Lecture 2 - Visualisations, density estimates and model choice

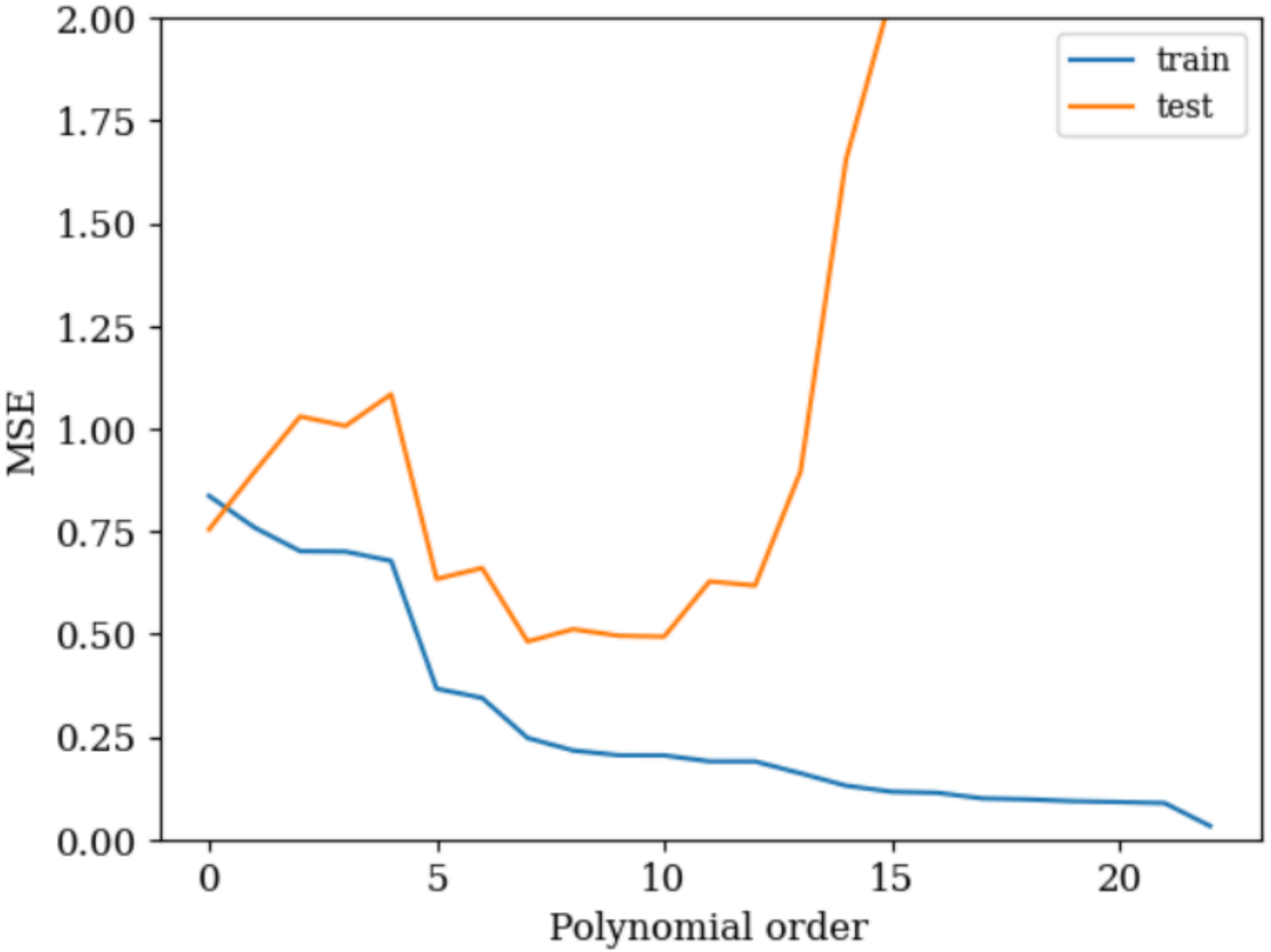
See https://github.com/jbrinchmann/MLD2024 as usual

Lectures/Lecture 2 has the PDF for today and

Lectures/Lecture 2/Notebooks for Jupyter notebooks used for this lecture

When is good good enough?

