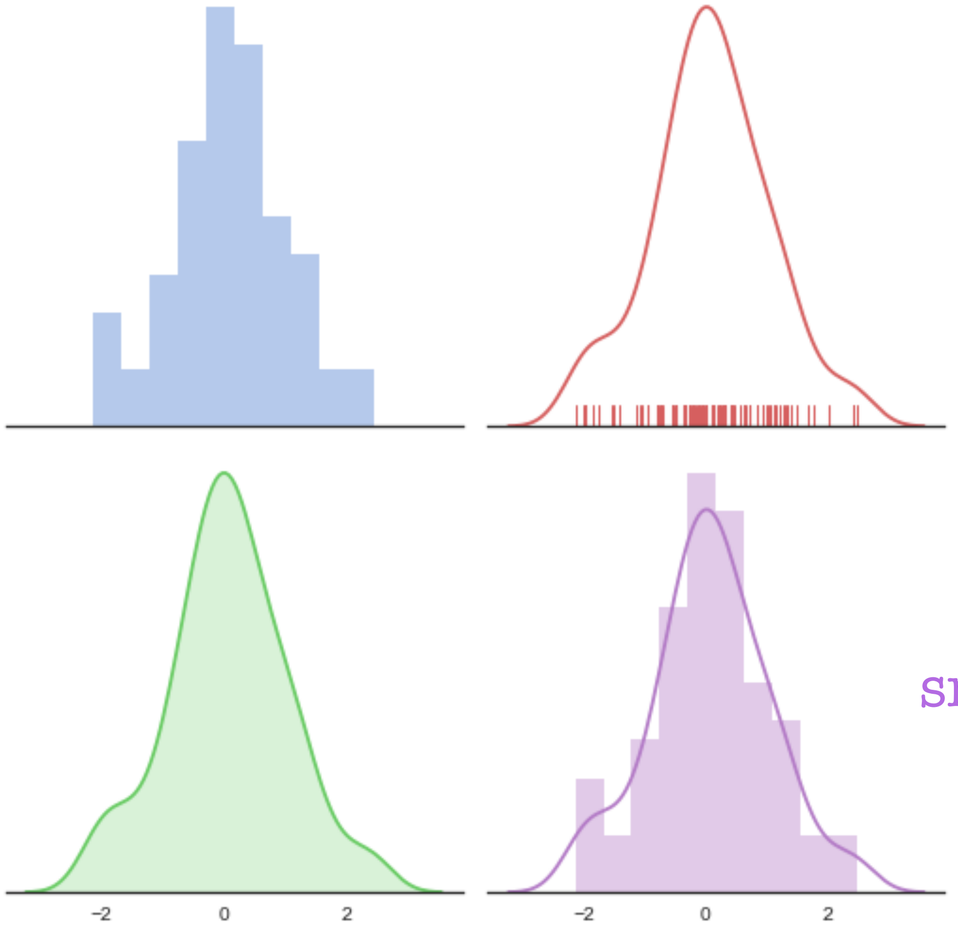

```
sns.distplot(x, hist=False)
```

Histograms are convenient but bin the data and are not differentiable - we'll look at kernel density plots later. Rug plots complement histograms and kernel density plots by showing the data.

http://seaborn.pydata.org/examples/distplot_options.html

Various ways to show 1D distributions

```
import matplotlib.pyplot as plt

import seaborn as sns
```



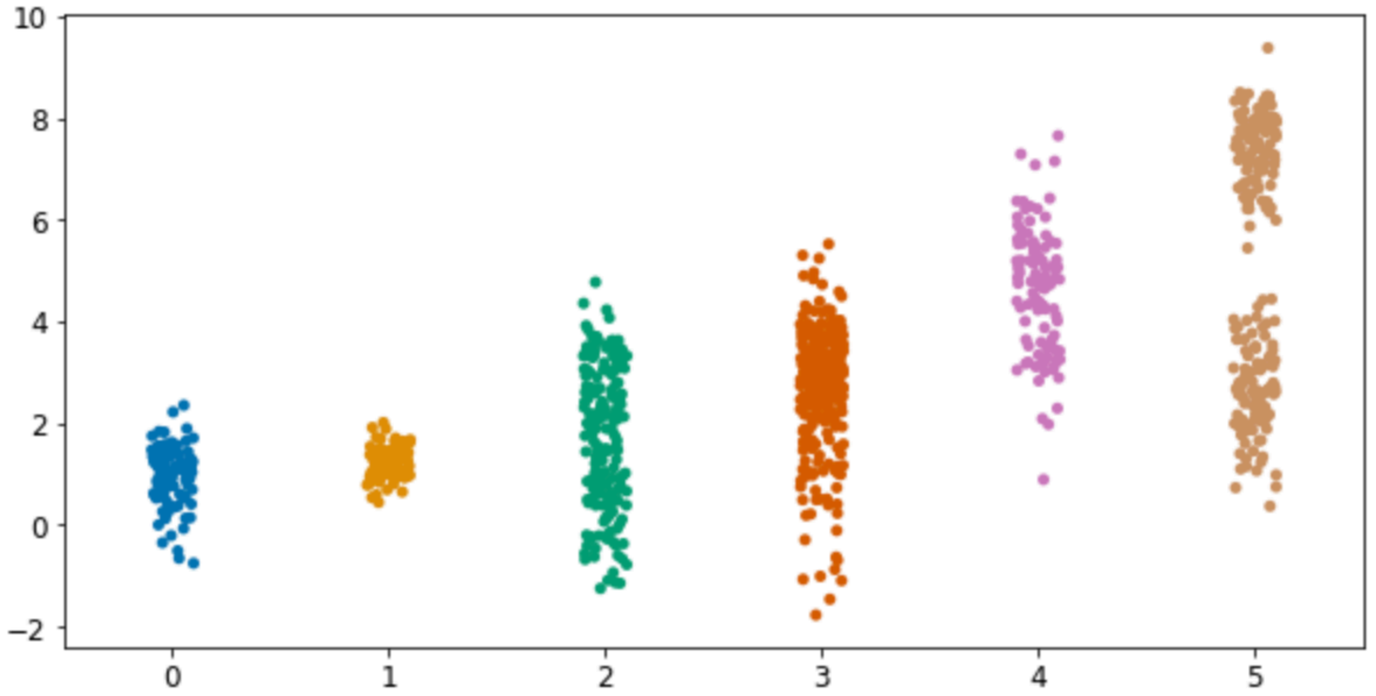

```
sns.distplot(x)
```

Histograms are convenient but bin the data and are not differentiable - we'll look at kernel density plots later. Rug plots complement histograms and kernel density plots by showing the data.

http://seaborn.pydata.org/examples/distplot_options.html

Comparing multiple 1D distributions

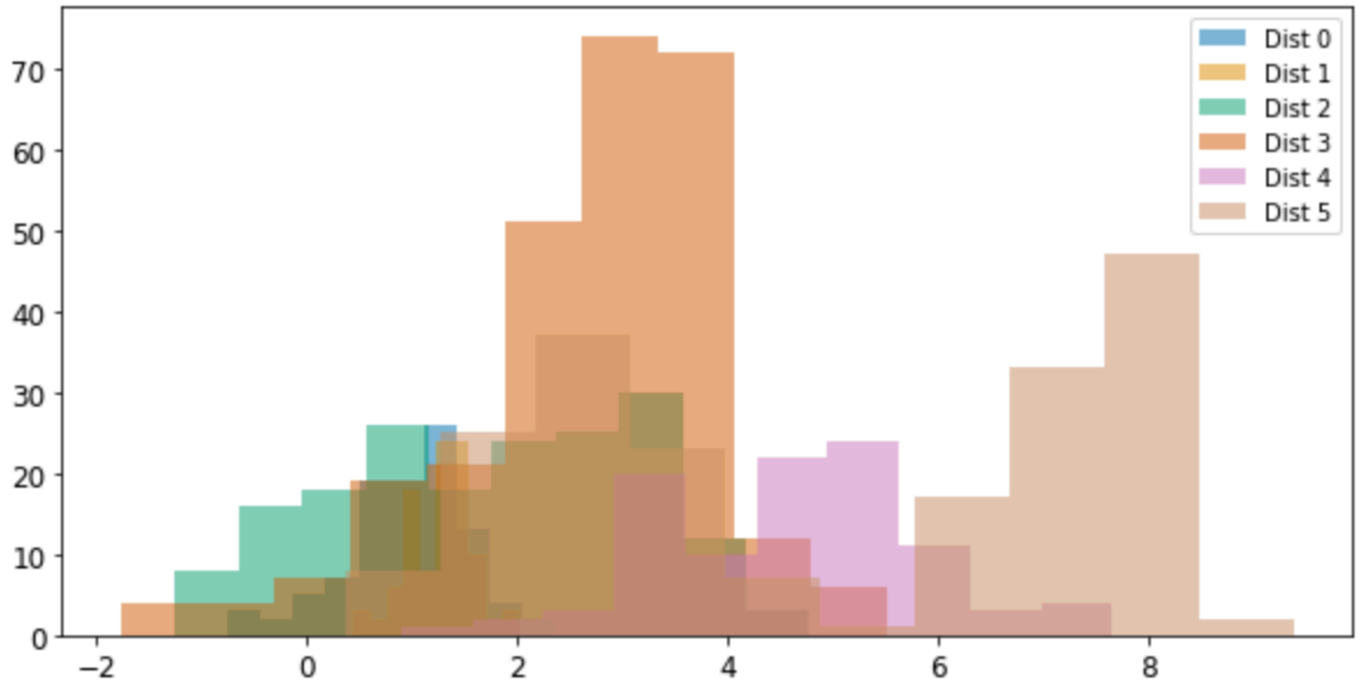
Scenario: 6 classes with measurements - how do we compare them?



See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

Scenario: 6 classes with measurements - how do we compare them?

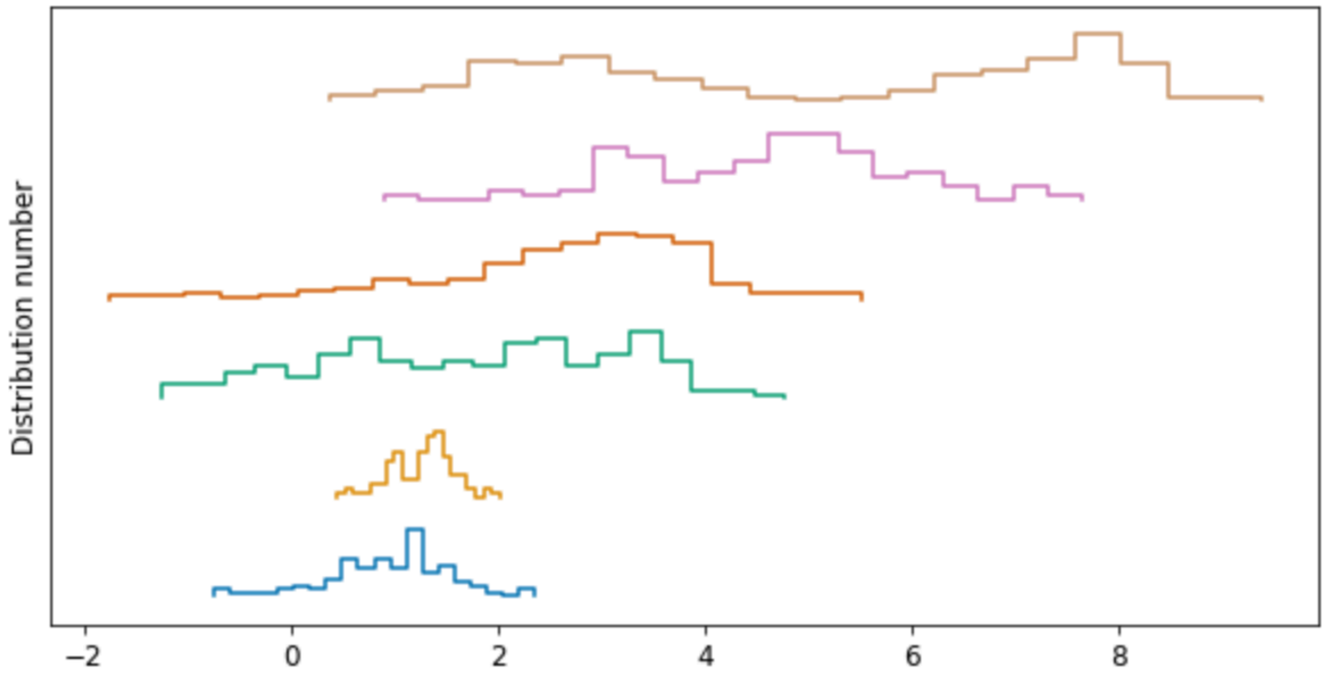


over-plotting histograms does not work well

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

Scenario: 6 classes with measurements - how do we compare them?

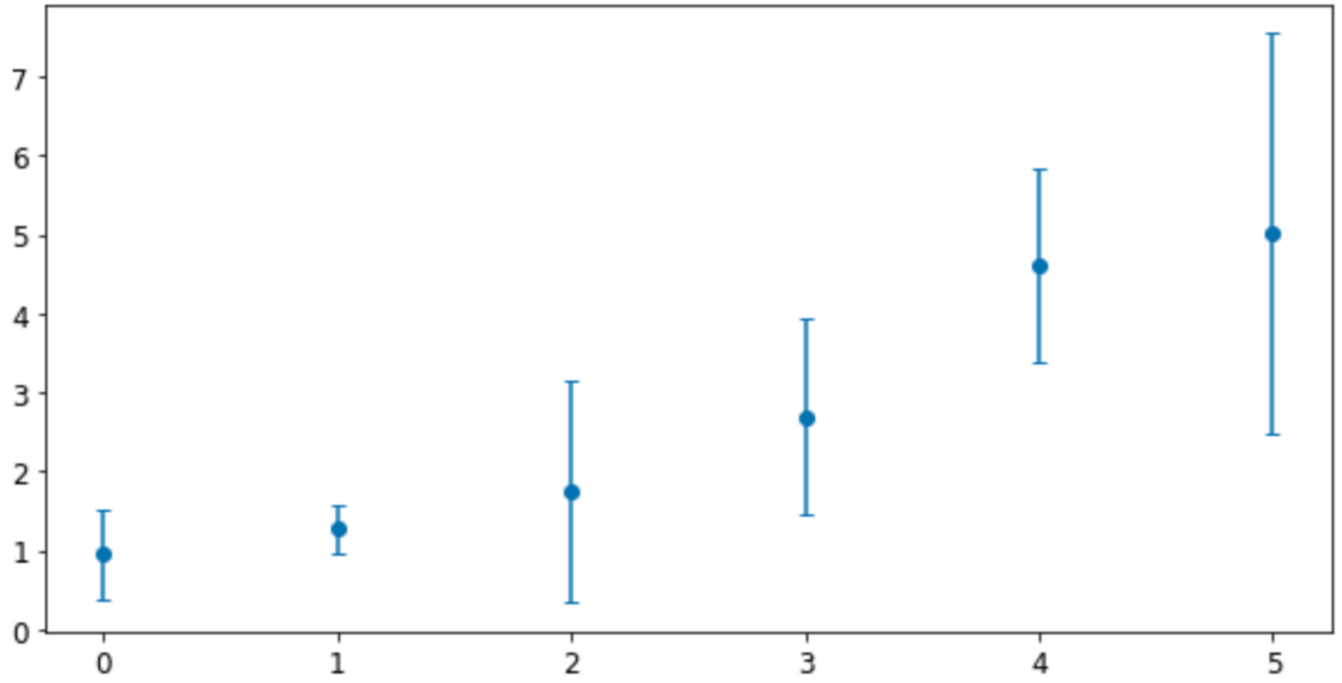


offsetting histograms is a possibility

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

Scenario: 6 classes with measurements - how do we compare them?

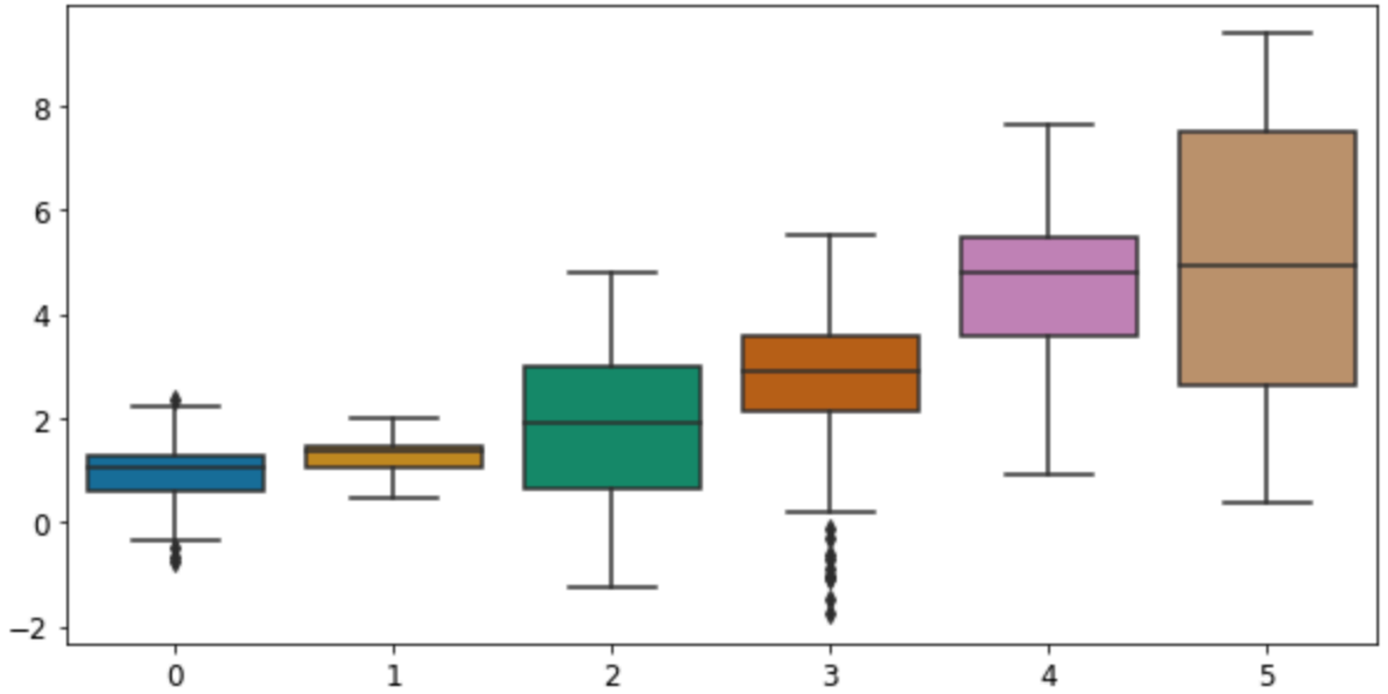


errorbars give some illustration of spread but lacks detail

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

Scenario: 6 classes with measurements - how do we compare them?

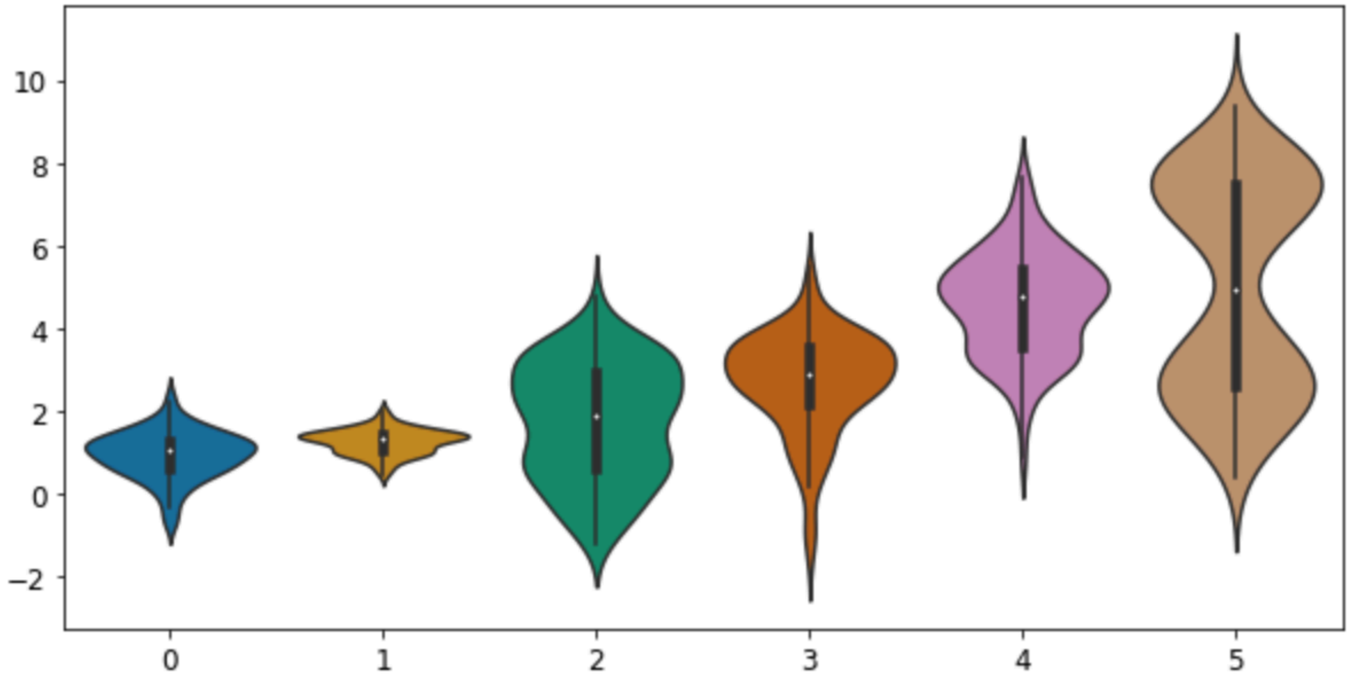


box plots are better - but only for uni-modal data

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

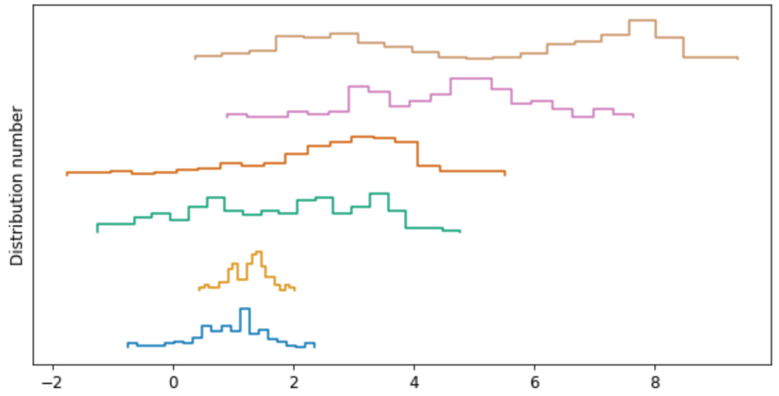
Scenario: 6 classes with measurements - how do we compare them?

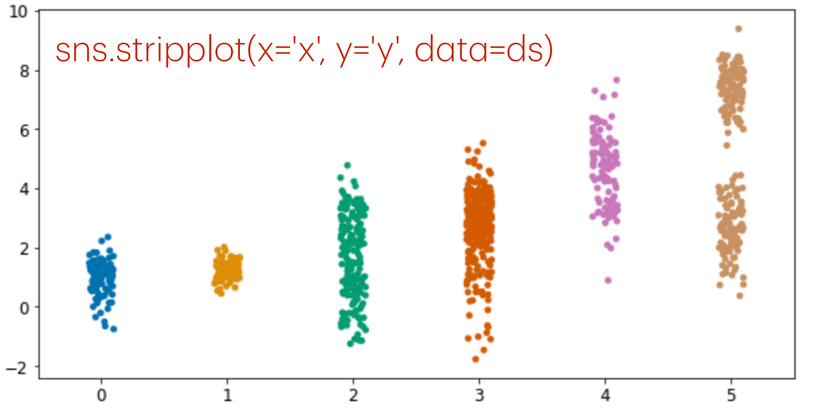


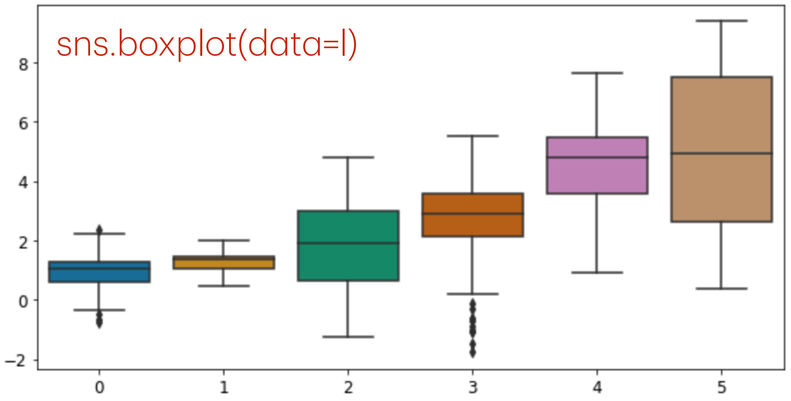
violin plots can be useful for this

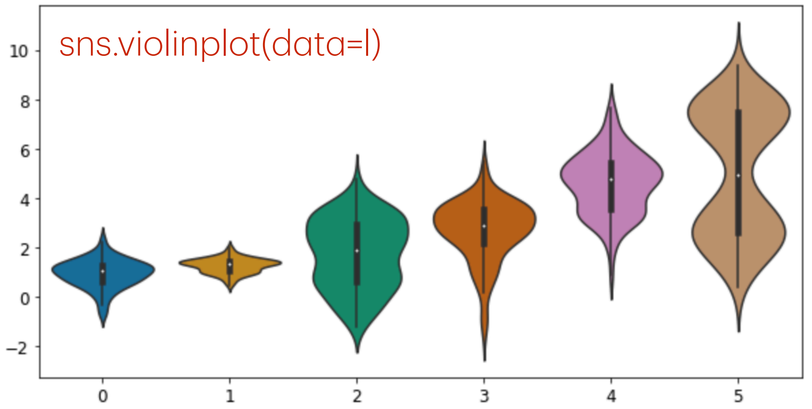
See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions



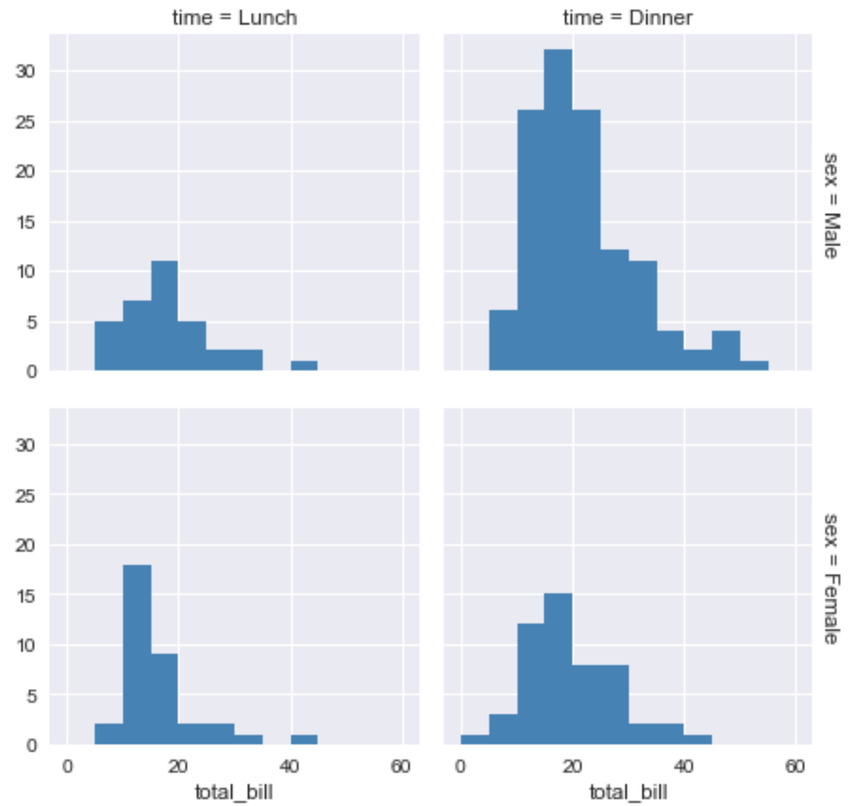






See the Distribution Illustration notebook in the Lecture directory

Various ways to show 1D distributions



Conditional plots (called FacetGrids in seaborn - in R the lattice package provides these) are very powerful tools for exploring complex multi-dimensional data. While shown for histograms here they can be used for all kinds of plots.

http://seaborn.pydata.org/examples/faceted_histogram.html

A quick interlude - try out:

- Get the file Datafiles/several_datasets.tsv from the course website.
- Read it in using e.g. astropy:

```
from astropy.table import Table
t = Table().read("several_datasets.tsv", format="ascii.fast_tab")
```

There are 13 datasets there - t['x1'], t['y1'] contains the x & y values for dataset 1 etc.

Calculate summary statistics (mean, standard deviation, correlation coefficient, maybe a linear fit) for each dataset - what do you conclude?

Let’s switch to Colab!

Notebook: MLD2025-04-Distribution illustrations

Direct link

Looking at odd data

x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

An example of *odd* data

x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

An example of *odd* data

x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

mean(x) = **9**, mean(y) = **7.5**

Var(x) = **11**, Var(y) = **4.12**

The correlation coefficient: **0.816**

The best-fit line: **y = 3 +0.5 x**

An example of *odd* data

x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

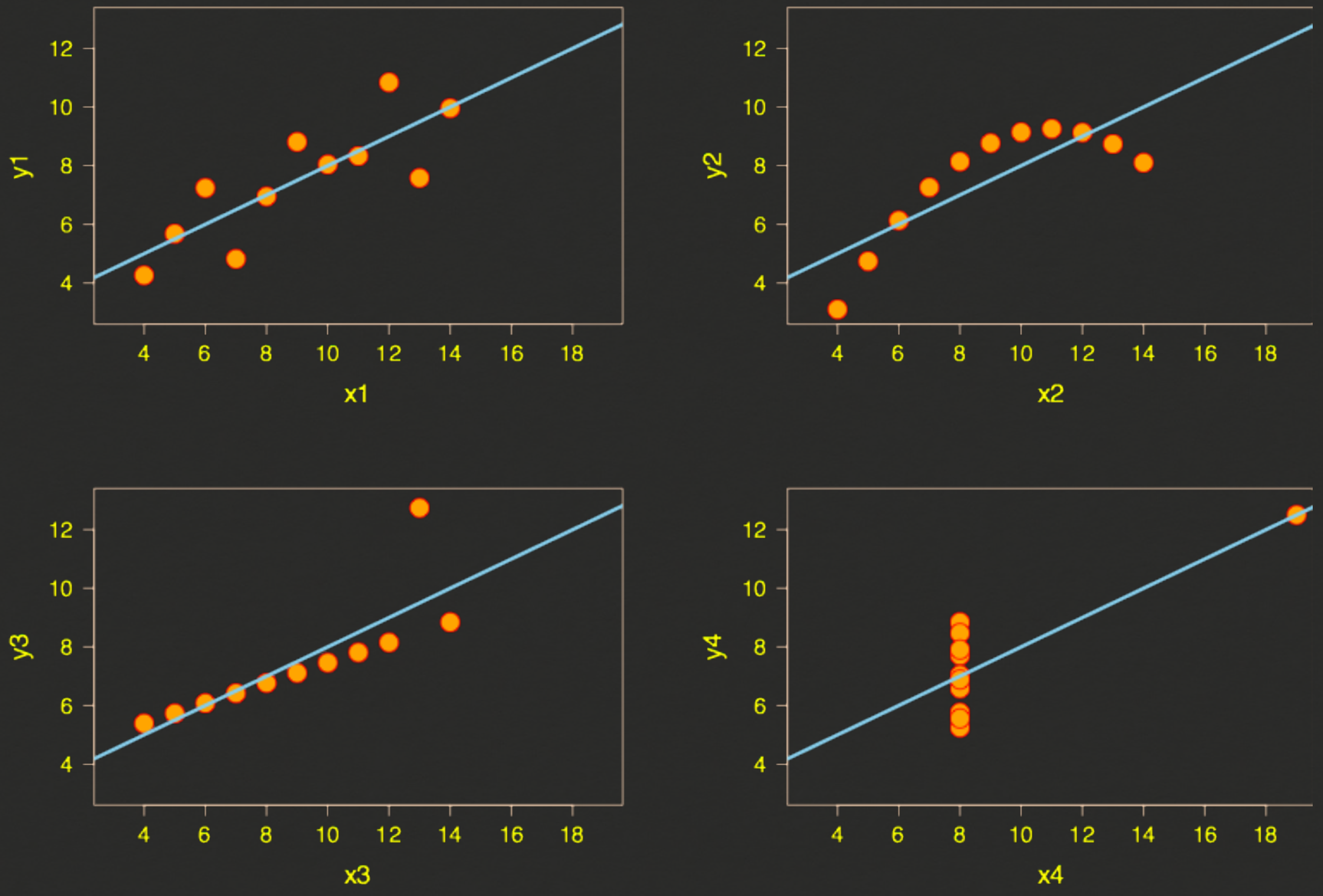
mean(x) = **9**, mean(y) = **7.5**

Var(x) = **11**, Var(y) = **4.12**

The correlation coefficient: **0.816**

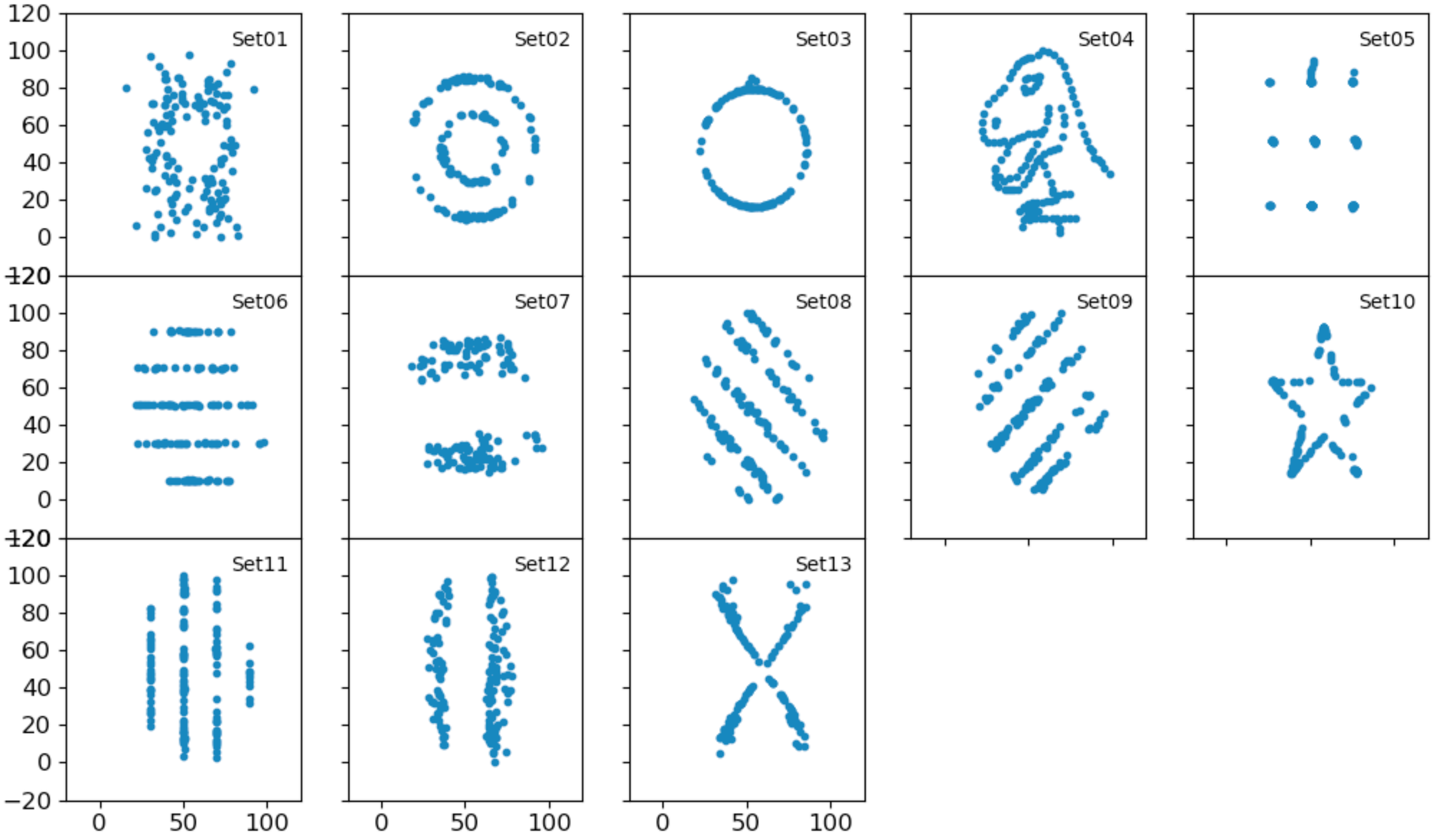
The best-fit line: **y = 3 +0.5 x**

So they seem to be very similar!



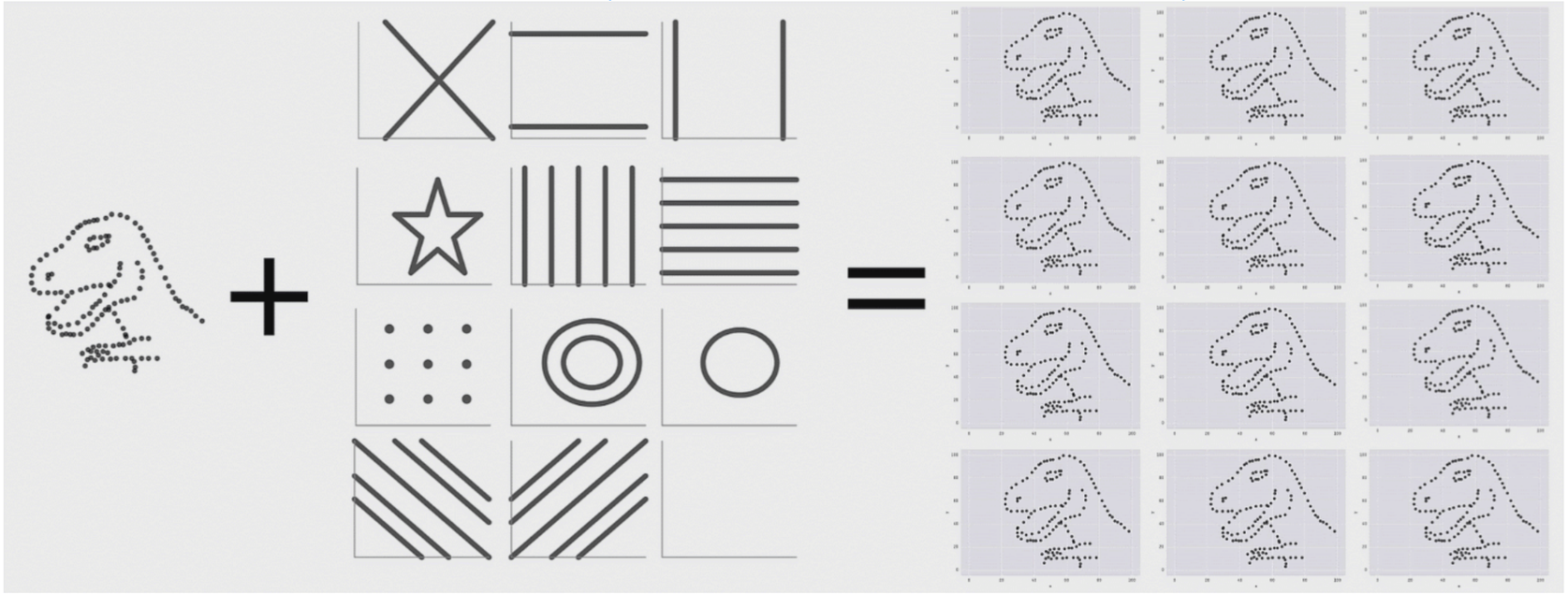
The bottom line: Always plot your data! (but you knew that already I hope!)
But how would you deal with this in N dimensions?

Visualisation - problem



But how did they do that?

From: https://www.autodeskresearch.com/publications/samestats

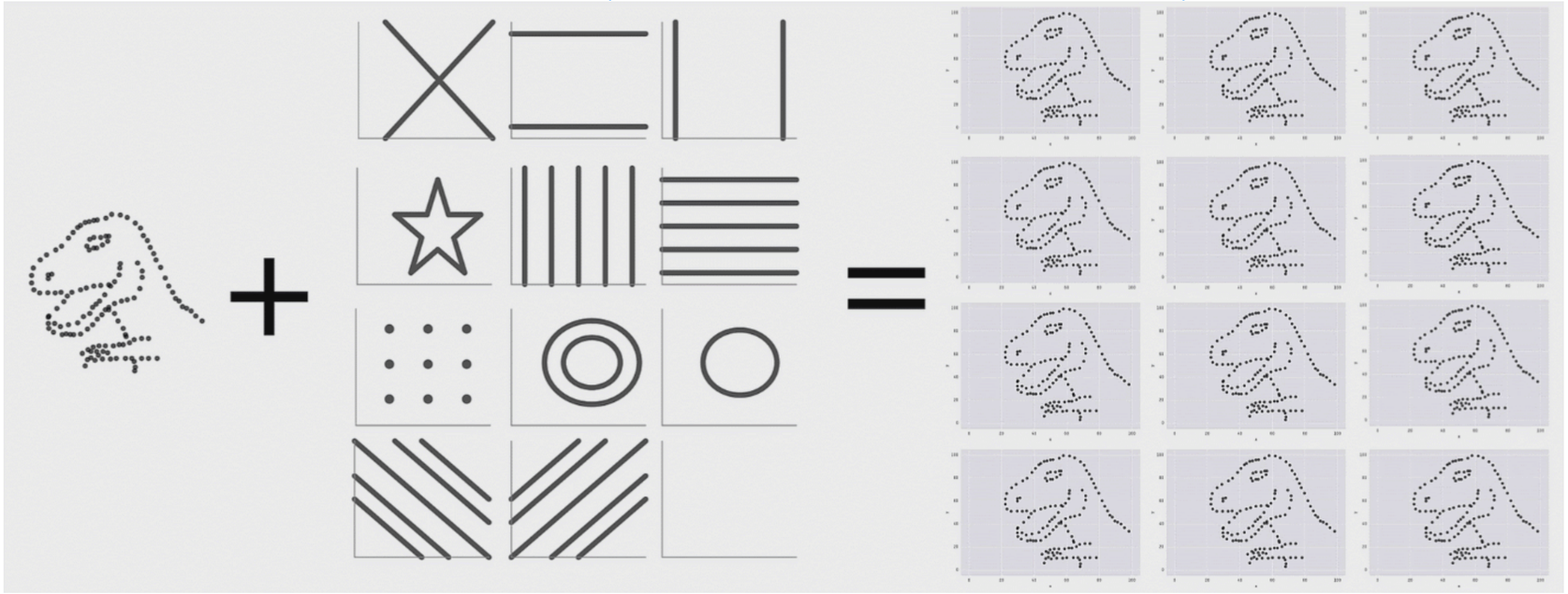


Start with a particular distribution, then shift points slightly making sure the summary statistics is the same and that you head towards a particular target statistic.

https://www.autodeskresearch.com/publications/samestats

But how did they do that?