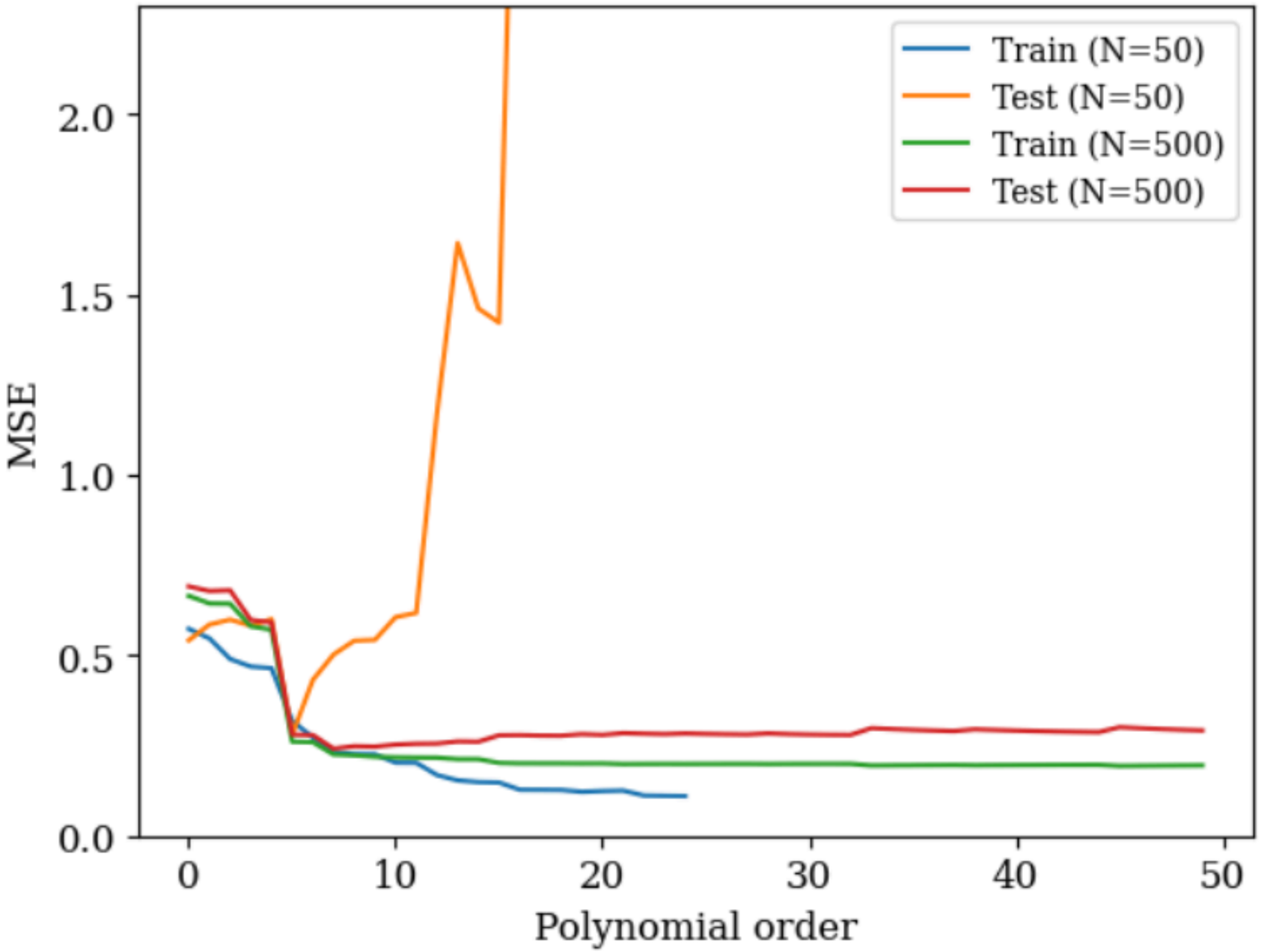
Polynomial fitting



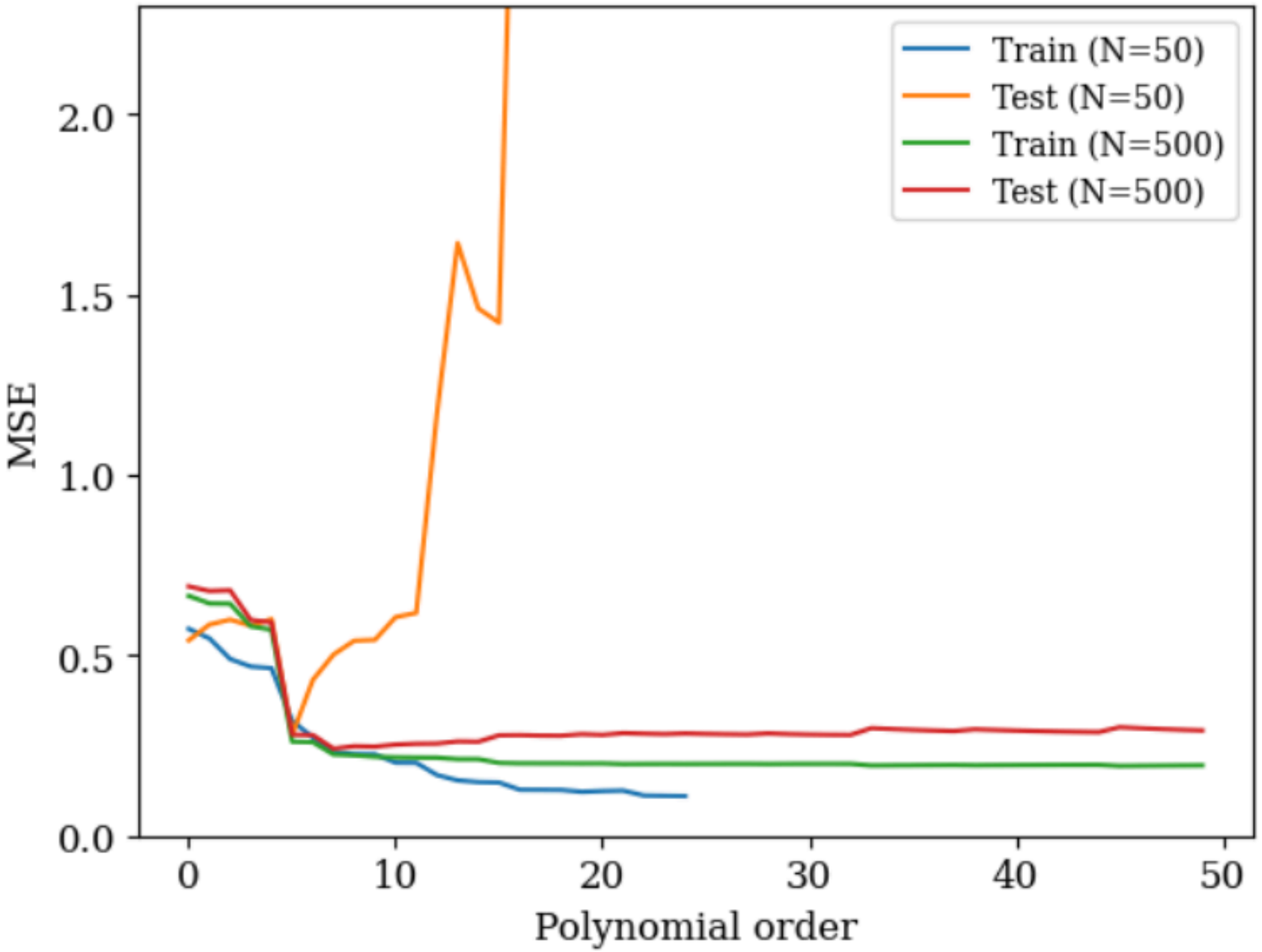
Points to note:

•

Polynomial fitting - many samples

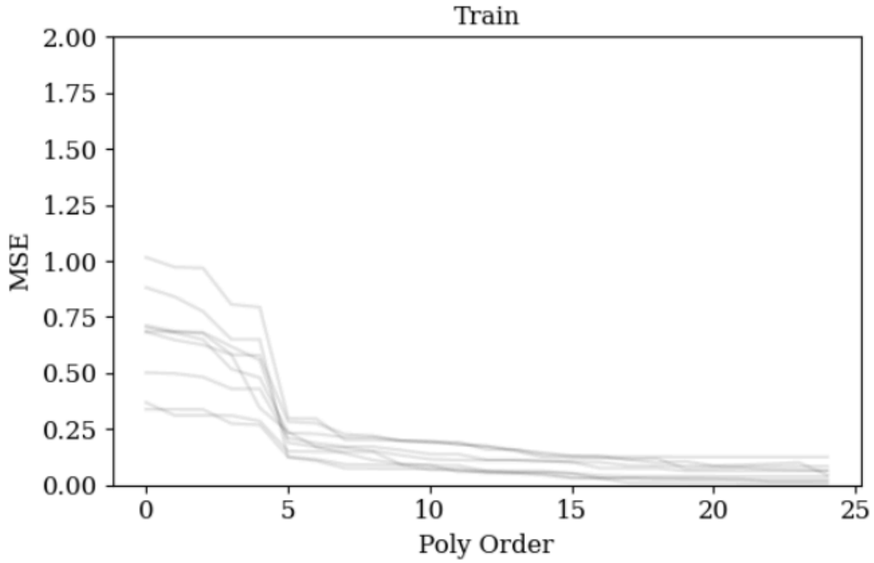


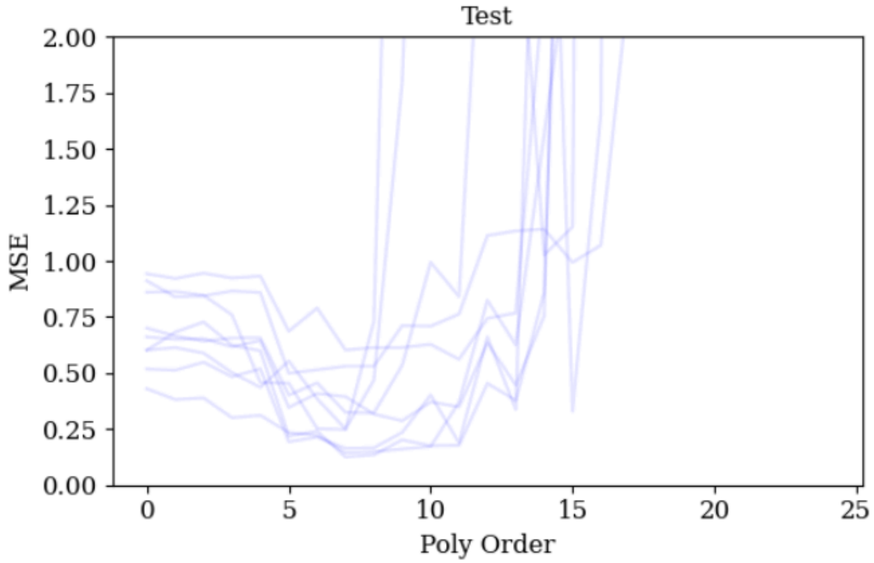
Polynomial fitting - many samples



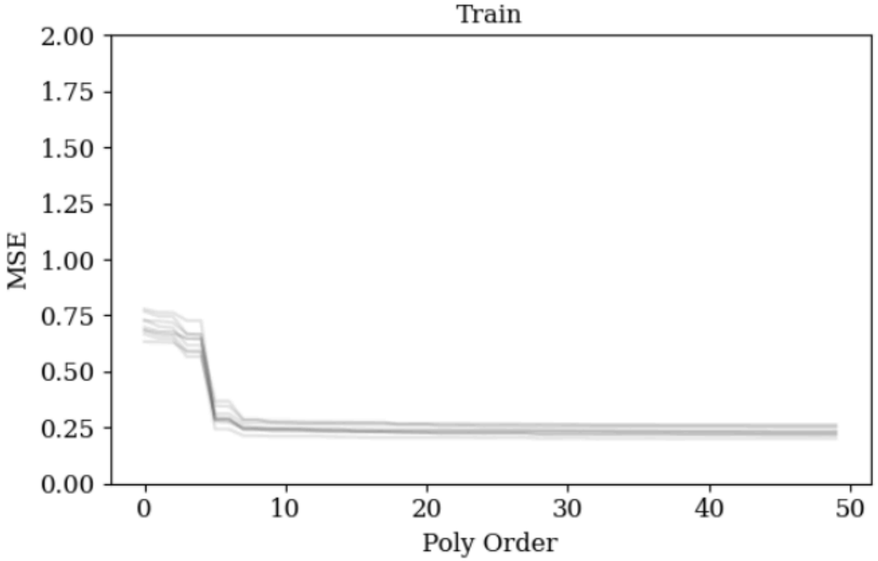
Polynomial fitting - scatter between runs

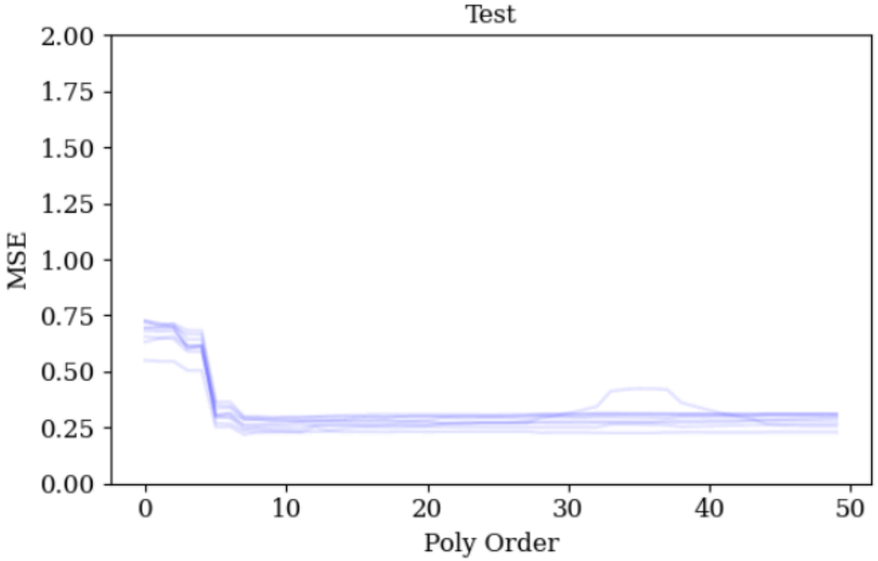
N=50





N=500





Why does it matter - the bias-variance trade-off

Complexity versus simplicity - the bias variance trade-off

Assume that the true relation between x & y is

$$y = f(x) + \epsilon \qquad \epsilon \sim N(0, \sigma_\epsilon)$$

And write the predicted value of y at x_0 $\hat{y}$

Let us ask what the expected mean square error in our prediction at $x{=}x_0$ is:

$$\mathrm{Err}(x_0) = E\left[\left(f(x_0) - \hat{y}\right)^2\right]$$

Small versus big bins - the bias variance trade-off

Expected mean square error at $x=x_0$:

$$\mathrm{E}\left[(f(x_0)-\hat{y})^2\right]$$

$$y = f(x) + \epsilon$$
$$\epsilon \sim N(0, \sigma_\epsilon)$$

We can expand this and add and subtract $(\mathrm{E}[\hat{y}])^2$

to get:

$$\mathrm{Err}(x_0) = \sigma_\epsilon^2 + \left(E\left[\hat{y}\right] - f(x_0)\right)^2 + E\left[\left(\hat{y} - E[\hat{y}]\right)^2\right]$$

Small versus big bins - the bias variance trade-off

Expected mean square error at $x=x_0$:

$$\mathrm{E}\left[(f(x_0)-\hat{y})^2\right]$$

$$y = f(x) + \epsilon$$
$$\epsilon \sim N(0, \sigma_\epsilon)$$

We can expand this and add and subtract $(\mathrm{E}[\hat{y}])^2$

to get:

$$\mathrm{Err}(x_0) = \sigma_\epsilon^2 + \left(E\left[\hat{y}\right] - f(x_0)\right)^2 + E\left[\left(\hat{y} - E[\hat{y}]\right)^2\right]$$

Bias^2

Small versus big bins - the bias variance trade-off

Expected mean square error at $x=x_0$:

$$\mathrm{E}\left[(f(x_0)-\hat{y})^2\right]$$

$$y = f(x) + \epsilon$$
$$\epsilon \sim N(0, \sigma_\epsilon)$$

We can expand this and add and subtract $(\mathrm{E}[\hat{y}])^2$

to get:

$$\mathrm{Err}(x_0) = \sigma_\epsilon^2 + \left(E[\hat{y}] - f(x_0)\right)^2 + E\left[\left(\hat{y} - E[\hat{y}]\right)^2\right]$$

Bias^2 Variance

Small versus big bins - the bias variance trade-off

Expected mean square error at $x=x_0$:

$$\mathrm{E}\left[(f(x_0)-\hat{y})^2\right]$$

$$y = f(x) + \epsilon$$
$$\epsilon \sim N(0, \sigma_\epsilon)$$

We can expand this and add and subtract $(\mathrm{E}[\hat{y}])^2$

to get:

$$\mathrm{Err}(x_0) = \sigma_\epsilon^2 + \left(E[\hat{y}] - f(x_0)\right)^2 + E\left[\left(\hat{y} - E[\hat{y}]\right)^2\right]$$

Irreducible Error

Bias^2

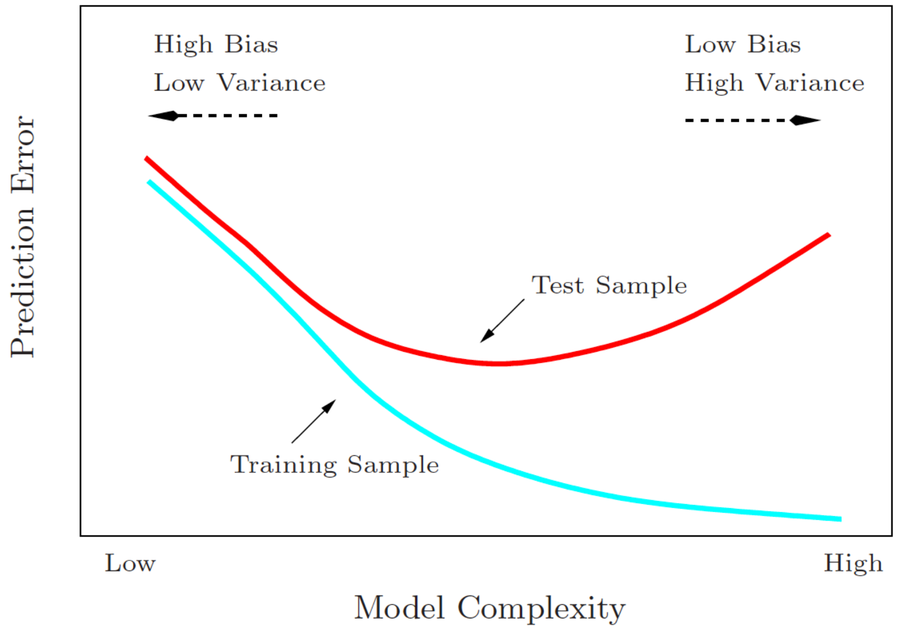
Variance

The bias variance trade-off

$$\mathrm{Err}(x_0) = (E\,[\hat{y}] - f(x_0))^2 + \mathrm{E}\left[(\hat{y} - \mathrm{E}[\hat{y}]^2\right]$$

Bias^2 Variance

This decomposition into two positive terms is a general result and usually we have to choose our method to have low bias *or* low variance but not both.



From "The Elements of Statistical Learning - Data Mining, Inference, and Prediction", Hastie, Tibshirani & Friedmann, second edition, Figure 2.11

Databases & archives - some examples

Sloan Digital Sky Survey - the reference database in astronomy

Millennium, EAGLE, Illustris, IlustrisTNG, FIRE - cosmological simulations

Kepler, TESS, GAIA - stellar treasure troves

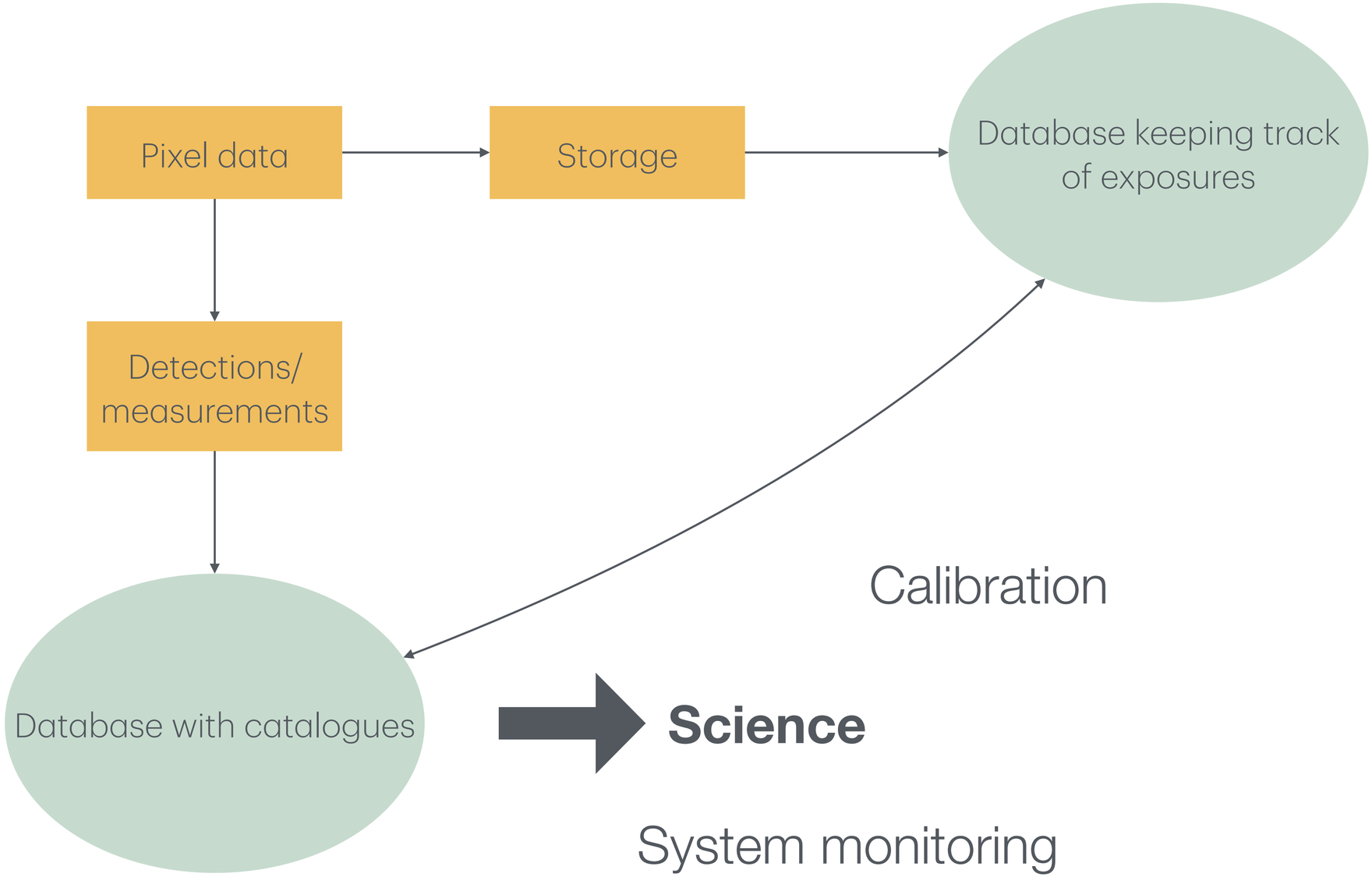
GALEX, 2MASS, etc etc.