From: https://www.autodeskresearch.com/publications/samestats

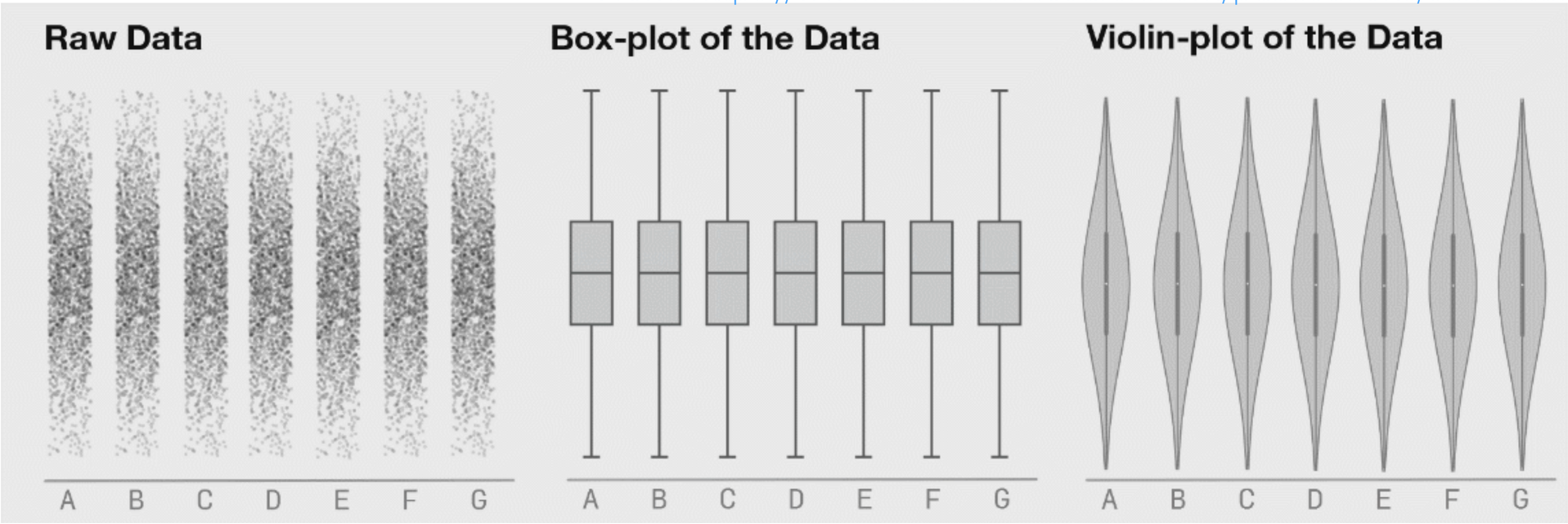


Start with a particular distribution, then shift points slightly making sure the summary statistics is the same and that you head towards a particular target statistic.

https://www.autodeskresearch.com/publications/samestats

Why boxplots can be untrustworthy

From: https://www.autodeskresearch.com/publications/samestats

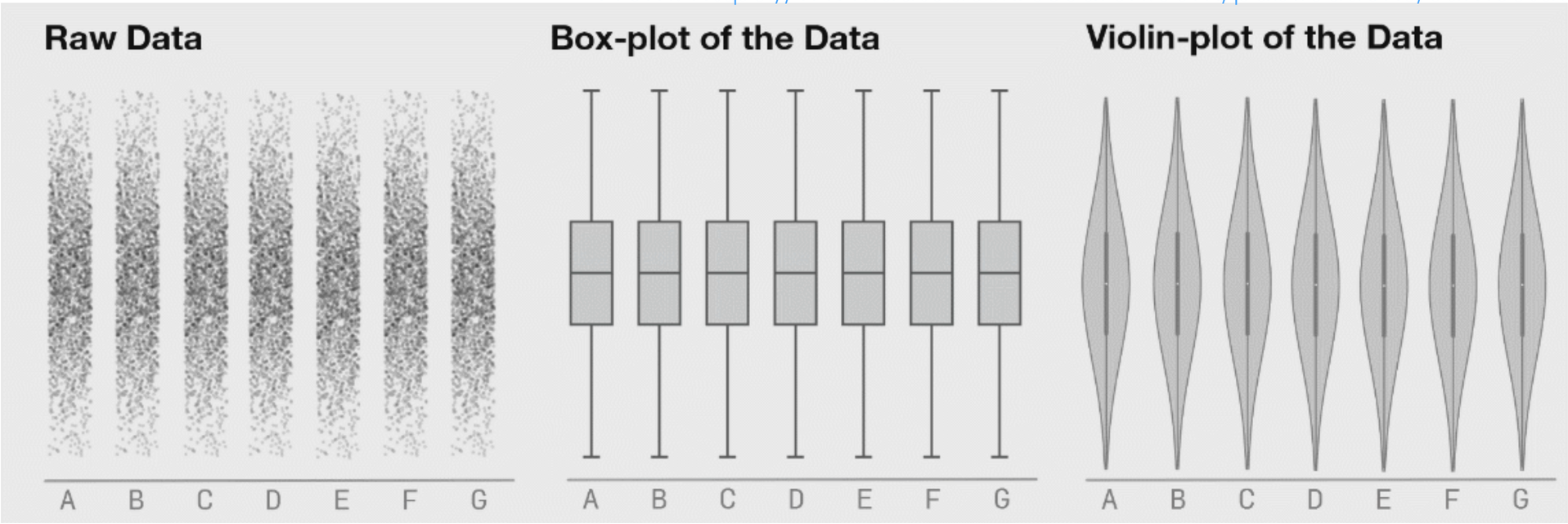


Again - same technique. Read more at

https://www.autodeskresearch.com/publications/samestats

Why boxplots can be untrustworthy

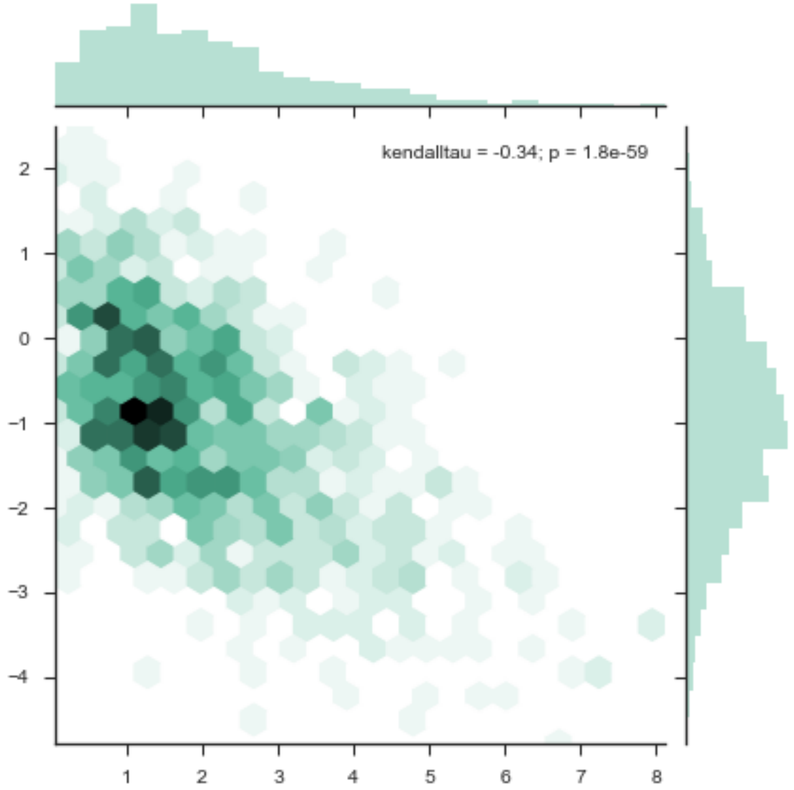
From: https://www.autodeskresearch.com/publications/samestats

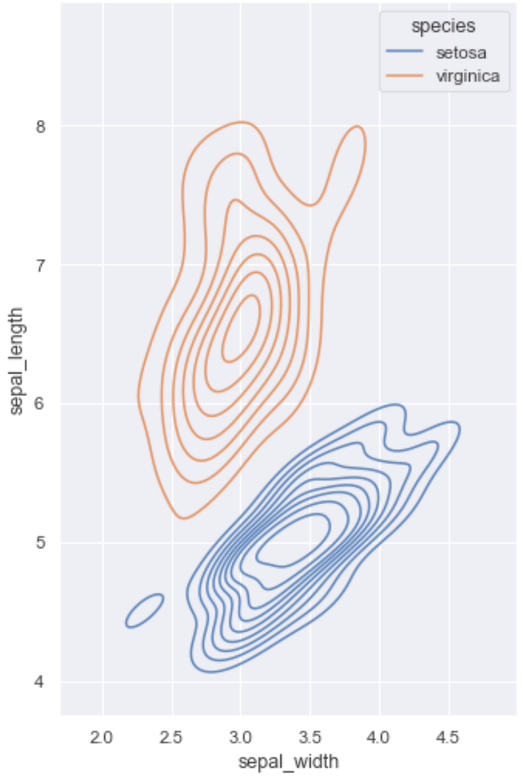


Again - same technique. Read more at

https://www.autodeskresearch.com/publications/samestats

Various ways to show 2D distributions

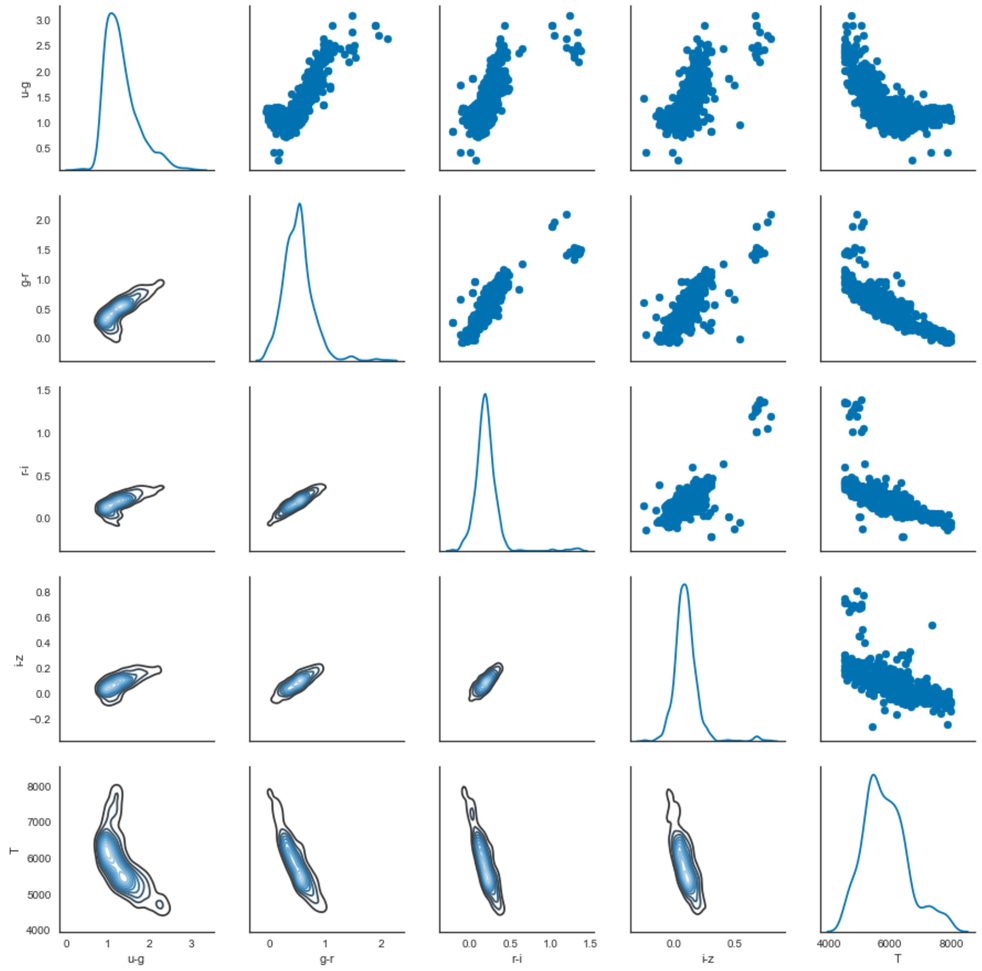




Hex-bin plots (left), or kernel density maps (right) are powerful ways to view 2D data structures and should be in your toolbox.

http://seaborn.pydata.org/examples/hexbin_marginals.html and https://seaborn.pydata.org/examples/multiple_bivariate_kde.html

Multi-dimensional data

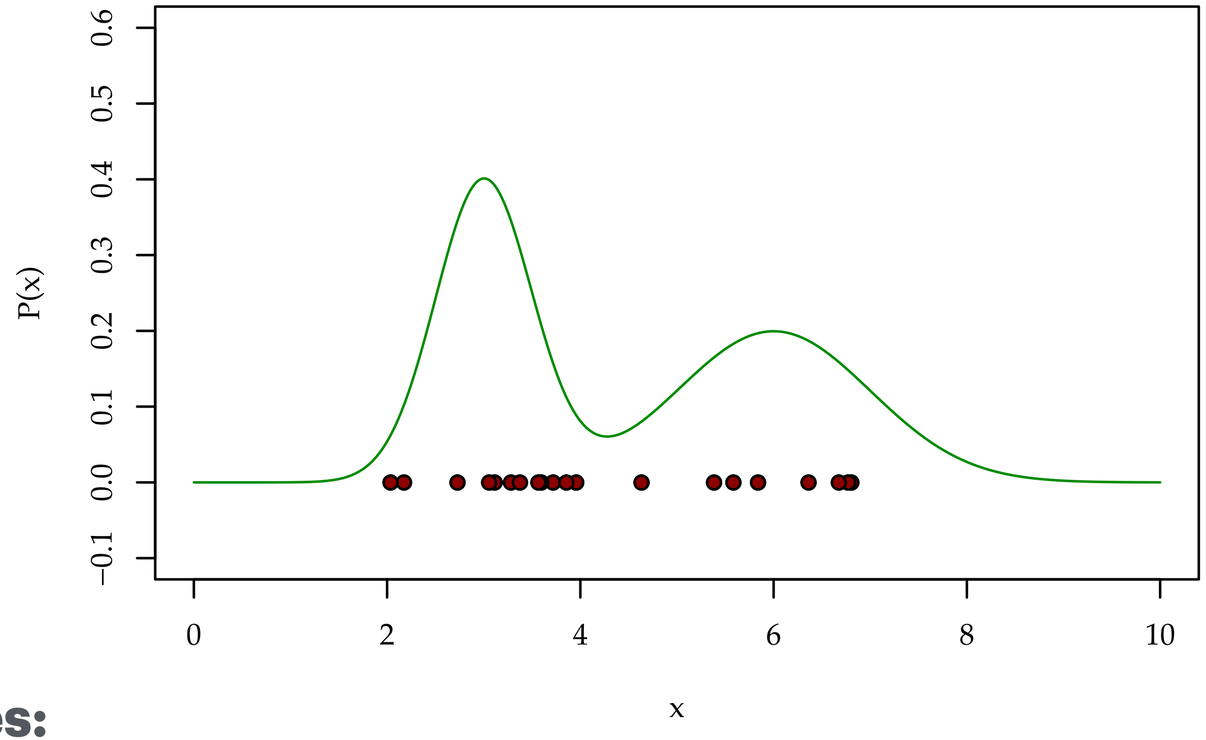


Scatter-plot matrices or pairplots are very powerful ways to explore the inter-relationship of many variables quickly. But always keep in mind that the true correlation might be hidden in these projections.

https://seaborn.pydata.org/generated/seaborn.pairplot.html

Density estimation

How do you recover the “true” density distribution?



Two main approaches:

Non-parametric - where no assumptions about the functional form is made.

Parametric - where we do assume a particular functional form (two Gaussians for instance), and fit for the parameters of the function

Density estimation - formally

Formally a density estimation problem is to find/learn a function

$$p_{\text{model}}(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$$

In most cases it is natural to interpret $p_{model}(\boldsymbol{x})$ as a probability density function.

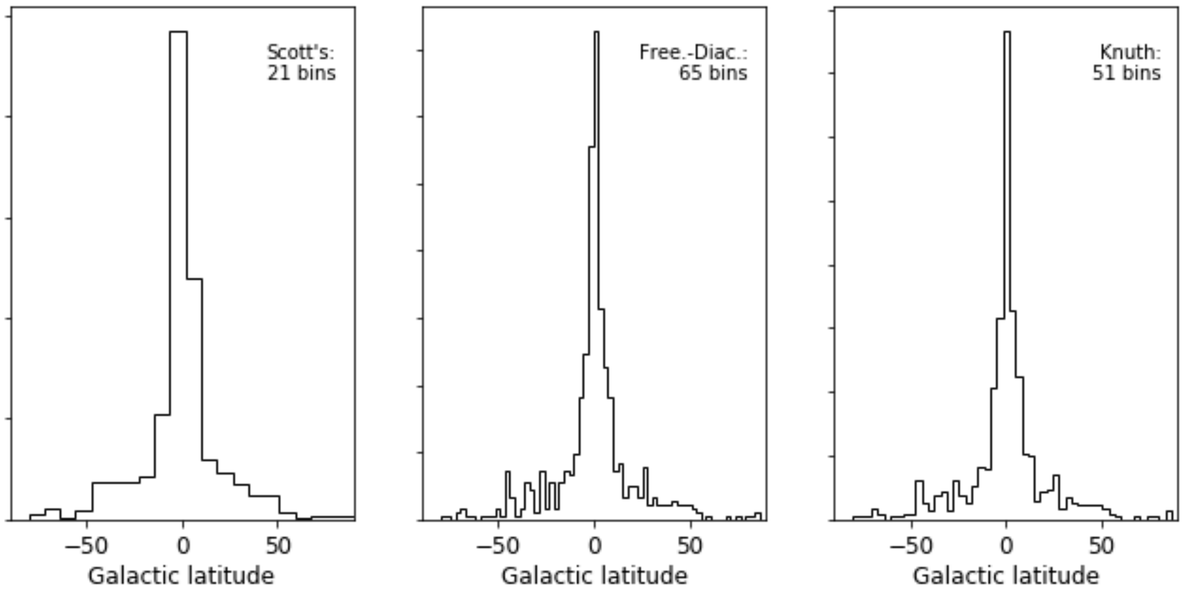
As soon as you have this function you could use it to predict missing data, take the derivative of it, create new data, or use it for visualisation - among other things.

Simple Non-parametric estimators

Histograms

Count objects in bins. Assume the bin size of bin i is Δ_i and that there are n_i objects in this bin. The probability of the value x_i corresponding to that bin is then:

$$p_i(x_i) = \frac{n_i}{N\Delta_i}$$

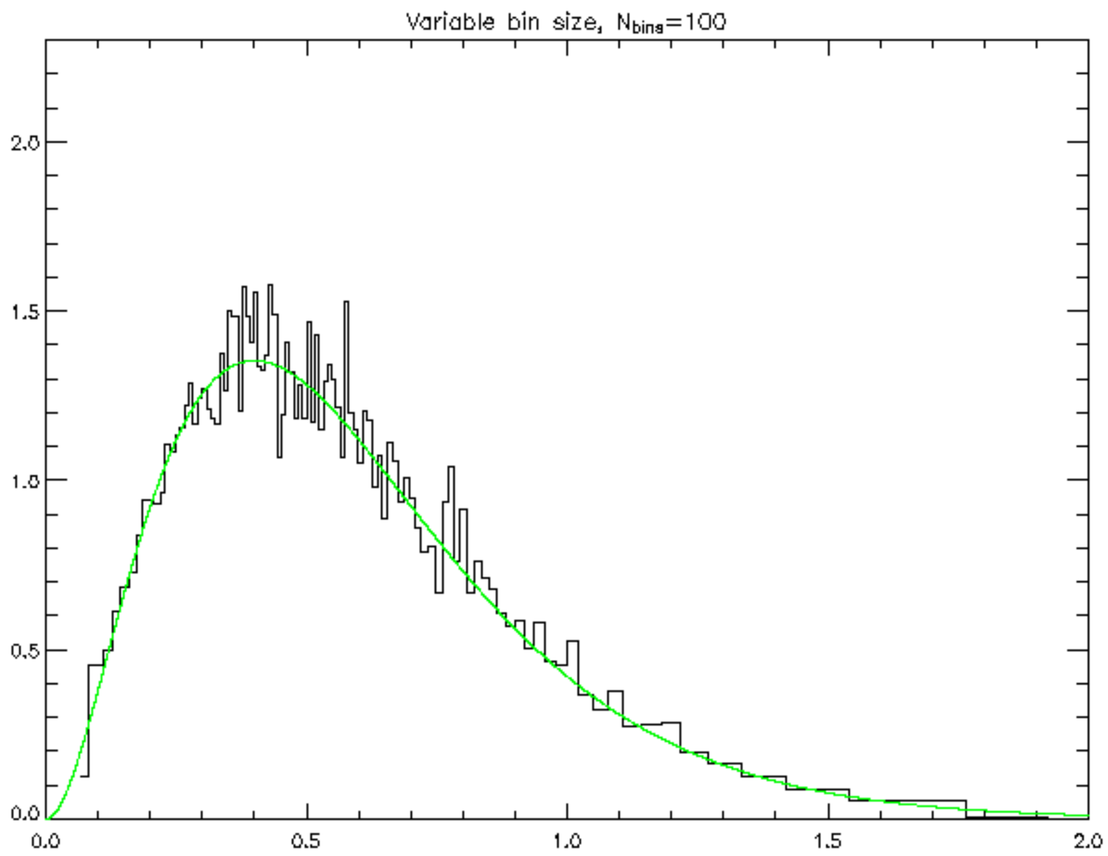


Histogram Problems

We can adapt the bin sizes to the data, but even so there are two major problems with the histogram:

a) It is discontinuous

b) It does not scale well to higher dimensions!

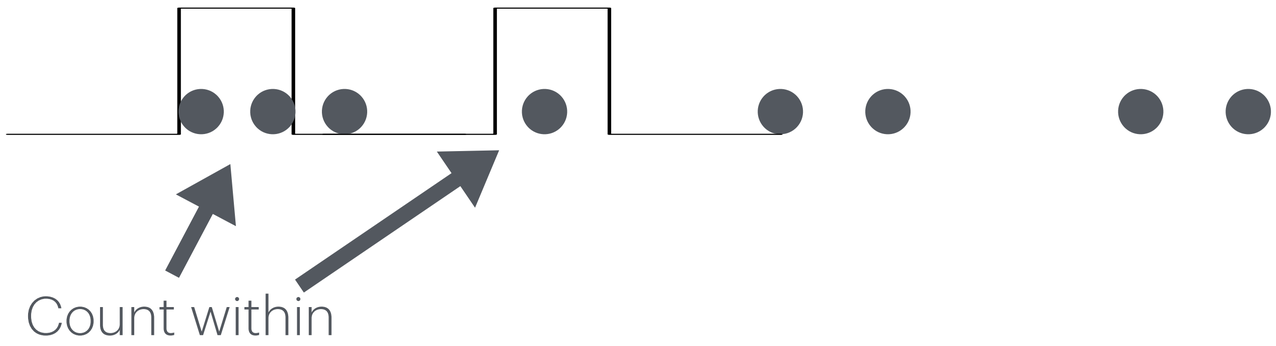


But it is still very useful! [some authors disregard it unnecessarily]

To do better we need to refine our density estimator.

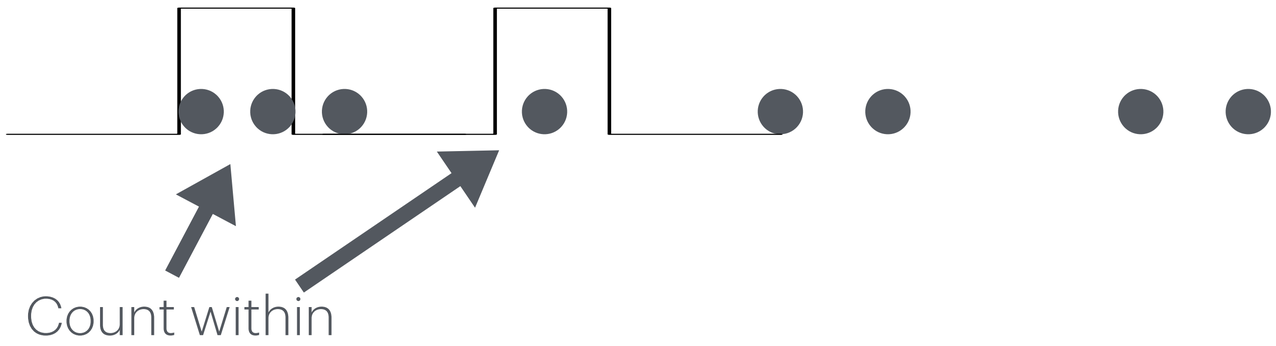
Histograms & kernels

We want to add together effects from points close to x.



Histograms & kernels

We want to add together effects from points close to x.

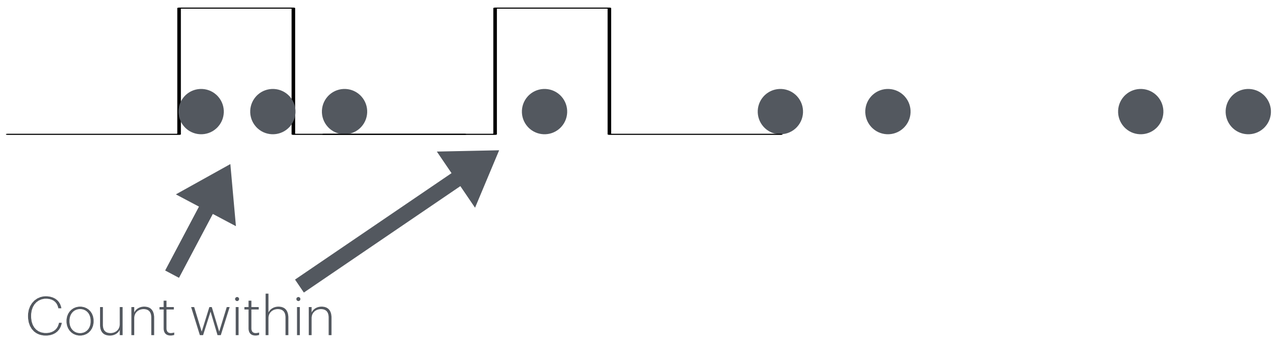


Define a k(x) so that:

$$k(\mathbf{u}) = 1 \text{ iff } \forall |u_i| < 1/2$$

Histograms & kernels

We want to add together effects from points close to x.



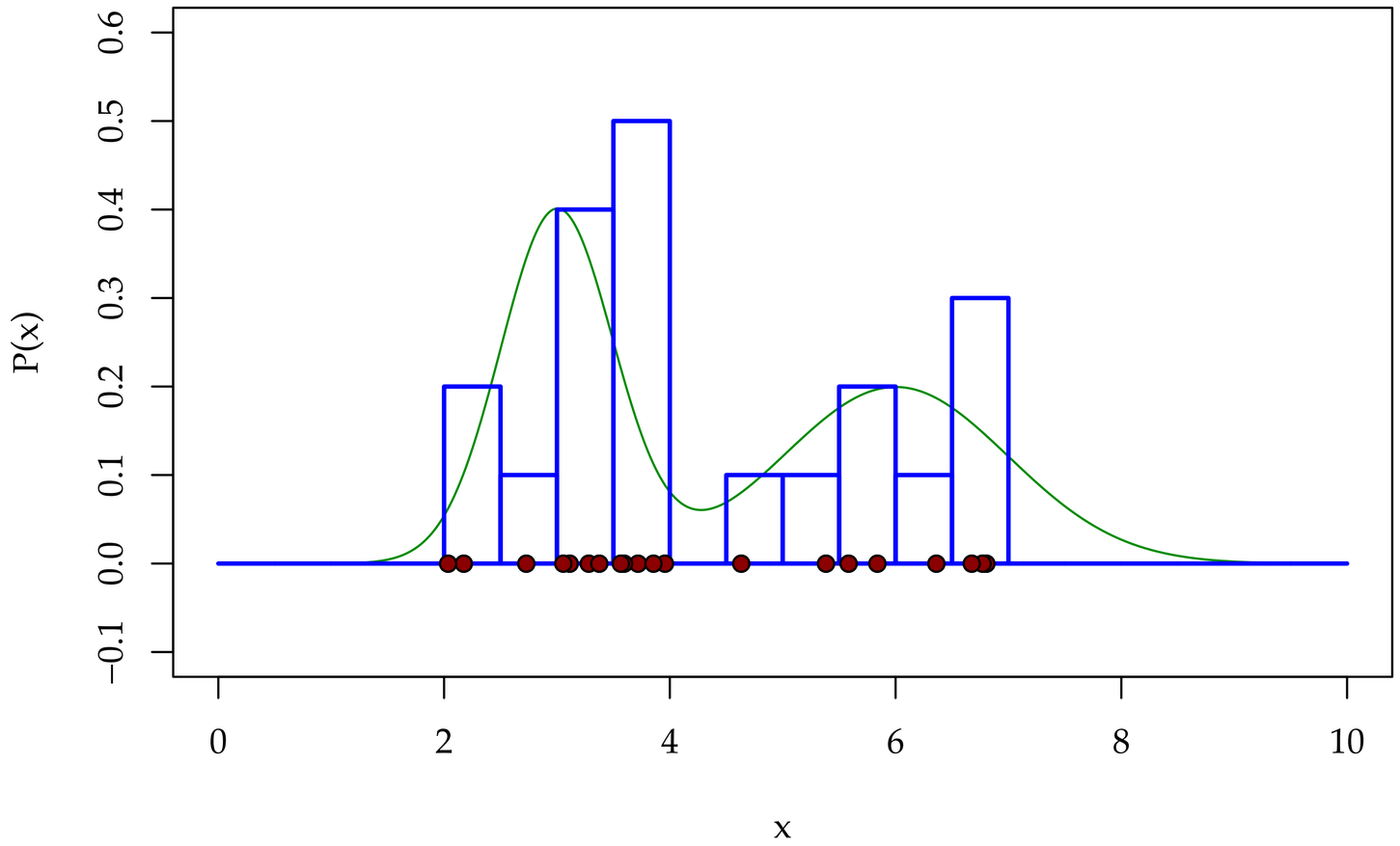
Define a k(x) so that:

$$k(\mathbf{u}) = 1 \text{ iff } \forall |u_i| < 1/2$$

Then the histogram (number of counts) can be defined as:

$$\#(\mathbf{x}) = \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_i}\right)$$

Histograms



What we are doing here is adding one box for each data point. This is our ‘kernel’.

Histograms & kernels

The histogram (number of counts) can be defined as:

$$\#(\mathbf{x}) = \sum_{i=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_i}\right)$$

Recasting this as a probability distribution we get:

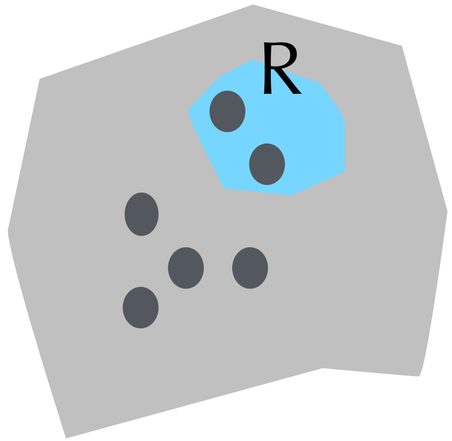
$$p_i(x_i) = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{h}K\left(\frac{x_i - x_j}{h}\right) \qquad K(x) = \begin{cases} 1, & |x| \leq \frac{1}{2} \\ 0, & |x| > \frac{1}{2} \end{cases}$$

this is a good starting point for density estimators

Simple Non-parametric estimators

Kernel estimators

Like in the histogram method we want to use objects near to a point to help us estimate the density there. If we look at a small region:



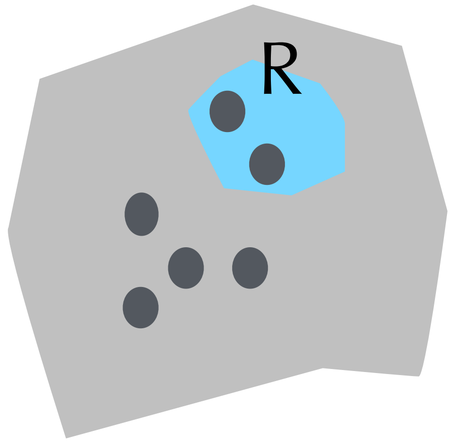
N observations in total
K inside R
V is volume of R

$$p(\mathbf{x}) = \frac{K}{NV}$$

Simple Non-parametric estimators

Kernel estimators

Like in the histogram method we want to use objects near to a point to help us estimate the density there. If we look at a small region:



N observations in total
K inside R
V is volume of R

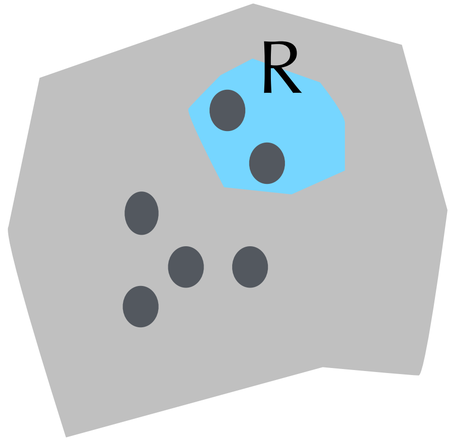
$$p(\mathbf{x}) = \frac{K}{NV}$$

So we can either fix K or fix V

Simple Non-parametric estimators

Kernel estimators

Like in the histogram method we want to use objects near to a point to help us estimate the density there. If we look at a small region:



N observations in total
K inside R
V is volume of R

$$p(\mathbf{x}) = \frac{K}{NV}$$

So we can either fix K or fix V

Kernel estimators fix V and the **k-nearest neighbours estimator** fixes K.

These are one type of instance-based learning methods

Instance-based learning

Basically: methods that use all training data to evaluate new data.

E.g. compare a linear method:

$$\hat{y}(x_i) = \theta_0 + \sum_{j=1}^{N} \theta_j \phi_j(x_i)$$

with using the nearest neighbours:

$$\hat{y}(x_i) = \frac{1}{k} \sum_{j:x_j \in N_k(x_i)} y_j$$

Instance-based learning

These are very useful techniques and are often overlooked.

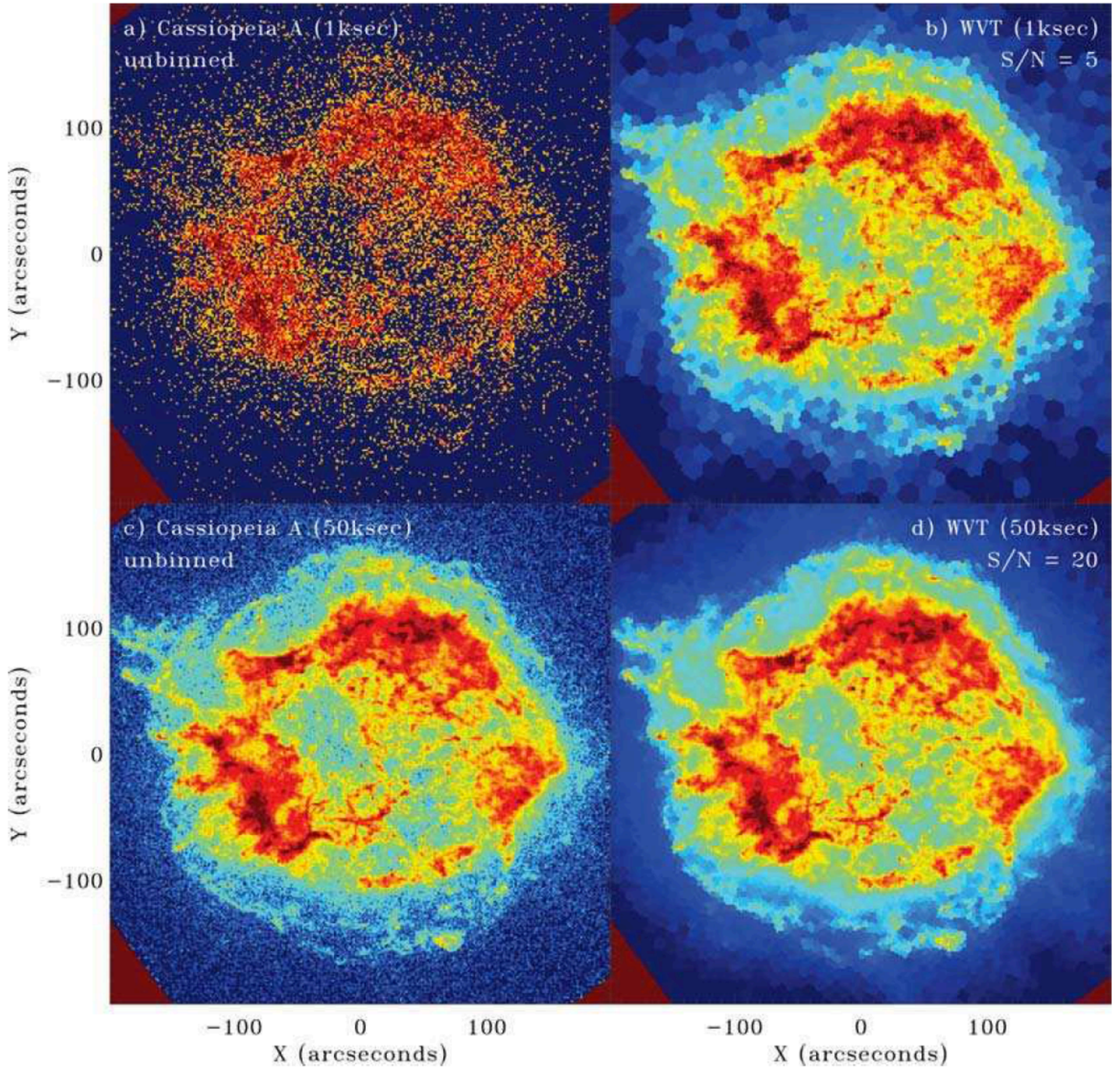
Pros:

- **Flexible** - can approximate very complex functions fairly easily
- **Can adapt to new data** - and throw away old
- **Simple**
- **Scales to large data**

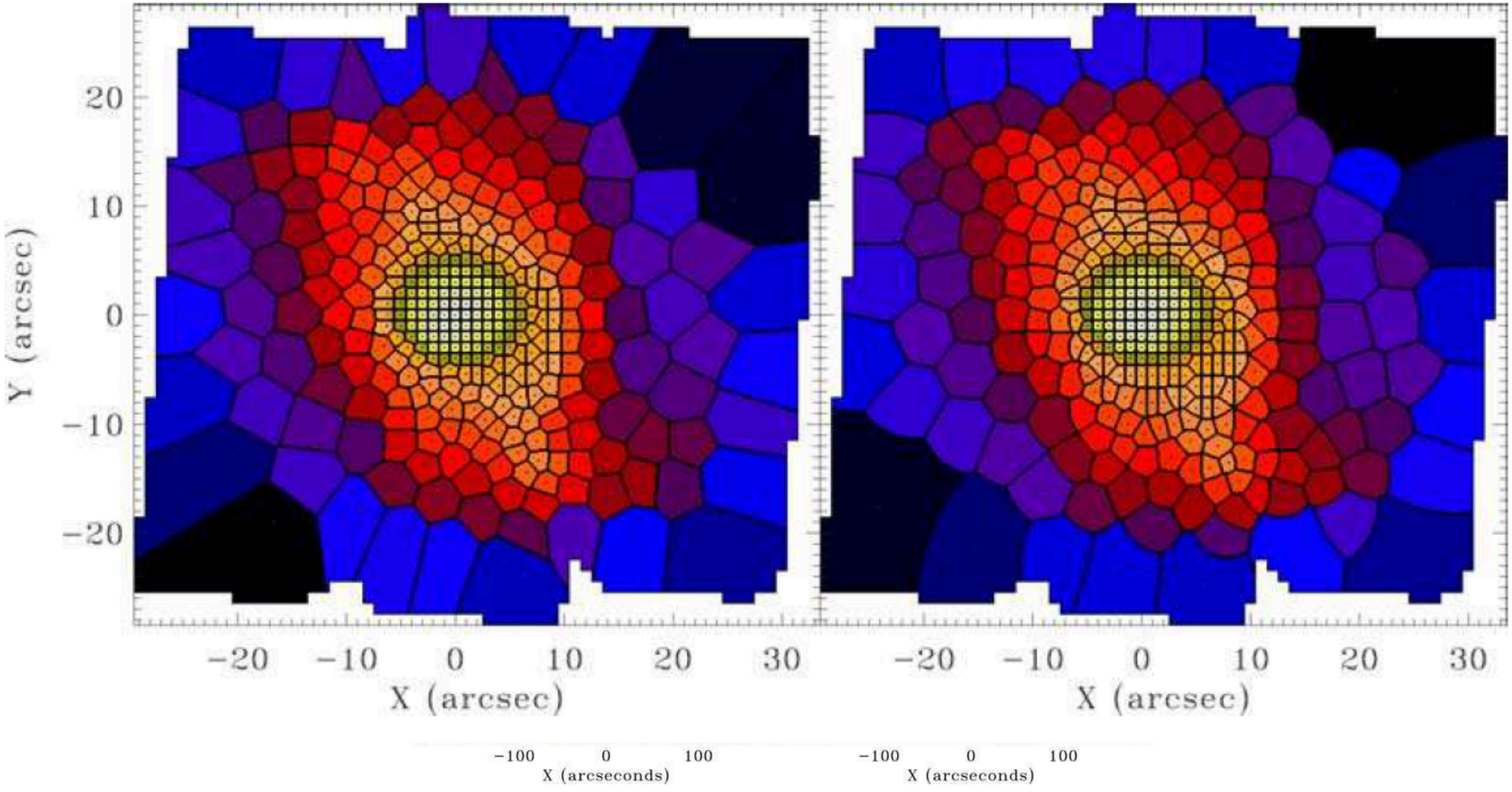
Cons:

- **Memory expensive** - all data needs to be in memory
- **Time expensive** - naïve implementations need comparison to all data when a new example is found
- **Opaque** - it is difficult to gain understanding from these

Diehl & Statler (2006)



Diehl & Statler (2006)



Kernel methods

We want to add together effects from points close to **x.**

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V_i}k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_i}\right)$$

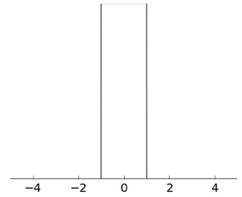
This is the basic equation for kernel estimation

Different kernels lead to somewhat different final distributions. The Epanchenikov kernel is the one that theoretically gives the minimum variance.

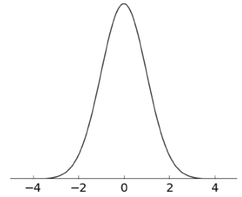
Kernel methods

We want to add together effects from points close to **x.**

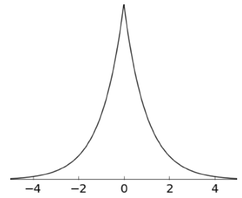
Square kernel: $k(\mathbf{u}) = 1 \text{ iff } \forall |u_i| < 1/2$



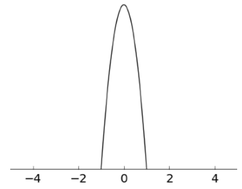
Gaussian kernel: $k(\mathbf{u}) = \frac{1}{(2\pi)^{1/D}} e^{-\|\mathbf{u}\|^2/2}$

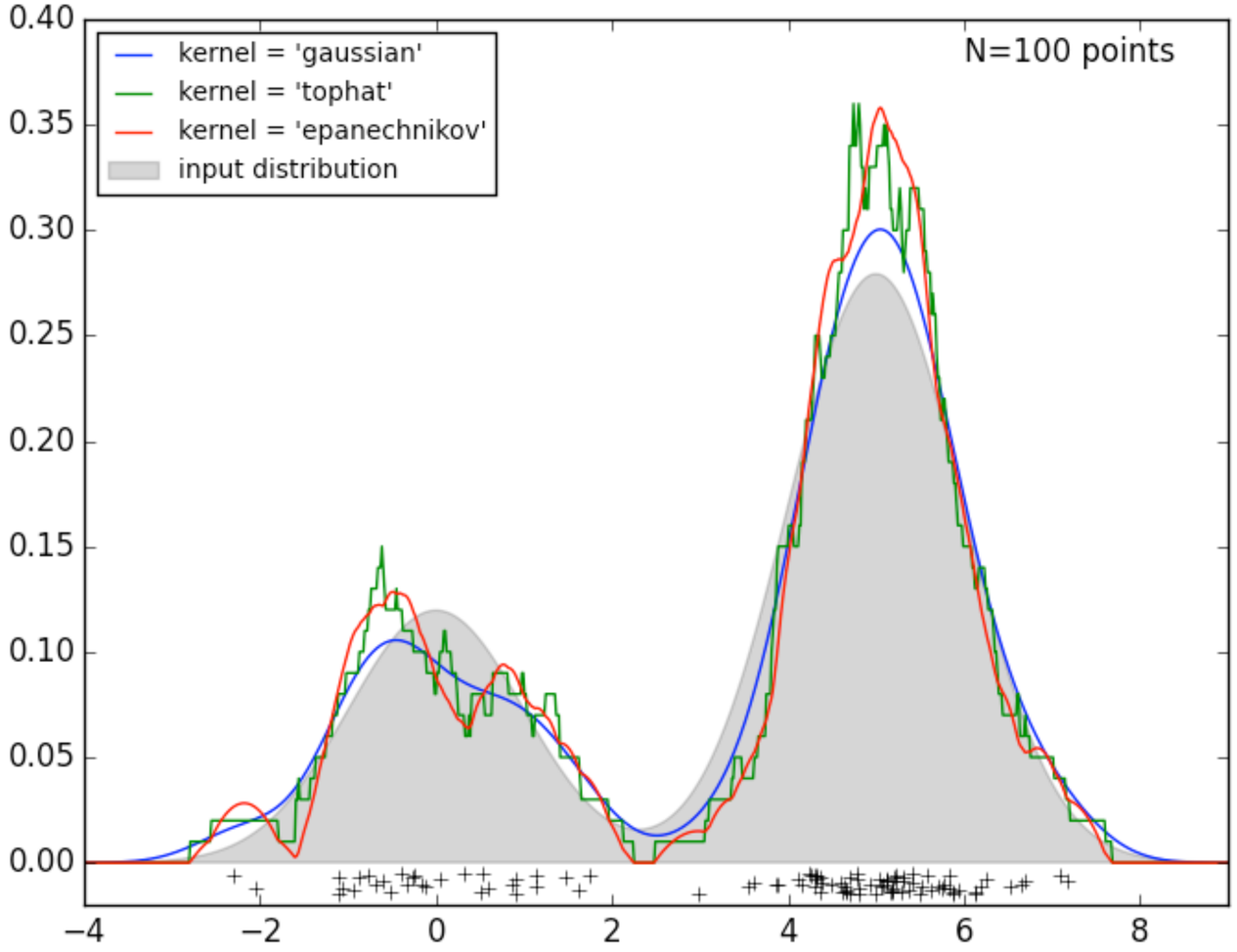


Exponential kernel: $k(\boldsymbol{u}) = \frac{1}{D! V_D(1)} e^{-|u|}$



Epanechnikov kernel: $k(\boldsymbol{u}) = \begin{cases} \frac{3}{4}\left(1 - u^2\right), & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$





Taken from: http://scikit-learn.org/stable/auto_examples/neighbors/plot_kde_1d.html

Kernel methods

We want to add together effects from points close to **x.**

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

This is the basic equation for kernel density estimation (KDE)

Note that this **has** a parameter: the bandwidth - changing this changes the results.