ESO archive - observational data from ESO telescopes

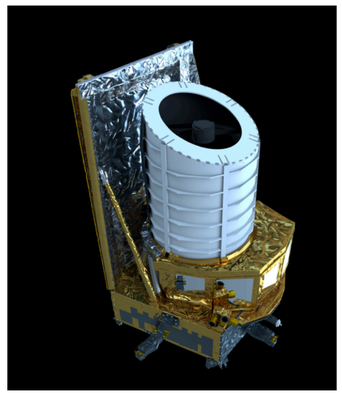
MAST archive - observational data from space telescopes

Euclid survey - the next reference database in astronomy? - the link is to the early release observations

The role of databases - a modern view



Euclid as an example



DPS

Data releases

SAS

General users

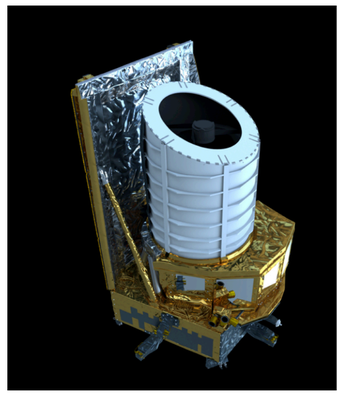
Calibration
Quality control

SDC

Data reduction
Analysis

Euclid consortium users

Euclid as an example



DPS

Data releases

Total amount:
300 Tb Raw data
~100 Pb total

SAS

General users

Calibration
Quality control

SDC

Data reduction
Analysis

Euclid consortium users

SQL - Structured Query Language

The standard way to interact with databases.

Used for instance for Gaia, Euclid, SDSS, Galex,

(although astronomical databases typically use a dialect called ADQL)

What we are doing next - and where it fits in

- The aim is always the data - today we will look a bit at how we can use a database to handle data.
- We will see how we can
  - Find data in a table
  - Combine tables
  - Possible exercise: Create tables in a data-base
- There will be some technical detail - this needs to be learned, but you should always keep in mind what your goal is: To do science with the data.

An example problem: keeping track of observations

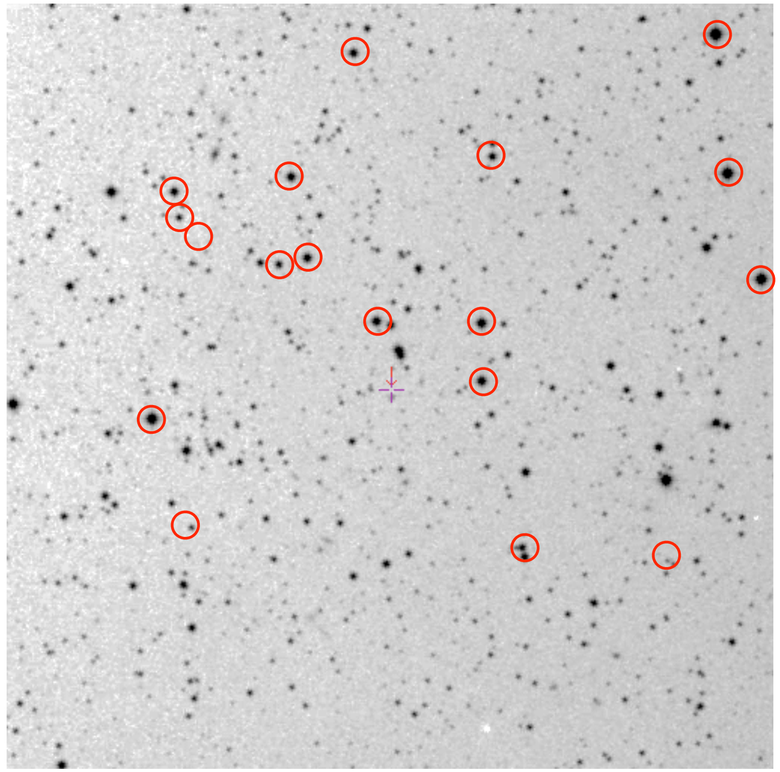
Observations:

```
         # Field      Date          Exptime   Quality   WhereStored
StF-043  92.9885764     23.2         1         /disks/yaeps-1/StF-043.fits
StF-044  97.3323764     30.2         1         /disks/yaeps-1/StF-044.fits
StF-045  93.5532134     29.5         0.5       /disks/yaeps-1/StF-045.fits
```

An example problem: keeping track of observations

Observations:

#	Field	Date	Exptime	Quality	WhereStored
StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits



Within each field we will detect a number of stars.

An example problem: keeping track of observations

Observations:

```
# Field      Date          Exptime   Quality   WhereStored
StF-043  92.9885764     23.2       1         /disks/yaeps-1/StF-043.fits
StF-044  97.3323764     30.2       1         /disks/yaeps-1/StF-044.fits
StF-045  93.5532134     29.5       0.5       /disks/yaeps-1/StF-045.fits
```

Stars:

```
# Star     Ra         Dec         g      r
 S1  198.8475000 10.5034722 14.5   15.2
 S2  198.5654167 11.0231944 15.3   15.4
 S5  198.9370833  9.9168889 16.4   15.8
 S7  199.2516667 10.3486944 14.6   14.1
```

If, for each star I keep information about the observations, I can waste a LOT of space. => Relational databases.

Relational databases:

Table A - lastnames

1 Smith

2 Kovacs

3 Tanahaka

.....

Table B - course grades

1 2.3 5.4 6.2 7.8

3 8.5 7.4 9.2 8.8

7 2.2 7.0 8.2 7.5

9 8.0 7.0 8.0 7.5

Relational databases:

Table A - lastnames

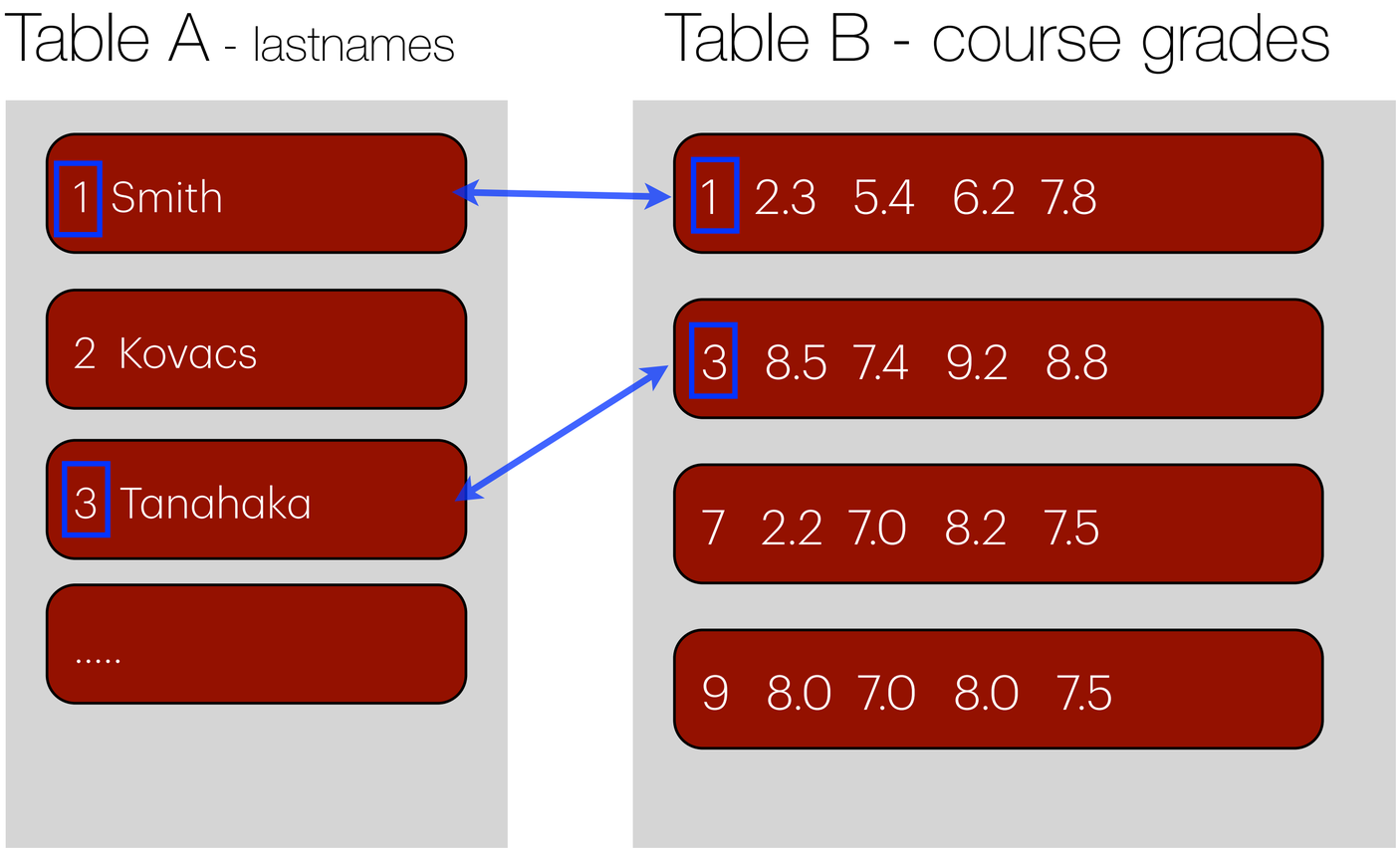
1 Smith
2 Kovacs
3 Tanahaka
.....

Table B - course grades

1	2.3	5.4	6.2	7.8
3	8.5	7.4	9.2	8.8
7	2.2	7.0	8.2	7.5
9	8.0	7.0	8.0	7.5

While a relational database is formally defined with no reference to tables, it is useful to think of it as a collection of tables where (some) rows can be related.

Relational databases:



While a relational database is formally defined with no reference to tables, it is useful to think of it as a collection of tables where (some) rows can be related.

Relational databases - the observing example

Observations:

```
# Field      Date          Exptime   Quality   WhereStored
StF-043  92.9885764     23.2       1         /disks/yaeps-1/StF-043.fits
StF-044  97.3323764     30.2       1         /disks/yaeps-1/StF-044.fits
StF-045  93.5532134     29.5       0.5       /disks/yaeps-1/StF-045.fits
```

Stars:

```
# Star     Ra         Dec        g      r
 S1  198.8475000 10.5034722 14.5   15.2
 S2  198.5654167 11.0231944 15.3   15.4
 S5  198.9370833  9.9168889 16.4   15.8
 S7  199.2516667 10.3486944 14.6   14.1
```

How should we link these?

One possibility: Create an ID column

Relational databases - the observing example

Table: Observations

#	ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits

Table: Stars

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Relational databases - the observing example

Table: Observations

#	ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits

Table: Stars

#	FieldID	StarID	Star	Ra	Dec	g	r
	1	1	S1	198.8475000	10.5034722	14.5	15.2
	1	2	S2	198.5654167	11.0231944	15.3	15.4
	3	3	S5	198.9370833	9.9168889	16.4	15.8
	2	4	S7	199.2516667	10.3486944	14.6	14.1

Note that there are two IDs in the Stars table:

The ID of each star (StarID) **Primary key**

The ID of the field it was observed id (FieldID) **Foreign key**

Relational databases - the observing example

# ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

Table Observations

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

Table Stars

We would like to ask questions like:

1. Give me all stars brighter than r=14.5
2. How many stars have 0.1 < g-r < 0.4?
3. When did we observe S2?
4. Where is the FITS image for star S5 stored?
5. Give me a list of all stars observed on the same FieldID

Choice of database solution

For concreteness I will use **sqlite** as my database as well as CasJobs

Lightweight & convenient

Advantages of sqlite: Light-weight, no need for complex setup, supports most of SQL. Easy to use for local work. Very widely used (e.g. Firefox, Chrome) and bindings for many languages.

Disadvantages: Not a client-server solution. Not all of SQL is supported and some features (e.g. ALTER TABLE) are only partially available.

Alternatives: MySQL, Oracle, PostgreSQL, Microsoft SQL Server.

Outline of creation of databases

- Determine the format for each table in your database - its *schema*. Insert this to create your table.

```
CREATE TABLE IF NOT EXISTS Stars (<schema>);
```

- Import data into each table.

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

The devil is in the details!

1. Go to the Github site for the course

2. Go to the ProblemSets/MakeTable directory and download the MLD2025.db file

3. Open this file using:
   sqlite3 MLD2025.db

4. Check which tables are available using
   sqlite> .tables
   Observations Stars

5. Follow what I do!

Querying databases - SQL

1. Give me all stars brighter than r=14.5

# FieldID	StarID	Star	Ra	Dec	g	r
1	1	S1	198.8475000	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

In SQL we do:

```
SELECT *
FROM Stars
WHERE r < 14.5
```

In sqlite:

```
sqlite> SELECT * FROM Stars WHERE r < 14.5;
4,2,S7,199.2516667,10.3486944,14.6,14.1
```

not the prettiest formatting (mysql is nicer) but good enough.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

```
FROM Stars
```

Use the table named Stars for this query.

Breaking down the SELECT query:

1. Give me all stars brighter than r=14.5

```
SELECT *
```

Return all columns for all the rows that matches the constraints. We can also specify specific columns.

```
FROM Stars
```

Use the table named Stars for this query.

```
WHERE r < 14.5
```

Only return those **rows** that satisfy the criteri(on/a) that we specify.

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
    FROM Stars
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
    FROM Stars
```

```
sqlite> SELECT Star, g, r FROM Stars;
S1,14.5,15.2
S2,15.3,15.4
S5,16.4,15.8
S7,14.6,14.1
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

```
SELECT Star, g, r
FROM Stars
WHERE r < 14.5
```

But what if I want some combination of columns?

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r	g-r
1	1	S1	198.8475	10.5034722	14.5	15.2	-0.7
1	2	S2	198.5654167	11.0231944	15.3	15.4	-0.1
3	3	S5	198.9370833	9.9168889	16.4	15.8	0.6
2	4	S7	199.2516667	10.3486944	14.6	14.1	0.5

```
SELECT Star, g, r, g-r as gr
FROM Stars
WHERE FieldID = 1
```

Choosing columns to output

FieldID	StarID	Star	Ra	Decl	g	r	g-r
1	1	S1	198.8475	10.5034722	14.5	15.2	-0.7
1	2	S2	198.5654167	11.0231944	15.3	15.4	-0.1
3	3	S5	198.9370833	9.9168889	16.4	15.8	0.6
2	4	S7	199.2516667	10.3486944	14.6	14.1	0.5

```
SELECT Star, g, r, g-r as gr
FROM Stars
WHERE FieldID = 1
```

```
sqlite> SELECT Star, g, r, g-r as gr FROM Stars WHERE FieldID
= 1;
S1,14.5,15.2,-0.699999999999999
S2,15.3,15.4,-0.0999999999999996
```

Getting a few values - SQL flavours...

When you have very large tables, you often want to try out statements or just get few examples back. This is easy but depends on the SQL flavour you use:

Microsoft SQL (used in SDSS):

```
SELECT TOP 2 r FROM STARS
```

MySQL and sqlite:

```
SELECT r FROM STARS LIMIT 2
```

Oracle (used in AstroWISE):

```
SELECT r FROM STARS WHERE ROWNUM < 2
```

Recall:

Table Observations

#	ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits

Table Stars

#	FieldID	StarID	Star	Ra	Dec	g	r
	1	1	S1	198.8475000	10.5034722	14.5	15.2
	1	2	S2	198.5654167	11.0231944	15.3	15.4
	3	3	S5	198.9370833	9.9168889	16.4	15.8
	2	4	S7	199.2516667	10.3486944	14.6	14.1

Now let us go back to our questions:

3. When did we observe S2?
4. Where is the FITS image stored for star S5?
5. Give me a list of all stars observed on the same FieldID

#	ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits

#	FieldID	StarID	Star	Ra	Dec	g	r
	1	1	S1	198.8475000	10.5034722	14.5	15.2
	1	2	S2	198.5654167	11.0231944	15.3	15.4
	3	3	S5	198.9370833	9.9168889	16.4	15.8
	2	4	S7	199.2516667	10.3486944	14.6	14.1

In these cases we need to be able to link information between two tables. In SQL we do this using JOINs

First a theoretical view:

Two sets of values: $\{x_i\}$ $\{y_j\}$ (the elements can be vectors/matrices etc)

Possible ways to combine:

Union: $\{x_i, y_j | i=1, n; j=1, m\}$ elements must be the same

Cross-join: $\{(x_i, y_j) | i=1, n; j=1, m\}$ ie. all possible pairs

Left Outer join: $\{(x_i, y_i)$ if y_i exists, $(x_i$, NULL) otherwise$\}$

Right Outer join: $\{(x_i, y_i)$ if x_i exists, (NULL, y_i) otherwise$\}$

Inner join: $\{(x_i, y_i)$ if y_i exists$\}$

All these are supported in SQL.

How do you combine?

ID	u
1	19.3
2	17.5
3	20.5

ID	EW
1	75Å
3	0.5Å

JOIN

ID	u	EW
1	19.3	75Å
3	20.5	0.5Å

How do you combine?

ID	u
1	19.3
2	17.5
3	20.5

ID	EW
1	75Å
3	0.5Å

JOIN

ID	u	EW
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.EW
FROM Photo as P
   JOIN  Spectro as S
ON P.ID=S.ID
```

How do you combine?

ID	u
1	19.3
2	17.5
3	20.5

ID	EW
1	75Å
3	0.5Å

JOIN

ID	u	EW
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.EW
FROM Photo as P
   JOIN  Spectro as S
ON P.ID=S.ID
```

OR

```
SELECT P.u, S.EW
FROM Photo as P,
Spectro as S
WHERE P.ID=S.ID
```

How do you combine?

ID	u
1	19.3
2	17.5
3	20.5

ID	EW
1	75Å
3	0.5Å

JOIN

ID	u	EW
1	19.3	75Å
3	20.5	0.5Å

```
SELECT P.u, S.EW
FROM Photo as P
   JOIN  Spectro as S
ON P.ID=S.ID
```

Explicit INNER JOIN

OR

```
SELECT P.u, S.EW
FROM Photo as P,
Spectro as S
WHERE P.ID=S.ID
```

Implicit INNER JOIN

(or *old-style* INNER JOIN)

Explicit vs implicit JOINs

JOIN ... ON a=b

or

WHERE a = b

Mostly up to you - there should be no significant difference between the two.

The main disadvantage of an implicit JOIN is that you have less control of the order things are done if you have more than two tables.

I personally prefer explicit JOINs because they show more clearly what your intention is and if you have a problem because your query runs too slowly, you can more easily figure out the execution order.

UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0

UNION

Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 1

Table 2

Try it in SDSS!

UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0

UNION

Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 1

Table 2

Ra	Dec

Try it in SDSS!