Kernel methods

We want to add together effects from points close to **x.**

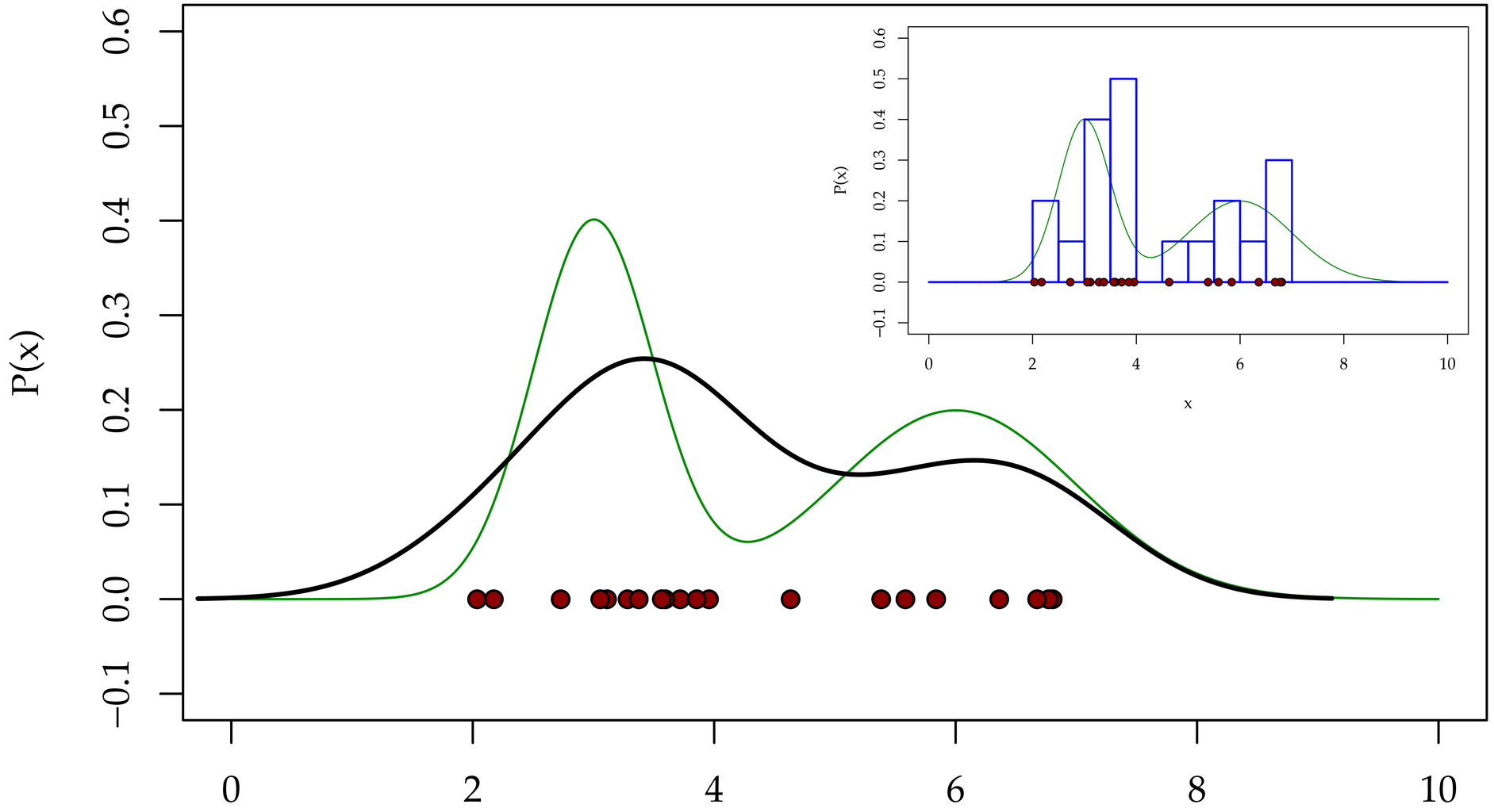
$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

Bandwidth

This is the basic equation for kernel density estimation (KDE)

Note that this **has** a parameter: the bandwidth - changing this changes the results.

Count the number weighted by a kernel



Practical use:

```
from sklearn.neighbors import KernelDensity

kde = KernelDensity(bandwidth=<bandwidth>,
kernel='gaussian').fit(X)
Xgrid = <same shape as X: N_samples x N_features>

log_dens = kde.score_samples(Xgrid)
```

Let’s switch to Colab!

Notebook: MLD2025-04-Density estimation

Direct link

Kernels to expansions - radial basis functions

Inspired by this we can expand the function p(x) using the kernel functions

$$p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$$

This gives radial basis function expansion. Very useful for interpolation.

$$f(\mathbf{x}) = \sum_{i=1}^{N}\beta_i K\left(\frac{\mathbf{x}-\boldsymbol{\xi}_i}{\lambda_i}\right)$$

[see e.g. chap. 6.7 in the Elements of Statistical Learning]

By making this a bit more flexible we get a very useful density estimation technique.

Gaussian Mixture Modeling

Gaussian mixture modelling

Let us assume that our unknown distribution can be written as a sum of Gaussians:

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^{M} \alpha_j N(x_i; \mu_j, \sigma_j)$$

We need to estimate the parameters and can write the log likelihood of all data as

$$\ln L = \sum_{i=1}^{N} \ln \left[\sum_{j=1}^{M} \alpha_j \, N(x_i; \mu_j, \sigma_j) \right]$$

Gaussian mixture modelling

Let us assume that our unknown distribution can be written as a sum of Gaussians:

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^{M} \alpha_j N(x_i; \mu_j, \sigma_j)$$

We need to estimate the parameters and can write the log likelihood of all data as

$$\ln L = \sum_{i=1}^{N} \ln \left[\sum_{j=1}^{M} \alpha_j \, N(x_i; \mu_j, \sigma_j) \right]$$

And this now needs to be maximised to determine the $3M\text{-}1$ parameters [why -1?]

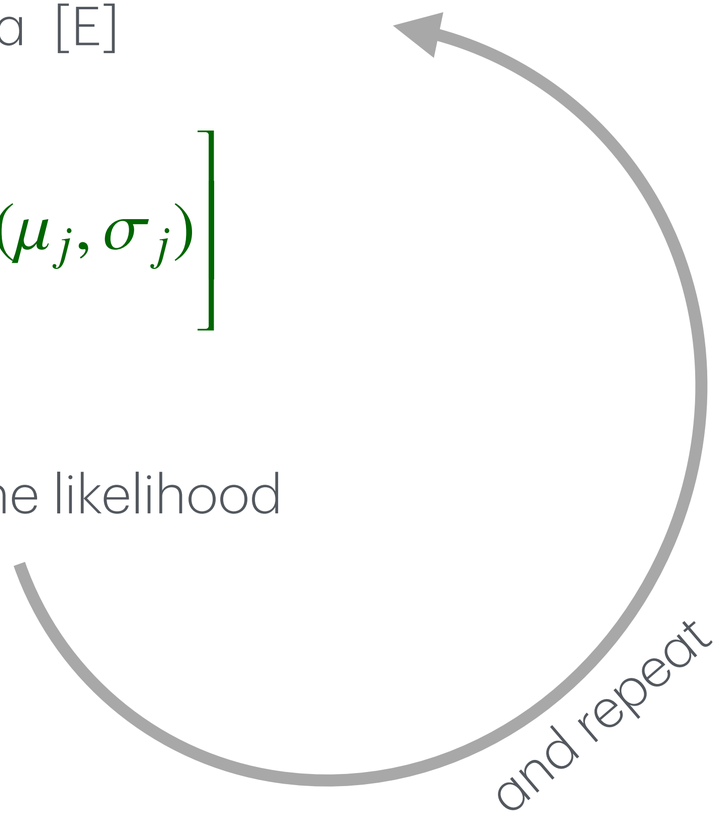
Gaussian mixture modelling

In this case, the parameters are found using the Expectation-Maximisation algorithm:

1. Calculate the log likelihood of the data [E]

$$\ln L = \sum_{i=1}^{N} \ln \left[\sum_{j=1}^{M} \alpha_j N(\mu_j, \sigma_j) \right]$$

2. Find the parameters that maximise the likelihood



Gaussian mixture modelling

In this case, the parameters are found using the Expectation-Maximisation algorithm:

1. Calculate the log likelihood of the data [E]

$$\ln L = \sum_{i=1}^{N} \ln \left[\sum_{j=1}^{M} \alpha_j N(\mu_j, \sigma_j) \right]$$

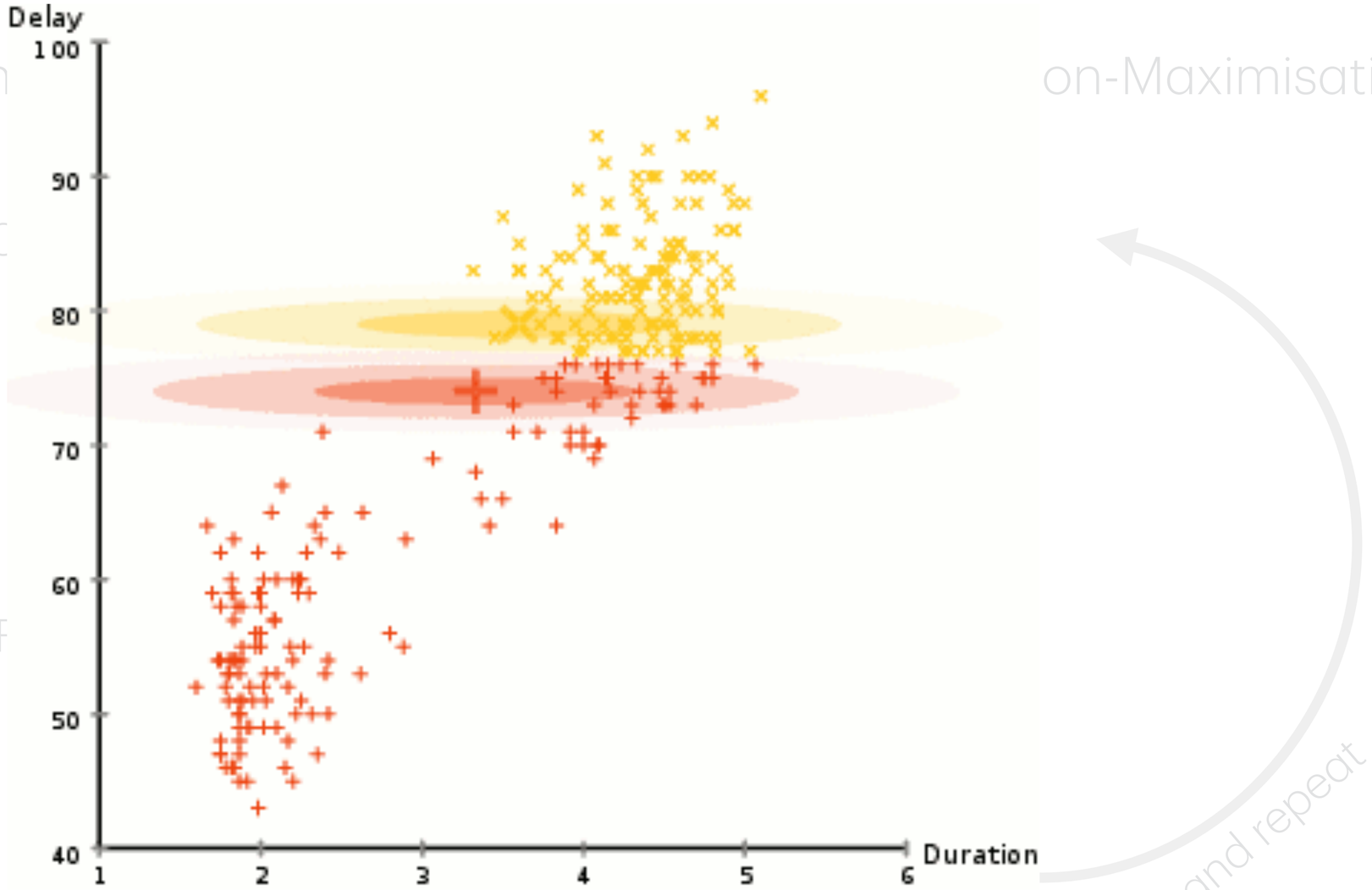
2. Find the parameters that maximise the likelihood

Gaussian mixture modelling

In this case, th on-Maximisation algorithm:

1. C

2. F



and repeat

https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm#/media/File:EM_Clustering_of_Old_Faithful_data.gif

Gaussian mixture modelling - in practice

```
from sklearn.mixture import GaussianMixture
```

(you can make mixtures of other things than Gaussians but they are not implemented in sklearn.mixture)

```
model = GaussianMixture(2)
```

Create a model with a given number of components - here two - this number can be known *a priori*, or will have to be determined as the kernel bandwidth.

```
res = model.fit(X)
```

fit the data - just as with regression or kernel density estimation

```
res.bic(X)
```

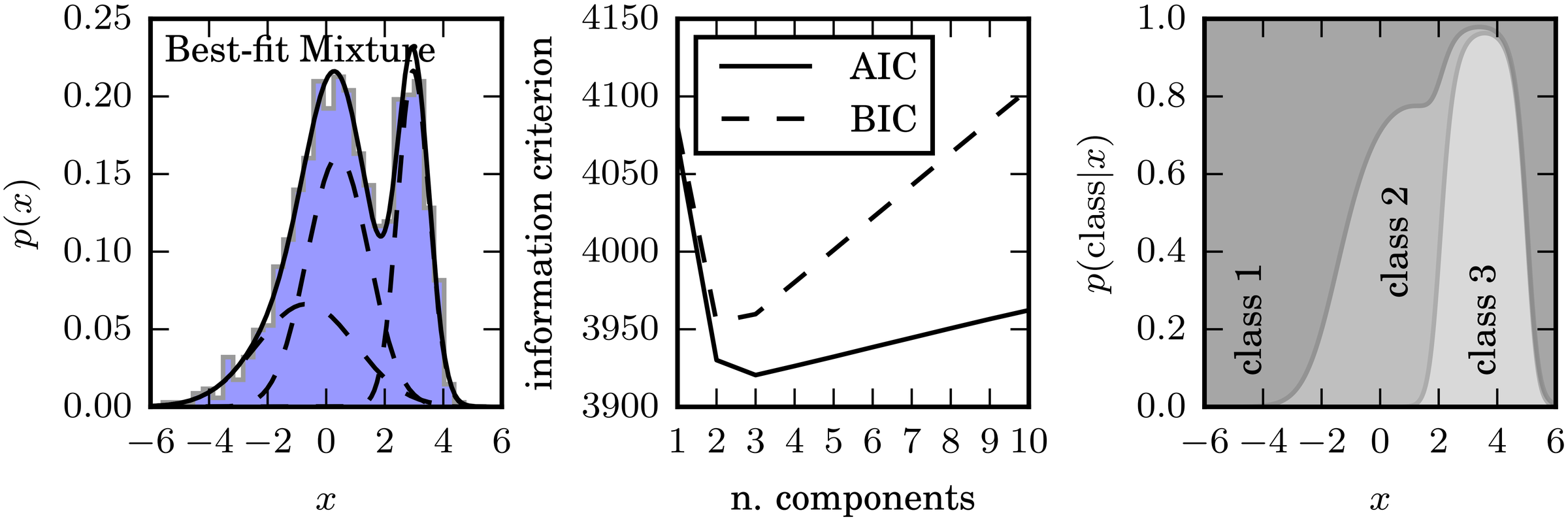
You can get the BIC or AIC easily

```
print(model.means_)
```

Getting the parameters of the Gaussians is usually quite important

Gaussian mixture modelling

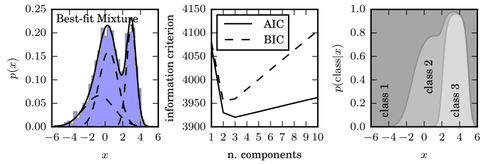
Using the BIC or AIC to find the best number of Gaussians



Based on Figure 4.2 from Statistics, Data Mining, and Machine Learning in Astronomy - see http://www.astroml.org/book_figures/chapter4/fig_GMM_1D.html

Gaussian mixture modelling

Use of AIC/BIC in practice:



X = X[:, np.newaxis]

```
from sklearn.mixture import GaussianMixture

# fit models with 1-10 components
N = range(1, 11)
models = [None for i in N]

for i in range(len(N)):
    models[i] = GaussianMixture(N[i]).fit(X)

# compute the AIC and the BIC
AIC = [m.aic(X) for m in models]
BIC = [m.bic(X) for m in models]
```

Based on Figure 4.2 from Statistics, Data Mining, and Machine Learning in Astronomy - see http://www.astroml.org/book_figures/chapter4/fig_GMM_1D.html

Gaussian mixture modelling

Getting fit results back:

```
logprob, p_component = M_best.score_samples(u)
```

`logprob:` The log p(x) for the overall model. So how likely data **u** is under this model.

`p_component:` The likelihood of the data **u** for each individual component.

If u has 10 elements and the GMM has 3 components, **`p_component`** will be 10x3.

Gaussian mixture modelling

When do we use GMM?

- When you think your distributions are reasonably described by Gaussians.

- As a way to describe distributions.

- When you want a simple approach that works often.

- For classification/assignment.

Gaussian mixture modelling - degeneracies

Consider the model

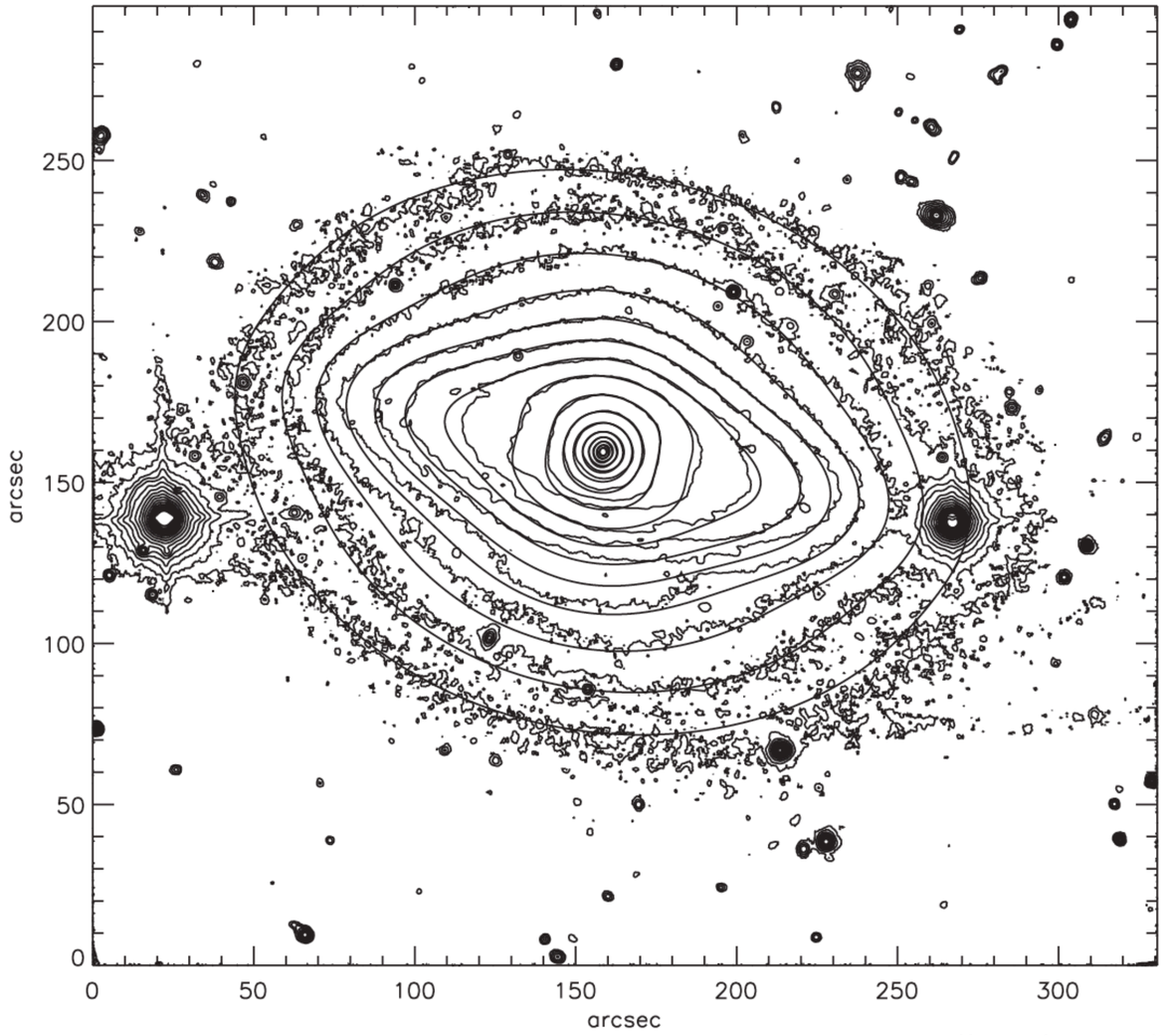
$$\ln L = \sum_{i=1}^{N} \ln \left[\sum_{j=1}^{M} \alpha_j \, N(x_i; \mu_j, \sigma_j) \right]$$

Assume now that $\mu_j = x_i$ for some i, j. In that case we have a term

$$p_j = \frac{1}{\sqrt{2\pi}\sigma_j}$$

which diverges for $\sigma_k \to 0$ - in which case that component takes all power.

Common example: Multi-gaussian expansion of galaxy images:



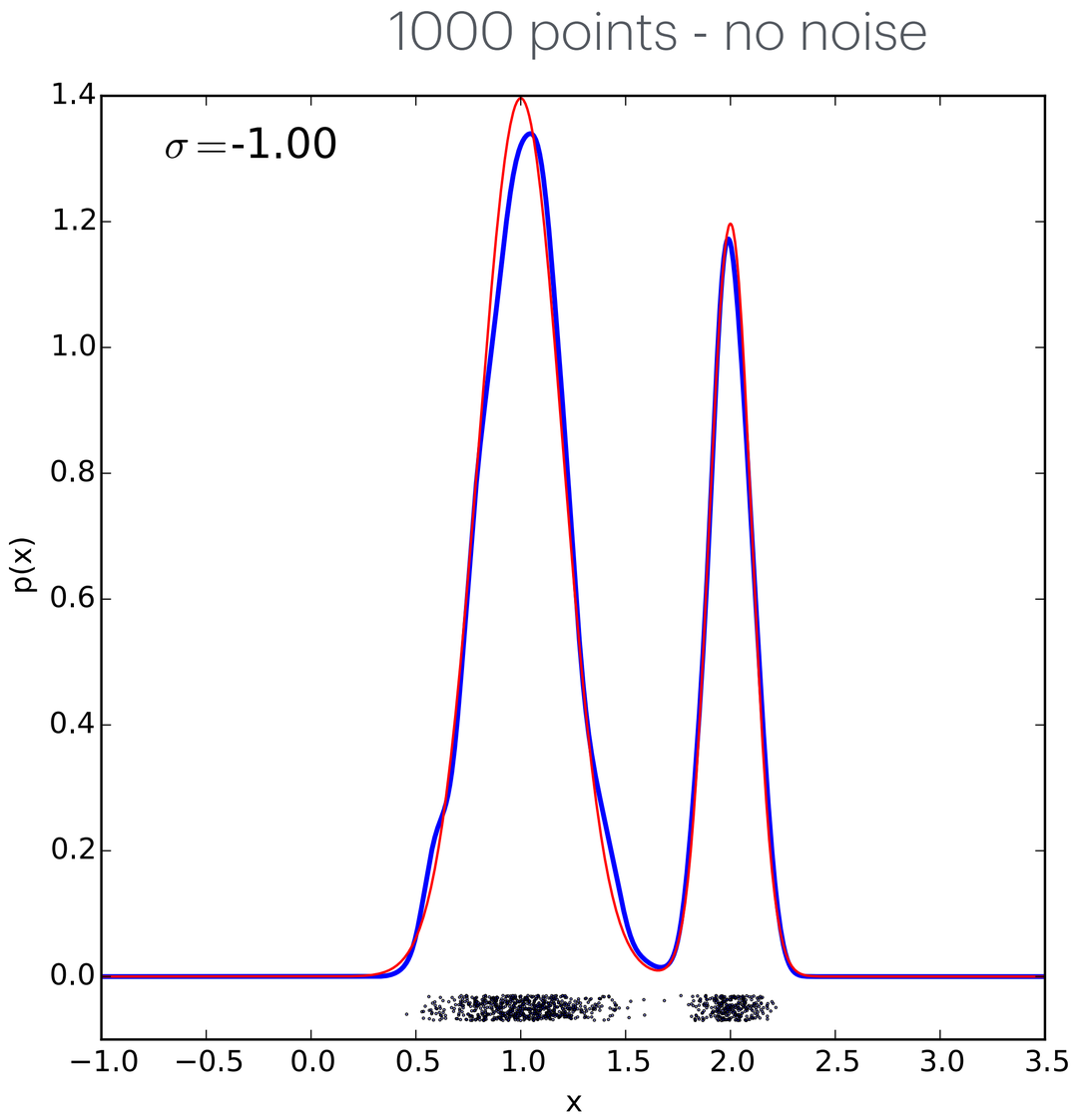
Cappellari (2002)

The effect of noise

Uncertainties in the data

True

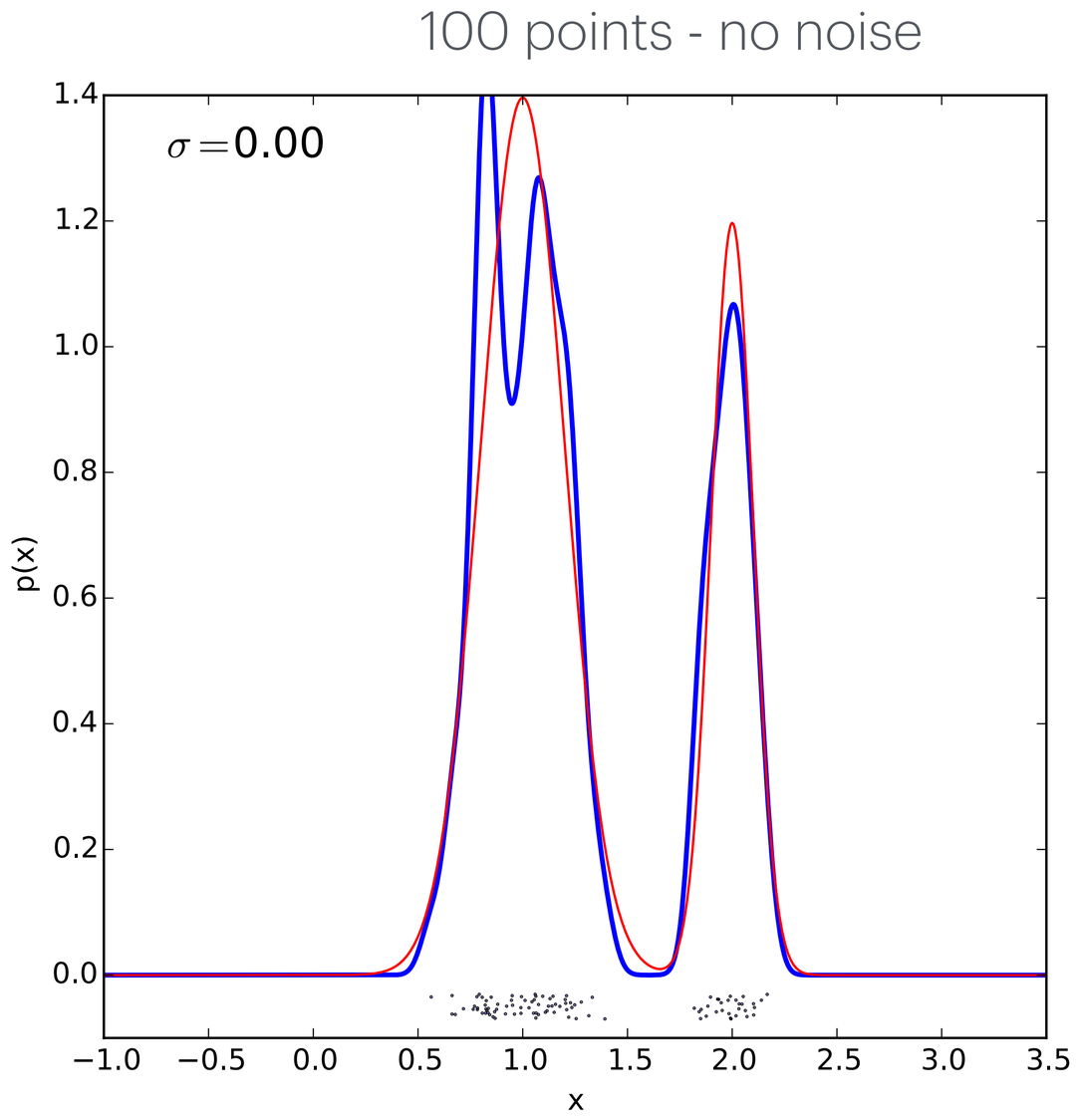
KDE



Uncertainties in the data

True

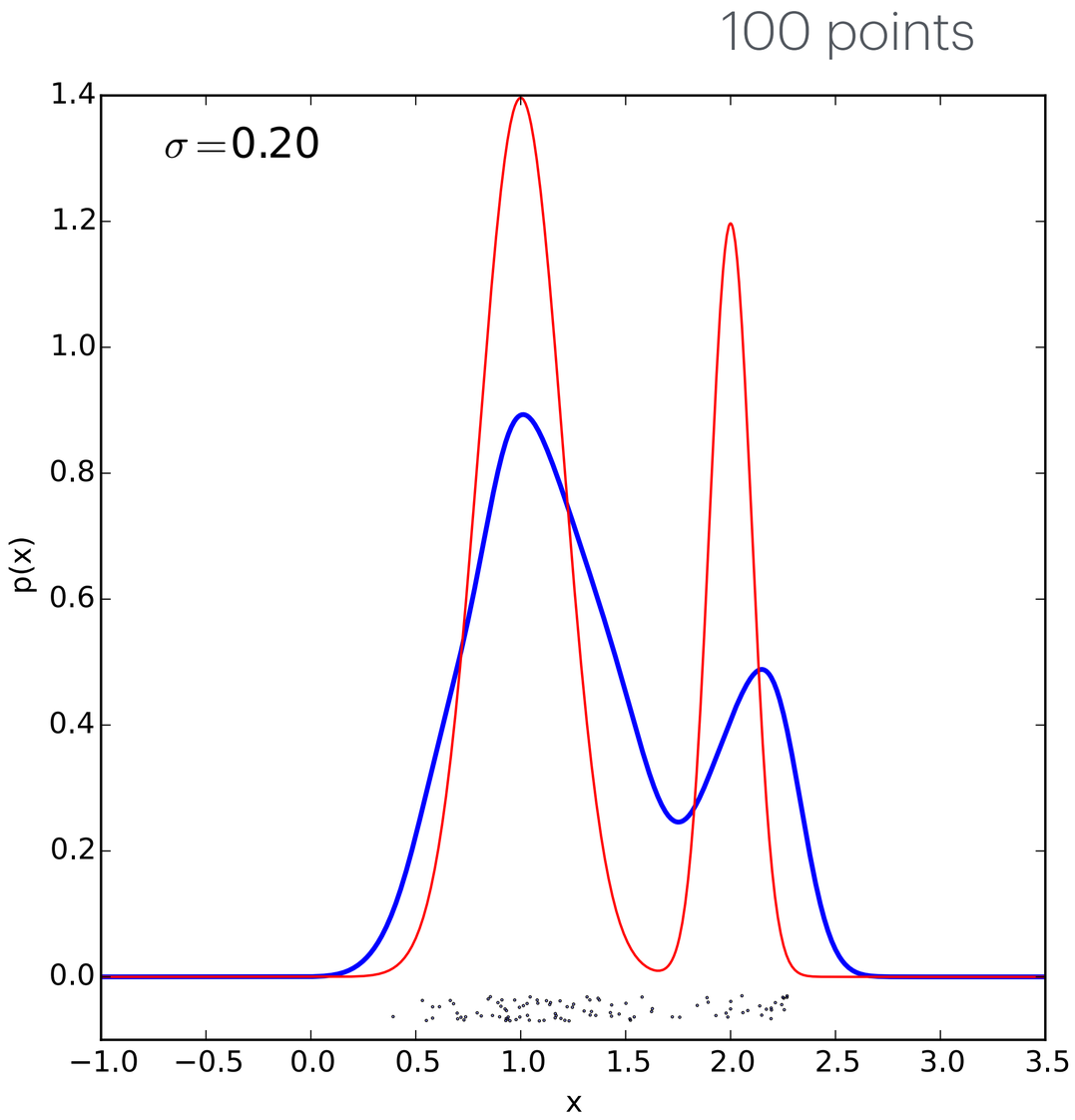
KDE



Uncertainties in the data

True

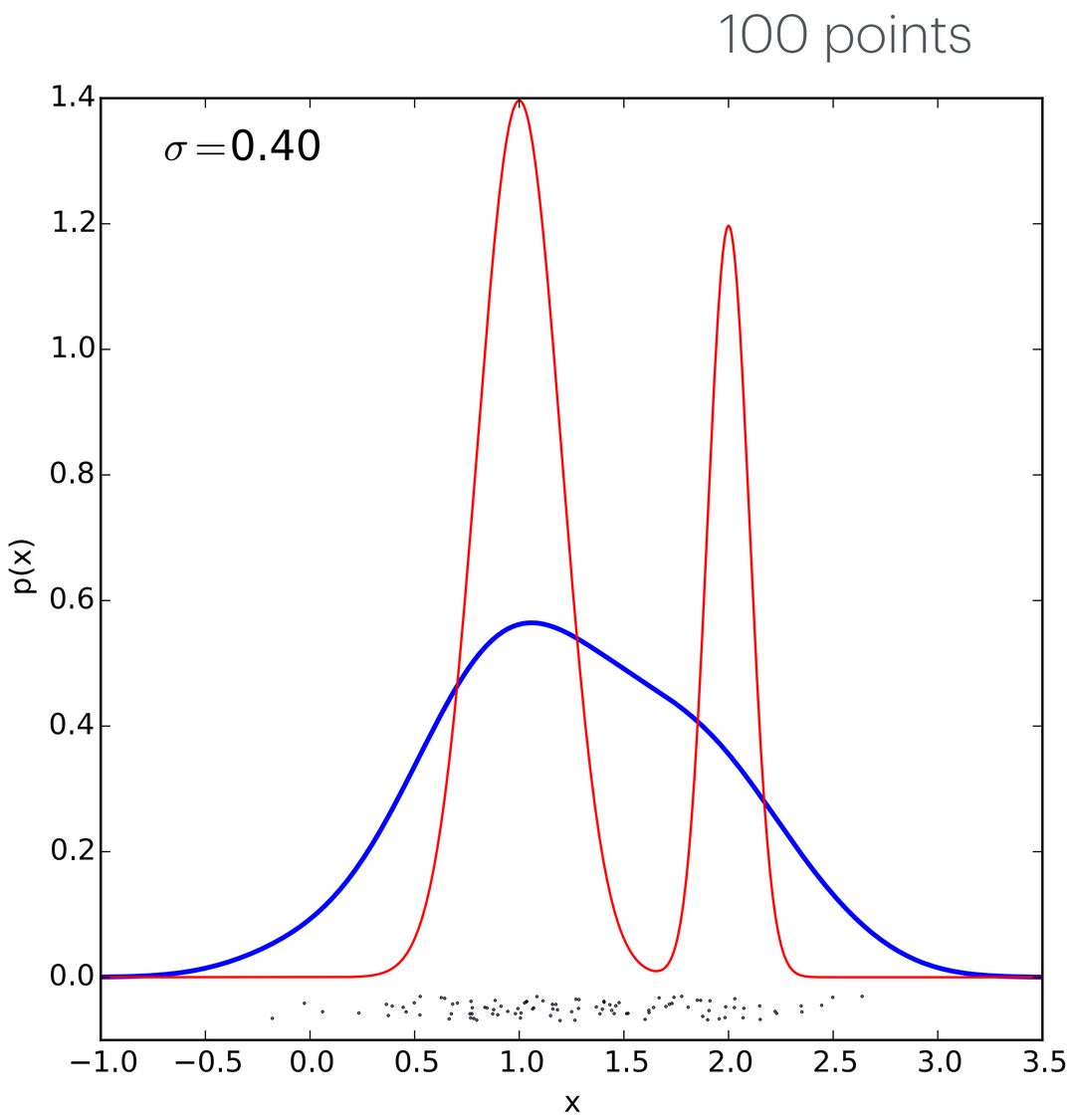
KDE



Uncertainties in the data

True

KDE



Uncertainties in the data - KDE

Your observations are (presumably) a random realisation of the underlying true value - this noise acts to broaden the recovered distribution. For equal uncertainties, σ, we have:

$$h_{\mathrm{obs}} = \left(h_{\mathrm{true}} \star g_\sigma\right)(x) = \int_{-\infty}^{\infty} h(u) g_\sigma(x-u)\, du$$

So can we recover a better estimate of the “true” distribution?

Uncertainties in the data - KDE

Your observations are (presumably) a random realisation of the underlying true value - this noise acts to broaden the recovered distribution. For equal uncertainties, σ, we have:

$$h_{\mathrm{obs}} = \left(h_{\mathrm{true}} \star g_\sigma\right)(x) = \int_{-\infty}^{\infty} h(u) g_\sigma(x-u)\, du$$

So can we recover a better estimate of the “true” distribution?

Yes: Deconvoluting KDE

(Delaigle & Meister (2008, Bernoulli, 14, 2, 562) - no standalone Python implementation afaik, but Matlab & R available at https://researchers.ms.unimelb.edu.au/%7Eaurored/links.html#Code

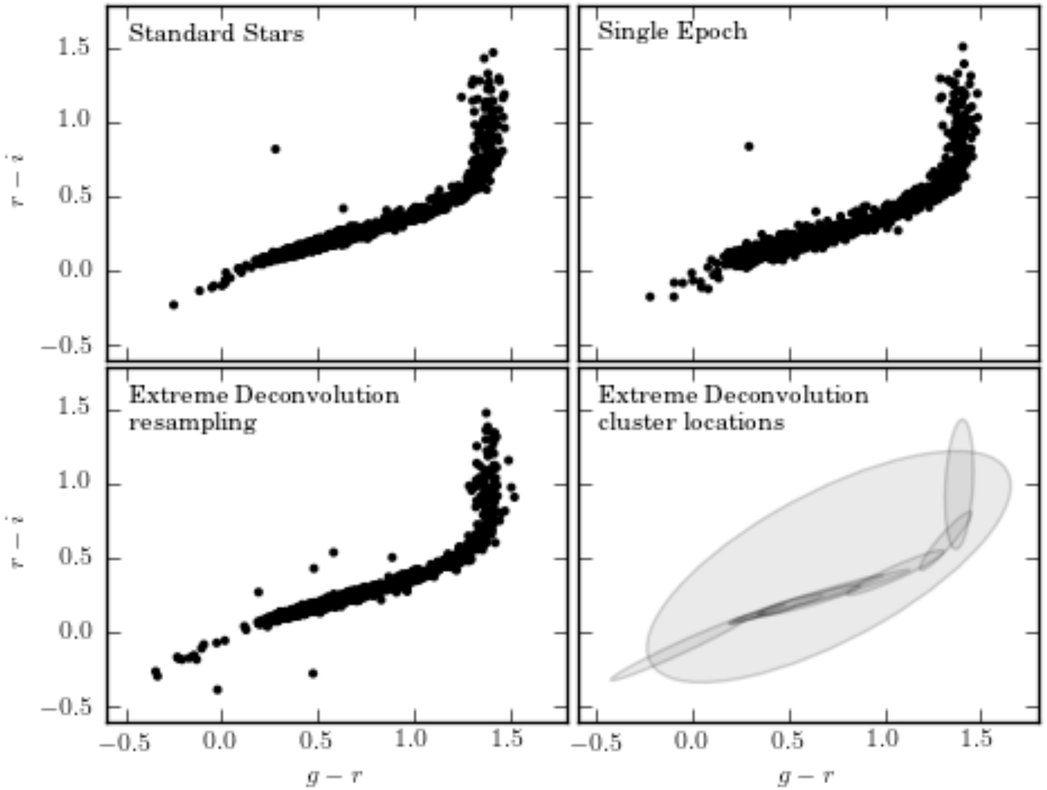
$$h_{\mathrm{est}} = \mathcal{F}^{-1}\left[\frac{\mathcal{F}(h_{\mathrm{obs}})}{\mathcal{F}(g)}\right]$$

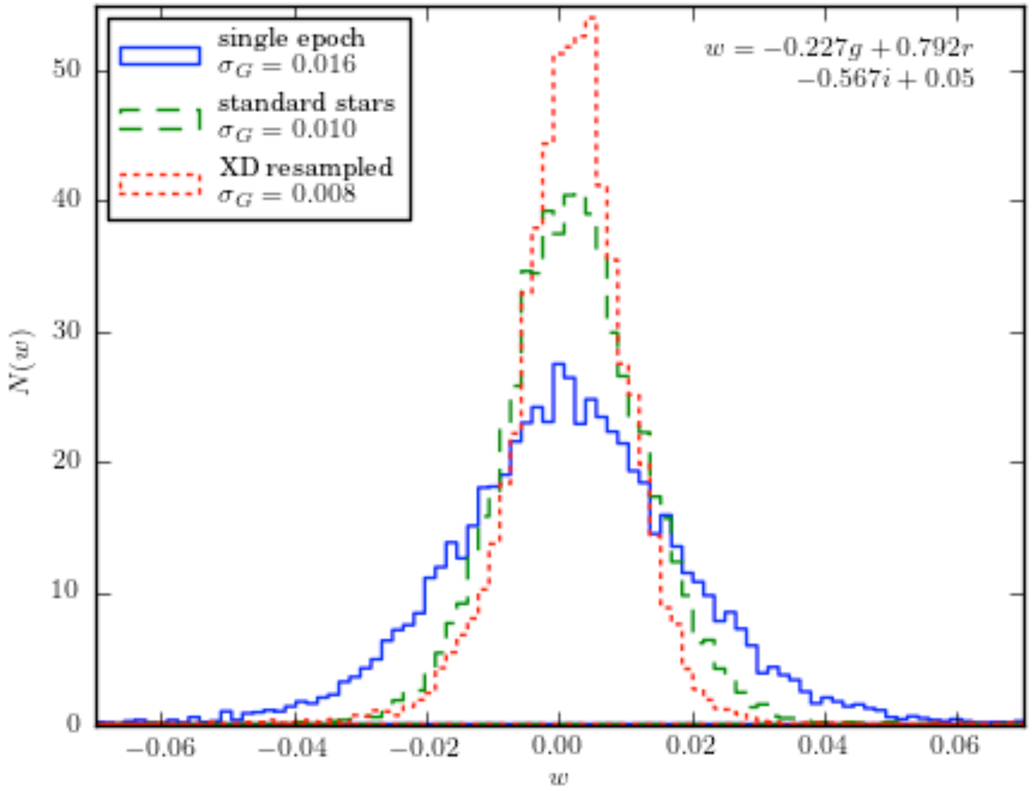
$\mathcal{F}$ being the Fourier transform

Uncertainties in the data - GMM

Extreme-deconvolution (Bovy, Hogg & Roweis 2011)

$$p(\boldsymbol{x}) + \sum_j \alpha_j N\left(\boldsymbol{x}|\mu_j, \boldsymbol{\Sigma}_j\right) \quad + \quad \boldsymbol{x}_i = \boldsymbol{R}_i \boldsymbol{v}_i + \epsilon_i$$





Uncertainties in the data - GMM

Trying it out (for after the lecture!):

```
from astroML.density_estimation import XDGMM

@pickle_results("XD_lgm-d4000.pkl")
def compute_XD_results(X, Xerr, n_components=10, n_iter=500):

    clf = XDGMM(n_components, n_iter=n_iter)
    clf.fit(X, Xerr)
    return clf
```

X: N_{obs} **x** $N_{features}$

Xerr: N_{obs} **x** $N_{features}$ **x** $N_{features}$

Data - e.g. x & y

Covariances

For serious use it is probably better to use the XDGMM wrapper class - https://github.com/tholoien/XDGMM which allows XDGMM to be used in the **sklearn** ecosystem.