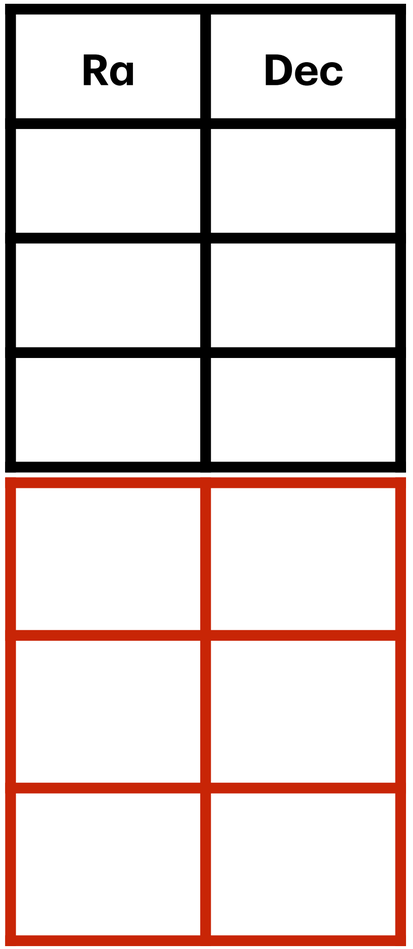
UNION

It must make sense to glue the tables together!

```
Select TOP 10
ra, dec
FROM SpecPhoto
WHERE ra > 120
AND DEC < 0

UNION

Select TOP 10
ra, dec
From SpecPhoto
WHERE ra < 10
AND DEC > 0
```

Table 1

Table 2



Try it in SDSS!

But if we want to keep **all** possible pairs we need an **OUTER JOIN**

```
SELECT P.u, S.z
FROM Photo as P
LEFT OUTER JOIN  Spectro as S
ON P.ID=S.ID
```

But if we want to keep **all** possible pairs we need an **OUTER JOIN**

```
SELECT P.u, S.z
FROM Photo as P
LEFT OUTER JOIN  Spectro as S
ON P.ID=S.ID
```

ID	u
1	19.3
2	17.5
3	20.5

ID	EW
1	75Å
3	0.5Å

ID	u	EW(Ha)
1	19.3	75Å
2	17.5	NULL
3	20.5	0.5Å

Returning to our questions:

3. When did we observe S2?

#	ID	Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1		/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1		/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5		/disks/yaeps-1/StF-045.fits

#	FieldID	StarID	Star	Ra	Dec	g	r
	1	1	S1	198.8475000	10.5034722	14.5	15.2
	1	2	S2	198.5654167	11.0231944	15.3	15.4
	3	3	S5	198.9370833	9.9168889	16.4	15.8
	2	4	S7	199.2516667	10.3486944	14.6	14.1

Our link is FieldID in Stars to ID in Observations

Returning to our questions:

3. When did we observe S2?

#	ID Field	Date	Exptime	Quality	WhereStored
1	StF-043	92.9885764	23.2	1	/disks/yaeps-1/StF-043.fits
2	StF-044	97.3323764	30.2	1	/disks/yaeps-1/StF-044.fits
3	StF-045	93.5532134	29.5	0.5	/disks/yaeps-1/StF-045.fits

#	FieldID	StarID	Star	Ra	Dec	g	r
	1	1	S1	198.8475000	10.5034722	14.5	15.2
	1	2	S2	198.5654167	11.0231944	15.3	15.4
	3	3	S5	198.9370833	9.9168889	16.4	15.8
	2	4	S7	199.2516667	10.3486944	14.6	14.1

Our link is FieldID in Stars to ID in Observations

```
select s.Star, o.Field, o.Date
from
   stars as s
   JOIN Observations as o
   ON s.fieldID = o.ID
Where s.Star = ‘S2’
```

Returning to our questions:

3. When did we observe S2?

```
select s.Star, o.Field, o.Date
from
  stars as s
  JOIN Observations as o
  ON s.fieldID = o.ID
Where Star = ‘S2’
```

We must specify what table to get a quantity from.

JOIN the tables explicitly

Choose our star

Useful: We can do **multiple** stars by changing the WHERE statement to:

Returning to our questions:

3. When did we observe S2?

```
select s.Star, o.Field, o.Date
from
  stars as s
  JOIN Observations as o
  ON s.fieldID = o.ID
Where Star = ‘S2’
```

We must specify what table to get a quantity from.

JOIN the tables explicitly

Choose our star

Useful: We can do **multiple** stars by changing the WHERE statement to:

```
Where Star IN (‘S2’, ‘S1’)
```

Returning to our questions:

3. When did we observe S2?

```
select s.Star, o.Field, o.Date
from
  stars as s
  JOIN Observations as o
  ON s.fieldID = o.ID
Where Star = ‘S2’
```

We must specify what table to get a quantity from.

JOIN the tables explicitly

Choose our star

Useful: We can do **multiple** stars by changing the WHERE statement to:

```
Where Star IN (‘S2’, ‘S1’)
```

That’s it for now - more advanced topics for SQL are included at the end of the slides for self-study

Let’s switch to Colab!

Notebook: MLD2025-01c-Python and SQL

Direct link

Python and SQL

Python & SQL

There are many ways to interact with databases in Python. I will cover two: an in-depth one and a simple one.

Not covered, but maybe of relevance:

MySQL interface for Python: MySQLdb

PostgresSQL with Python: psycopg2

Oracle with Python: cx_Oracle

There are also many places you can learn much more about this topic:

I quite like the compilation of links on Full Stack Python:

https://www.fullstackpython.com/databases.html

Google will give you many other options - it is a very popular topic...

Python & SQL

There are many ways to interact with databases in Python. I will cover two: an in-depth one and a simple one.

I will assume that you have a database file already created for the following slides. If you do not, then you can either get ProblemSets/MakeTables/MLD2024.db from the GitHub site, or you can play with e.g. the places.sqlite database in your Firefox profile directory.

Python & sqlite3 the comprehensive way

```
import sqlite3 as lite
```

Load what is necessary

The database must be created first!

```
con = lite.connect(database)
```

Connect to database

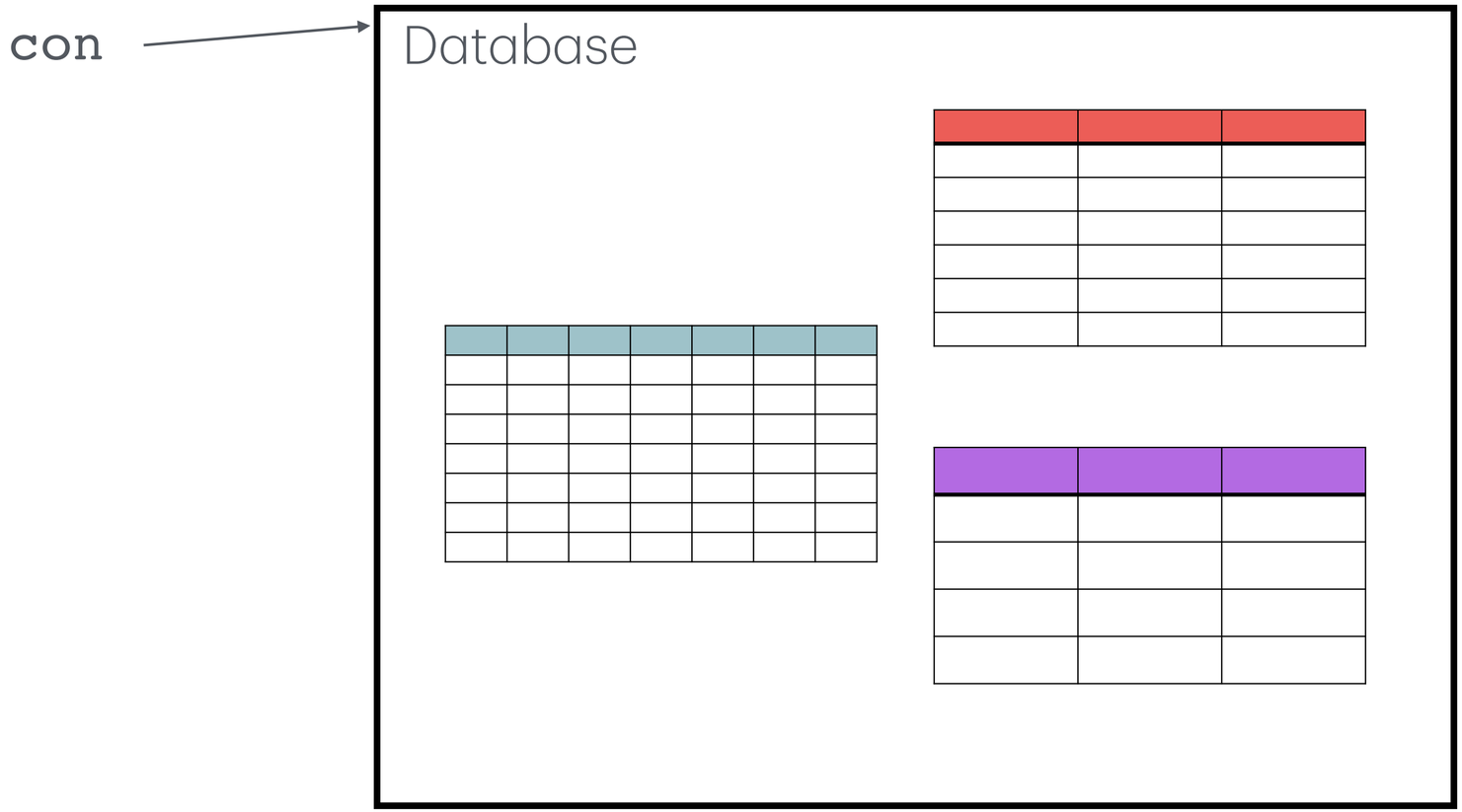
```
with con:

  # Get a cursor.
  cur = con.cursor()

  # Execute commands
  cur.execute(command)
```

The Cursor object is one way to pass SQL statements to the database - I like to think about it as the finger reading through a table.

This is frequently seen - but it is not strictly necessary. I would recommend not creating an explicit cursor object like this for simplicity.

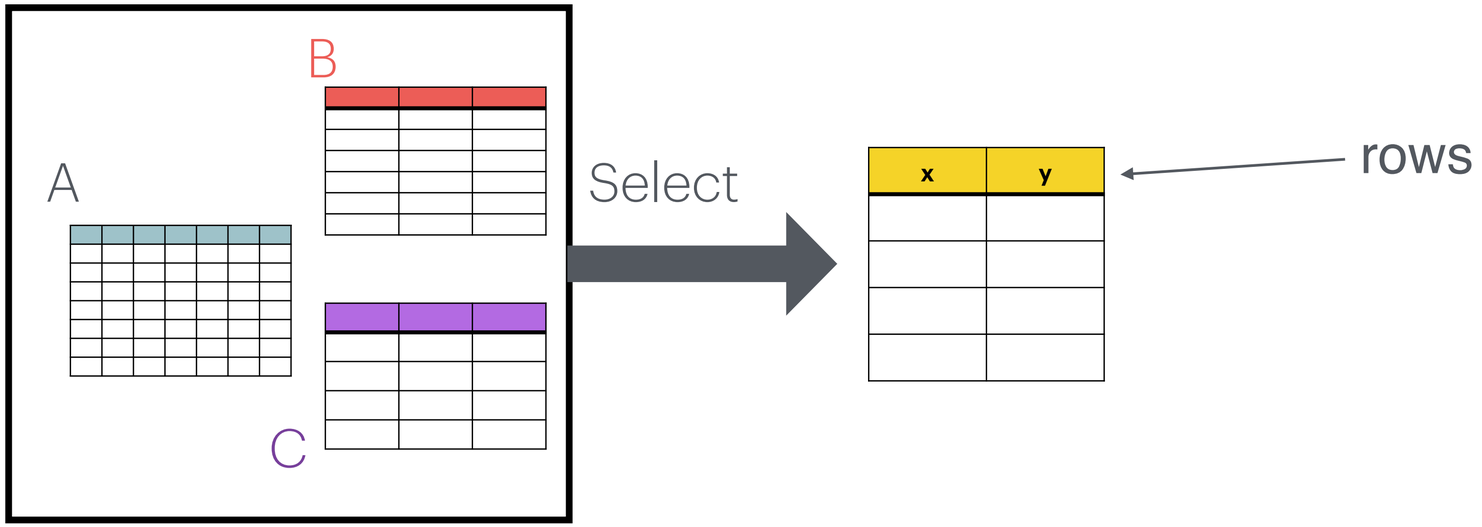
Python & SQL - the Cursor object

```
con = lite.connect(database)
```



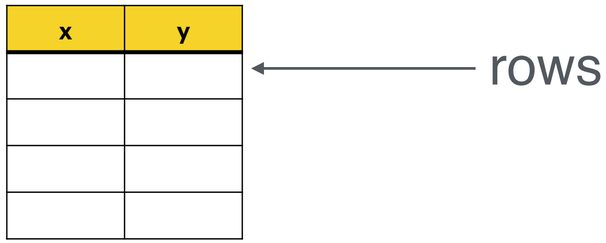
Using python to query the database:

```
In [1]: import sqlite3 as lite;

In [2]: con = lite.connect("MLD2025.db")

In [4]: query = "select s.star, o.Date from stars as s JOIN observations as
o ON s.fieldID=o.ID"

In [3]: rows = con.execute(query)
```



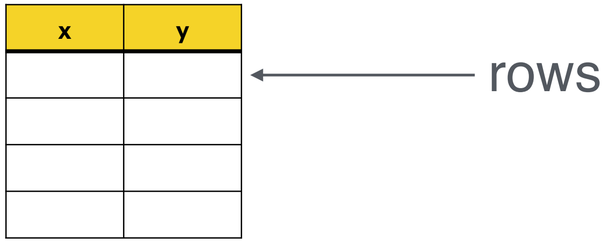
Using python to query the database:

```
In [4]: row = rows.fetchone()

In [5]: print row
Out[57]: ('S1', 92.9885764)
```



Using python to query the database:

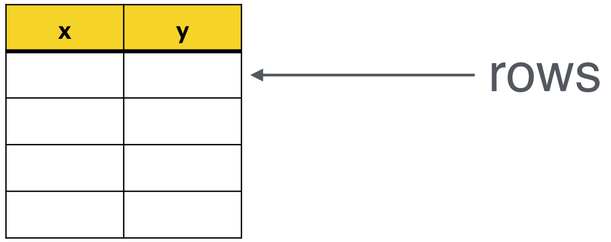
```
In [4]: row = rows.fetchone()

In [5]: print row
Out[5]: ('S1', 92.9885764)
```




```
In [6]: for row in rows:
   ....:     print(row)
   ....:
('S2', 92.9885764)
('S5', 93.5532134)
('S7', 97.3323764)
```

Using python to query the database:

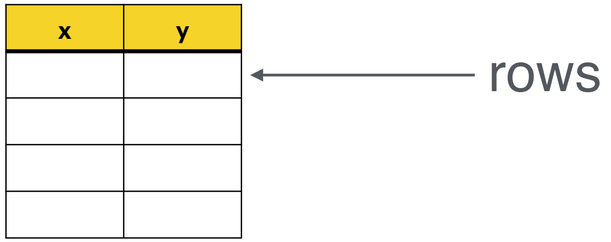
```
In [4]: row = rows.fetchone()

In [5]: print row
Out[5]: ('S1', 92.9885764)
```




```
In [6]: for row in rows:
   ....:     print(row)
   ....:
('S2', 92.9885764)
('S5', 93.5532134)
('S7', 97.3323764)
```

Using python to query the database:

```
In [4]: row = rows.fetchone()

In [5]: print row
Out[5]: ('S1', 92.9885764)
```




```
In [6]: for row in rows:
   ....:     print(row)
   ....:
('S2', 92.9885764)
('S5', 93.5532134)
('S7', 97.3323764)
```

Watch out!

```
In [7]: row = rows.fetchone()
In [8]: row
```

Because we used up all rows in the previous call! If you use cursors this would lead row to contain the last row - a bug waiting to happen.

The simplest(?) way of dealing with tables in Python

pandas:

```
In [35]: import pandas as pd

In [36]: con = lite.connect(“MLD2025.db")

In [37]: t = pd.read_sql_query('Select ra, decl from Stars', con)

In [38]: t
Out[38]:
            ra        decl
0   198.847500   10.503472
1   198.565417   11.023194
2   198.937083    9.916889
3   199.251667   10.348694
```

The simplest(?) way of dealing with tables in Python

pandas:

```
In [35]: import pandas as pd

In [36]: con = lite.connect(“MLD2025.db")

In [37]: t = pd.read_sql_query('Select ra, decl from Stars', con)

In [38]: t
Out[38]:
           ra        decl
0  198.847500  10.503472
1  198.565417  11.023194
2  198.937083   9.916889
3  199.251667  10.348694
```

and it is easy to make new tables:

```
In [39]: t.to_sql(‘Stars2’, con)
```

That’s it.

pandas vs normal sqlite

sqlite3

```
# Next, we create a connection to the database.
con = lite.connect(database)


with con:

    table = "Stars"
    # Create the command to create the table. I use a
    # multiline string to ease readability here.
    command = """CREATE TABLE IF NOT EXISTS {0} (StarID INT,
         FieldID INT, Star varchar(10), ra DOUBLE,
         decl DOUBLE, g FLOAT, r FLOAT,
         UNIQUE(StarID), PRIMARY KEY(StarID),
         FOREIGN KEY(FieldID) REFERENCES Observations(ID))""".format(table)

    # Next, actually execute this command.
    con.execute(command)

    # Now that this is working, let us loop over the table entries
    # and insert these into the table.
    for row in cat:
        command = "INSERT INTO Stars VALUES({0},{1},'{2}',{3},{4},{5},{6})".format(row[0], row[1], row[2], row[3],
row[4], row[5], row[6])
        print command
        con.execute(command)
```

pandas

```
In [39]: t.to_sql('Stars2', con)
```

So why not only pandas?

pandas vs normal sqlite

sqlite3

```
# Next, we create a connection to the database.
con = lite.connect(database)


with con:

    table = "Stars"
    # Create the command to create the table. I use a
    # multiline string to ease readability here.
    command = """CREATE TABLE IF NOT EXISTS {0} (StarID INT,
         FieldID INT, Star varchar(10), ra DOUBLE,
         decl DOUBLE, g FLOAT, r FLOAT,
         UNIQUE(StarID), PRIMARY KEY(StarID),
         FOREIGN KEY(FieldID) REFERENCES Observations(ID))""".format(table)

    # Next, actually execute this command.
    con.execute(command)

    # Now that this is working, let us loop over the table entries
    # and insert these into the table.
    for row in cat:
        command = "INSERT INTO Stars VALUES({0},{1},'{2}',{3},{4},{5},{6})".format(row[0], row[1], row[2], row[3],
row[4], row[5], row[6])
        print command
        con.execute(command)
```

pandas

```
In [39]: t.to_sql('Stars2', con)
```

So why not only pandas?