Uncertainties in the data - GMM

Working with the result:

```
sample = clf.sample(1000)
```

Draw from the fitted distribution

```
clf.mu
```

Position of each component

```
clf.V
```

Covariance matrix of each component

Inference

Degree of belief

f(x) - the degree of belief about the value of a quantity

Example:

We observe X photons from a source, as long as the number of photons is large we might say that we observed a flux of

$$X \pm \sqrt{X}$$

What we mean then is that our "degree of belief" about the value of X is given by (in this case):

$$f(x) \propto e^{-(x-\hat{\mu})^2/2\hat{\sigma}^2} = e^{-(x-X)^2/2X}$$

This can be formalised (Jaynes 1998) using probabilities

Statistical context

Assume: $$\mathbf{x} \sim p\left(\mathbf{x}; \boldsymbol{\theta}\right)$$

We call $p(x; \theta)$ a probability distribution function (PDF) with parameters θ

The cumulative probability distribution function (CDF) is defined as

$$\mathrm{CDF}(x; \theta) = p(\ < x) = \int_{-\infty}^{x} p(x; \theta)\, dx$$

in the case of a single continuous random variable.

Statistical context

Assume: $\mathbf{x} \sim p\left(\mathbf{x}; \boldsymbol{\theta}\right)$

If you know p(x;) you might want to estimate its parameters $\boldsymbol{\theta}$.

Inference/ estimation

If you know p(x; $\boldsymbol{\theta}$) you might want to draw random variables from it.

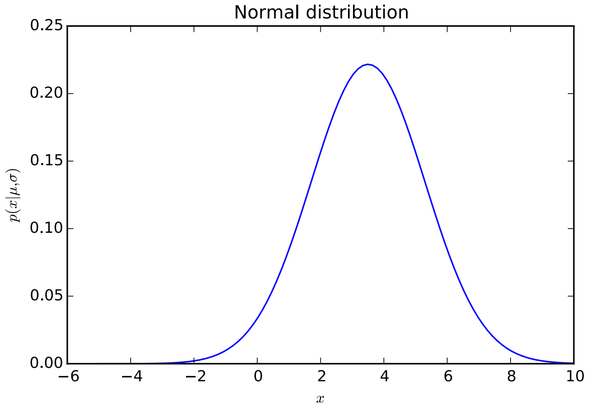
Simulation

If you know x but not p(x), you might want to estimate p(x).

Density estimation

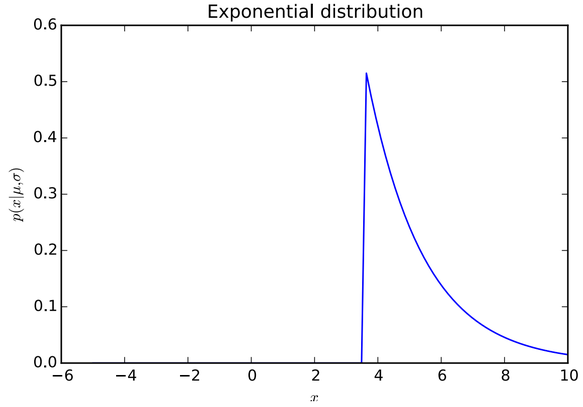
See GitHub for a IPython notebook on distributions. (Lecture 2/Notebooks)

Summarising distributions/data



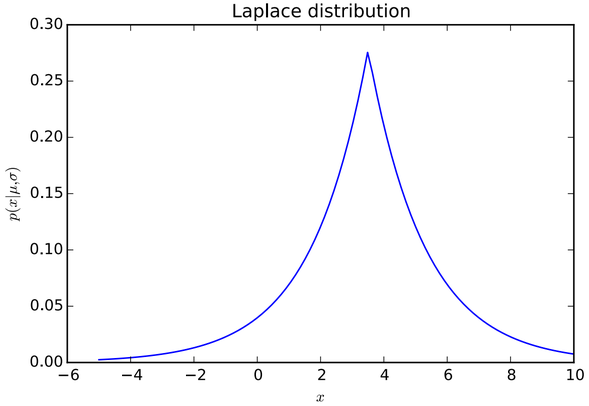
$$p(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

$$E[x] = \mu \qquad V = E[(x-\mu)^2] = \sigma^2$$



$$p(x|\mu,\sigma) = \frac{1}{2\sigma} e^{-|x-\mu|/\sigma} H(x-\mu)$$

$$E[x] = \mu + \sigma \qquad V = \sigma^2$$



$$p(x|\mu,\sigma) = \frac{1}{2\sigma} e^{-|x-\mu|/\sigma}$$

$$E[x] = \mu \qquad V[x] = 2\sigma^2$$

The Normal distribution

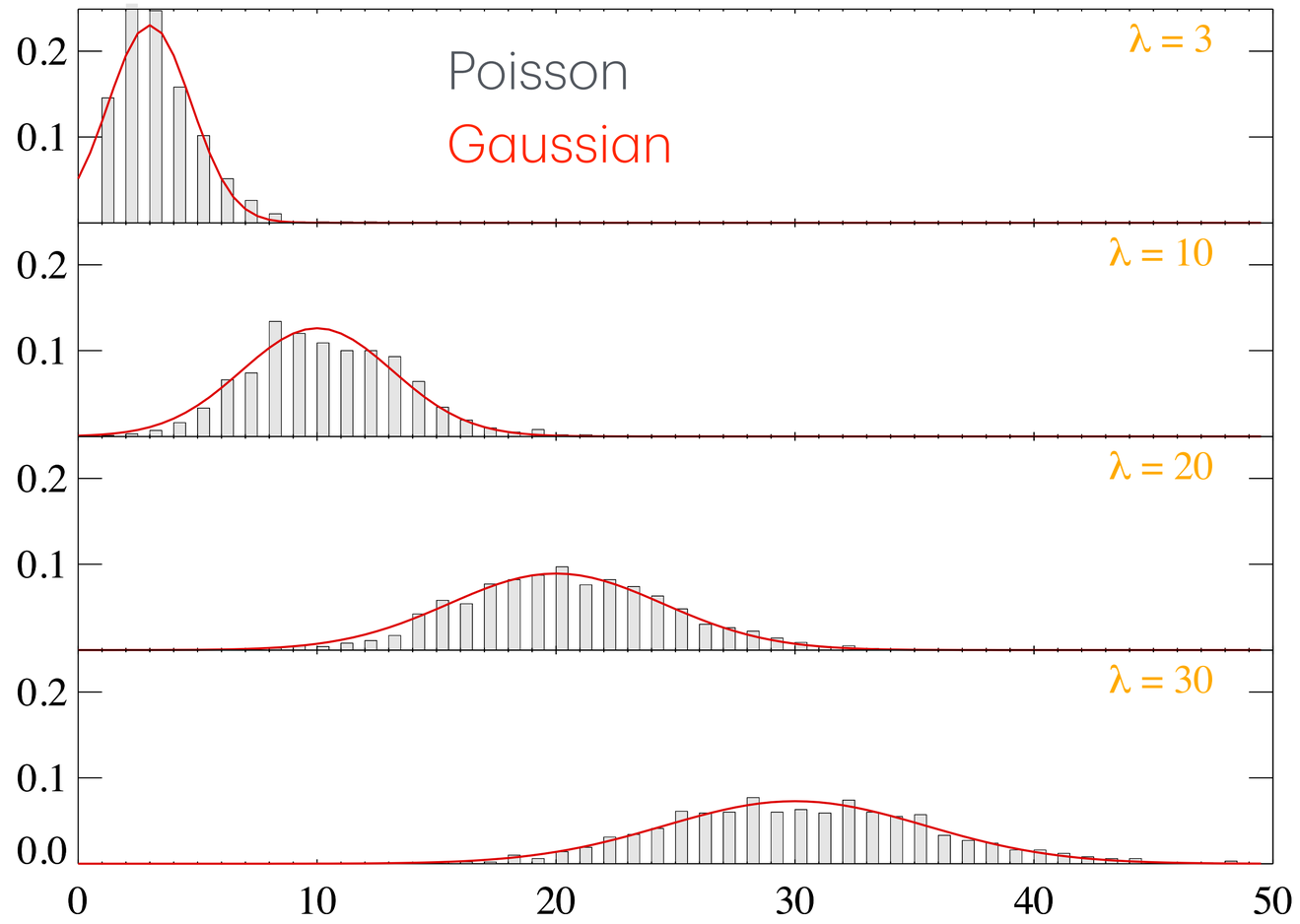
A particularly important distribution is the normal distribution. It is important in data mining because it is easy to deal with analytically and because the mean of large numbers of random numbers is distributed as a normal distribution.

In astronomy, the key reason is that if you observe a source emitting N photons you will detect n, where n is distributed as a Poisson distribution. However as soon as n > 10 (or so), that is essentially a Gaussian distribution.

The Normal distribution

A particularly i . It is
important in d(/tically and
because the m istributed
as a normal di:

In astronomy, t mitting N
photons you w
distribution. Hc ally a
Gaussian distr



The Central Limit Theorem

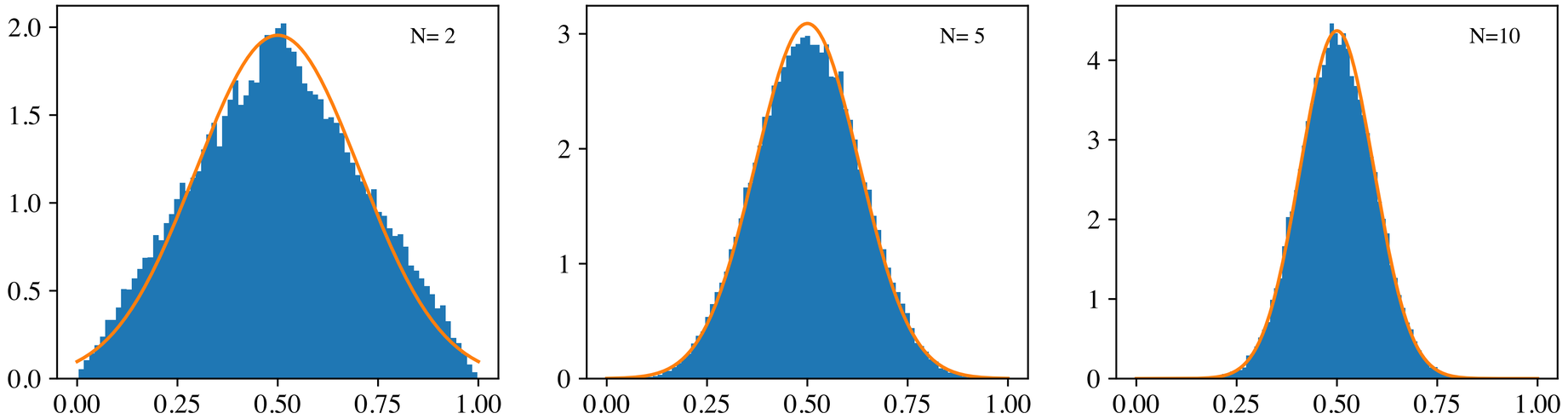
Arbitrary distribution $h(x)$ with mean μ and standard deviation σ

Draw M values x from this distribution - the mean of x will follow

$$\bar{x} \sim N(\mu, \sigma/\sqrt{M})$$

from astroML.org

Illustration for the mean of N = 2, 3 & 10 values
drawn from a uniform distribution.



The normal distribution - multi-dimensional

$$P(\boldsymbol{x}) = \frac{1}{(2\pi)^{1/D}} \frac{1}{|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

Where $\boldsymbol{\Sigma}$ is the covariance matrix:

$$\Sigma = E\left[(x-\mu)(x-\mu)^T\right]$$

This is a useful distribution in many cases and underlies a lot of the techniques we will be using.

Drawing from a distribution

This is a common task. Most computer languages provide you with a (more or less useful) random number generator:

R:

```
x = runif(N, [min=0, max=1])
```

```
x = runif(1000, min=-1, max=1)
```

IDL:

```
x = randomu(seed, N)
```

```
x = 2*(randomu(ss, 1000)-0.5)
```

Python (after doing: import numpy as np)

```
x = np.random.uniform(a, b, N)
```

```
x = np.random.uniform(-1, 1, 1000)
```

More complex distributions:

$$r = \mathrm{CDF}^{-1}(x)$$

where x is uniformly distributed

In many-D this is more complex and other algorithms might be used.

A brief intermezzo - storing and caching

Some of the calculations you will do can be time-consuming.

When nothing has changed there is no reason to redo the calculation - but of course you need to store the result. You can do this on disk as an intermediate file - that may or may not be a good idea (writing to disk is slow!)

As an alternative you can keep it in memory through clever caching of intermediate results. The general programming technique for this is called memoizing, in case you cared, but we will typically want to store the results of calculations to disk.

Pickling

```
def pickle_to_file(data, fname):
    """A simple function to pickle some data to a file"""
    try:
        fh = open(fname, 'wb')
        pickle.dump(data, fh)
        fh.close()
    except:
        print("Pickling failed!", sys.exc_info()[0])

def pickle_from_file(fname):
    """The corresponding simple routine to load from a file"""
    try:
        fh = open(fname, 'rb')
        data = pickle.load(fh)
        fh.close()
    except:
        print("Loading pickled data failed!", sys.exc_info()[0])
        data = None
```

A standard way to store data in Python - can store complex data (but in general use other formats for this).

Decorating

This is a standard Pythonic way to modify functions.

```
def get_text(name):
   return "l ({0}) am a decorator".format(name)

def p_decorate(func):
   def wrapper(name):
       return "*** {0} ***".format(func(name))
   return wrapper

my_get_text = p_decorate(get_text)
```

“Nicer” way (more Pythonic):

```
@p_decorate
def get_text_d(name):
   return "l ({0}) am a decorator".format(name)
```

Why should you care?

astroML has a very convenient way to store results of heavy calculations:

```
from astroML.decorators import pickle_results

@pickle_results("GMM_three_Gaussians.pkl")
def compute_GMM(data, N, n_iter=1000):
    # Setup the output array
    models = [None for n in N]
    for i in range(len(N)):
        models[i] = GMM(n_components=N[i], n_iter=n_iter,
                        covariance_type=covariance_type)
        models[i].fit(data)

    return models
```

When the function is called with the same arguments - the results are read from the file & not redone!.

Classical inference

The broad idea

We want to compare data, $\{y_i\}$, to a model $M(a, b, \ldots)$

To do so we need:

A way to compare our data to the model (cost functions)

e.g.:

$$\int \left(f(x) - f_{\text{true}}(x)\right)^2 \, dx$$

$$\left|f(x) - f_{\text{true}}(x)\right|$$

A way to rank/compare different models Likelihood ratios, information criteria (AIC) etc.

A way to assess whether the fit we obtained is "good" e.g. χ^2 tests

This is arguably the approach most widely used in Machine Learning/Data Mining.

Maximum Likelihood

How were the data generated?

$p(D|M)$

1. Define a likelihood of the data given a model

I will write the parameters of the model θ and the model $M(\theta)$

sometimes $\hat{}$

2. Find $\boldsymbol{\theta} = \boldsymbol{\theta}^0$ that maximise $p(D\,|\,M)$

point estimates

For this we use a minimization routine (e.g. **`scipy.optimize`**)

3. Find confidence levels for $\boldsymbol{\theta}^0$

$$\sigma^2_{j,k} = -\left.\frac{d^2 \ln L}{d\theta_j d\theta_k}\right|_{\theta=\theta_0}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $y_i = \alpha + \beta x_i + \epsilon_i$ $\epsilon_i \sim N(0, \sigma_i^2)$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

So log likelihood: $$\ln L = -\frac{1}{2} \ln 2\pi - \frac{1}{2} \sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

ML - Simple example - Gaussian errors

$\boldsymbol{\theta} = (\alpha, \beta)$

Model y as $$y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$$

Likelihood of data:

$$P(x_i, y_i, \sigma_i | \alpha, \beta) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-[y_i - (\alpha + \beta x_i)]^2 / 2\sigma_i^2}$$

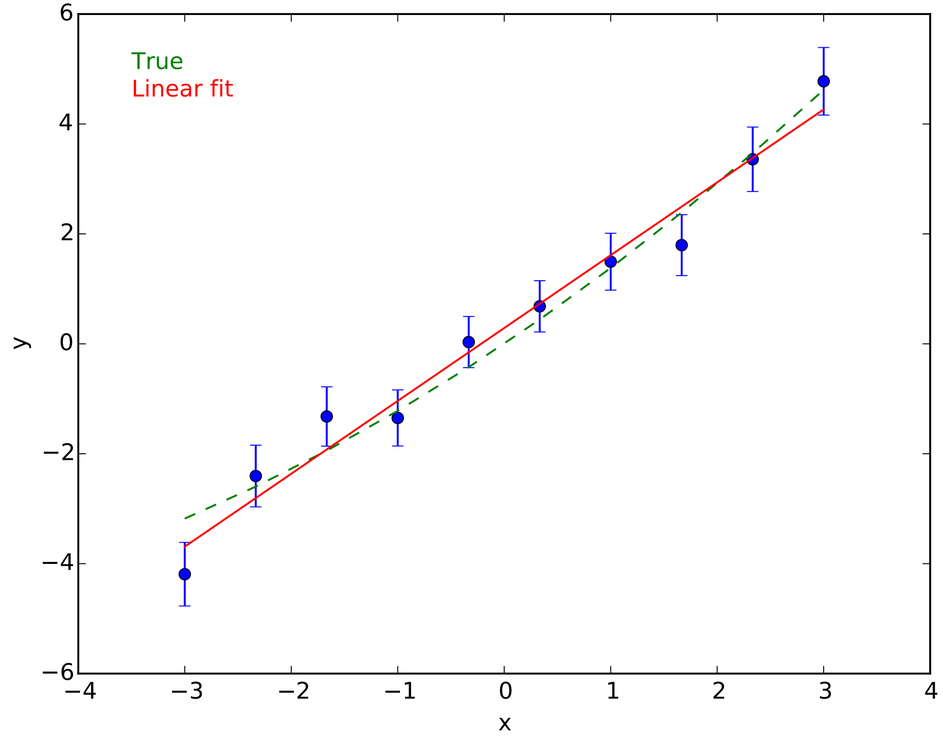
Full Likelihood $$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\sigma} | \alpha, \beta) = \prod_i P(x_i, y_i, \sigma_i | \alpha, \beta)$$

So log likelihood: $$\ln L = -\frac{1}{2} \ln 2\pi - \frac{1}{2} \sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

Minimisation of this is the same as least squares minimisation.

ML - Simple example - Gaussian errors, standard function

```
from scipy.optimize import curve_fit

def func_line(x, a, b):
    return a + b*x

pars, cov = curve_fit(func_line, x, y_obs)
```



This is equivalent to the regression solution you have seen before:

```
from astroML.linear_model import LinearRegression
m = LinearRegression()
m.fit(x[:, None], y_obs, sigma)
a, b = m.coef_
```

ML - Simple example - explicit likelihood

```
def neglnL(theta, x, y, yerr):
    a, b = theta
    model = b * x + a
    inv_sigma2 = 1.0/(yerr**2)

    return 0.5*(np.sum((y-model)**2*inv_sigma2))
```

$$-\ln L = \frac{1}{2}\sum_i \frac{\left[y_i - (\alpha + \beta x_i)\right]^2}{\sigma_i^2}$$

```
import scipy.optimize as op
result = op.minimize(neglnL, [1.0, 0.0], args=(x, y_obs, sigma))
a_ml, b_ml = result["x"]
```

More work in this case - but a much more general approach.

Bayesian inference

Bayesian statistics

This allows us to calculate the likelihood distribution of parameters

$$p(\text{Model}|\text{Data}) = p(M|D) = \frac{p(D|M)p(M)}{p(D)}$$

The key part here is that we need to specify what **prior** information we have on the model parameters.

$$p(M, \boldsymbol{\theta}|D, I) = \frac{p(D|M, \boldsymbol{\theta}, I)p(M, \boldsymbol{\theta}|I)}{p(D|I)}$$

The overall plan of attack

- Define the likelihood - the best is a generative one that mimic how you think the data were created. $p(D|\boldsymbol{\theta}, M, I)$
- Decide on the prior - ie. what range of model parameters are likely. $p(\boldsymbol{\theta}, M|I)$
- Use Bayes' theorem to calculate the posterior likelihood distribution. $p(M|D, I)$
- To calculate the posterior distribution we often use Markov Chain Monte Carlo calculations. MCMC

The result of the calculation can be summarised - the maximum of $p(M|D,I)$ gives the maximum a posteriori (**MAP**) estimate - means, medians are also reasonable options.

A practical example - line fitting - model

Generative model: $y_i = \alpha + \beta x_i + \epsilon_i \qquad \epsilon_i \sim N(0, \sigma_i^2)$

So log likelihood: $$\ln L = -\frac{1}{2}\ln 2\pi - \sum_i \ln \sigma_i - \frac{1}{2}\sum_i \frac{[y_i - (\alpha + \beta x_i)]^2}{\sigma_i^2}$$

(note that since σ_i are known, the first two terms are constants and can be ignored for maximisation

```
def lnL(theta, x, y, yerr):
    a, b = theta
    model = b * x + a
    inv_sigma2 = 1.0/(yerr**2)

    return -0.5*(np.sum((y-model)**2*inv_sigma2))
```

A practical example - line fitting - prior

Prior:

$$p(a, b, M|I) = \begin{cases} \text{const}, & \text{if } a \in [-5, 5] \text{ and } b \in [-10, 10]; \\ 0 & \text{otherwise} \end{cases}$$

```
def lnprior(theta):
    a, b = theta
    if -5.0 < a < 5.0 and -10.0 < b < 10.0:
        return 0.0
    return -np.inf
```

(the -np.inf is because p=0 means ln p = -∞)

A practical example - line fitting - posterior

Putting it together: $p(D|a,b,M,I)p(a,b,M|I)$

```
def lnprob(theta, x, y, yerr):
    """
    The likelihood to include in the MCMC.
    """
    lp = lnprior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + lnL(theta, x, y, yerr)
```

Note that I ignore the normalisation $p(D|I)$

A practical example - line fitting

```
import emcee

# Use ML to get a starting point.
# result = run_ml()
p_init = np.array([ 0.28233725,  1.31299656])

# Set up the properties of the problem.
ndim, nwalkers = 2, 100

# Setup a number of initial positions.
pos = [p_init + 1e-4*np.random.randn(ndim) for i in range(nwalkers)]

# Create the sampler.
sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=(x, y_obs, sigma))

# Run the process.
sampler.run_mcmc(pos, 500)
```

A practical example - importing the package

```
import emcee
```

We will use the MCMC Hammer library (emcee) -
http://dan.iel.fm/emcee/current/

it is a pure Python implementation (as pymc3), flexible and reasonably fast - for challenging problems other packages (MultiNest, BUGS, JAGS, STAN) might be better choices but it is well worth starting with this as the learning curve is less steep.

On your laptop or in Google Colab you can install it with:

pip install emcee
pip install corner

A practical example - starting position

```
# Use ML to get a starting point.
# result = run_ml()
p_init = np.array([ 0.28233725,  1.31299656])
```

The MCMC process will randomly (but cleverly) step around in your parameter space. To do this well you need a good starting position. A maximum-likelihood solution gives a good starting point.

I used scipy.optimize.minimize here. I set the result to a variable p_init.

A practical example - line fitting

```
# Set up the properties of the problem.
ndim, nwalkers = 2, 100
```

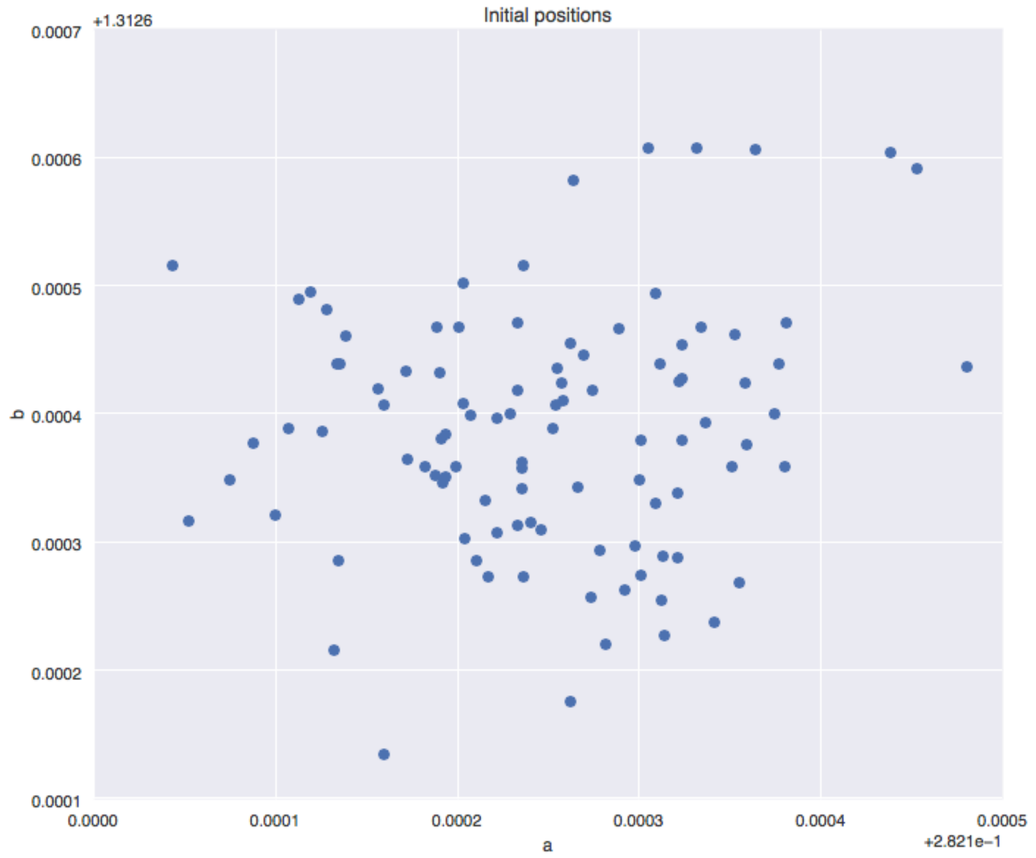
We have two parameters - so the dimensionality is 2

And we also need “walkers” - view these as random processes that explore your parameter space along different paths. The **emcee** documentation recommends using as many as you can get away with.

A practical example - line fitting

```
# Setup a number of initial positions.
pos = [p_init + 1e-4*np.random.randn(ndim) for i in range(nwalkers)]
```

This creates a set of different starting positions - one for each walker.



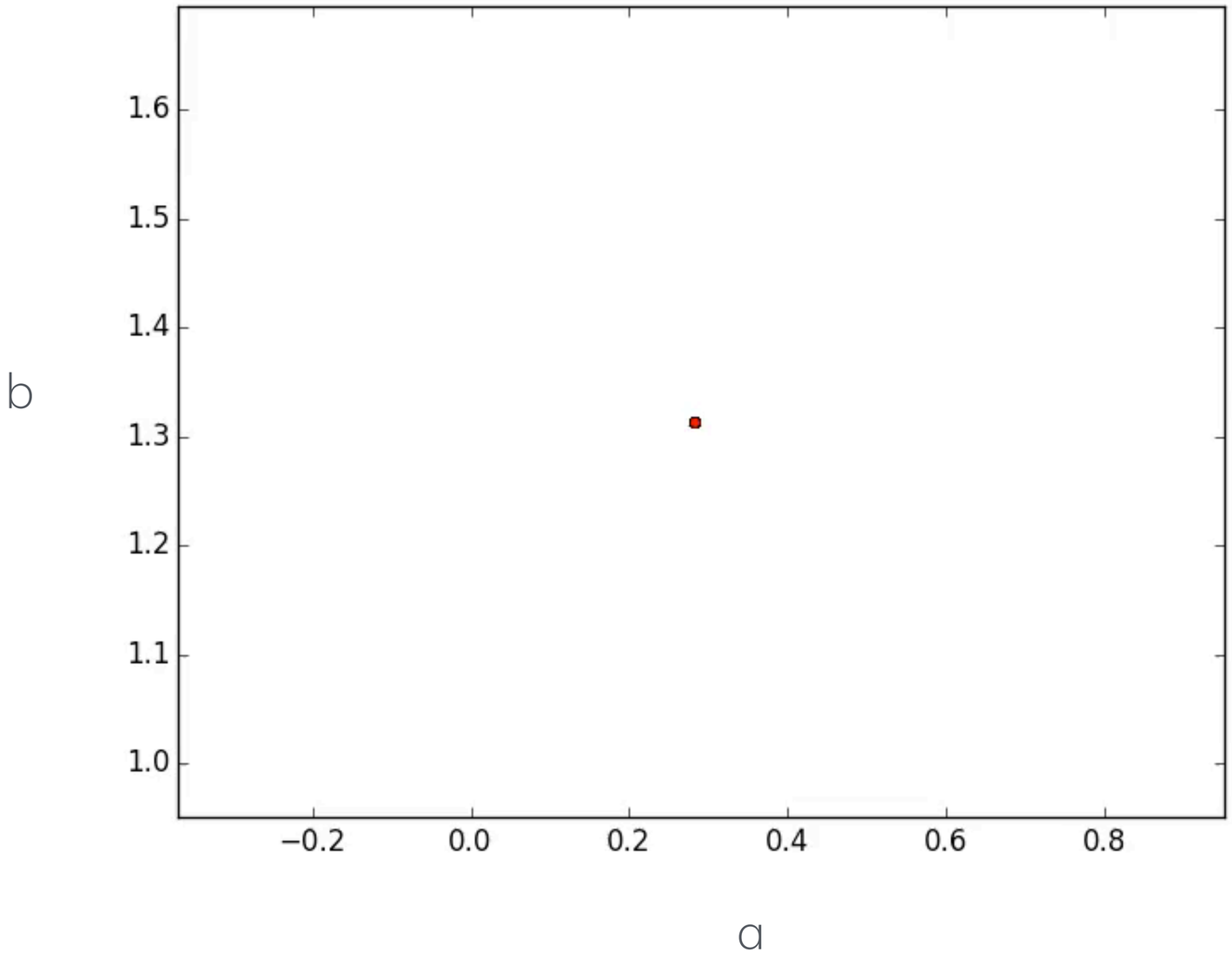
A practical example - line fitting

```
# Create the sampler.
sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=(x, y_obs, sigma))
```

This sets up the MCMC process - we give the function `lnprob` and the data as a tuple.

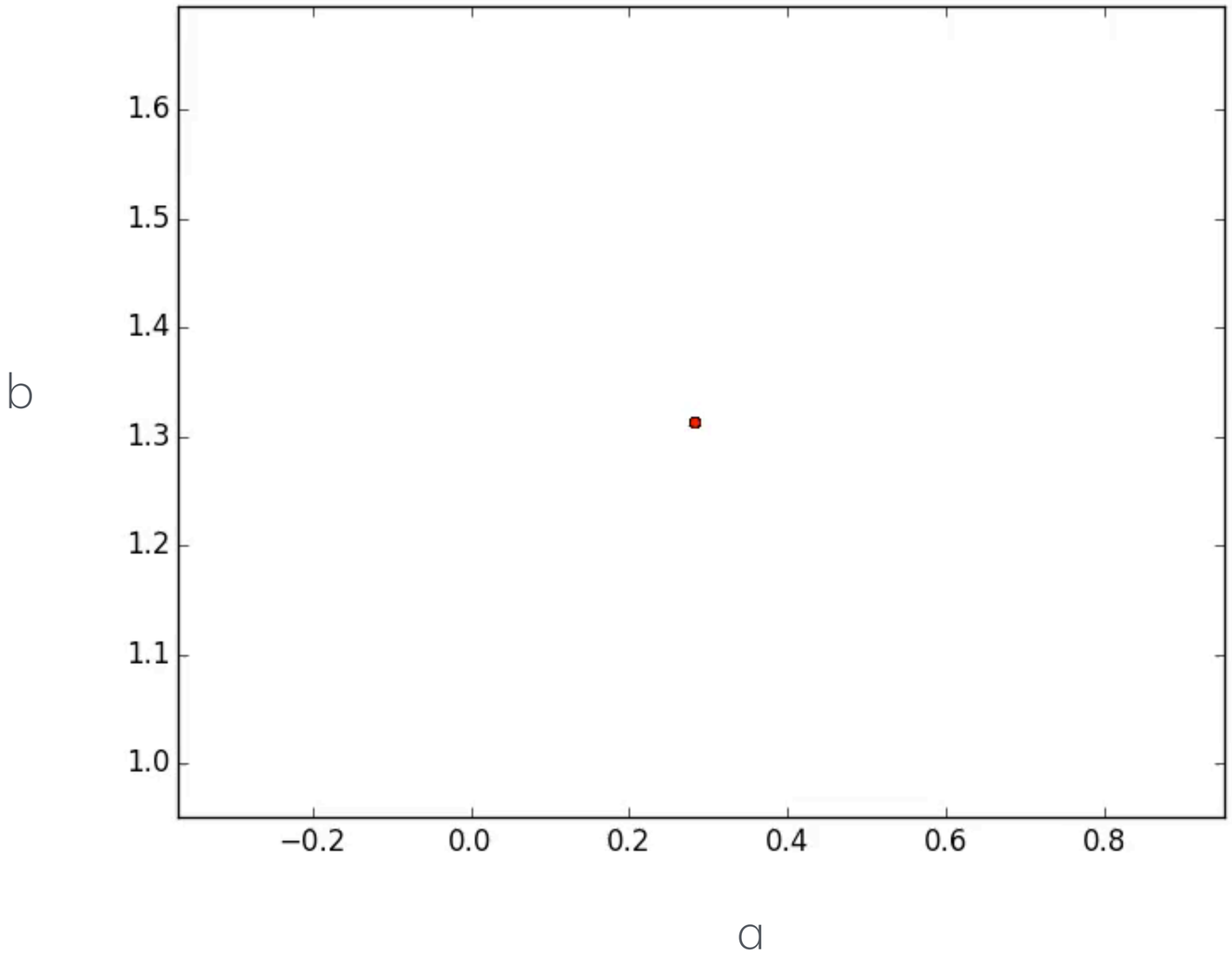
A practical example - line fitting

```
# Run the process.
sampler.run_mcmc(pos, 500)
```



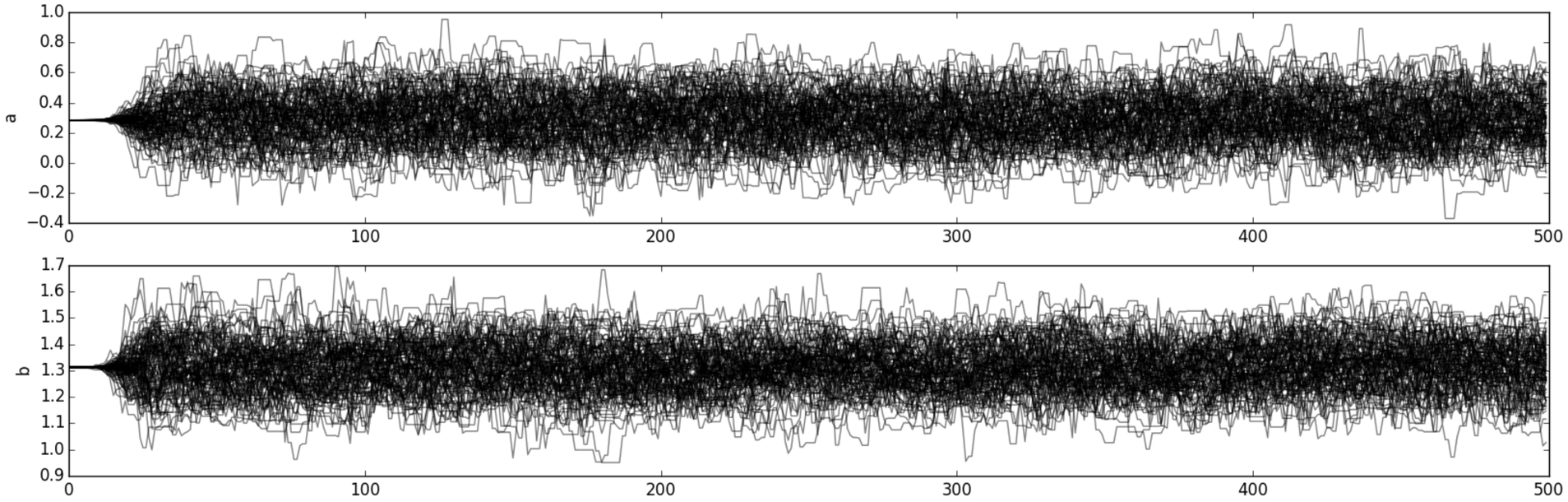
A practical example - line fitting

```
# Run the process.
sampler.run_mcmc(pos, 500)
```



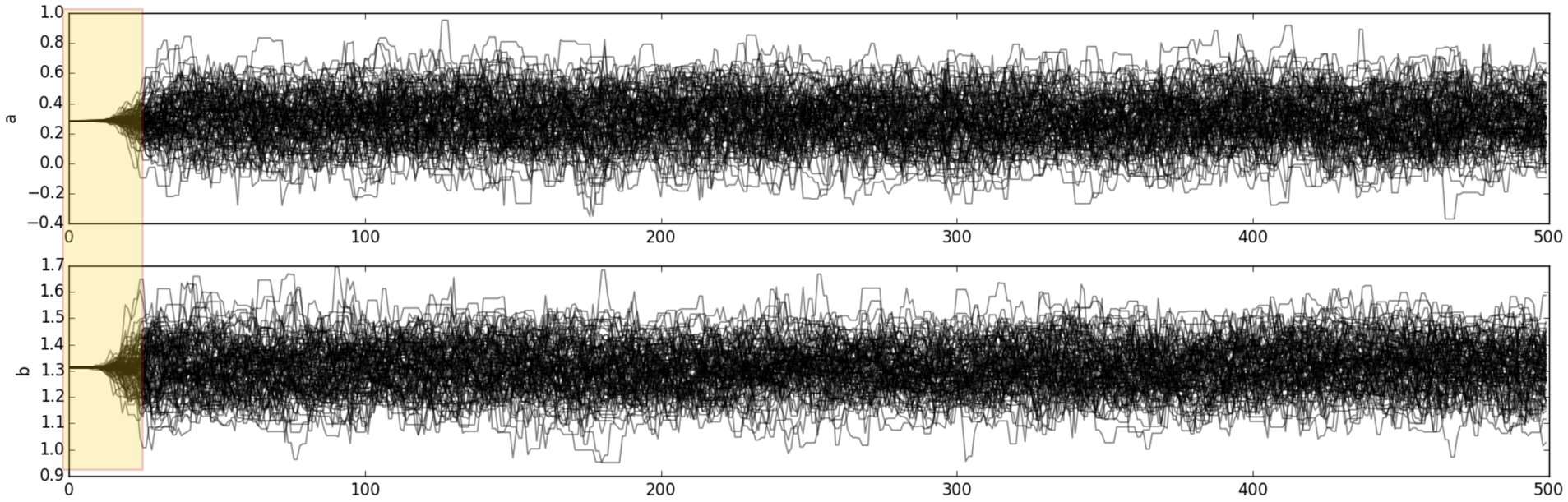
Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



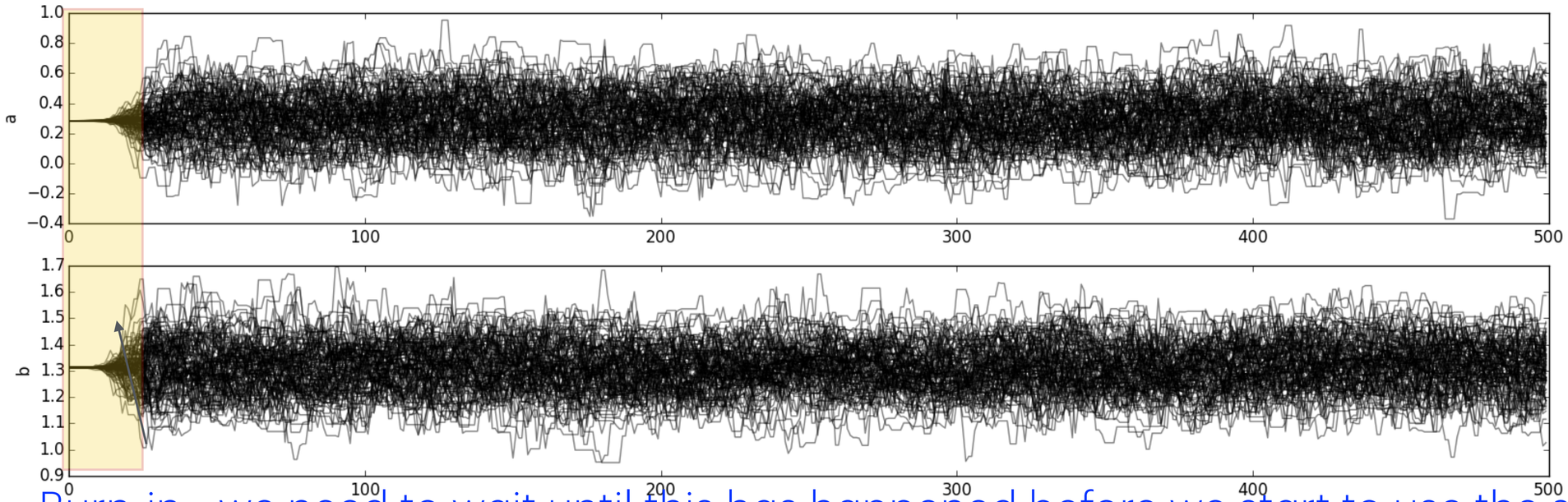
Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



Look at chain behaviour

```
labels = [‘a’, ‘b’]
chain = sampler.chain
for i_dim in range(2):
    plt.subplot(2,1,i_dim+1)
    plt.ylabel(labels[i_dim])

    for i in range(100):
        plt.plot(chain[i,:,i_dim],color='black', alpha=0.5)
```



Burn-in - we need to wait until this has happened before we start to use the samples.

A practical example - showing the result

Let us extract the samples from 50 onwards and collapse the different walkers

```
samples = sampler.chain[:, 50:, :].reshape((-1, 2))
```

We then show the result using the corner package:

```
import corner

fig = corner.corner(samples, labels=["$a$", “$b$"],
                    truths=[0.0, 1.3], quantiles=[0.16, 0.84])
fig.show()
```

A practical example - showing the result

Let us extract

nt walkers

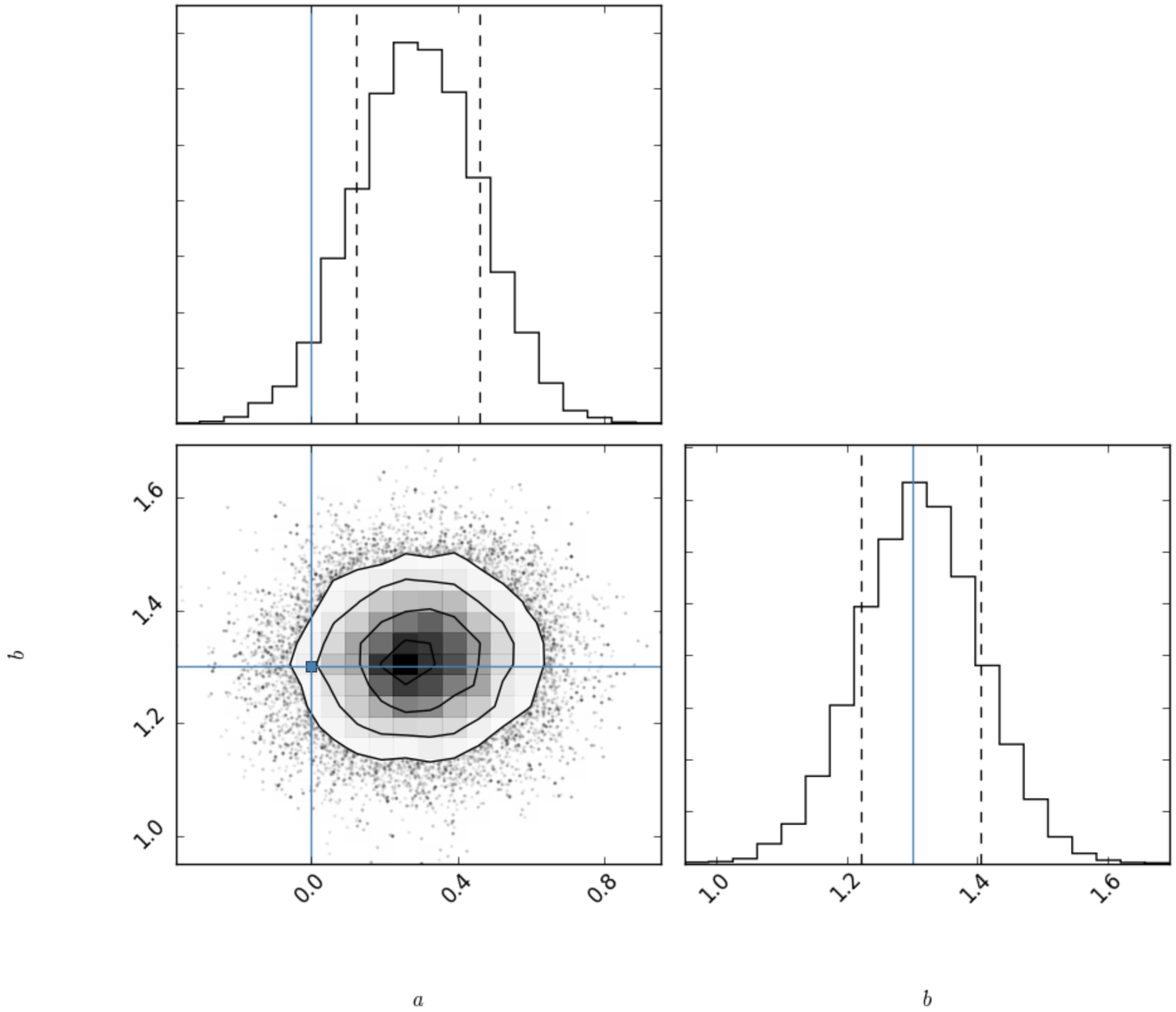
```
samples = s
```

We then show

```
import corn
```

```
fig = corne
```

```
fig.show()
```




```
16, 0.84])
```

Bayesian model selection

This is arguably the most rigorous but has its own problems.

$$p(M, \boldsymbol{\theta}|D, I) = \frac{p(D|M, \boldsymbol{\theta}, I)p(M, \boldsymbol{\theta}|I)}{p(D|I)}$$

What we want here is $p(M \,|\, D, I)$ so that we can compare against a different model. Thus we need to integrate out θ

This can be very challenging to impossible in multi-D situations.

Credible intervals vs confidence intervals

Frequentist view of the measurement process:

The probability is related to the frequency of repeated events.

In this view, models parameters are fixed and data are random, so we do not talk about the probability of the model parameter. The confidence interval is then random.

Bayesian way to look at it:

The probability is related to the degree of certainty about values.

In this view, model parameters are random (have a distribution) and data are fixed. Thus we can talk about the probability distribution of a parameter.

Credible intervals vs confidence intervals

Frequentists: confidence intervals

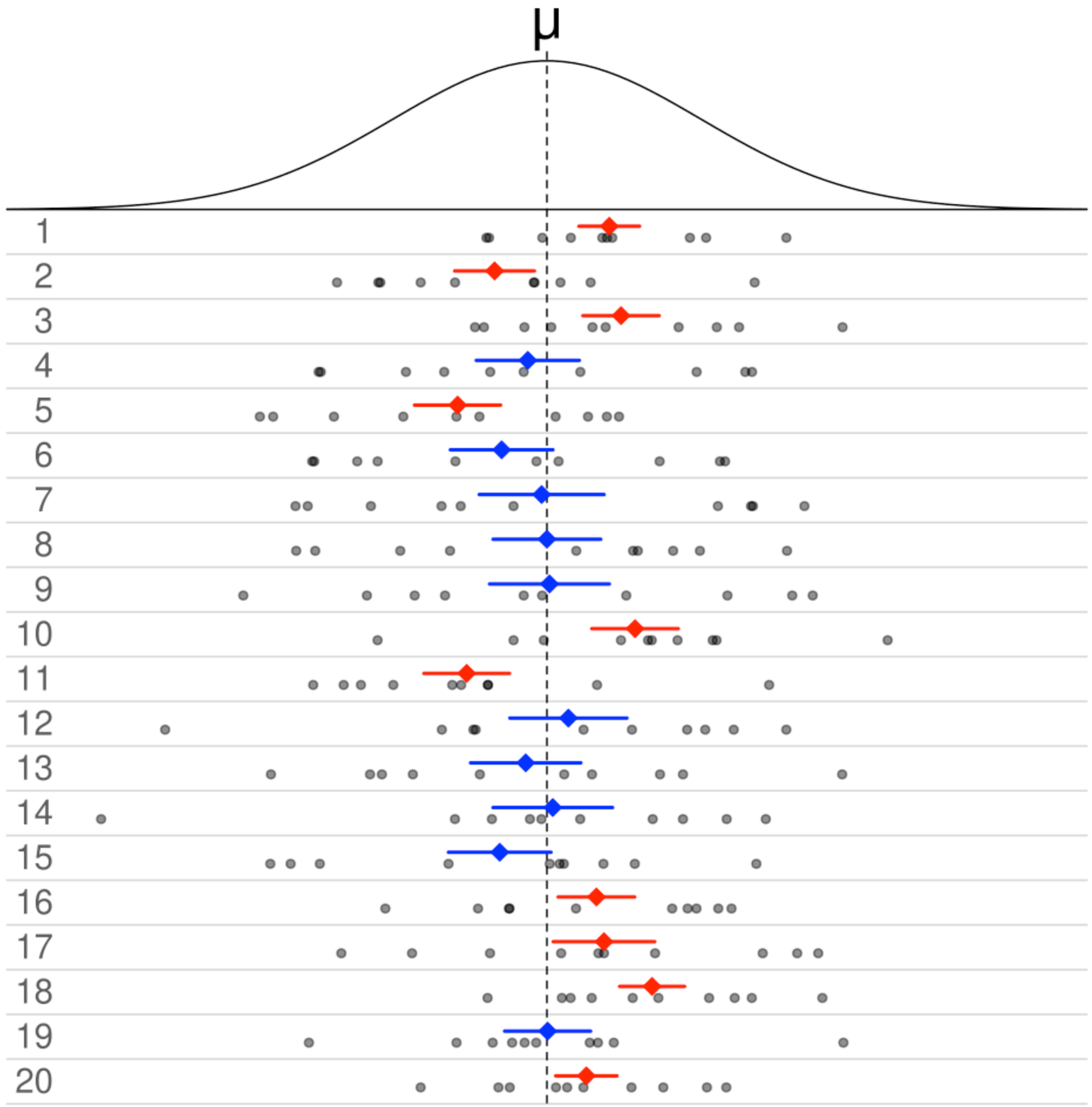
A range of values that are designed to include the true value of the parameter with some minimum probability (e.g. 95%). This is calculated based on the data and is a random variable as it depends on the data (viewed as random).

NB! This does not mean that there is a 95% probability that the true parameter lies within the interval. It does or does not in the frequentist interpretation.

Credibl ntervals

Frequentists: c

A range of e of the
parameter s
calculated t depends
on the datc



NB! This do the true
parameter frequentist
interpretati

Credible intervals vs confidence intervals

The Bayesian way - credible intervals:

An interval within which the true parameter value falls with a given (say 95%) probability.

Most of the time, that is what you want! But there are some subtleties - see next page!

For more details, Jake VanderPlas's write-up is interesting (and opinionated!):

https://jakevdp.github.io/blog/2014/06/12/frequentism-and-bayesianism-3-confidence-credibility/

And a (much) older consideration comes from E. T. Jaynes:

https://bayes.wustl.edu/etj/articles/confidence.pdf

Credible intervals - a subtle point:

It is not always clear what it means - two ways to define it:

1. Choose the narrowest interval containing a particular probability. This is the Highest Posterior Density Region
2. Choose the interval that has equal probability below and above the interval. This can be called the equal-tailed interval, or the Central Credible Region.

Credible intervals - a subtle point:

Highest Posterior Density Region

$$1 - \alpha = \int_{\theta : p(\theta|D) \geq p^*} p(\theta \,|\, D) d\theta$$

So find a p* so that the summed probability above this is equal to the probability we want.

Central Credible Region

$$C_\alpha(D) = (l, u) : P(l \leq \theta \leq u \,|\, D) = 1 - \alpha$$

Usually we want this to have equal probability mass below l and above u.
In which case this is an equal tail central credible region.

See this old Stackoverflow discussion on the topic which also has code: https://stackoverflow.com/questions/22284502/highest-posterior-density-region-and-central-credible-region

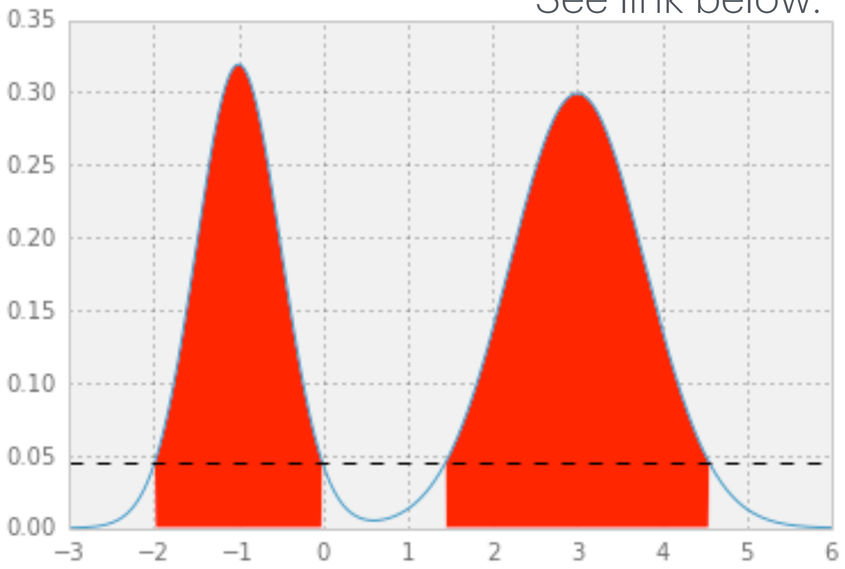
Credible intervals - a subtle point:

Highest Posterior Density Region

$$1 - \alpha = \int_{\theta : p(\theta|D) \geq p^*} p(\theta \,|\, D) d\theta$$

So find a p* so that the summed probability above this is equal to the probability we want.

See link below:



Central Credible Region

$$C_\alpha(D) = (l, u) : P(l \leq \theta \leq u \,|\, D) = 1 - \alpha$$

Usually we want this to have equal probability mass below l and above u.
In which case this is an equal tail central credible region.

See this old Stackoverflow discussion on the topic which also has code: https://stackoverflow.com/questions/22284502/highest-posterior-density-region-and-central-credible-region

Trying out MCMC

51 Peg b - the first confirmed exo-planet around a normal star

We will model radial velocities. Here we will consider a circular orbit with period P, inclination i, and the mass of the planet, M_P, and the star, $M_\star$. In this case the amplitude of the variation in the radial velocity, v_R, is given by

$$k = \left(\frac{2\pi G}{P}\right)^{1/3} \frac{M_P \sin i}{\left(M_P + M_\star\right)^{2/3}}$$

So we want P and k which we will estimate from the measured radial velocities using the following model:

$$v_R = k \sin\left(\frac{2\pi(t - t_0)}{P}\right) + v_0 = k \sin\left(2\pi(f + f_0)\right) + v_0$$

Thus in total we have four parameters: $\theta = (P, k, f_0, v_0)$

Trying out MCMC

1. Get the file 51Peg mayorqueloz95.dat.

The file provides the original dataset from 1995 which was used to find the exo-planet. The final consists of three columns of data. The first column gives the Julian date of the observations relative to some reference date. The second column gives v_R measured in meters per second and the third column gives the estimated uncertainty on v_R.

2. Get the notebook - Starting 51 Peg b MCMC - skeleton - from the Github site and run this.

Use this as a starting point for estimating the parameters - if you have time also try to get the planet mass.

A brief peek outside academia - Kaggle

Kaggle - data science competitions

But also a community - supposedly the largest data scientist community on the internet (>10^6 users)

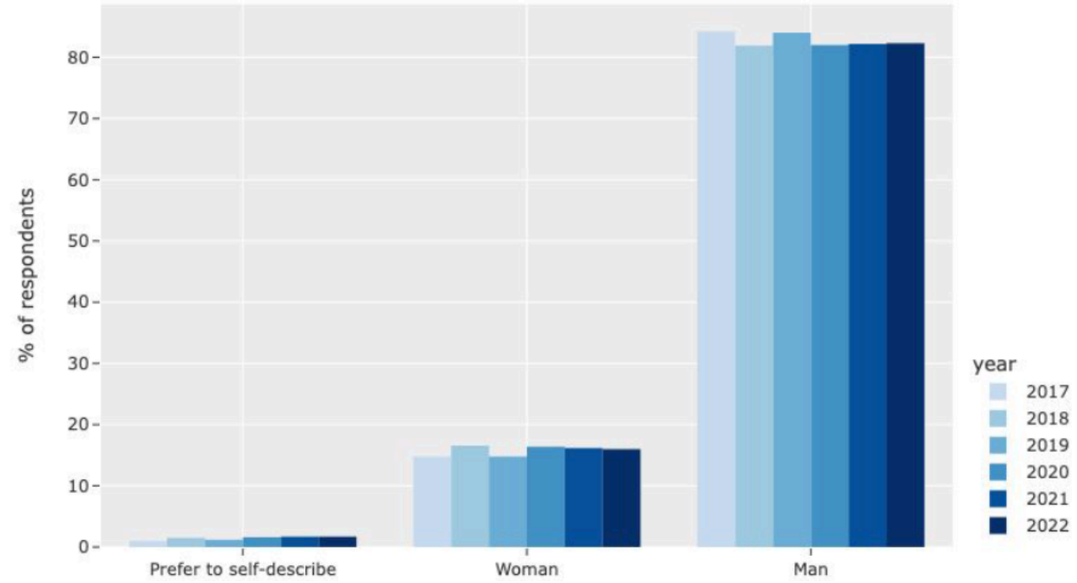
Lots of data sets (>296 000)

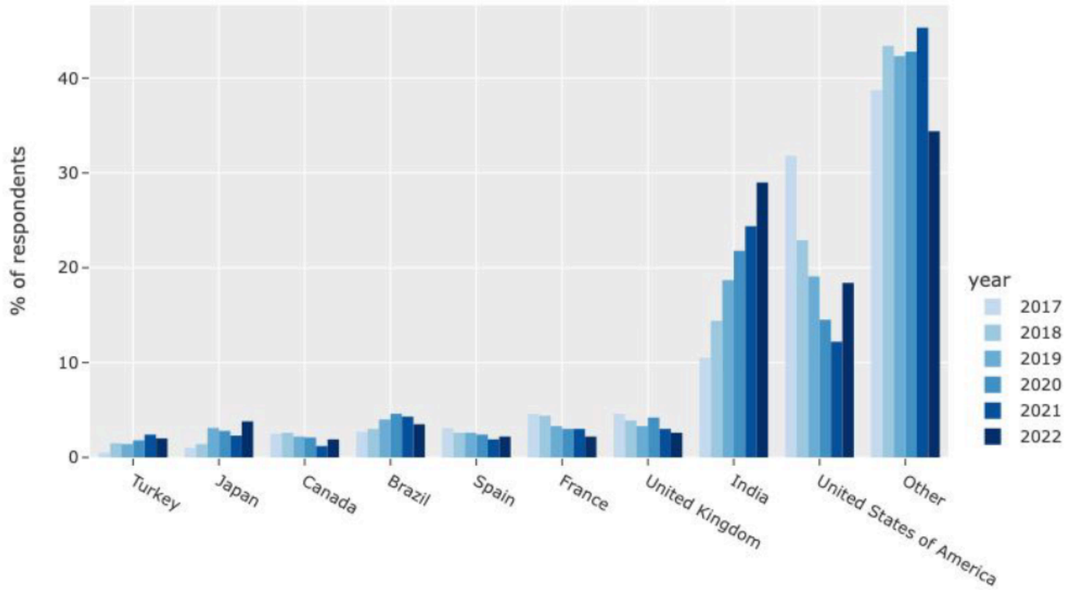
Lots of notebooks (>968 000)

Even if you do not want to compete, might be a good place to consult.

kaggle.com - note, owned by Google.

Kaggle's user survey (also freely available):



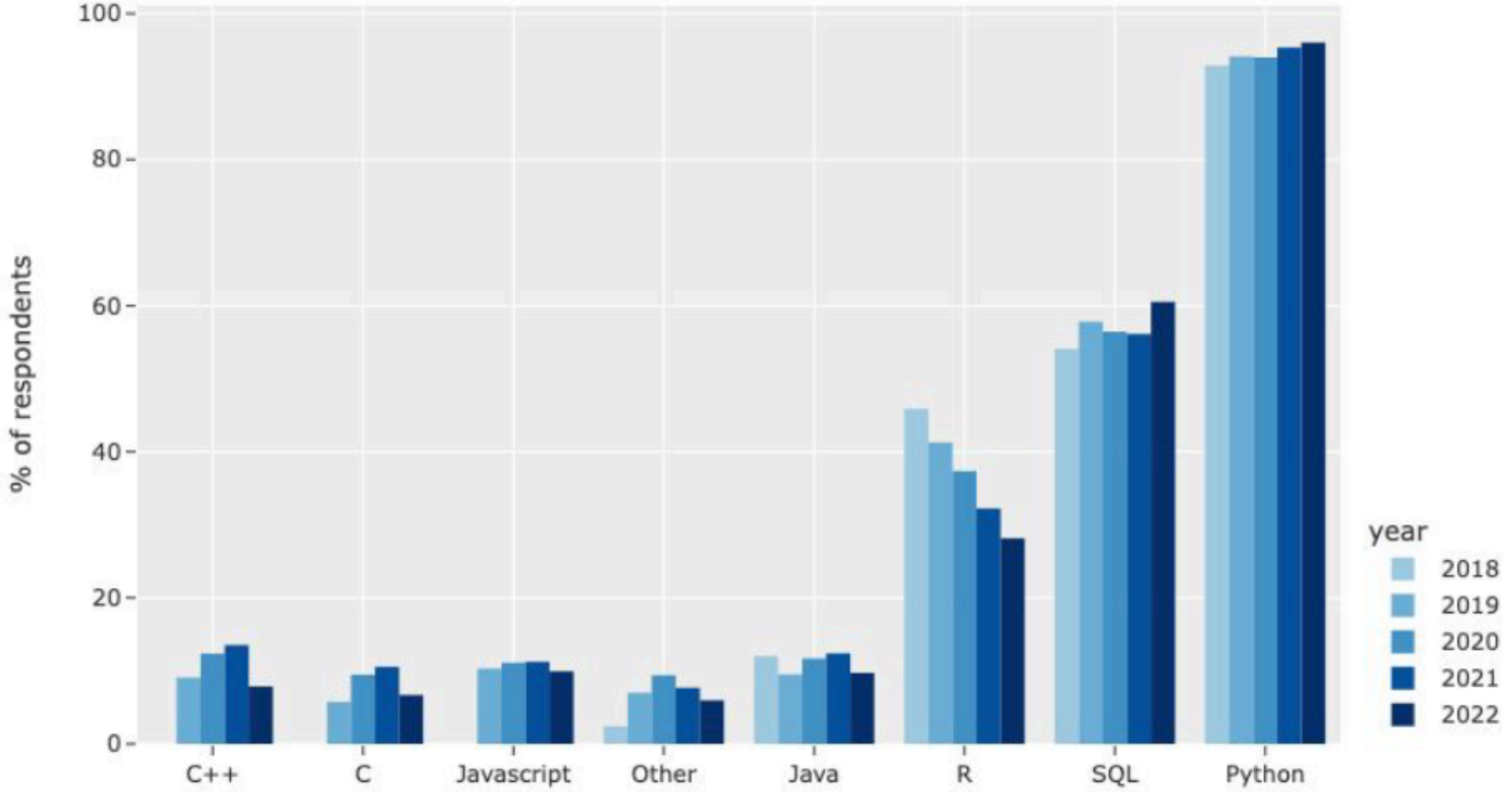


Are you surprised?

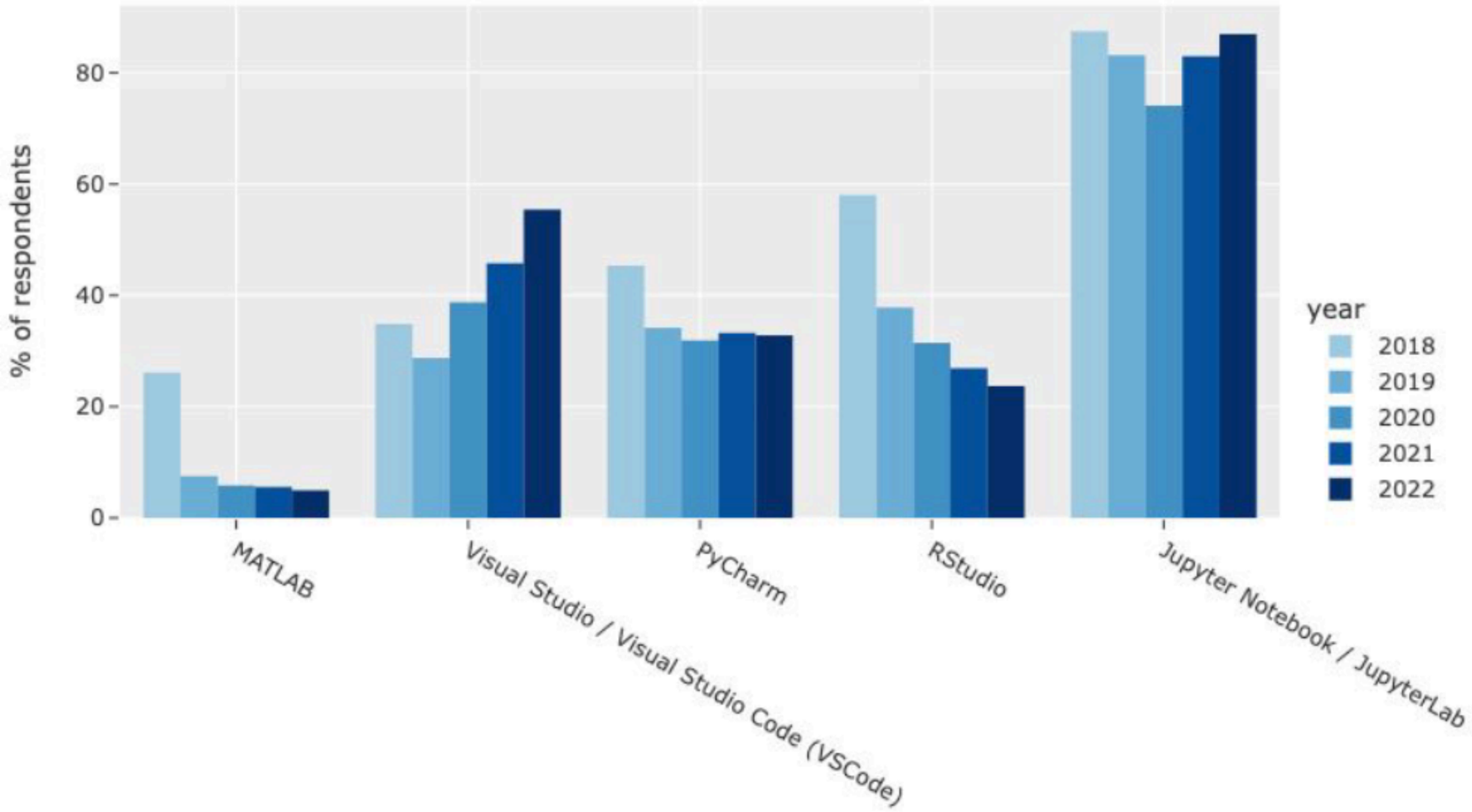
87 = 0.36%

from Portugal

Kaggle user's skills

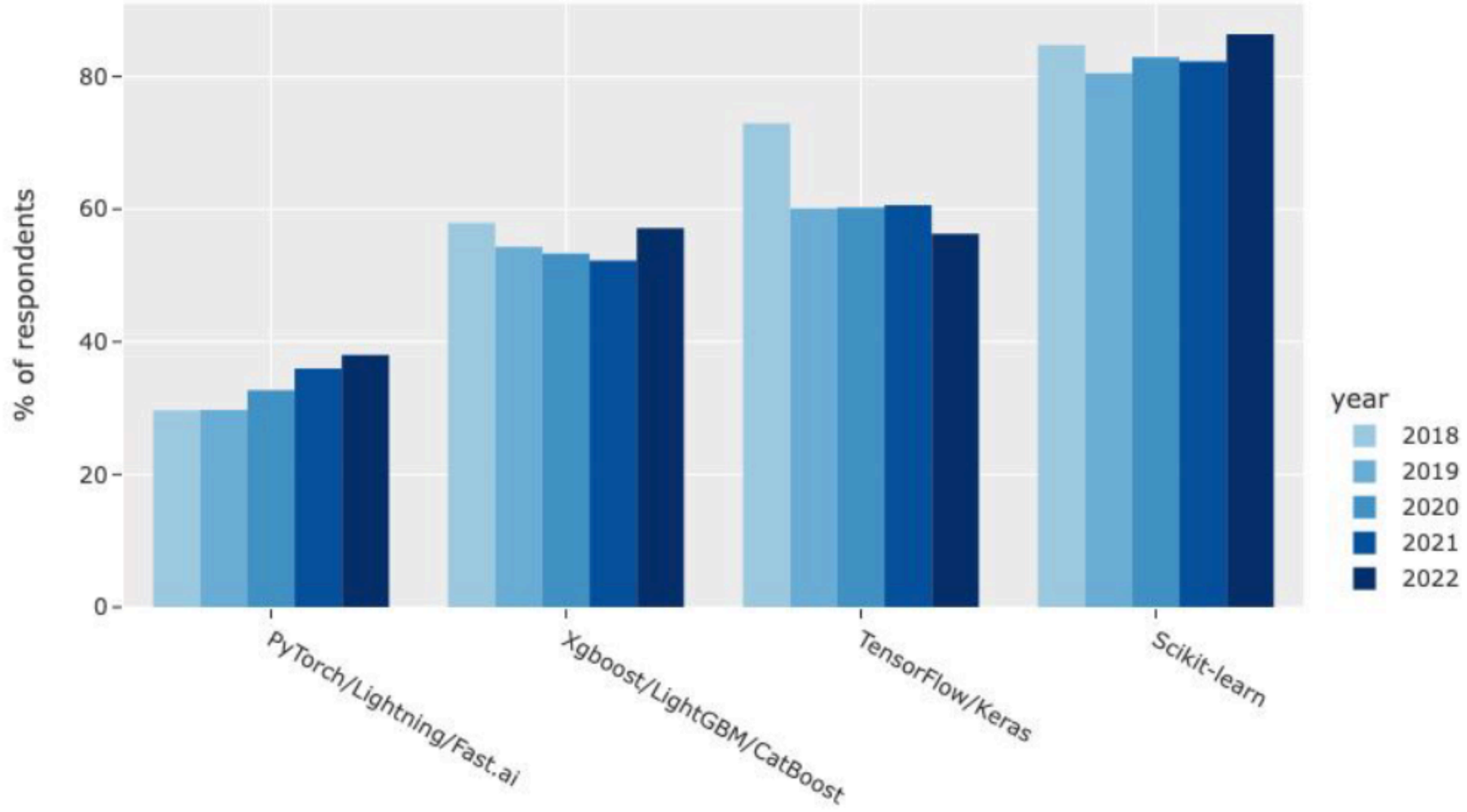


IDEs used by Kaggle data scientists



But note that only ~23,000 out of 10^6 answered the survey

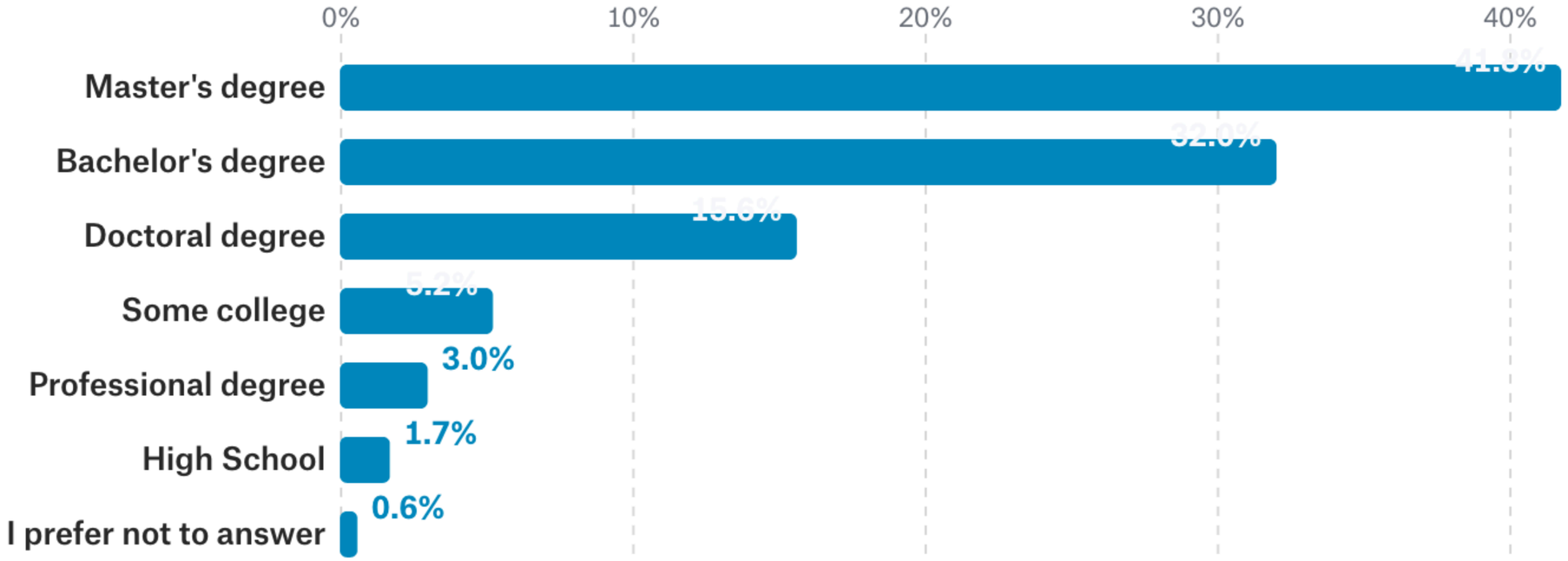
Frameworks used



Some fun survey results

From Kaggle's 2017 user survey (their first):

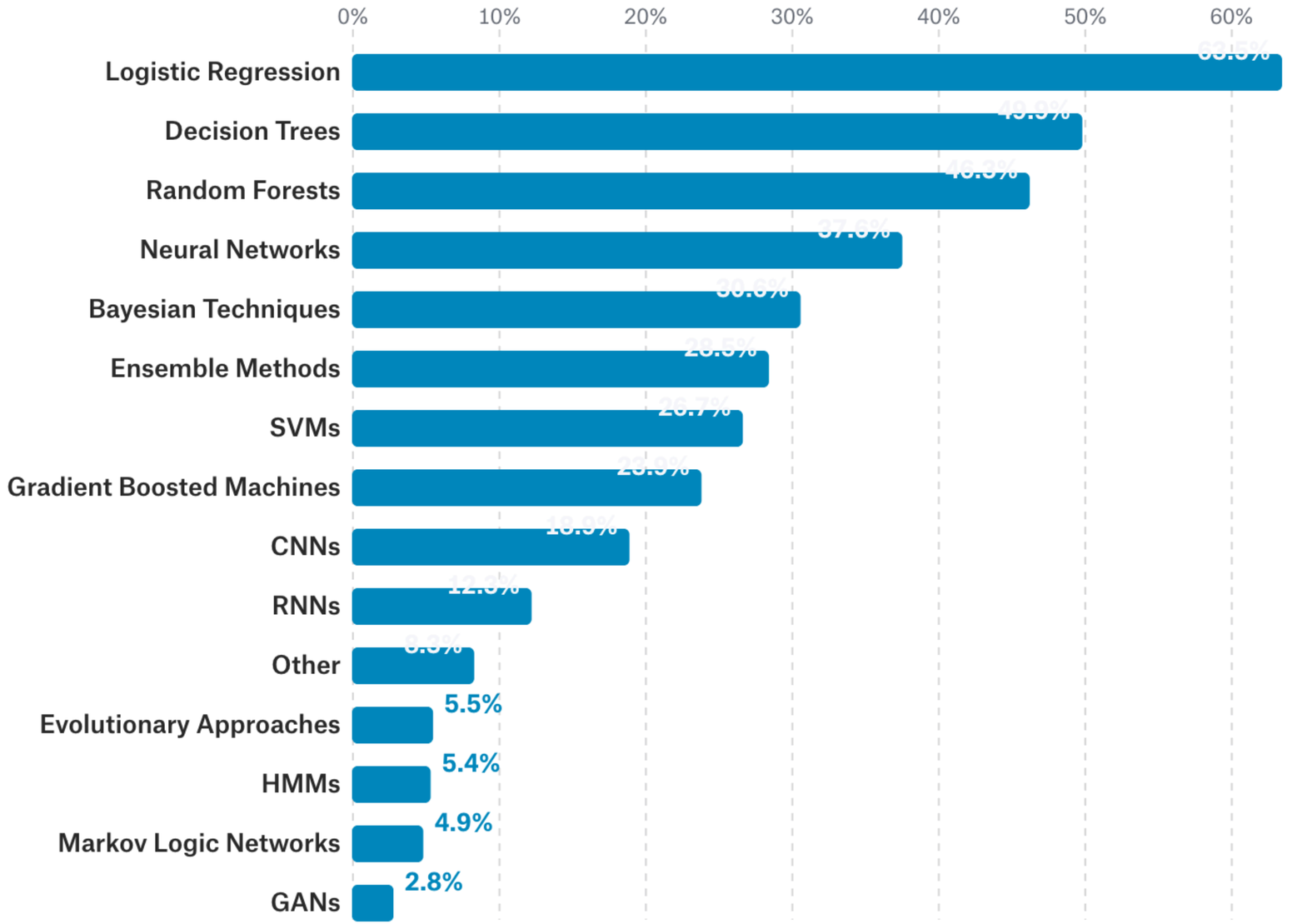
Participant's highest education level:



15,015 responses

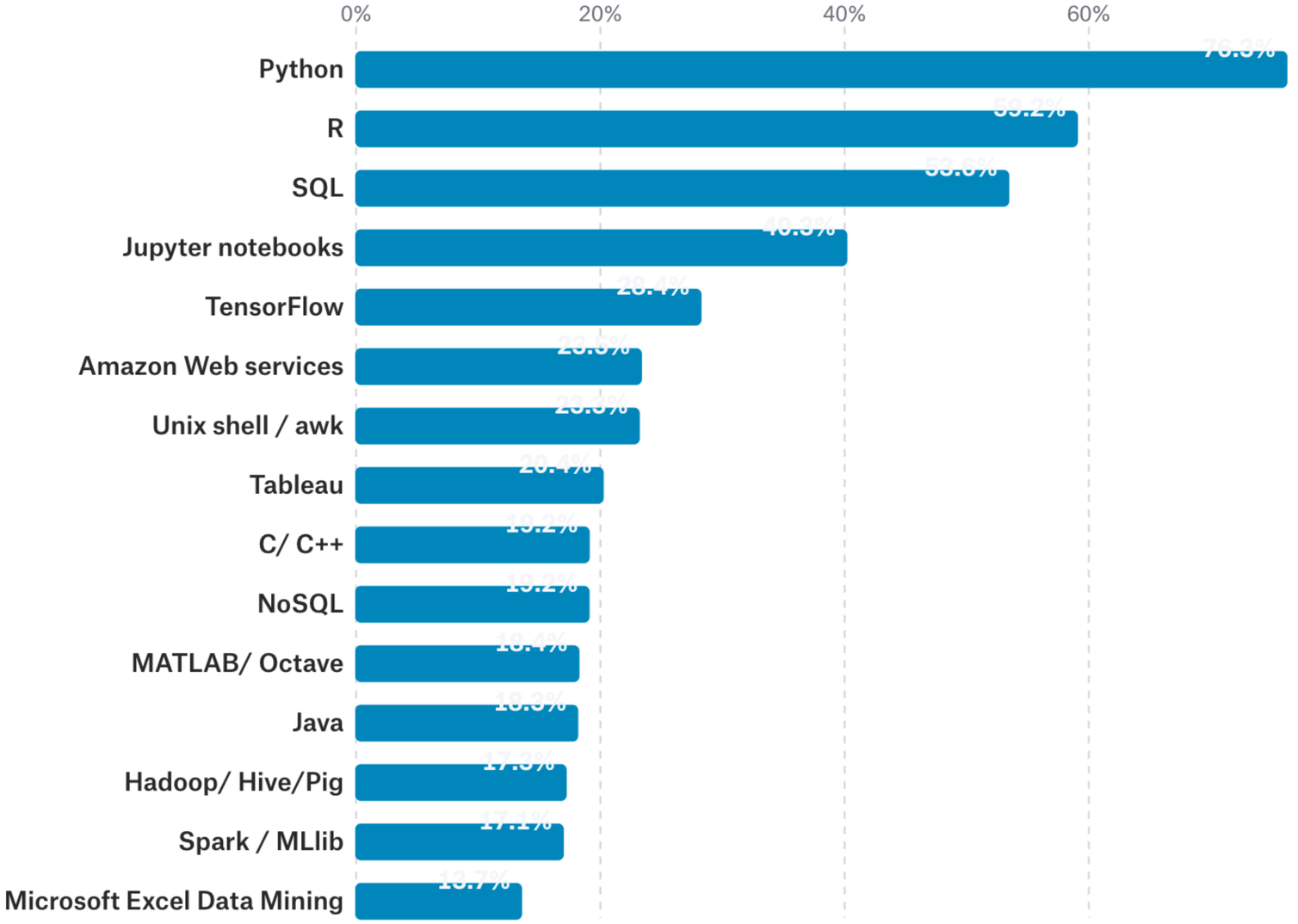
View code in Kaggle Kernels

Techniques used at work (lots we have not discussed!):



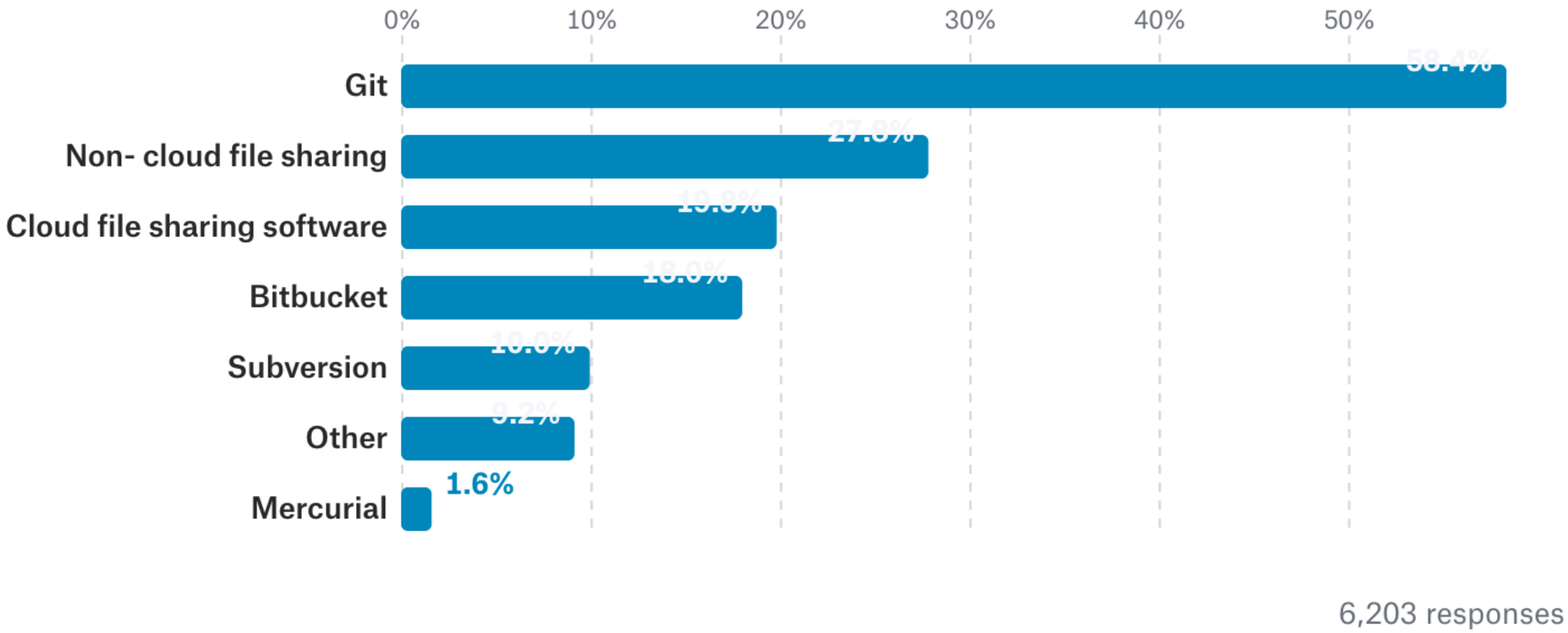
7,301 responses

Tools used at work:



7,955 responses

Code sharing used at work:



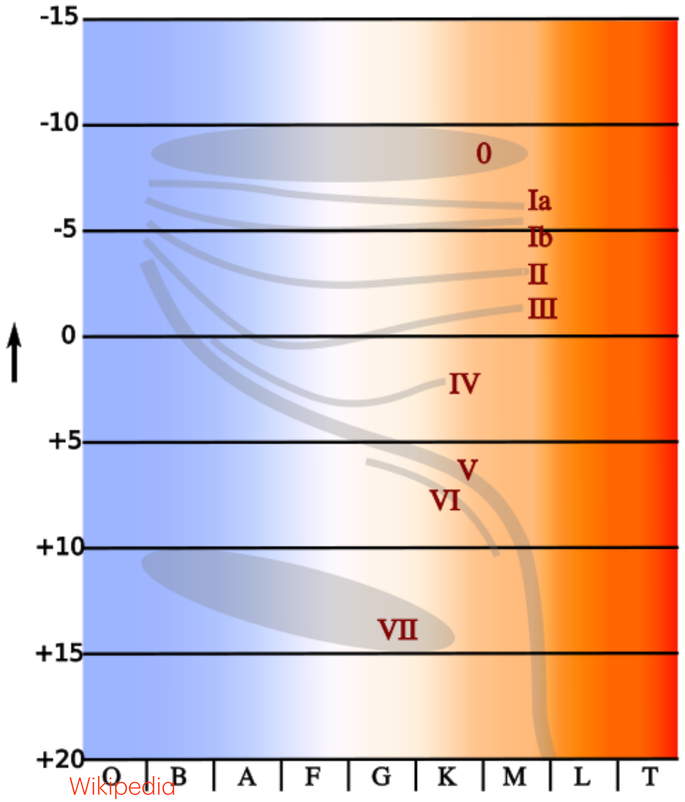
See more at:

https://www.kaggle.com/code/mhajabri/what-do-kagglers-say-about-data-science

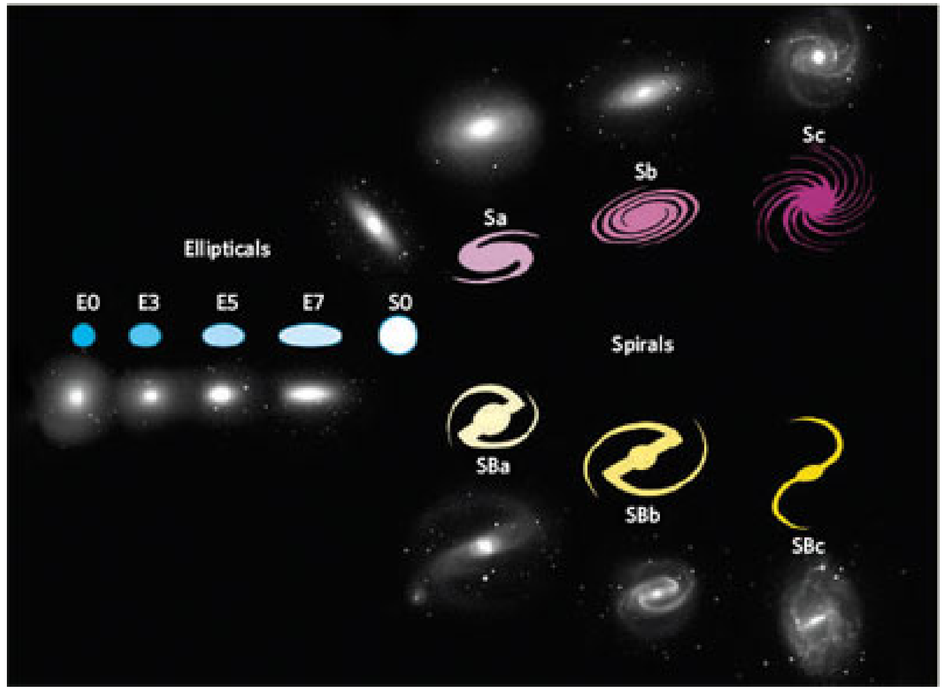
Classification

Classifying data

Astronomers very often classify stuff - this is one classical data-mining task that is very common in astronomy.



Wikipedia



Kennicutt (2006), Nature

The meaning of classification

I. Combination into known classes:

E.g GAIA:

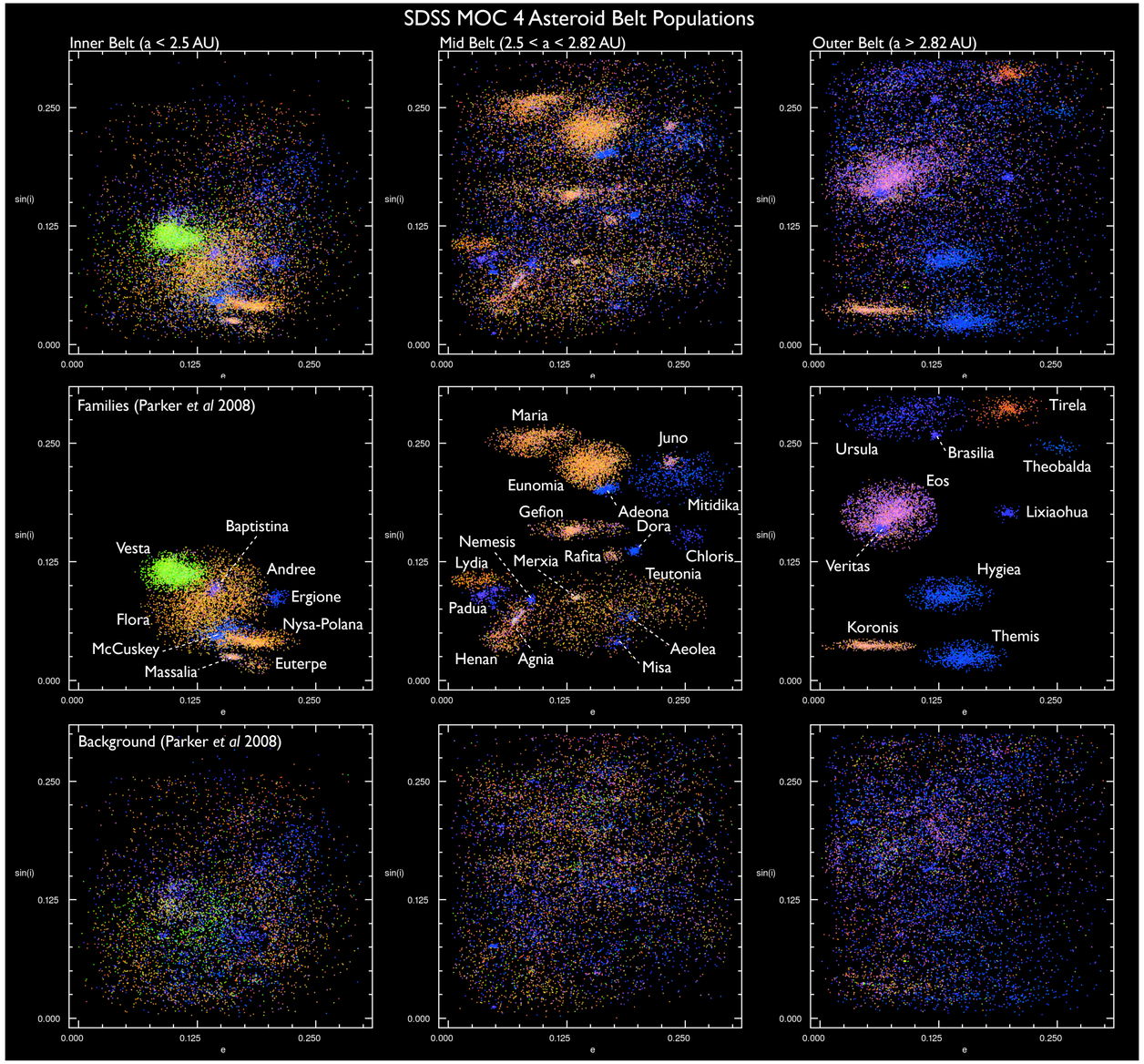
Have: Photometric observations of N objects and low-resolution spectroscopic observations of M objects.

Want: Classification of objects into stars, galaxies, quasars etc. For the stars we also want T_{eff}, log g, [Fe/H] etc. [this is actually at least two different problems!]

So how do you do that?

II. Group objects to explore their nature

Finding and classifying asteroids



Parker et al (2008, Icarus 198,138-155)

Supervised & unsupervised classification

Supervised classification:

Given: Objects with “features” $\{x_i\}$, and known class ω_j

Needed: For an object with features $\{y_i\}$, assign a class ω

It is supervised, because we know of the existence of the classes before and we have some objects that have been reliably classified in advance.

Classification of stellar spectra and galaxy images can fall into this category.

Supervised & unsupervised classification

Unsupervised classification:

Given: Objects with “features” $\{x_i\}$

Needed: Assign these to N (which can be known or given) classes.
Then use this to assign classes to new data.

In this case we know nothing, or very little about the classification of the data. Thus this is often exploratory and the results can be a lot harder to interpret.

Finding asteroid families fall in this category.

Techniques for classification

- Bayesian decision criteria
- Nearest neighbour methods & friends.
- Linear models.
- Artificial Neural Networks.
- And many more (cladistics, hierarchical trees, **support vector machines** etc etc.)

For all:

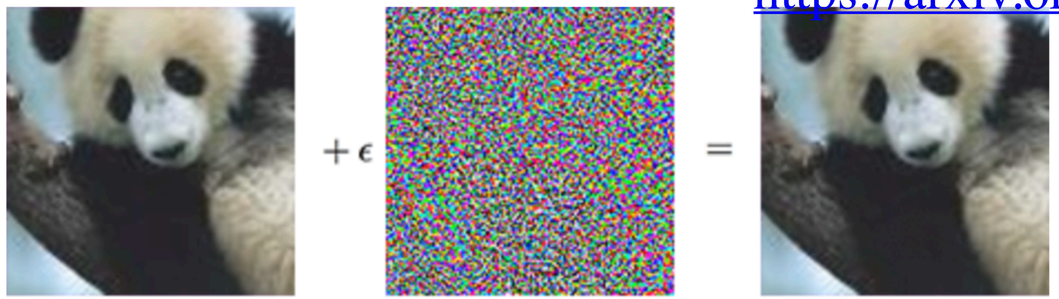
What is feature extraction?

- A crucial part of data classification/data mining - it is how you go about choosing the input data to the training algorithm.
- Depends strongly on the scientific question under consideration.
- For spectra it could be emission lines and absorption lines.
- For an image magnitudes, colours, light concentration, light profiles etc.
- Often it can be advisable to use something like principal component analysis (later) to guide your choice.
- It could also be the full image/full spectrum - this is often the way taken by deep learning algorithms (but with some modifications).

Classification - and training

Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

https://arxiv.org/abs/1412.6572



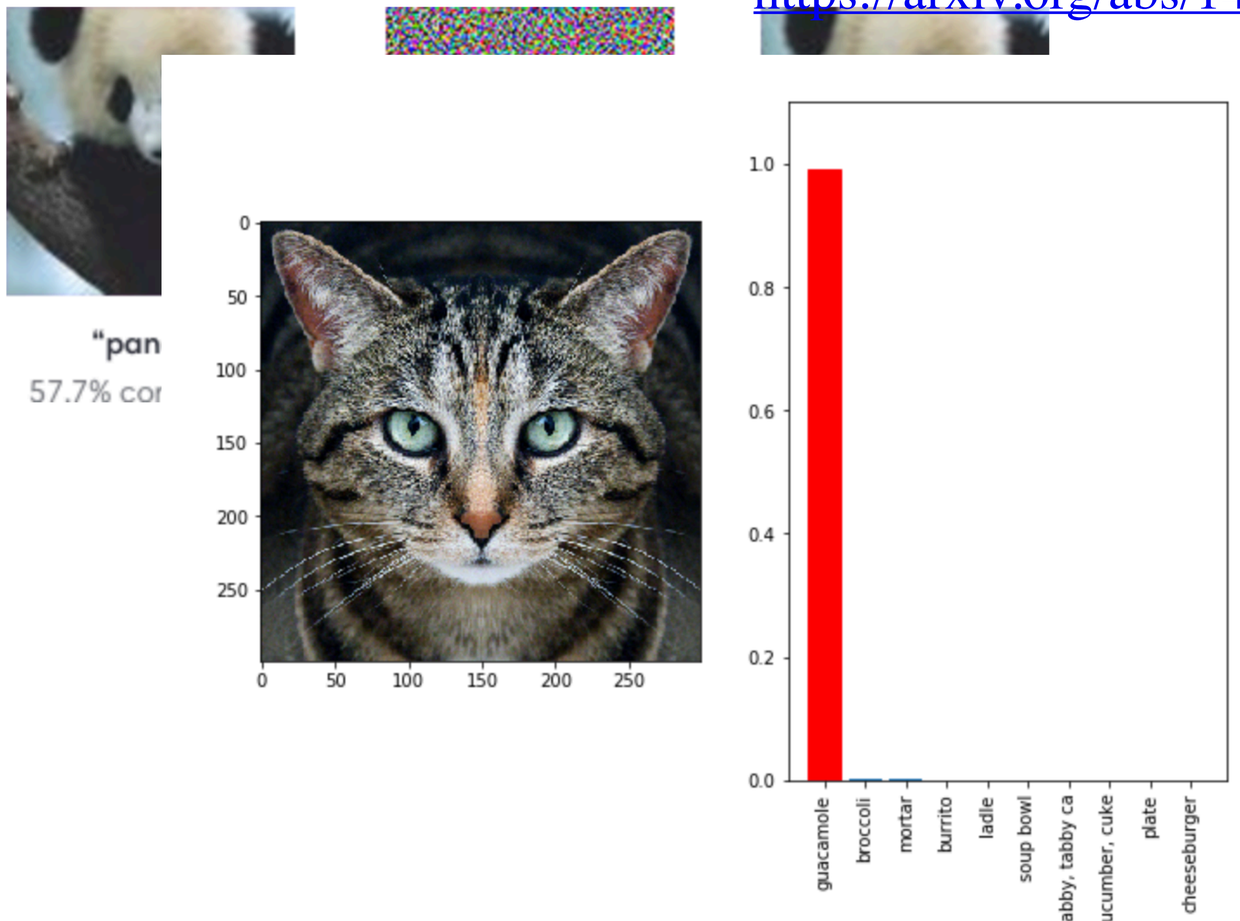
"panda"
57.7% confidence

"gibbon"
99.3% confidence

Classification - and training

Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

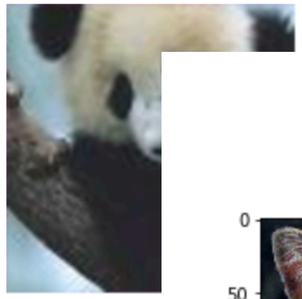
https://arxiv.org/abs/1412.6572



Classification - and training

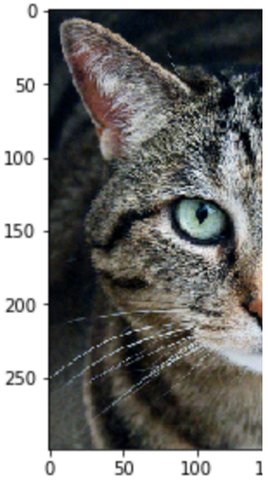
Machine learning typically relies on you having good training data covering the possible results, but even so it sometimes fails:

https://arxiv.org/abs/1412.6572

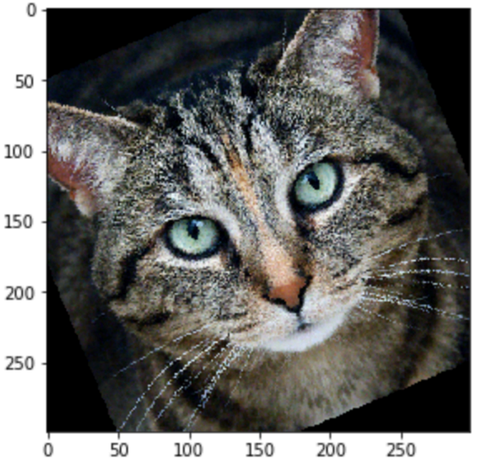


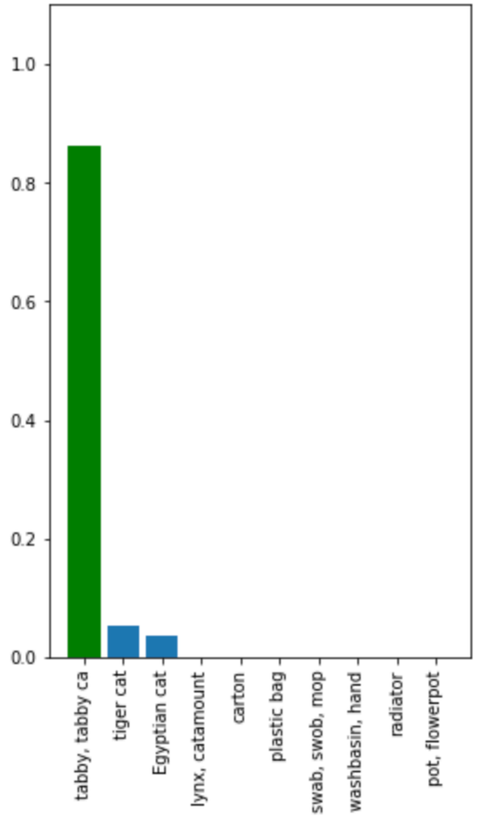
"pan

57.7% cor



However:





An example: Bayesian classification of galaxies

Starting point: Morphological classifications by Nair & Abraham (2010), downloaded from Vizir & matched to SDSS absolute magnitudes I calculated separately.

The task: Design a classifier that given a galaxy colour will return the morphological class.

Notation:

ω_0	Elliptical
ω_1	Spiral
$\vec{x}$	Observables/Features - here: g-r colour

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

$p(\omega_i|\vec{x})$ is the **posterior** probability

$p(\vec{x}|\omega_i)$ is the **likelihood**

$p(\omega_i)$ is the **prior** probability of a class

$$p(\vec{x}) = \sum_i p(\vec{x}|\omega_i)p(\omega_i)$$

ensures the probabilities are normalised but as it is independent of the classes we can ignore it here.

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

Classification is then very straightforward:

$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class } 0 \text{ (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class } 1 \text{ (spiral)}$$

But this supposes we know $p(\vec{x}|\omega_i)$ and $p(\omega_i)$

Bayes' theorem & optimal class assignment

Bayes' theorem for our case:

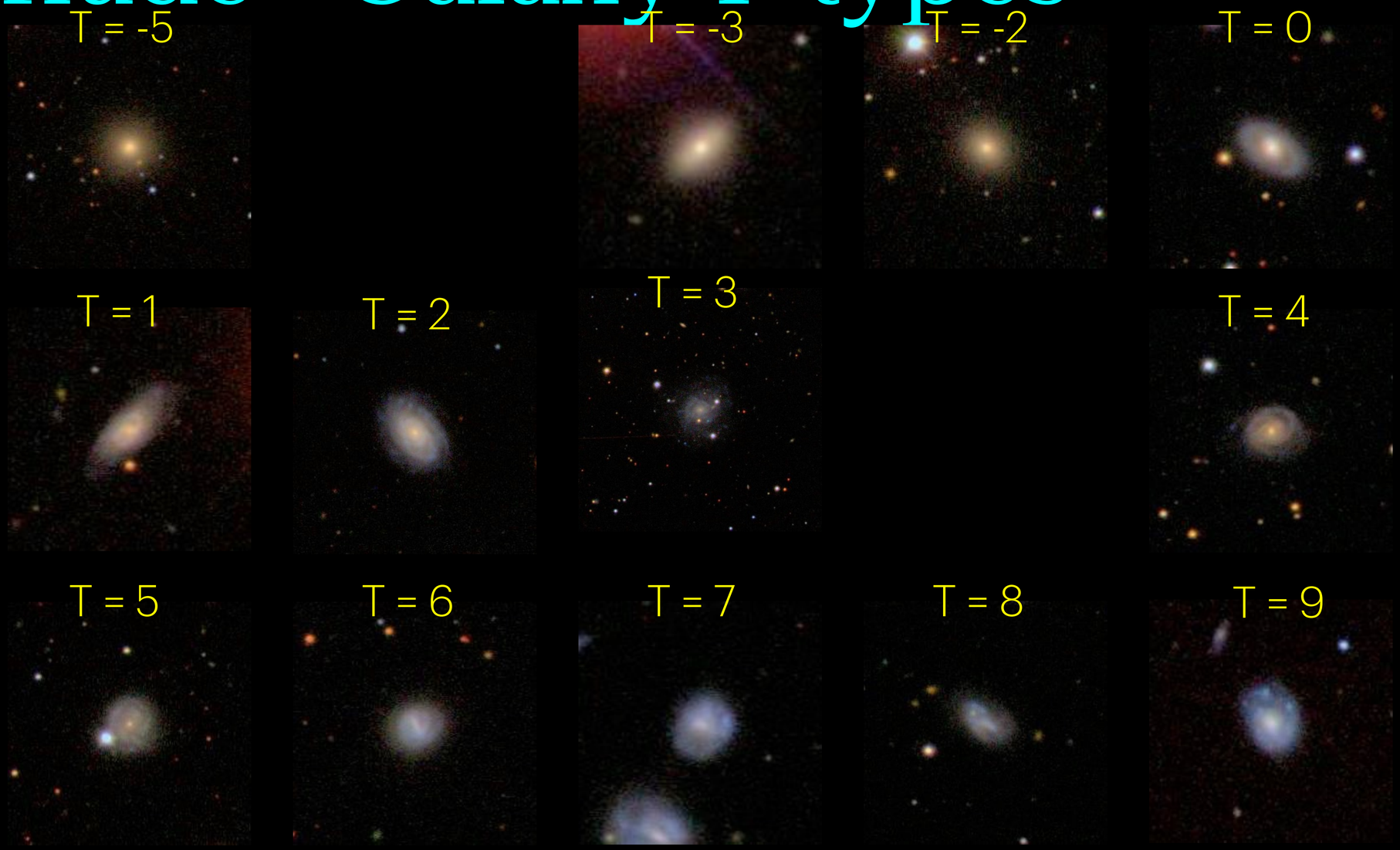
$$p(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i)p(\omega_i)}{p(\vec{x})}$$

Classification is then very straightforward:

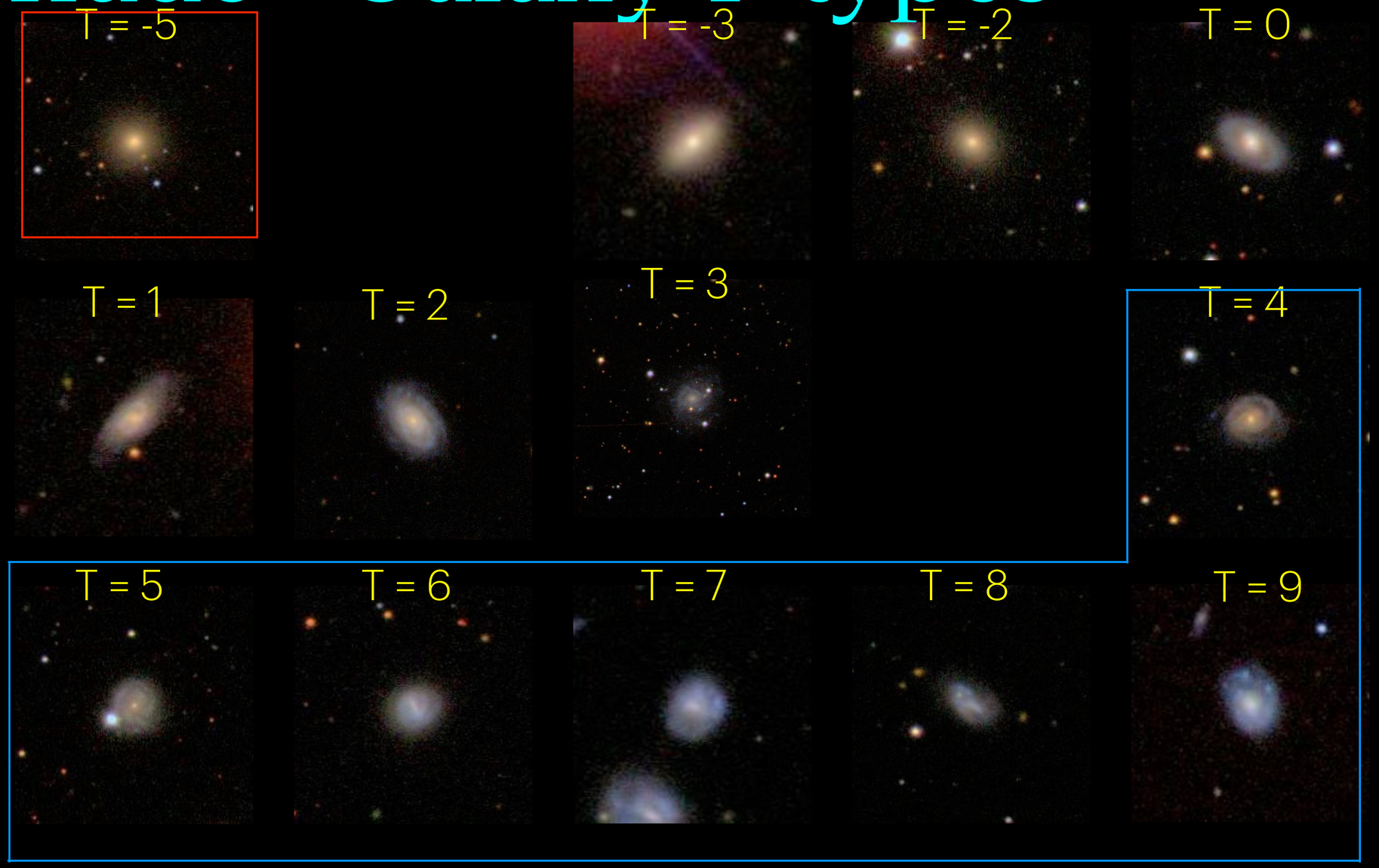
$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class } 0 \text{ (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class } 1 \text{ (spiral)}$$

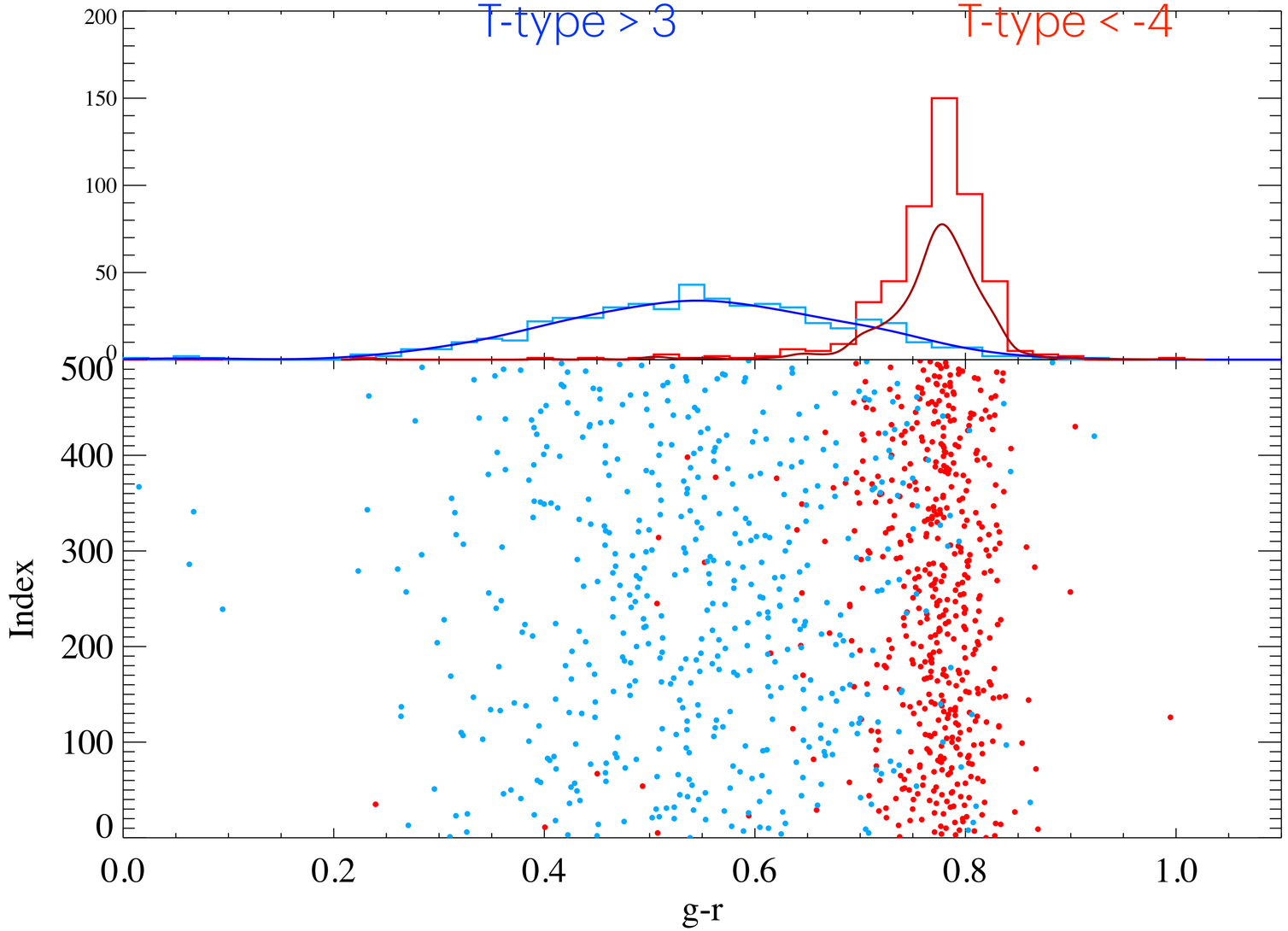
But this supposes we know $p(\vec{x}|\omega_i)$ and $p(\omega_i)$
If we assume all classes are equally probable, we just need $p(\vec{x}|\omega_i)$ and for this we can use the tools from earlier: Histograms, kernel estimates or estimating the mean and standard deviation of a Gaussian.

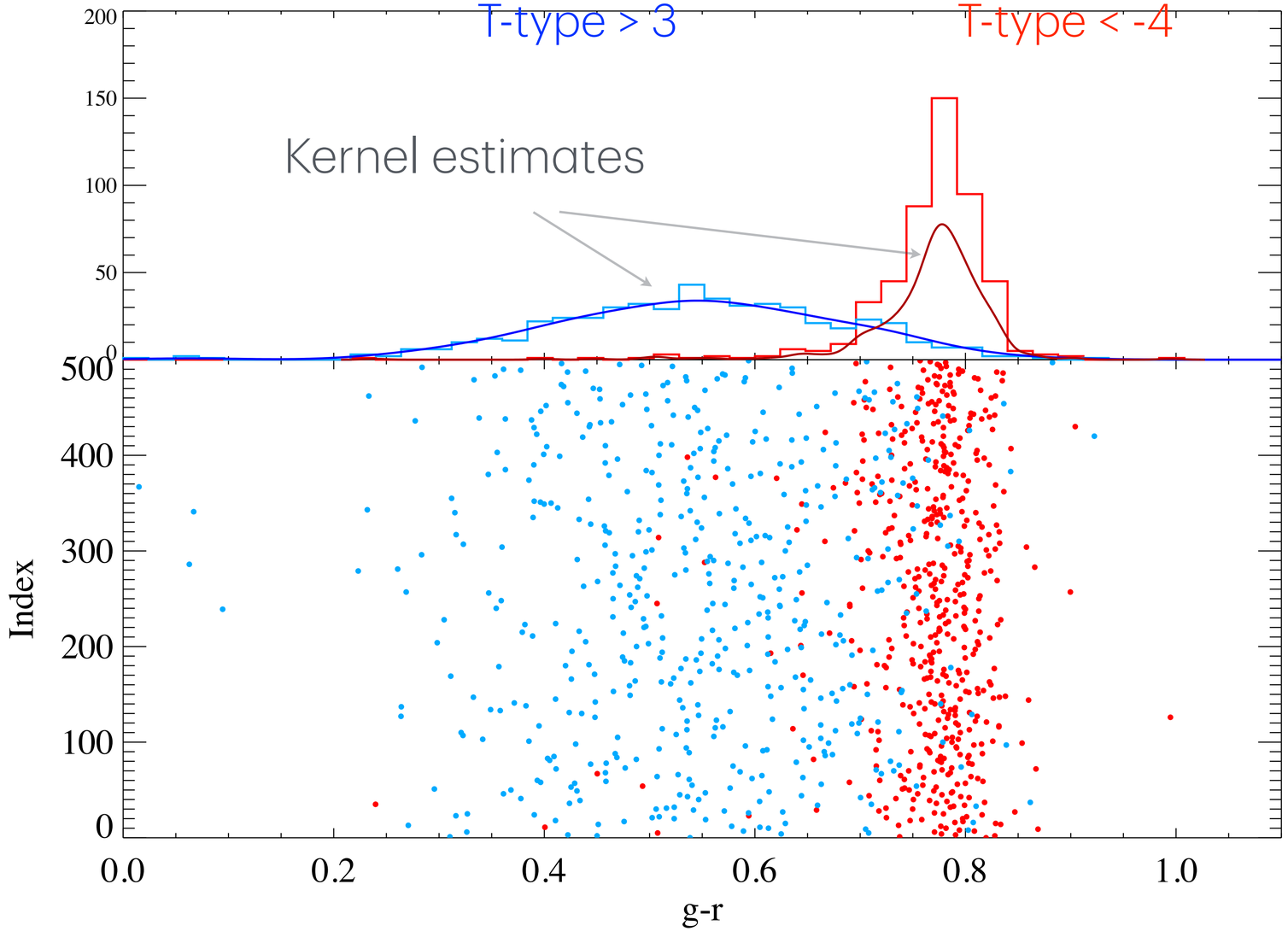
Interlude - Galaxy T-types

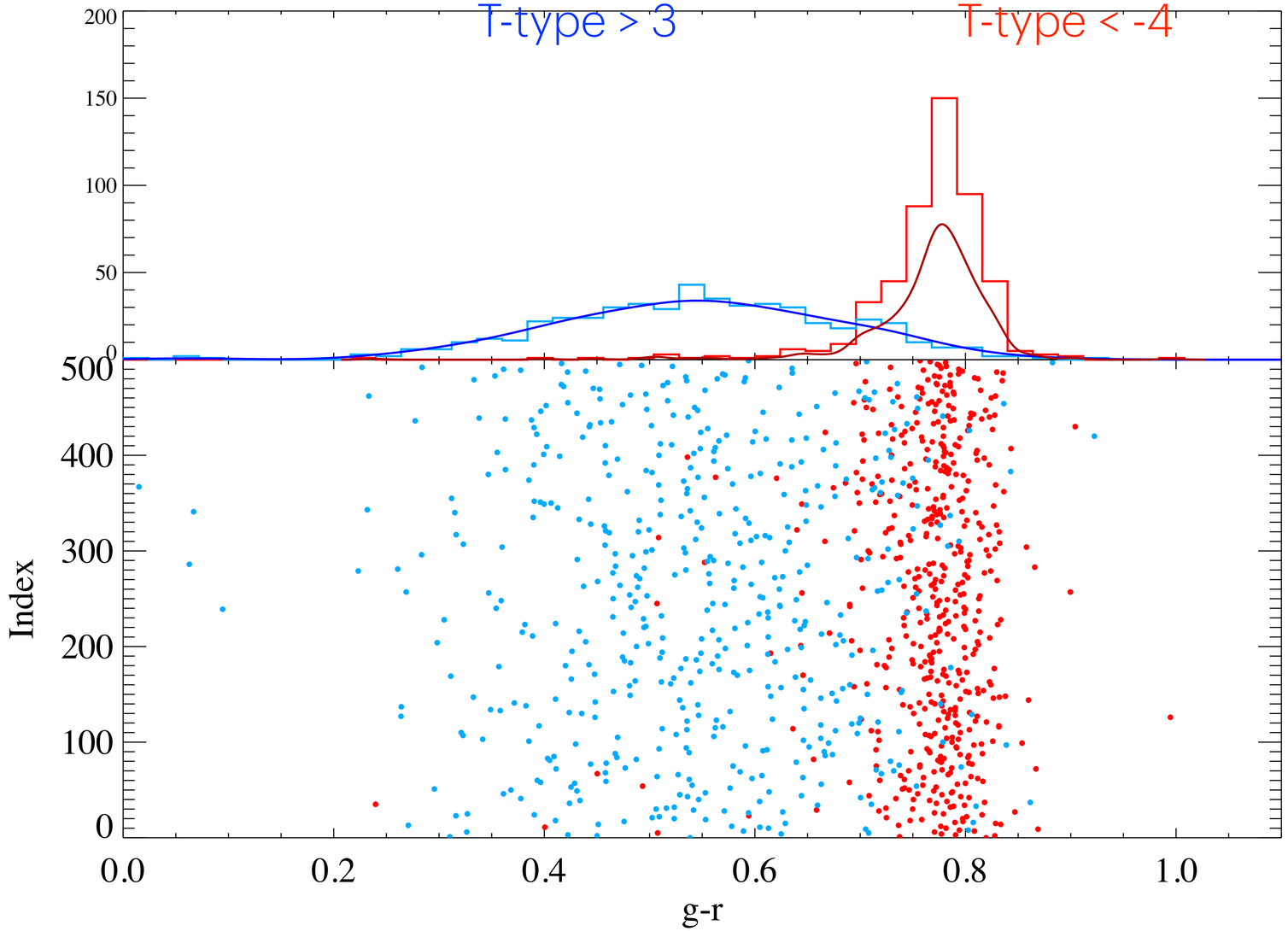


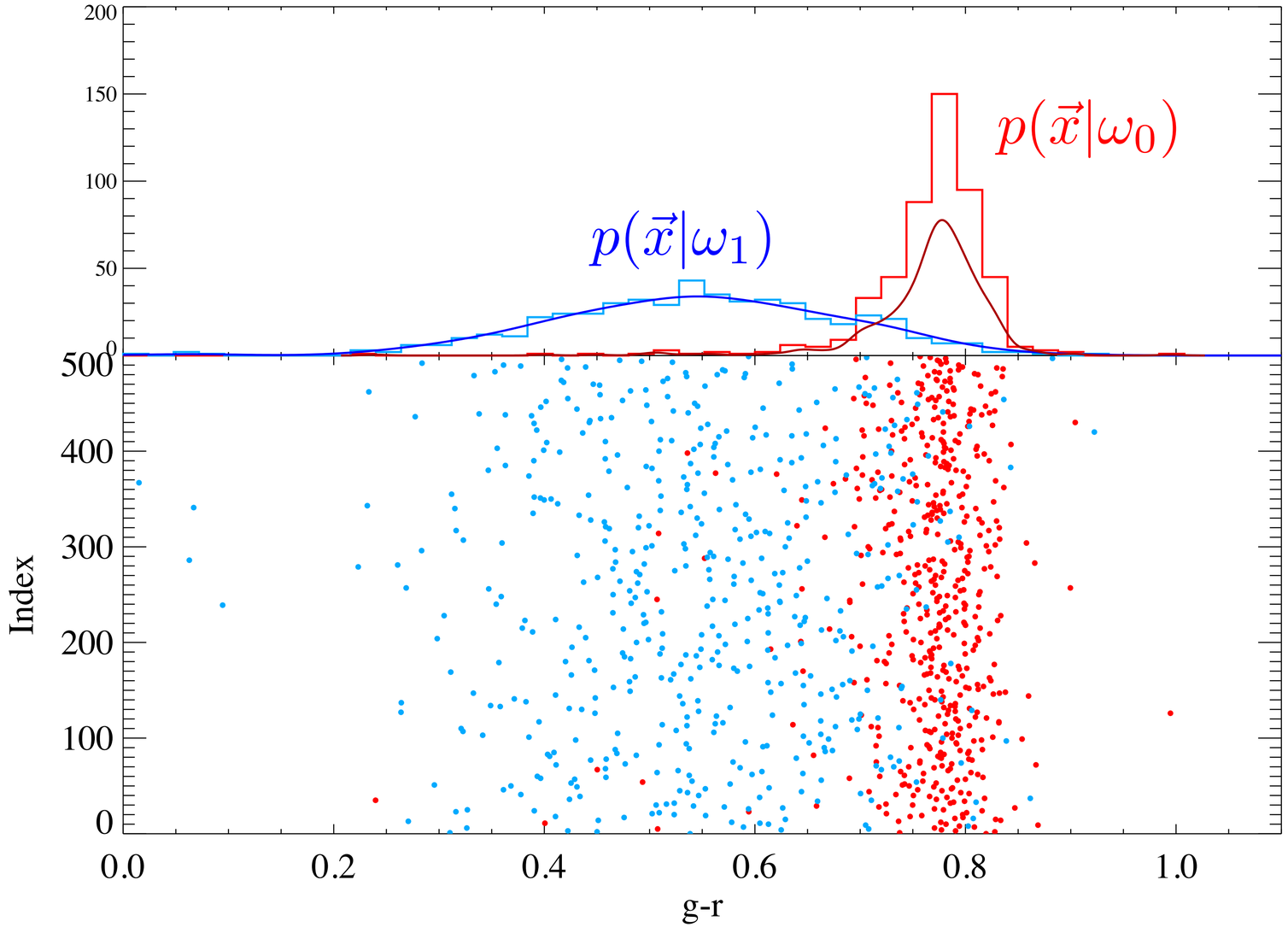
Interlude - Galaxy T-types



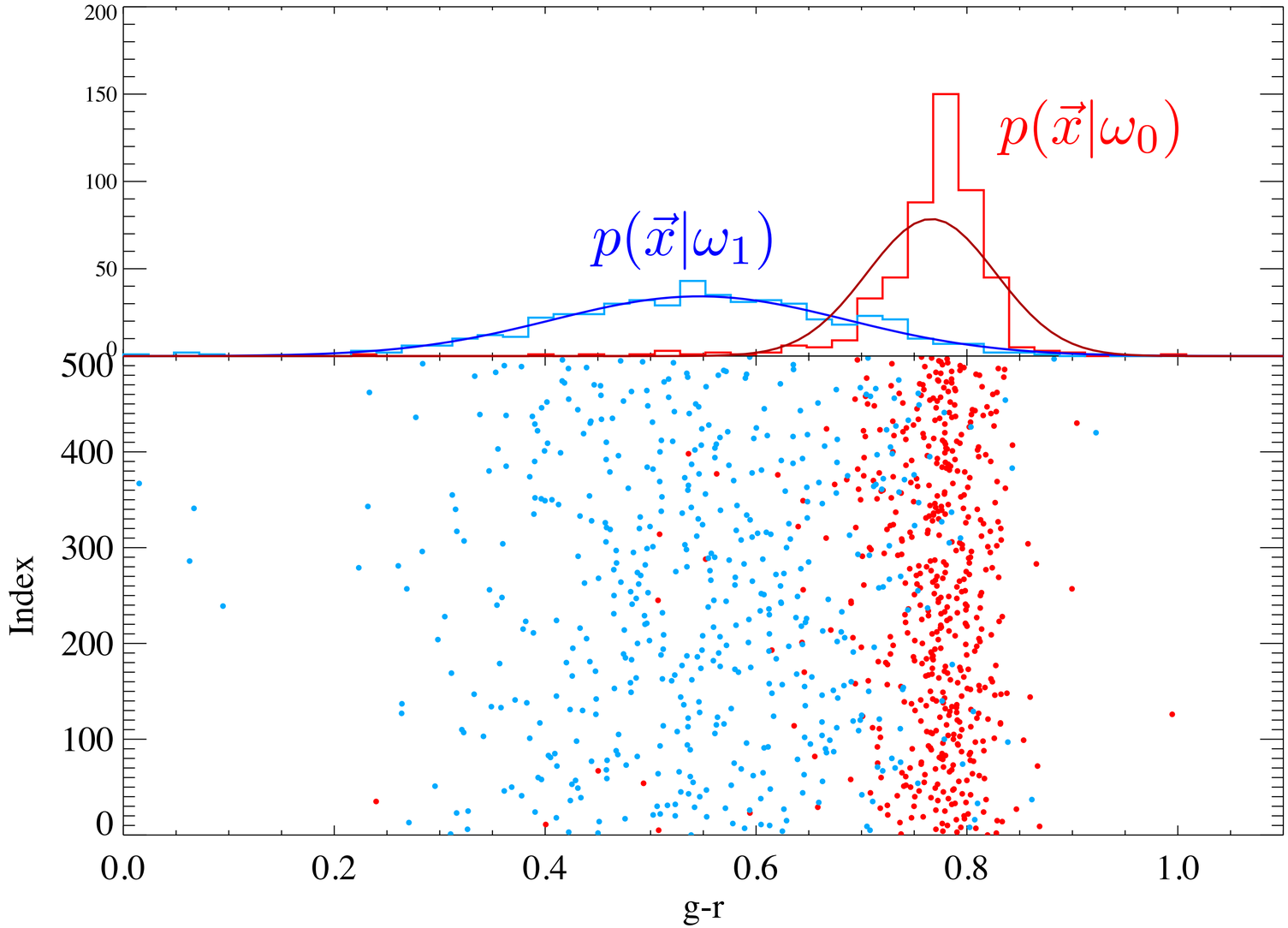

T-type > 3
T-type < -4
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r


T-type > 3
T-type < -4
Kernel estimates
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r

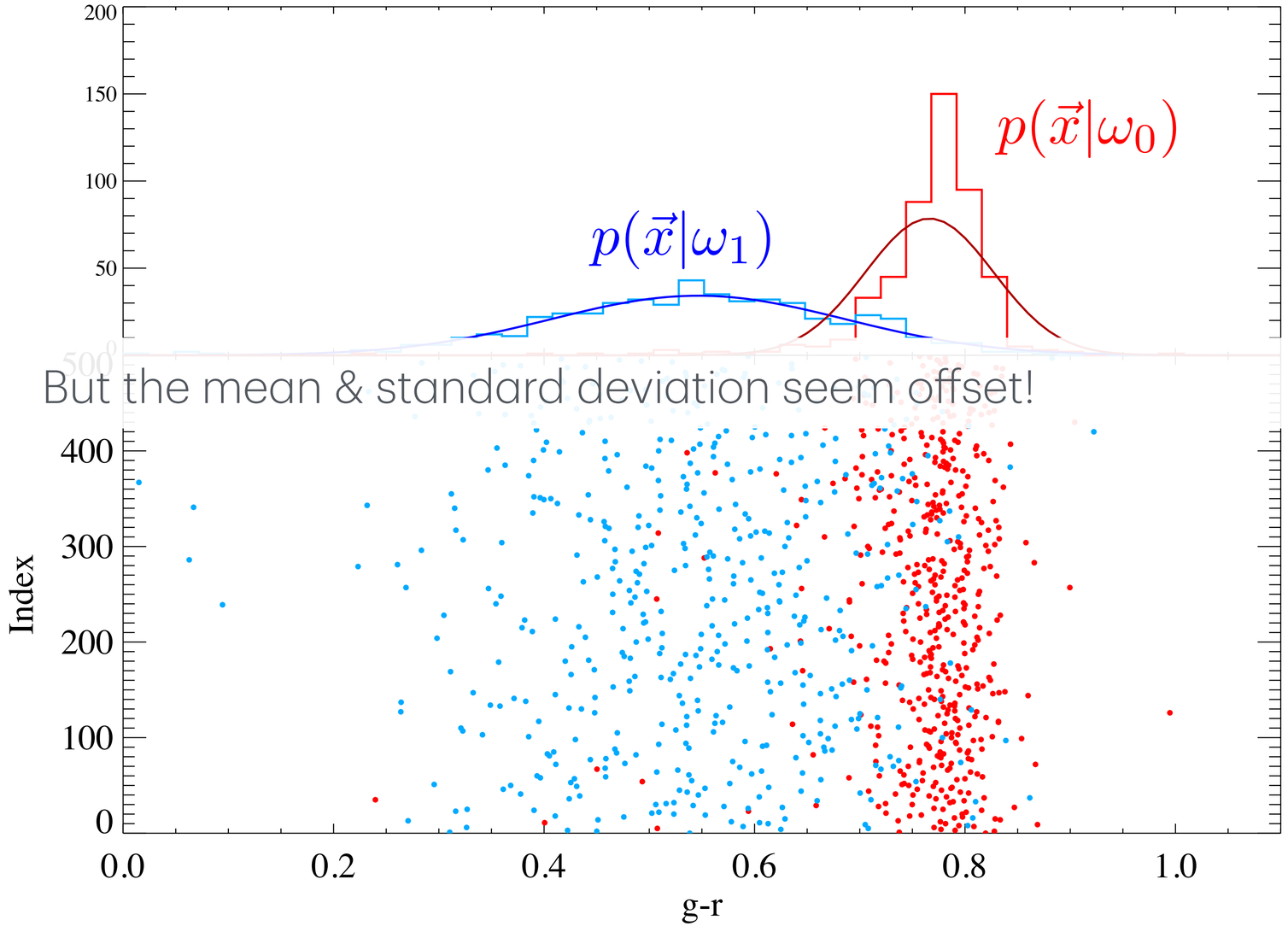

T-type > 3
T-type < -4
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r


$p(\vec{x}|\omega_1)$
$p(\vec{x}|\omega_0)$
200
150
100
50
0
500
400
300
200
100
0
Index
0.0
0.2
0.4
0.6
0.8
1.0
g-r

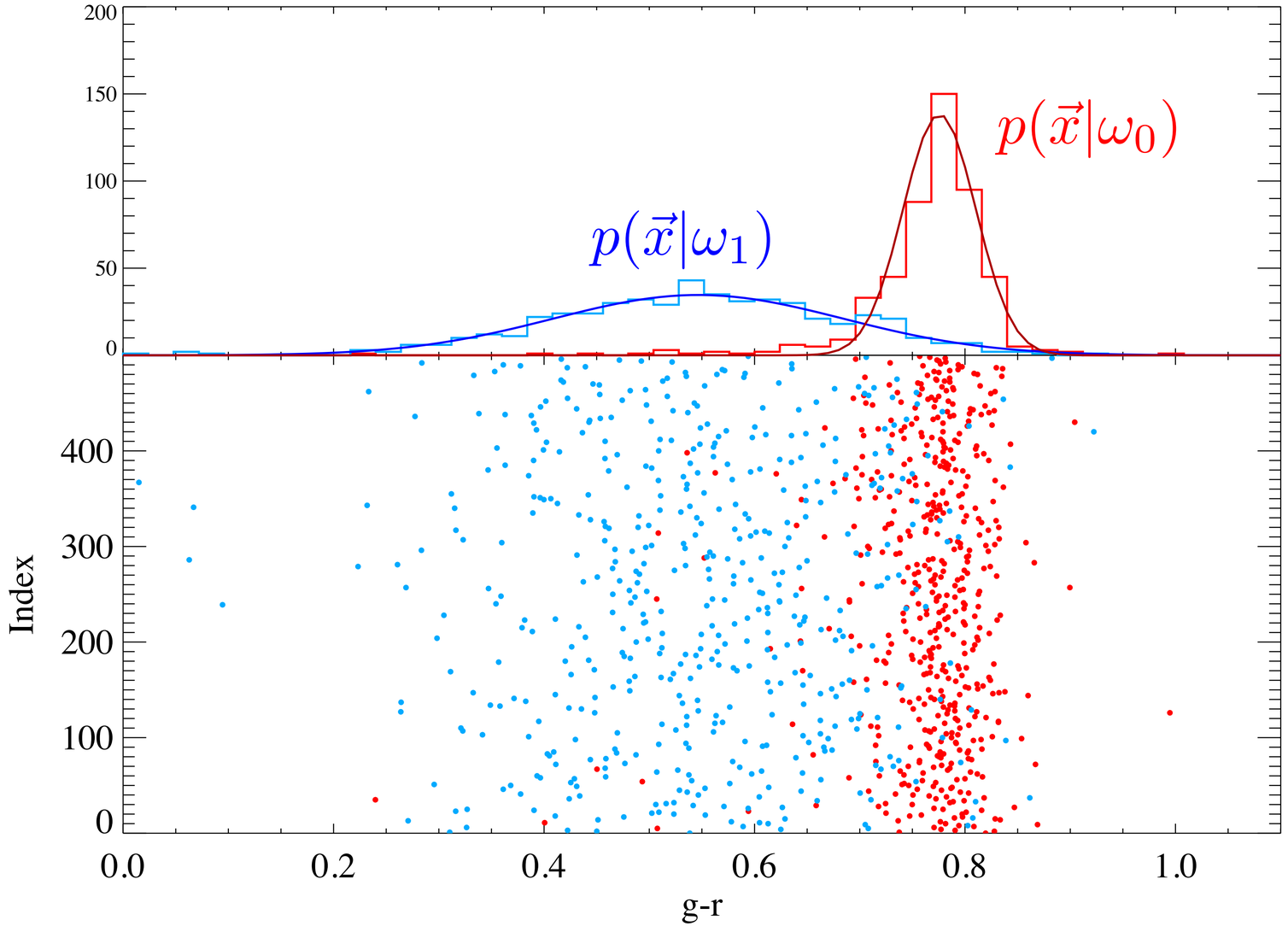
A Gaussian approximation is easier to work with:

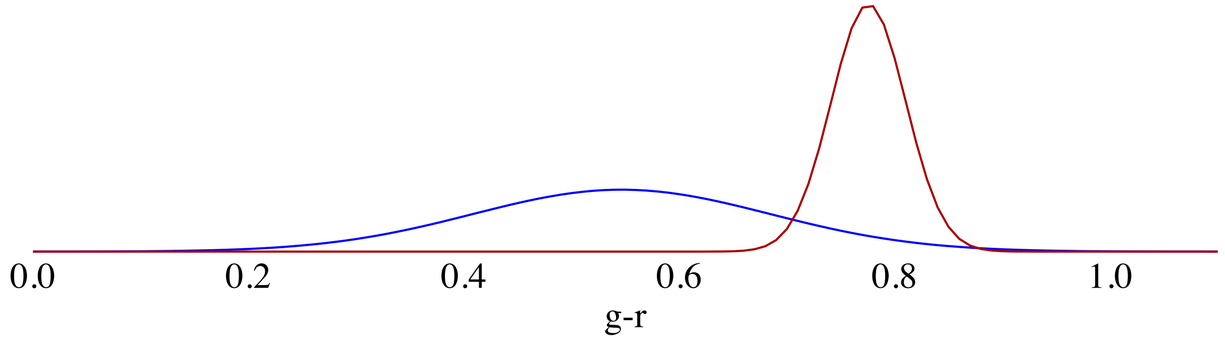


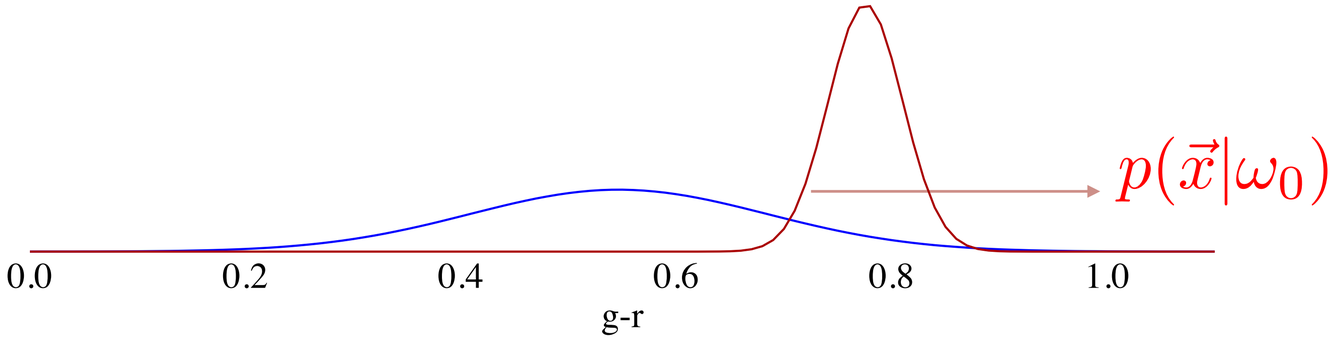
A Gaussian approximation is easier to work with:



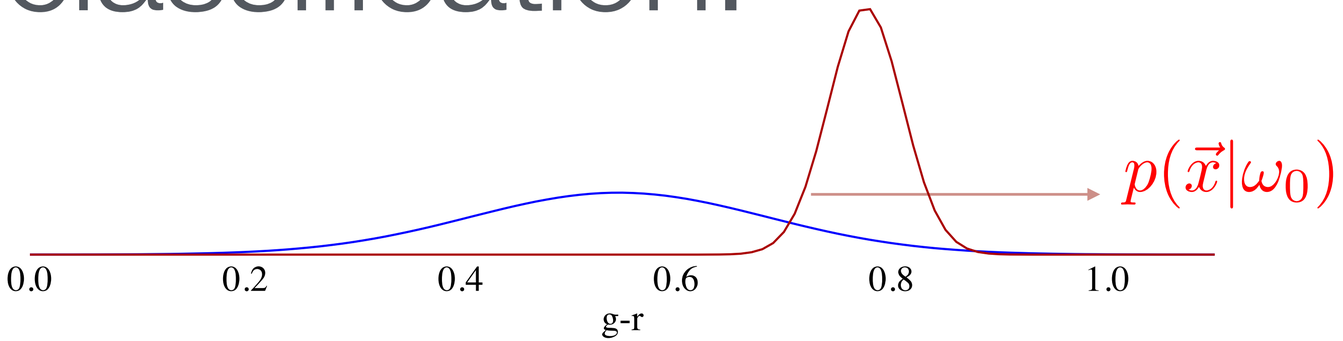
Maybe robust estimators are better (median, MAD):



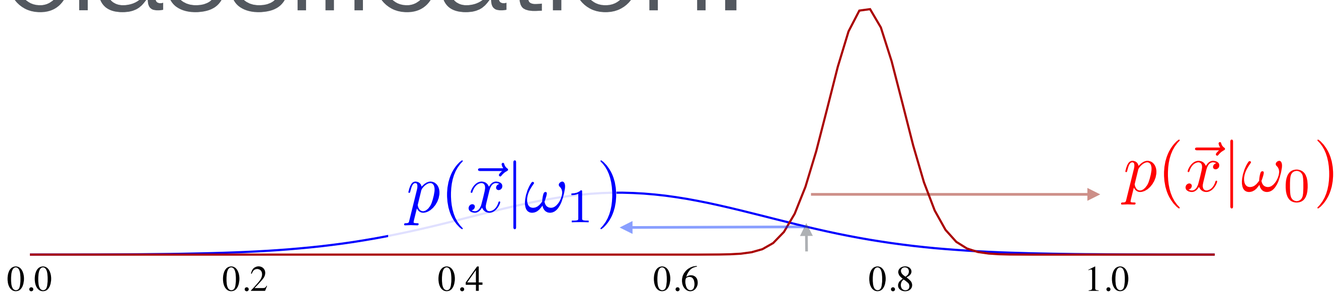

0.0
0.2
0.4
0.6
0.8
1.0
g-r


$p(\vec{x}|\omega_0)$
0.0
0.2
0.4
0.6
0.8
1.0
g-r

Bayesian classification:

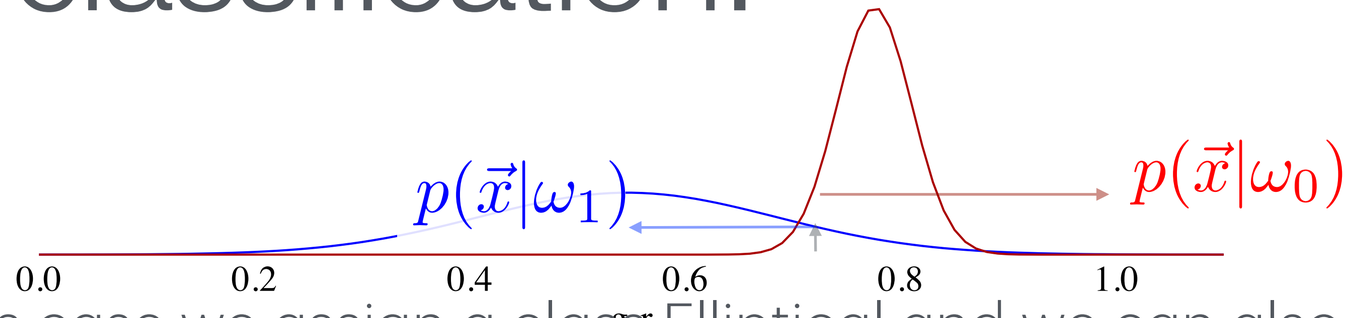


Bayesian classification:

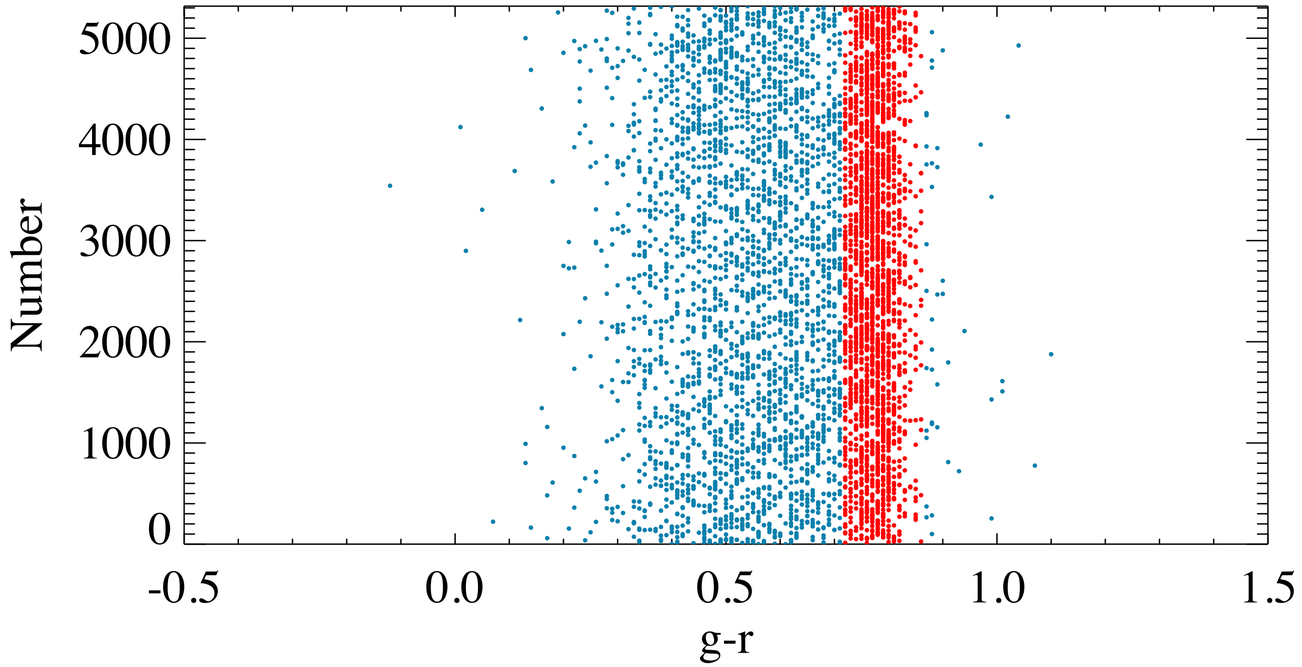


So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:

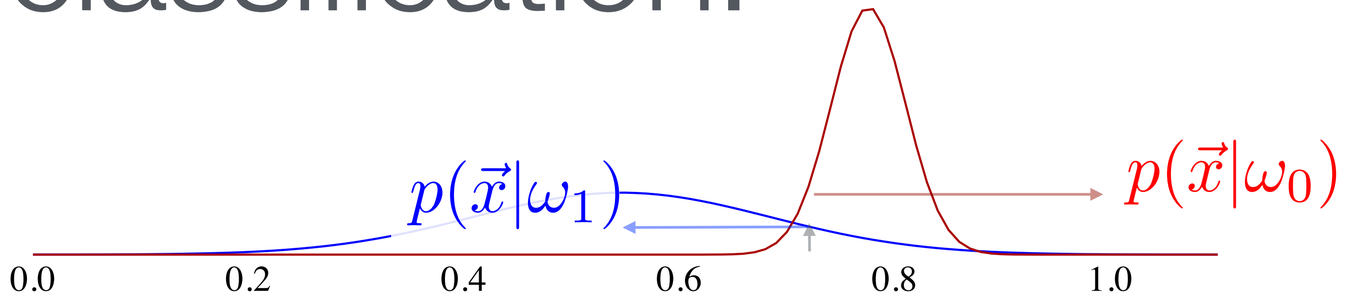
Bayesian classification:



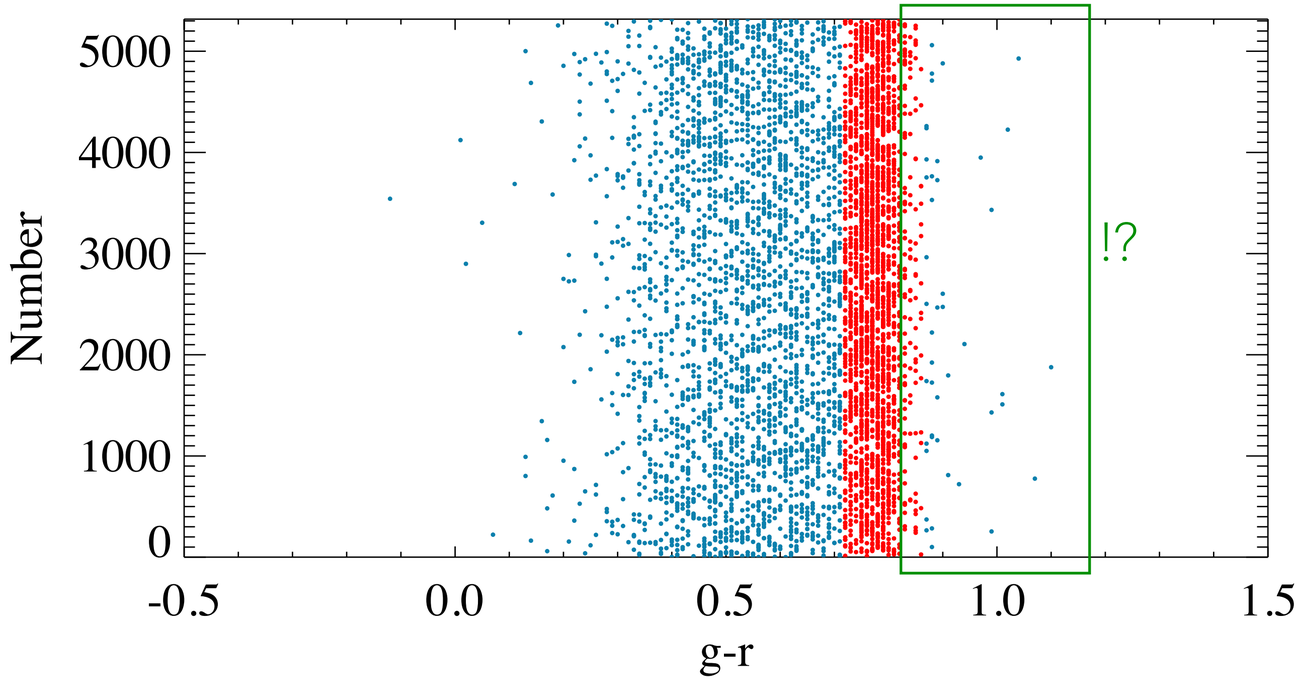
So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



Bayesian classification:



So in this case we assign a class Elliptical and we can also give the likelihood relative to the spiral class. We do this for all objects:



Bayesian classification - decision surfaces

We can write our class separation criterion as:

or $$p(\omega_0|\vec{x}) = p(\omega_1|\vec{x})$$

$$\ln p(\vec{x}|\omega_0)p(\omega_0) = \ln p(\vec{x}|\omega_1)p(\omega_1)$$

This will define decision regions, sometimes a surface

Taking logarithms and assuming a Gaussian PDF we have:

$$\left(\frac{x-\mu_0}{2\sigma_0}\right)^2 - \left(\frac{x-\mu_1}{2\sigma_1}\right)^2 = \ln\frac{\sigma_1}{\sigma_0}$$

$$p(\vec{x}|\omega_0) = \frac{1}{\sqrt{2\pi}\sigma_0}e^{-(x-\mu_0)^2/2\sigma_0^2}$$

$$p(\vec{x}|\omega_1) = \frac{1}{\sqrt{2\pi}\sigma_1}e^{-(x-\mu_1)^2/2\sigma_1^2}$$

So generally you get two solutions!

Validation

Getting a solution is a start - now we need to test (validate) the solution.
For this we have another sample which has known class - we apply our classifier and compare the predicted class with the known class.

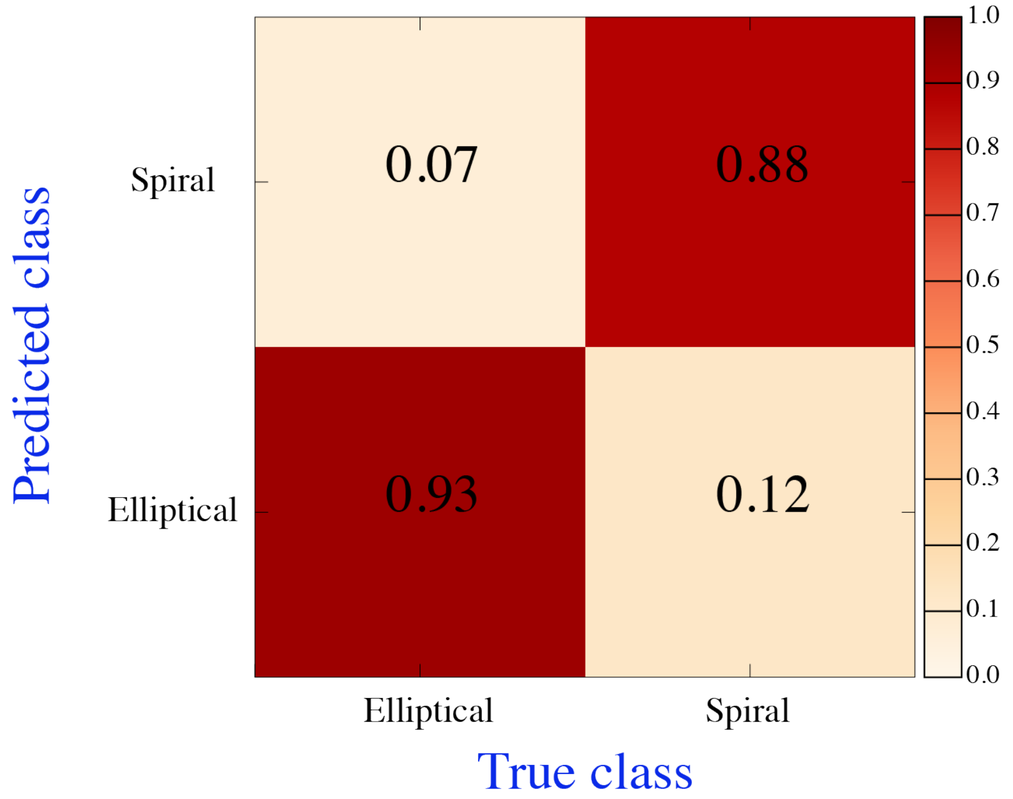
$$\frac{\#\text{ cases where predicted class != true class}}{\#\text{ of cases}} = \text{Misclassification rate}$$

In our case: 10% (also if we use kernel estimation)

But if we remove our gap in T-types - the misclassification rate goes up to 25%...

Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.



In python:

```
confusion_matrix in sklearn.metrics.
```

In Julia:

```
ConfusionMatrix in EvalMetrics.jl
```

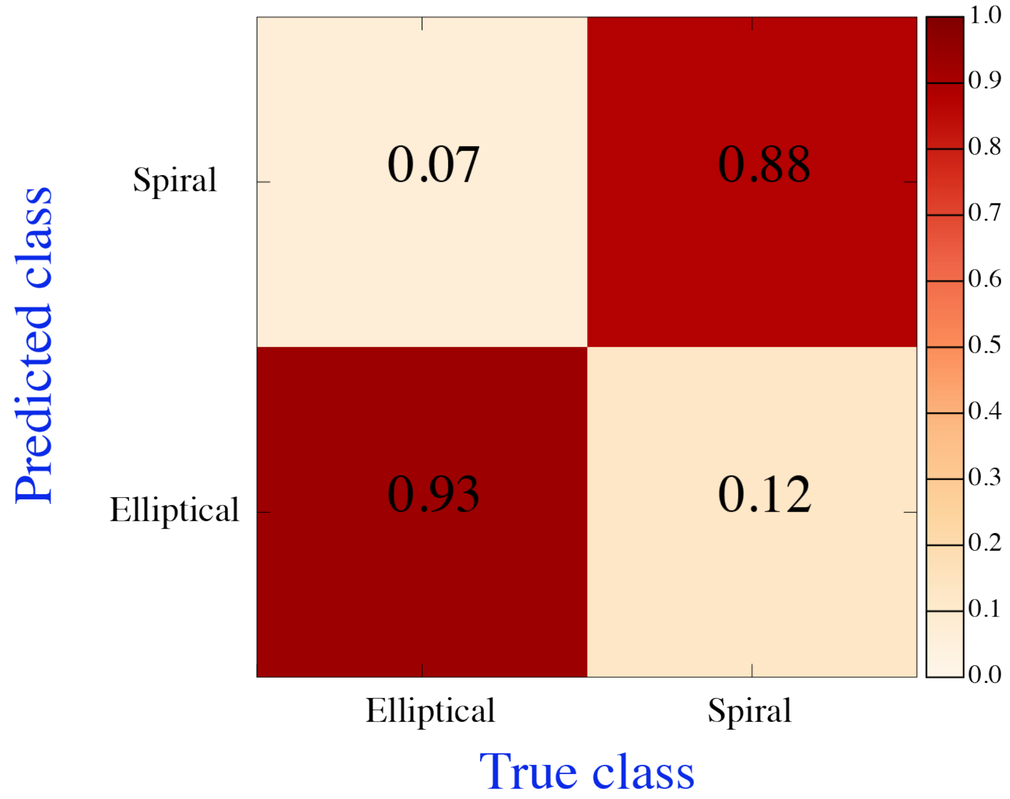
In R:

Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

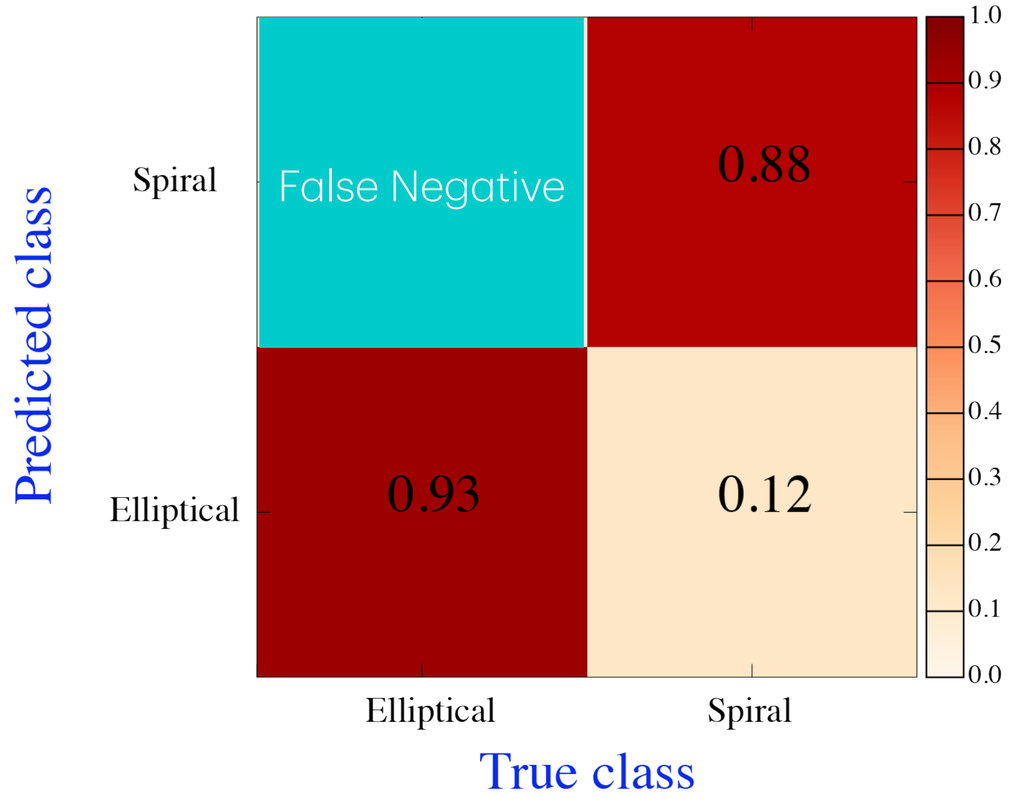


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

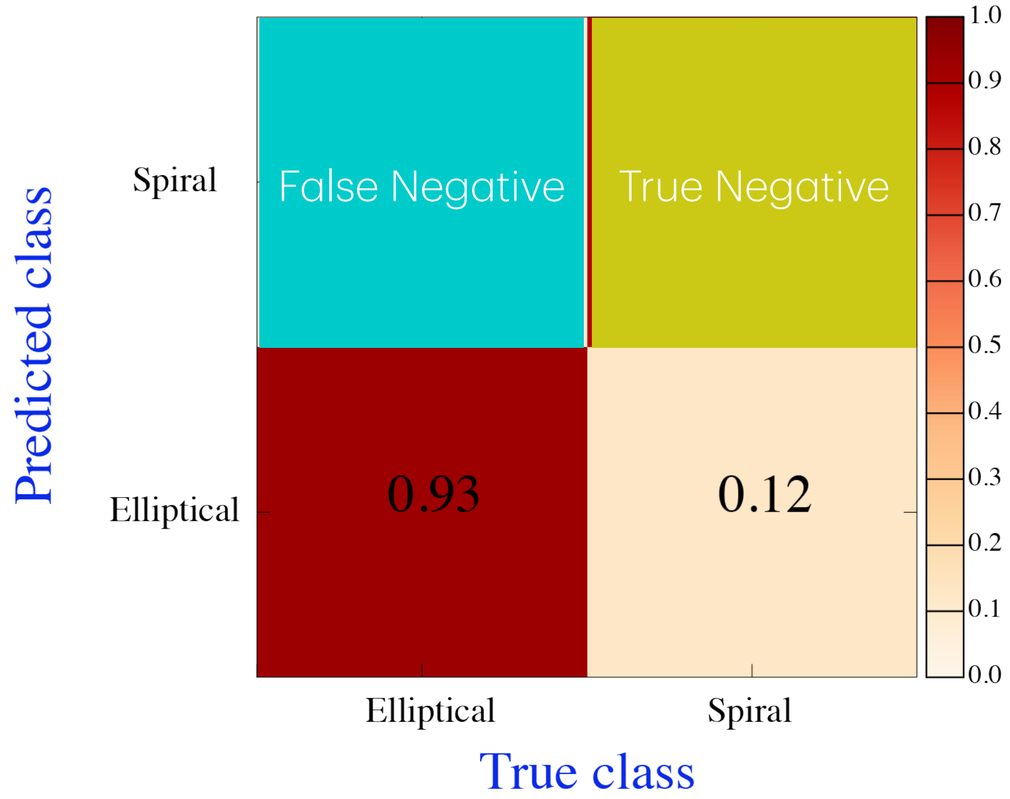


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

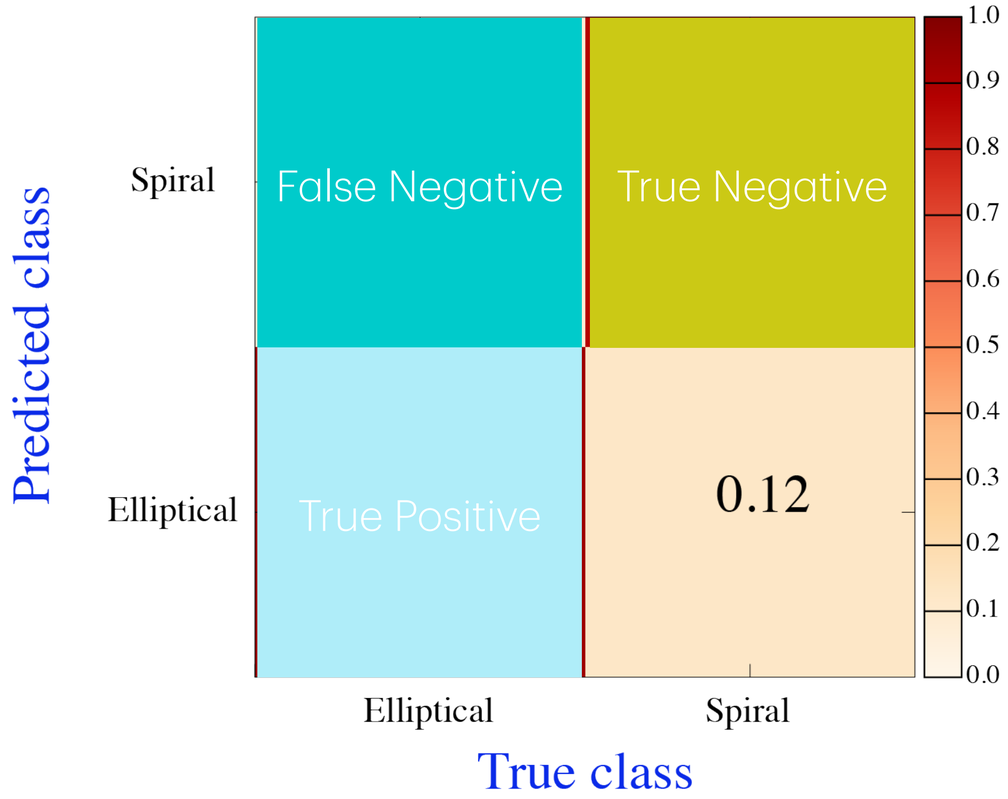


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways

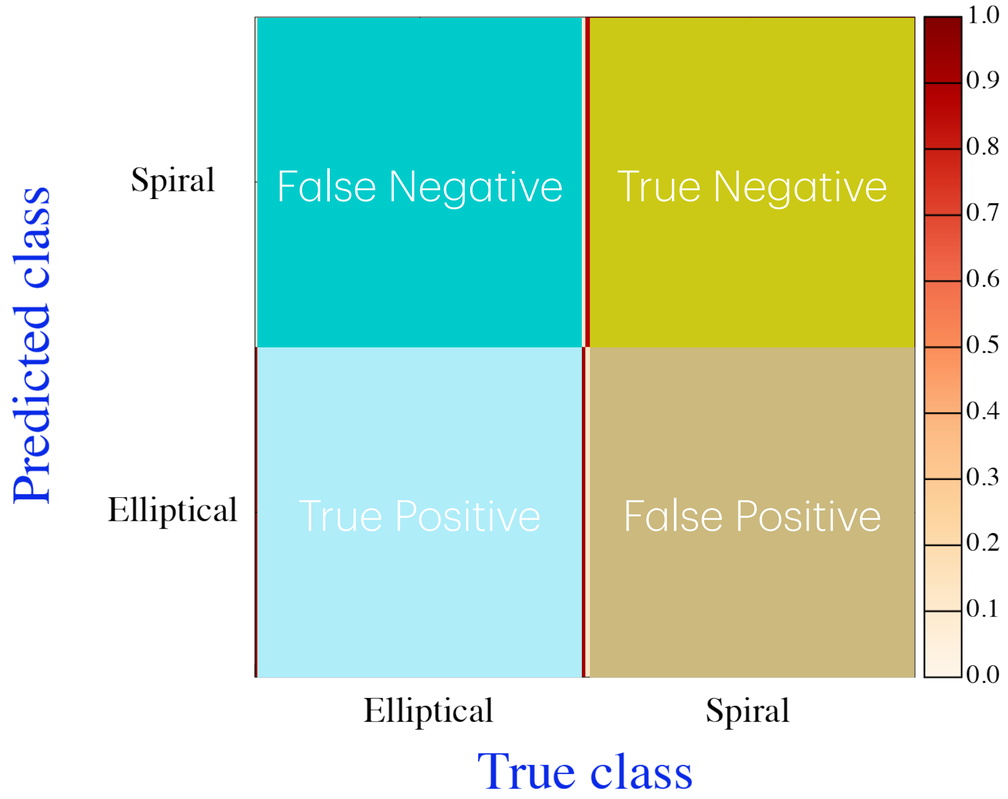


Validation - confusion matrix & misclassifications

The **confusion matrix** is a very handy way to summarise how well a classification method works.

Often you are only interested in one class - let us say elliptical. In that case it is usual to call this “Positive” and spiral “Negative”.

The resulting confusion matrix can then be summarised in many ways



Validation - confusion matrix & misclassifications

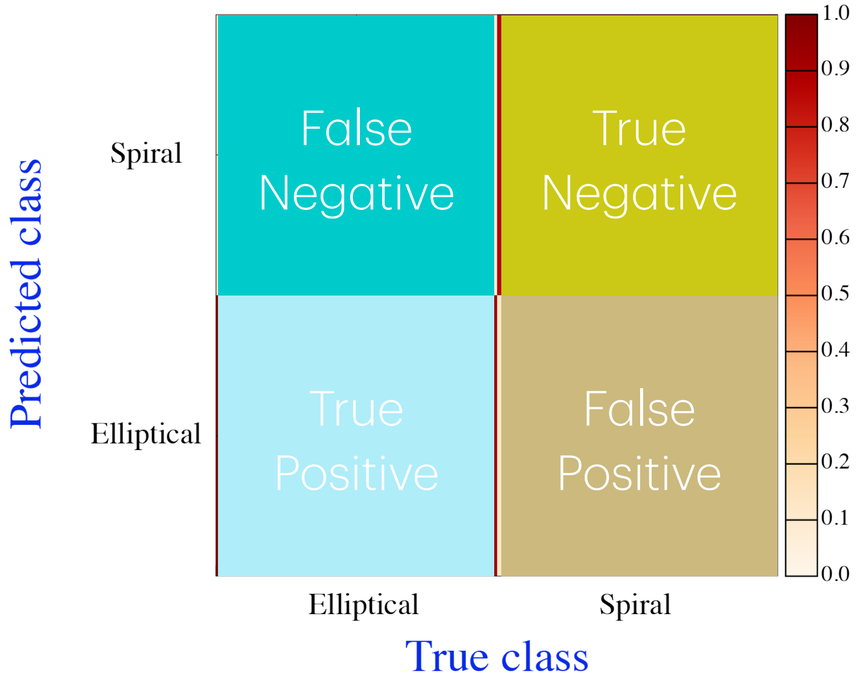
False Negative = **FN**

True Negative = **TN**

True Positive = **TP**

False Positive = **FP**

Some commonly seen terms:



Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

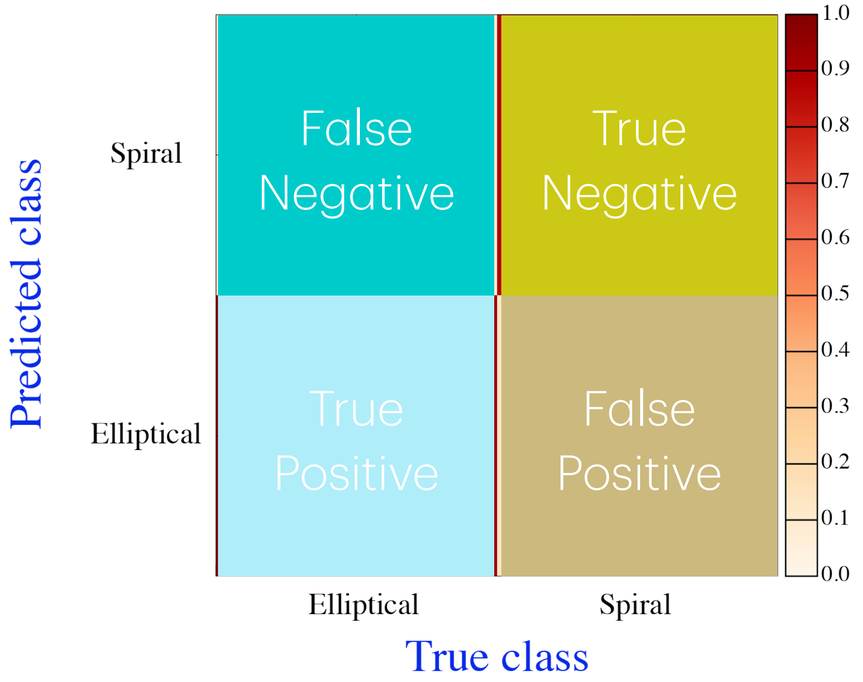
True Positive = **TP**

False Positive = **FP**

Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$



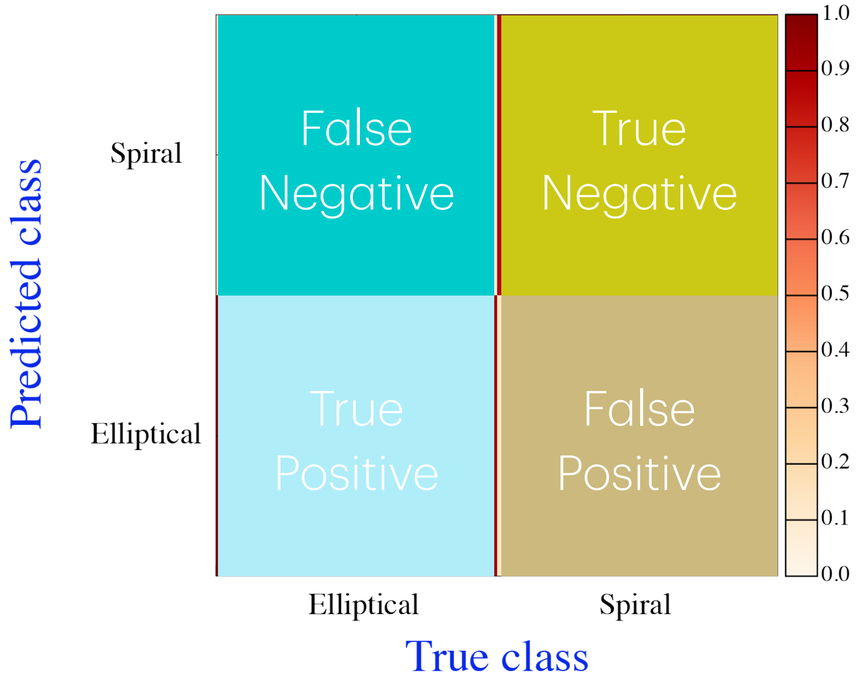
Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

True Positive = **TP**

False Positive = **FP**



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

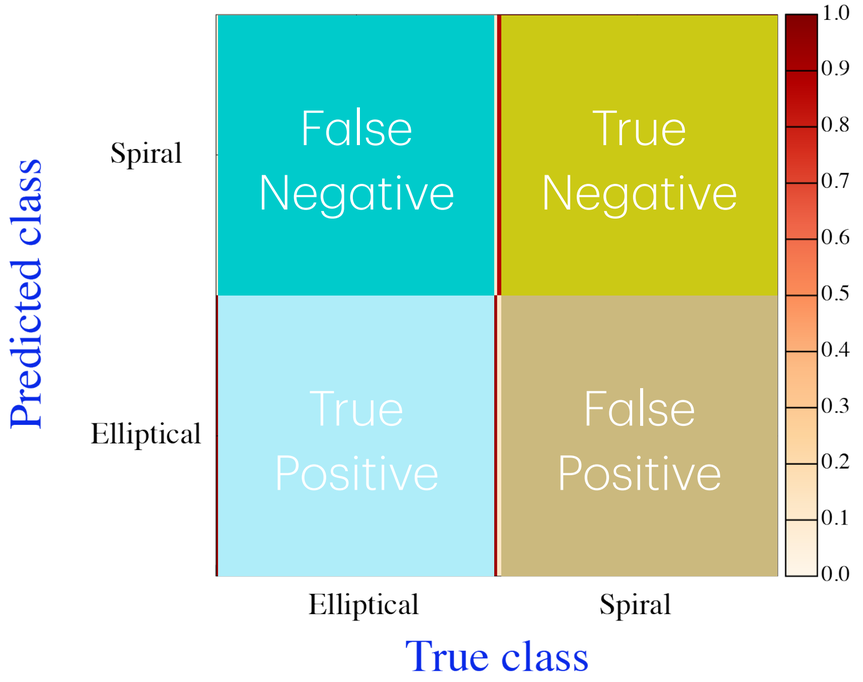
Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

True Positive = **TP**

False Positive = **FP**



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

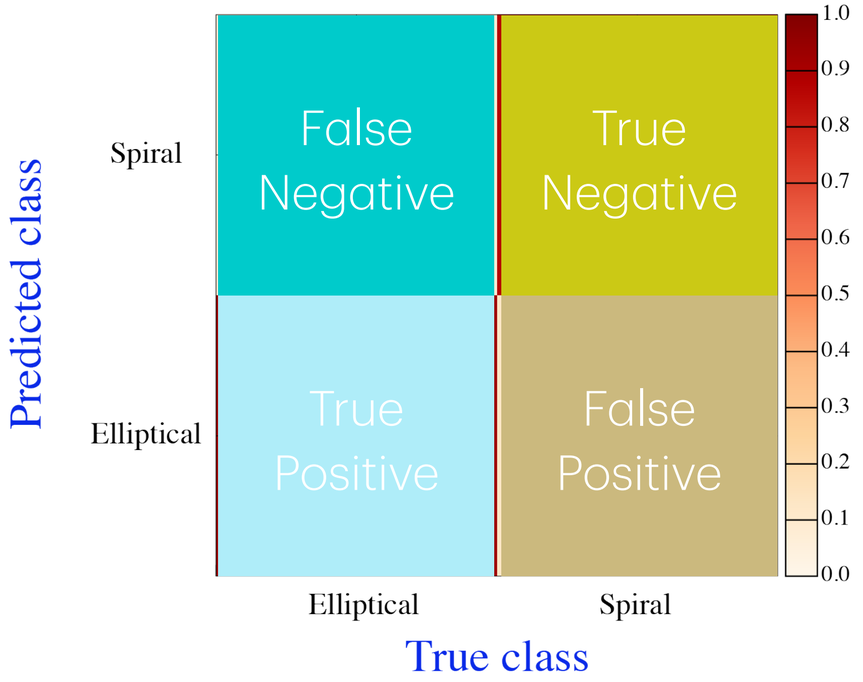
Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

True Positive = **TP**

False Positive = **FP**



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Accuracy:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$$

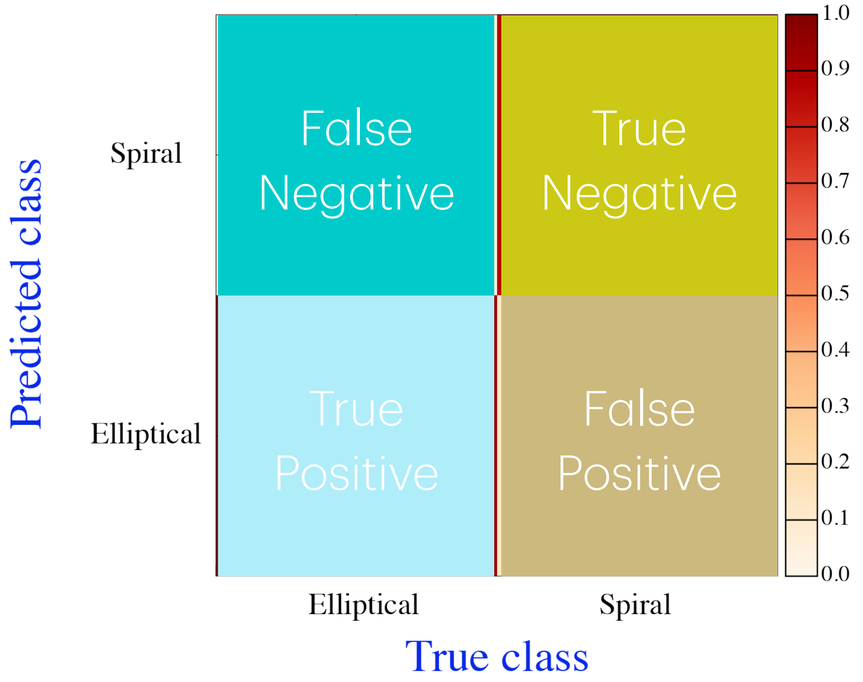
Validation - confusion matrix & misclassifications

False Negative = **FN**

True Negative = **TN**

True Positive = **TP**

False Positive = **FP**



Some commonly seen terms:

Recall, sensitivity, true positive rate:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Precision, positive predictive value:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

False negative rate:

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Accuracy:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$$

F1 score:

$$\mathrm{F}_1 = 2\frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}}$$

There are many more! See e.g. https://en.wikipedia.org/wiki/Confusion_matrix

Validation - confusion matrix & misclassifications

So what should you use?

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{TPR}\times\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2\times\frac{\mathrm{PPV}\times\mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP}\times\mathrm{TN}-\mathrm{FP}\times\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP}\times\frac{TP}{TP+FN}} = \sqrt{PPV\times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR}+\mathrm{TNR}-1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV}+\mathrm{NPV}-1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[5] Piryonesi and El-Diraby (2020),[6] Powers (2011),[4] Ting (2011),[4] CAWCR,[2]
D. Chicco & G. Jurman (2020, 2021, 2023),[4][7][8] Tharwat (2018),[4] Balayla (2020)[9][1]

Validation - confusion matrix & misclassifications

So what should you use?

It depends on your question!

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[5] Piryonesi and El-Diraby (2020),[6] Powers (2011),[4] Ting (2011),[7] CAWCR,[8]
D. Chicco & G. Jurman (2020, 2021, 2023),[9][10][3] Tharwat (2018).[2] Balayla (2020)[11]

Validation - confusion matrix & misclassifications

So what should you use?

If you are doing diagnosis of a deadly disease you are (hopefully) most interested in the false negatives.

It depends on your question!

If you want to create as pure a sample as possible, you probably want to focus on false positive rate.

If you are interested in a good overall metric, the F1 score is possibly your thing.

Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (φ or r_φ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[5] Piryonesi and El-Diraby (2020),[6] Powers (2011),[4] Ting (2011),[4] CAWCR,[2]
D. Chicco & G. Jurman (2020, 2021, 2023),[1][7][8] Tharwat (2018),[9] Balayla (2020)[10]

Validation - confusion matrix & misclassifications

So what should you use?

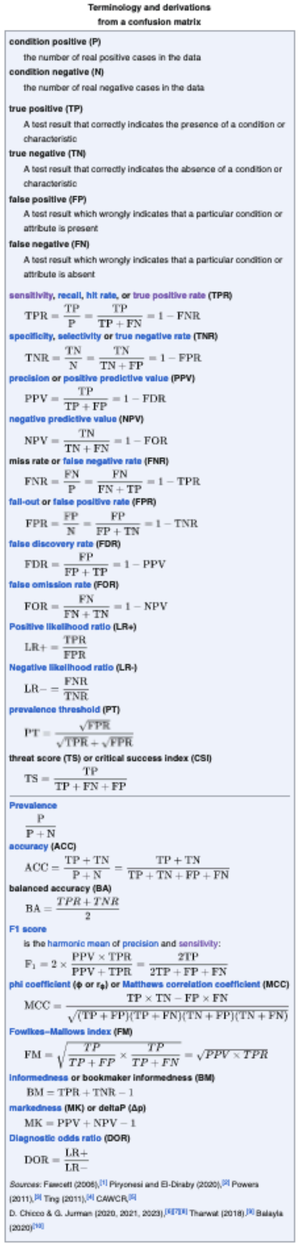
It depends on your question!

If you are doing diagnosis of a deadly disease you are (hopefully) most interested in the false negatives.

If you want to create as pure a sample as possible, you probably want to focus on false positive rate.

If you are interested in a good overall metric, the F1 score is possibly your thing.

The most important: think about what you need - the actual metric will come almost automatically.



Terminology and derivations from a confusion matrix

condition positive (P)
the number of real positive cases in the data
condition negative (N)
the number of real negative cases in the data

true positive (TP)
A test result that correctly indicates the presence of a condition or characteristic
true negative (TN)
A test result that correctly indicates the absence of a condition or characteristic
false positive (FP)
A test result which wrongly indicates that a particular condition or attribute is present
false negative (FN)
A test result which wrongly indicates that a particular condition or attribute is absent

sensitivity, recall, hit rate, or true positive rate (TPR)
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$
specificity, selectivity or true negative rate (TNR)
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$
precision or positive predictive value (PPV)
$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$
negative predictive value (NPV)
$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$
miss rate or false negative rate (FNR)
$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$
fall-out or false positive rate (FPR)
$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$
false discovery rate (FDR)
$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$
false omission rate (FOR)
$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TN}} = 1 - \mathrm{NPV}$$
Positive likelihood ratio (LR+)
$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$
Negative likelihood ratio (LR-)
$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$
prevalence threshold (PT)
$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}}+\sqrt{\mathrm{FPR}}}$$
threat score (TS) or critical success index (CSI)
$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$$

Prevalence
$$\frac{\mathrm{P}}{\mathrm{P}+\mathrm{N}}$$
accuracy (ACC)
$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{P}+\mathrm{N}} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$$
balanced accuracy (BA)
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$
F1 score
is the harmonic mean of precision and sensitivity:
$$\mathrm{F_1} = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$
phi coefficient (ϕ or r_ϕ) or Matthews correlation coefficient (MCC)
$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$
Fowlkes–Mallows index (FM)
$$\mathrm{FM} = \sqrt{\frac{TP}{TP+FP} \times \frac{TP}{TP+FN}} = \sqrt{PPV \times TPR}$$
informedness or bookmaker informedness (BM)
$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$
markedness (MK) or deltaP (Δp)
$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$
Diagnostic odds ratio (DOR)
$$\mathrm{DOR} = \frac{\mathrm{LR+}}{\mathrm{LR-}}$$

Sources: Fawcett (2006),[5] Piryonesi and El-Diraby (2020),[6] Powers (2011),[4] Ting (2011),[4] CAWCR,[2]
D. Chicco & G. Jurman (2020, 2021, 2023),[1][7][8] Tharwat (2018),[9] Balayla (2020)[10]

High dimensionality - Naïve Bayes & Bayesian networks

That appears to be a simple technique, but estimating $p(\vec{x}|\omega_i)$ is challenging in high dimensions - if you need N objects for a 1D PDF, you need N^d data points for d-dimensional PDF.

A simple way to reduce the requirements is to assume that all variables are independent, so that you have

$$p(\vec{x}|\omega_j) = \prod_i p(x_i|\omega_j)$$

Now you can just repeat what I showed for each 'feature' and multiply the results together - this is known as **Naïve Bayes**.

A bit more sophisticated is to partially factorize $p(\vec{x}|\omega_i)$. This leads to **Bayesian networks**. But the process is very similar to what we have seen.

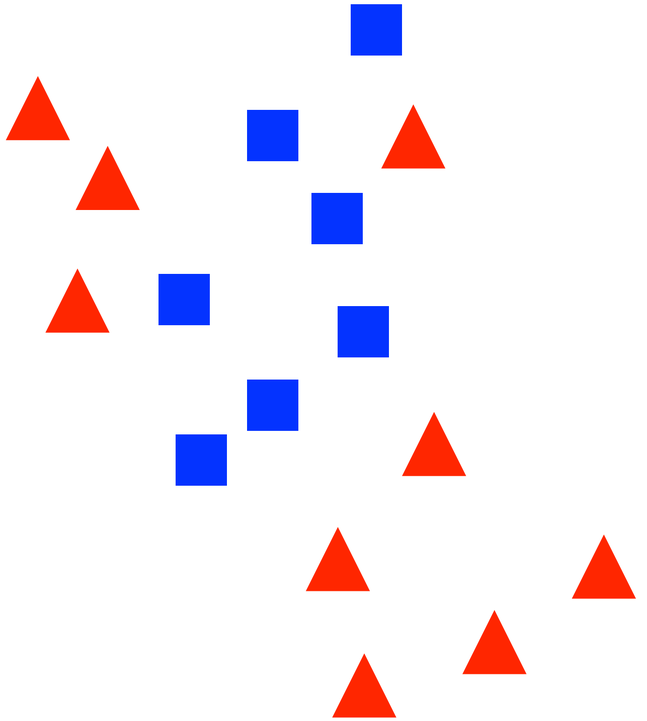
Other classification/grouping methods

In many situations a full application of Bayesian statistics is difficult to carry out, particularly in high dimension. There are therefore a range of alternative methods that are less “optimal” but can provide very flexible and useful techniques.

It is worth keeping in mind however, that the Bayesian technique not only is optimal (if you know the PDFs…), but it also assigns a probability for belonging to a particular class.

k-Nearest neighbours

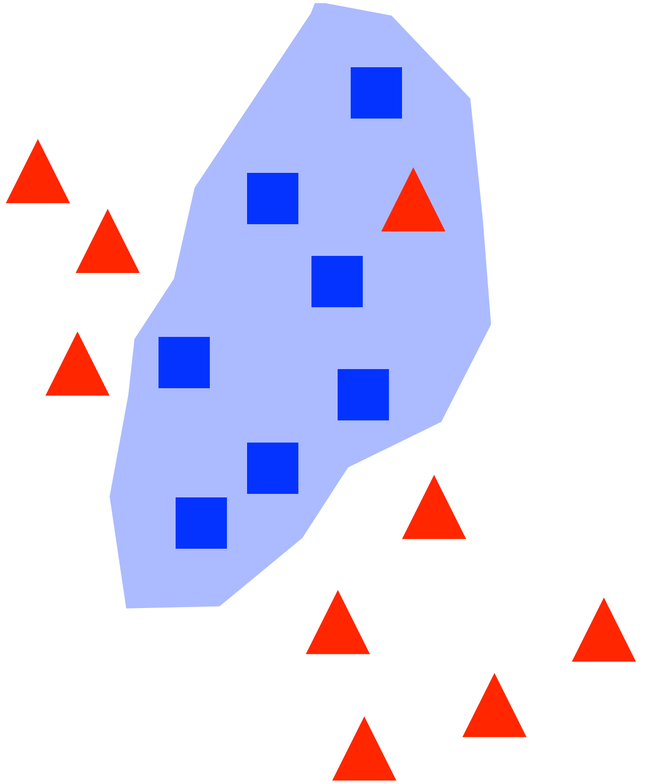
Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.



In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

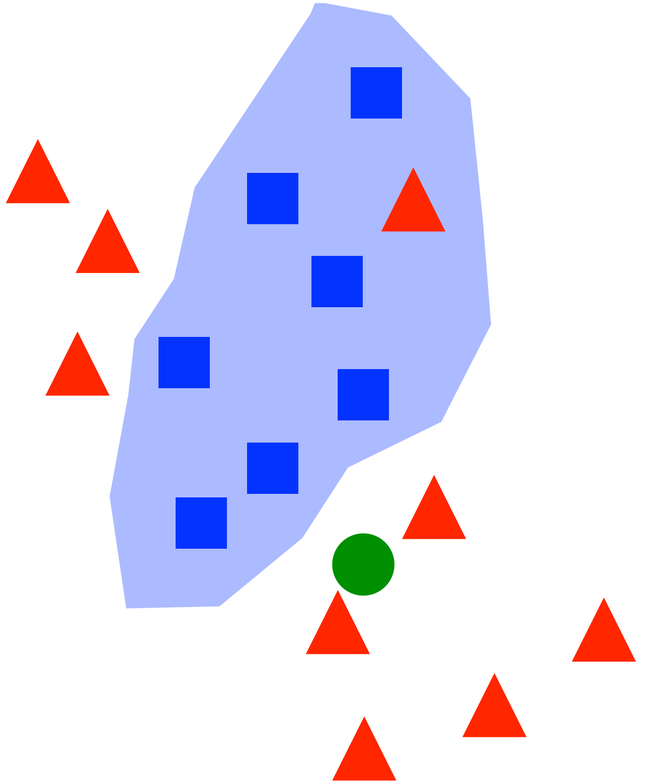


In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

Application: Find the nearest k objects and assign the majority class.

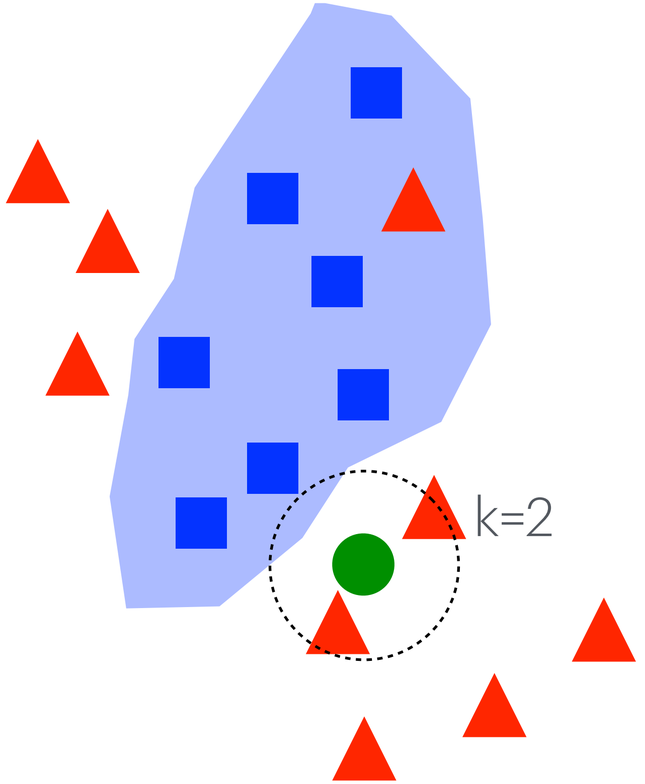


In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

Application: Find the nearest k objects and assign the majority class.

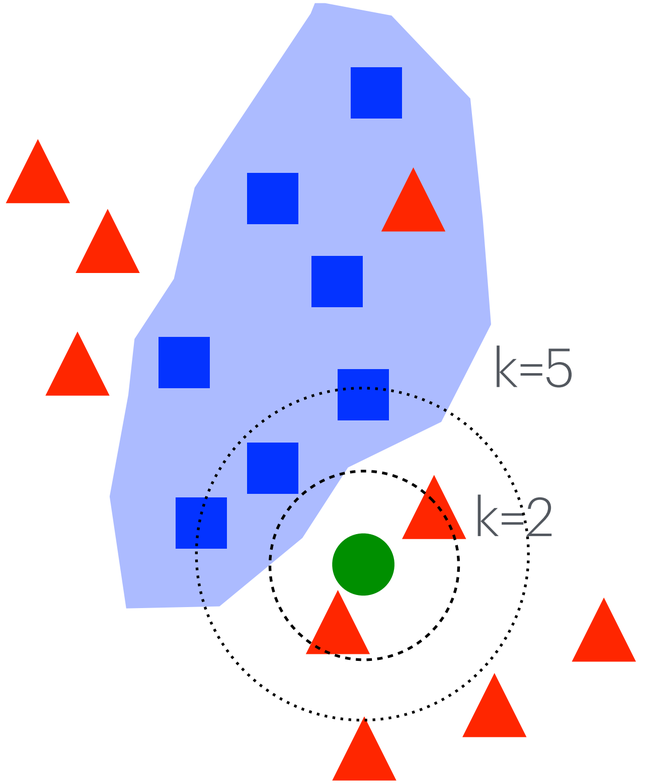


In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Training: For each object, find the nearest k objects - assign the class of the majority of the neighbours.

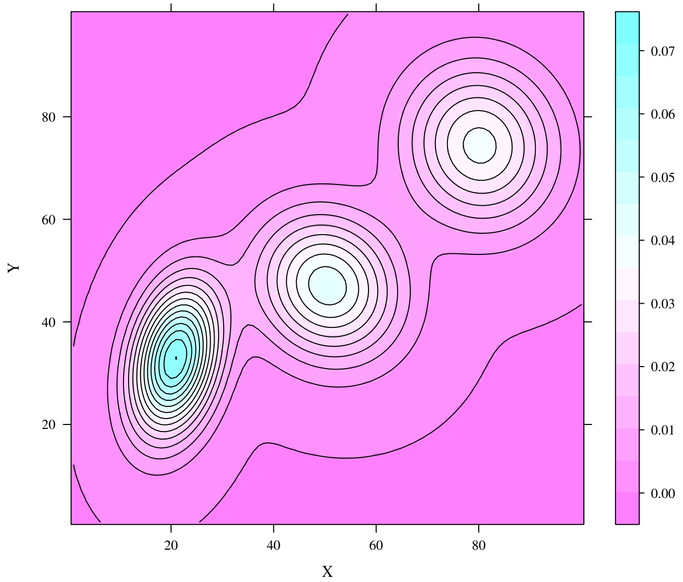
Application: Find the nearest k objects and assign the majority class.



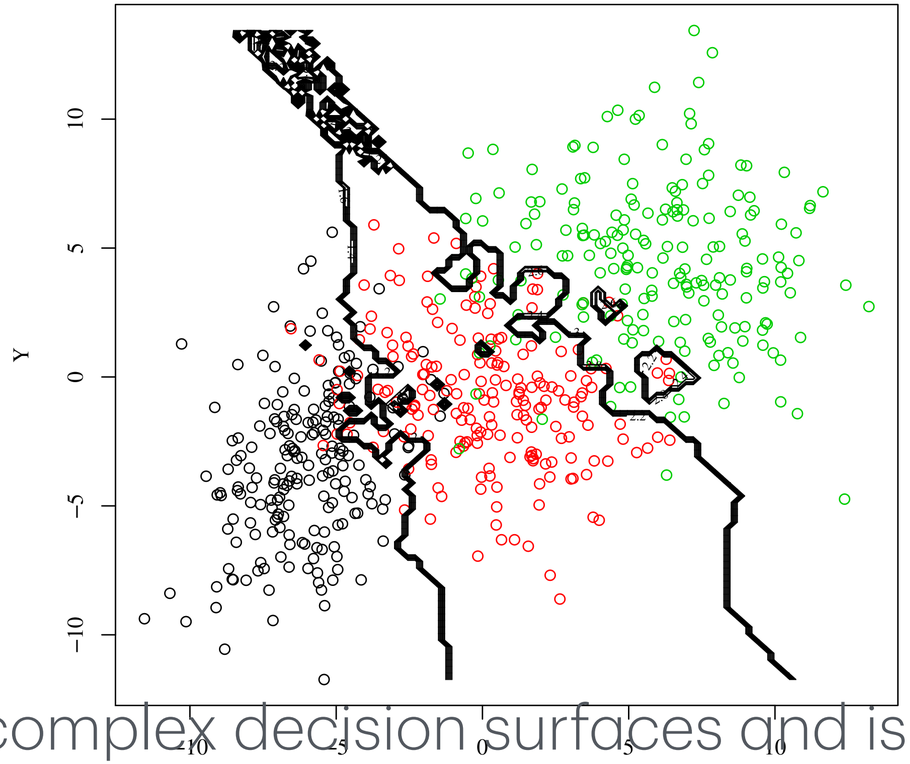
In python: `from sklearn.neighbors import KNeighborsClassifier`

k-Nearest neighbours

Underlying distributions



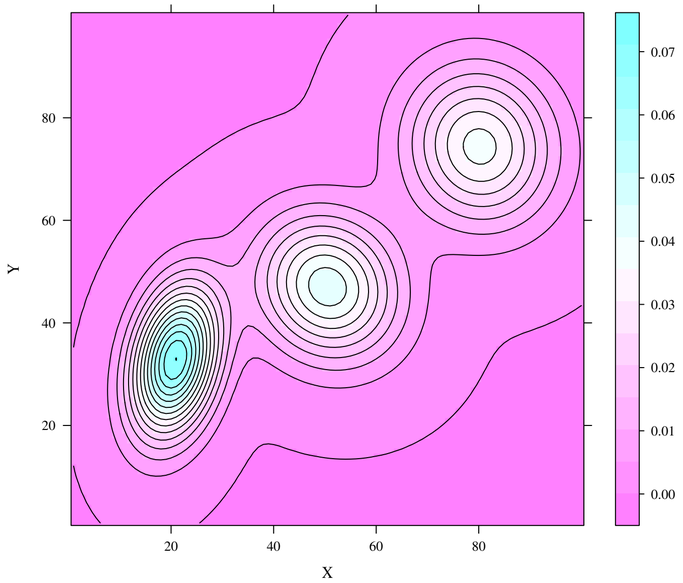
The result of applying knn



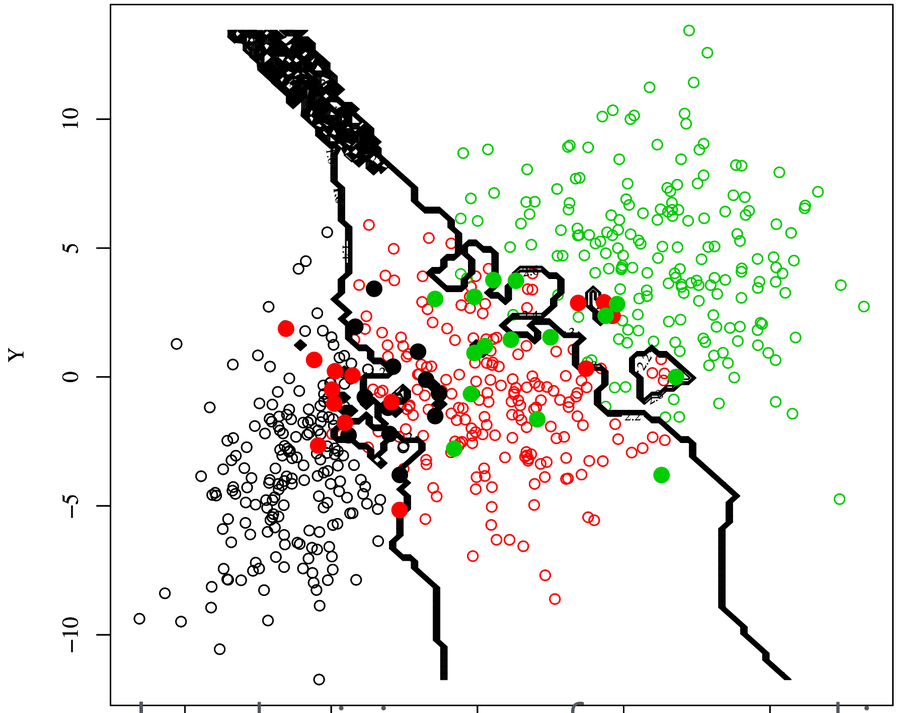
knn can lead to rather complex decision surfaces and is therefore not always optimal for **understanding** what is going on.

k-Nearest neighbours

Underlying distributions



The result of applying knn



knn can lead to rather complex decision surfaces and is therefore not always optimal for **understanding** what is going on.

Dimensional reduction

Fitting models & dimensionality

Assume you wish to fit a model to a high order polynomial:

$$y = w_0 + \sum_i w_i x_i + \sum_i \sum_j w_{ij} x_i x_j + \sum_i \sum_j \sum_k w_{ijk} x_i x_j x_k$$

The number of terms grows as d^M where d is the number of input variables and M is the polynomial order. So for higher order functions you need to constrain a large number of terms and need huge training datasets.

Fitting models & dimensionality

Assume you wish to fit a model to a high order polynomial:

$$y = w_0 + \sum_i w_i x_i + \sum_i \sum_j w_{ij} x_i x_j + \sum_i \sum_j \sum_k w_{ijk} x_i x_j x_k$$

The number of terms grows as d^M where d is the number of input variables and M is the polynomial order. So for higher order functions you need to constrain a large number of terms and need huge training datasets.

There is a theorem (Barron 1993) saying that for polynomials the error in an approximation goes as $O(1/M^{2/D})$ - D is the dimensionality. While for non-linear functions it goes as O(1/M). For large D you therefore need many more term for polynomial fits than for non-linear fits.

An aside: Higher dimensions - weirdities

You might think that the optimal way to sample spaces is in regular bins - and that the cube & the sphere cover similar volumes.

$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$

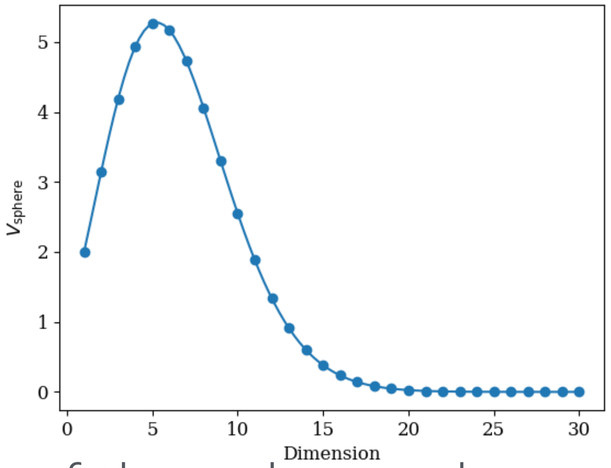
in D dimensions, and since:

$$\frac{V_{\text{sphere}}(1) - V_{\text{sphere}}(1-\epsilon)}{V_{\text{sphere}}(1)} = 1 - (1-\epsilon)^D$$

most of this volume is in a thin shell for high D. And indeed the volume of a sphere goes to zero when D goes to infinity!

Higher dimensions - weirdities

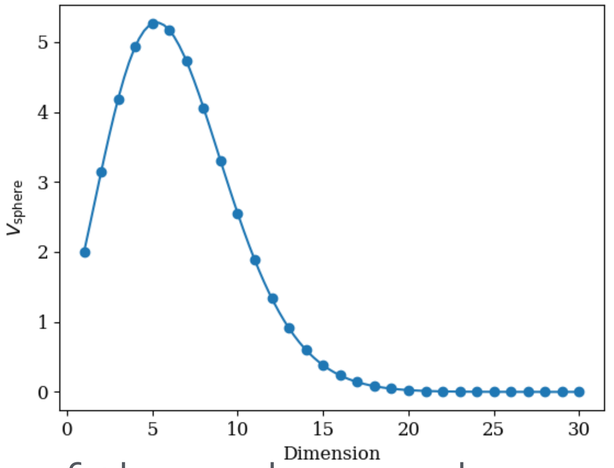
$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$



We can also compare this to the volume of the cube, and we find:

Higher dimensions - weirdities

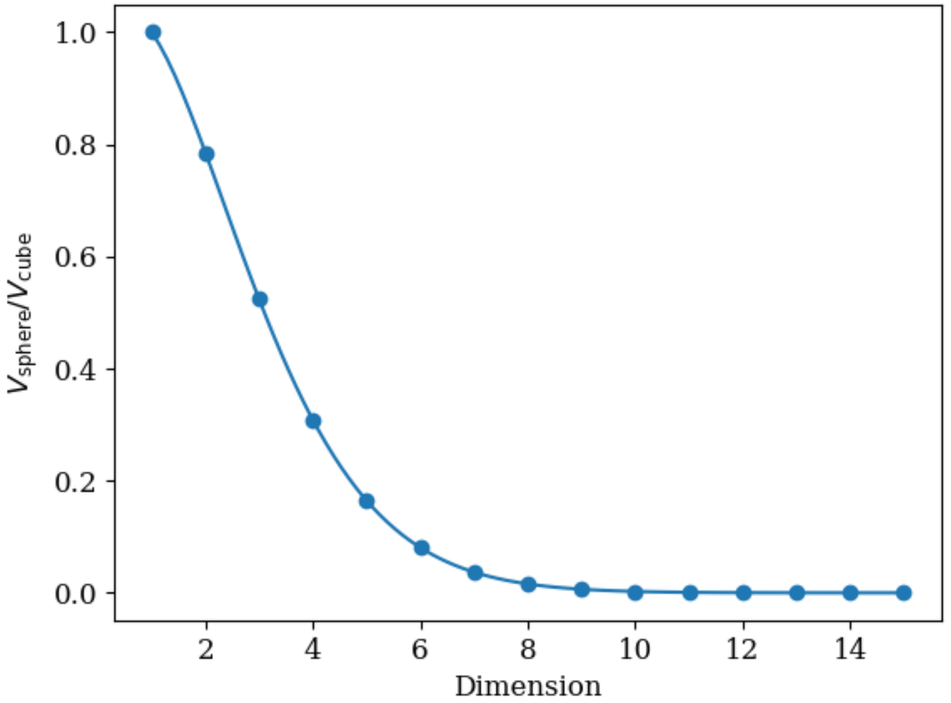
$$V_{\text{sphere}}(r) = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)} r^D$$



We can also compare this to the volume of the cube, and we find:

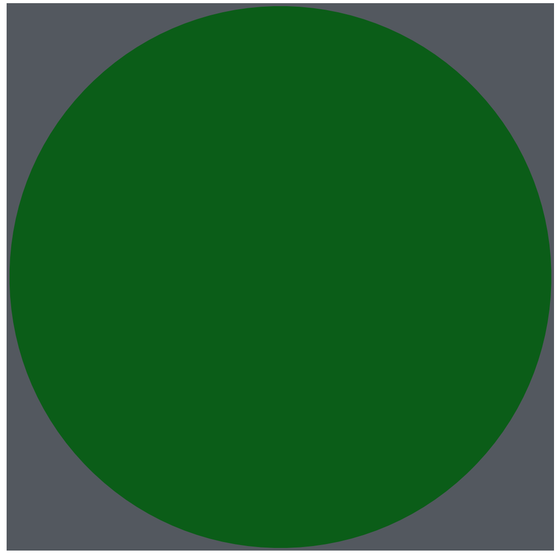
$$\frac{V_{\text{sphere}}}{V_{\text{cube}}} = \frac{\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2}+1)}$$

Very few points lie within a unit radius from the origin!



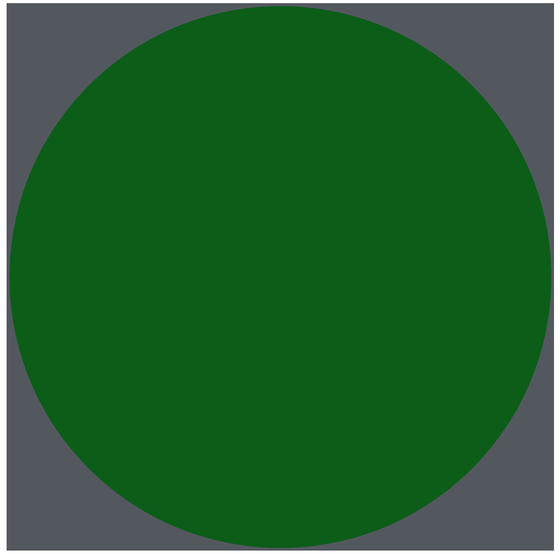
The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques



The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques



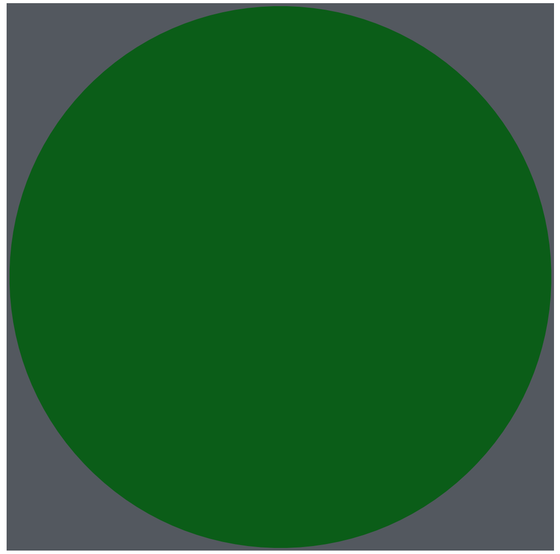
If you want to calculate

$$\int\int f(x,y)\,dxdy$$

over the green disk - you might opt for a Monte Carlo integration technique where you draw random numbers within the gray square and reject those that lie outside the green disk.

The curse of dimensions

So does this matter? It matters for instance in Monte Carlo techniques

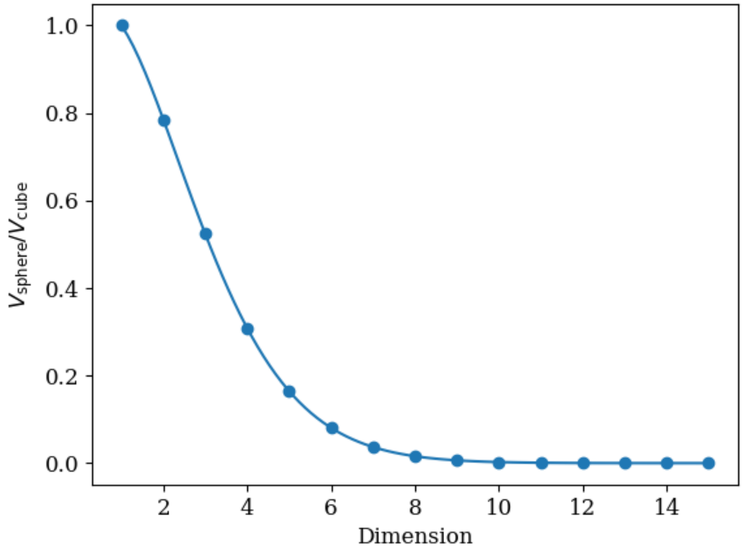


If you want to calculate

$$\int\int f(x,y)\,dxdy$$

over the green disk - you might opt for a Monte Carlo integration technique where you draw random numbers within the gray square and reject those that lie outside the green disk.

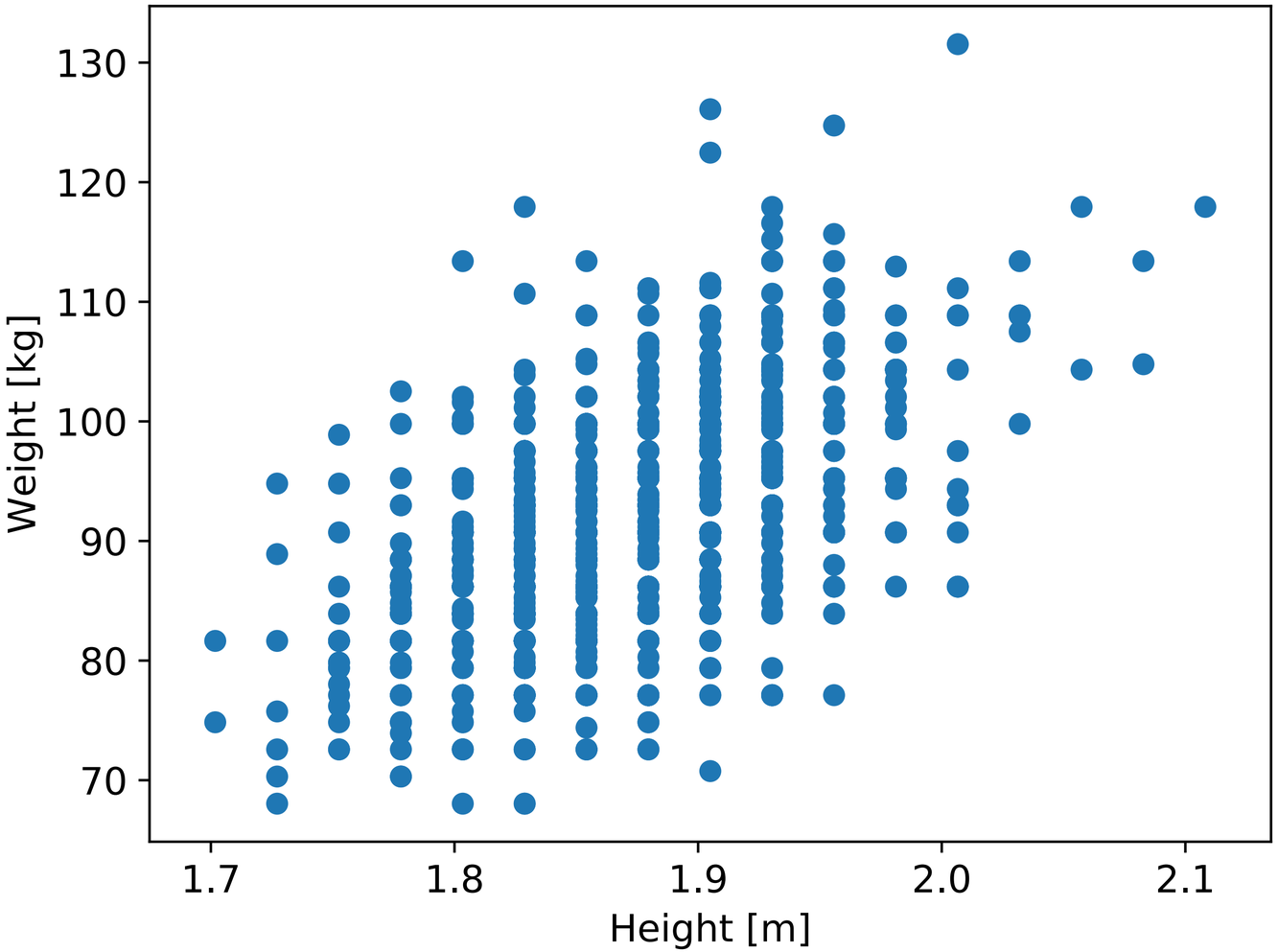
But in high-D, the chances of a point falling in the green sphere is very low!

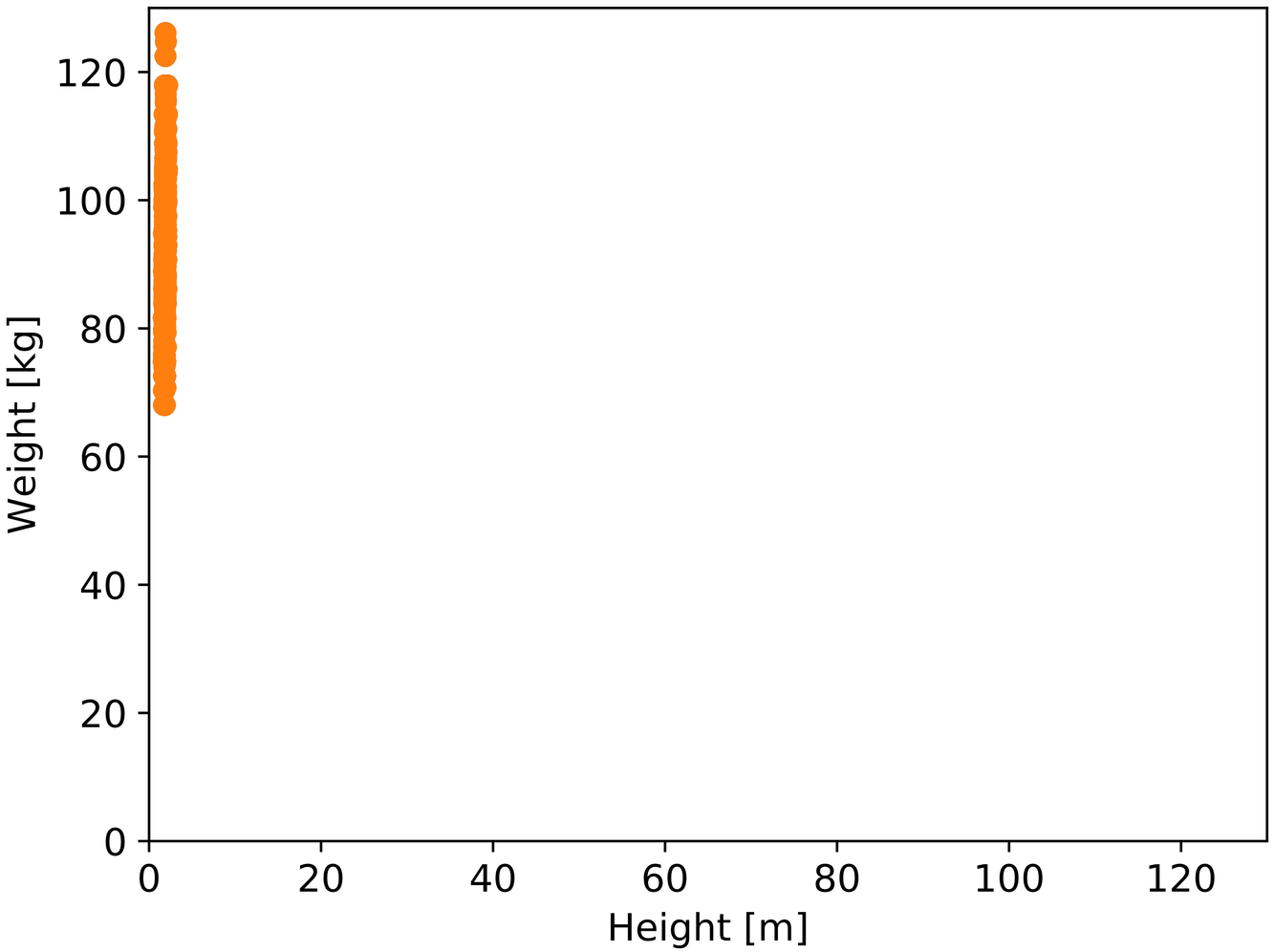


But in reality, life is better

Real physical objects do not fill “space” arbitrarily - thus we can look for ways to **reduce** the effective **dimension** of the problem.

But first - you might need to pre-process your data:


130
120
110
100
90
80
70
Weight [kg]
1.7
1.8
1.9
2.0
2.1
Height [m]


Weight [kg]
120
100
80
60
40
20
0
0
20
40
60
80
100
120
Height [m]

Rescaling data

In many algorithms it is highly desirable that the ranges probed in each dimension are comparable. To achieve that we **scale** our data.

Scale range to -1 to 1, or close to it:

$$x_i \to \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

min-max scaling

$$x_i \to \frac{x_i - \bar{x}}{\sigma_x}$$

standardizing

Often called whitening

Rescaling data

In many algorithms it is highly desirable that the ranges probed in each dimension are comparable. To achieve that we **scale** our data.

Scale range to -1 to 1, or close to it:

$$x_i \to \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

min-max scaling

$$x_i \to \frac{x_i - \bar{x}}{\sigma_x}$$

standardizing

Often called whitening

Approaches to standardizing

$$x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$$

1. Use sklearn convenience function:

```
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)
```

2. Calculate μ & σ yourself & scale the data manually

Approaches to standardizing

$$x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$$

1. Use sklearn convenience function:

```
from sklearn.preprocessing import StandardScaler

X_std = StandardScaler().fit_transform(X)
```

2. Calculate μ & σ yourself & scale the data manually

1. has the advantage that it can be chained into **pipelines** in sklearn and keeps track of things for you. But if your data are badly affected by outliers it will not detect that for you.

2. has the advantage that you know what you are doing, and if you want to use a robust estimator for the mean or variance you can. The downside is that you need to keep track of the values and remember to apply them!

Standardized data