★ Control
★ Memory usage

Creation of databases & using them from Python

Creating a sqlite database

Let us set up a simple database to start with:

```
> sqlite3 MLD2025.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .tables
sqlite> .exit
```

This should give you nothing because there are no tables.
We need to make one.

Step 2 - inserting a table

Get: YAEPS.stars-table-sqlite.dat and sqlite3-make-stars-table.sql from the GitHub site (ProblemSet/MakeTables). Edit the latter to reflect the location of YAEPS.stars-table-sqlite.dat

When that is done:

```
> sqlite3 MLD2025.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .read sqlite3-make-stars-table.sql
sqlite> .tables
Stars
```

Magic?

First we need to create a schema

To do this we need to understand our data:

```
# StarID FieldID  Star     Ra          Dec         g      r
   1         1      S1   198.8475000 10.5034722 14.5    15.2
   2         1      S2   198.5654167 11.0231944 15.3    15.4
   3         3      S5   198.9370833  9.9168889 16.4    15.8
   4         2      S7   199.2516667 10.3486944 14.6    14.1
```

Column	Type	SQL type	Other
StarID	Integer	INT	PrimaryKey,
FieldID	Integer	INT	ForeignKey
Star	String	varchar(10)	Length < 10
Ra	Real number	DOUBLE	
Dec	Real number	DOUBLE	
g	Real number	DOUBLE	
r	Real number	DOUBLE	

Schema

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
   StarID INT,
   FieldID INT,
   Star varchar(10),
   ra DOUBLE,
   decl DOUBLE,
   g FLOAT,
   r FLOAT,
   UNIQUE(StarID),
   PRIMARY KEY(StarID),
   FOREIGN KEY(FieldID) REFERENCES
   Observations(ID)
   );
```

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
   StarID INT,
   FieldID INT,
   Star varchar(10),
   ra DOUBLE,
   decl DOUBLE,
   g FLOAT,
   r FLOAT,
   UNIQUE(StarID),
   PRIMARY KEY(StarID),
   FOREIGN KEY(FieldID) REFERENCES
   Observations(ID)
   );
```

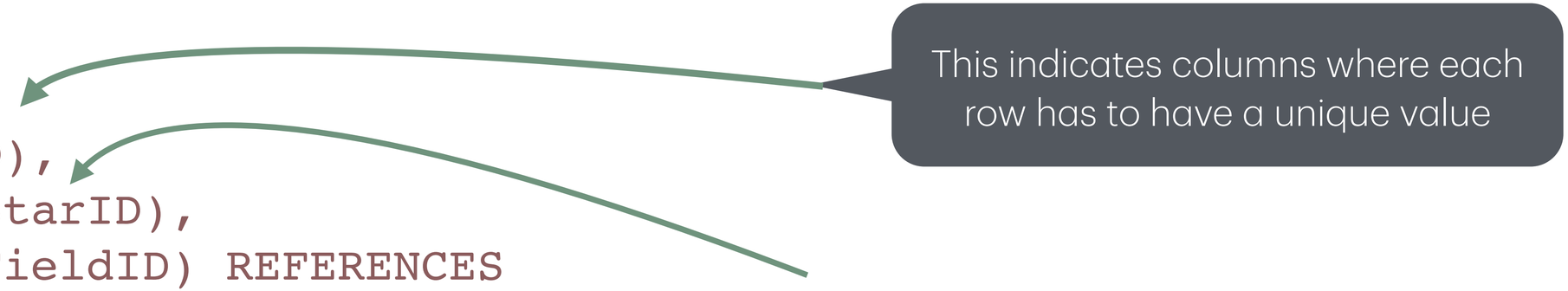
Notice the definition of strings

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
   StarID INT,
   FieldID INT,
   Star varchar(10),
   ra DOUBLE,
   decl DOUBLE,
   g FLOAT,
   r FLOAT,
   UNIQUE(StarID),
   PRIMARY KEY(StarID),
   FOREIGN KEY(FieldID) REFERENCES
   Observations(ID)
   );
```



You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
   StarID INT,
   FieldID INT,
   Star varchar(10),
   ra DOUBLE,
   decl DOUBLE,
   g FLOAT,
   r FLOAT,
   UNIQUE(StarID),
   PRIMARY KEY(StarID),
   FOREIGN KEY(FieldID) REFERENCES
   Observations(ID)
   );
```

This can optionally be used to indicate the primary key in this database

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

Our star table is created with:

```
CREATE TABLE IF NOT EXISTS Stars (
   StarID INT,
   FieldID INT,
   Star varchar(10),
   ra DOUBLE,
   decl DOUBLE,
   g FLOAT,
   r FLOAT,
   UNIQUE(StarID),
   PRIMARY KEY(StarID),
   FOREIGN KEY(FieldID) REFERENCES
   Observations(ID)
   );
```

Optional: Use this to indicate foreign keys in the table. This can be useful for clarity at least.

You can type this on the command line, but it is easier to store it in a file!

Creating databases and tables

well, we need some more (this is database specific:)

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

These quick import routines are not part of SQL so vary from database to database (but they are handy!)

Creating databases and tables

well, we need some more (this is database specific:)

```
.separator ,
.import YAEPS.stars-table-sqlite.dat Stars
```

These quick import routines are not part of SQL so vary from database to database (but they are handy!)

In MySQL for instance it would be:

```
LOAD DATA INFILE 'YAEPS.stars-table-sqlite.dat' INTO TABLE Stars
  FIELDS TERMINATED BY ‘,’;
```

here I can also add IGNORE 1 LINES at the end if I want to skip a header line.

Getting rid of a table & altering it

Everyone makes mistakes. Sometimes the best is to just get rid of a table. It is easy:

```
DROP TABLE Galaxy;
```

It is also possible to modify (alter) a table - but note that this does not fully work in sqlite!

```
ALTER TABLE Galaxy ADD rmag  FLOAT
```

Adding column

```
ALTER TABLE Galaxy DROP COLUMN rmag
```

Deleting column

```
ALTER TABLE Galaxy ALTER rmag DOUBLE
```

Changing the type of a column

Putting data into the database - row by row

Adding data one row at a time is done using INSERT:

```
INSERT INTO Galaxy VALUES (1,
12.334, 14.433);
```

GalaxyID	ra	decl

GalaxyID	**ra**	**decl**
1	12.334	14.433

You can also insert only some values but then you have to say what columns they are for:

```
INSERT INTO Galaxy (GalaxyID,
decl) VALUES (2, 17.5);
```

GalaxyID	**ra**	**decl**
1	12.334	14.433
2	NULL	17.5

Note: You can also insert multiple rows if you copy from one table to another.

Putting data into the database - in one go

INSERT is quite slow, in part because the database is reorganised after each insert. This is fine for small jobs but not for 100,000s of entries. For this case we use LOAD DATA.

```
LOAD DATA INFILE <filename> INTO TABLE
Stars
  FIELDS TERMINATED BY ',' IGNORE 1 LINES;
```

(optional)

will load from a file where each column is separated by a comma (the default is TAB), and rows by newline but it will skip the first line. The column types must match the Table definition - so in this case a file that can be loaded would be:

```
# StarID FieldID Star Ra Dec    g    r
1,1,S1,198.8475000,10.5034722,14.5,15.2
2,1,S2,198.5654167,11.0231944,15.3,15.4
```

This is fine for those situations where your data are already known - for instance if you downloaded a catalogue.

Updating data

Most astronomical data can change (calibration files can be updated, measurement techniques improved etc.)

We often then want to create a new table, but sometimes you want to update a row instead. This is done in SQL using the UPDATE command.

```
UPDATE Galaxy SET ra=11.3 WHERE decl=17.5
```

This is ok, in particular where information is acquired after most of the table is assembled.

Next: A more fancy criterion

2. How many stars have 0.1 < g-r < 0.4?

```
SELECT *
FROM Stars
WHERE g-r BETWEEN 0.1 AND 0.4
```

or

```
SELECT *
FROM Stars
WHERE g-r > 0.1
  AND g-r < 0.4
```

Try this now for your newly created table - remember to put a ; at the end of each line!

Reminder:

```
# StarID FieldID  Star     Ra          Dec         g      r
   1         1      S1   198.8475000 10.5034722 14.5   15.2
   2         1      S2   198.5654167 11.0231944 15.3   15.4
   3         3      S5   198.9370833  9.9168889 16.4   15.8
   4         2      S7   199.2516667 10.3486944 14.6   14.1
```

Column	Type	SQL type	Other
StarID	Integer	INT	PrimaryKey,
FieldID	Integer	INT	ForeignKey
Star	String	varchar(10)	Length < 10
Ra	Real number	DOUBLE	
Dec	Real number	DOUBLE	
g	Real number	DOUBLE	
r	Real number	DOUBLE	

Create a new table - now for the Observations

```
# FieldID  Field      Date       Exptime   Quality          WhereStored
   1       StF-043  92.9885764    23.2       1        /disks/yaeps-1/StF-043.fits
   2       StF-044  97.3323764    30.2       1        /disks/yaeps-1/StF-044.fits
   3       StF-045  93.5532134    29.5       0.5      /disks/yaeps-1/StF-045.fits
```

Column	Type	SQL type	Other
FieldID			
Field			
Date			
Exptime			
Quality			
WhereStored			

In practice:

Get YAEPS.observations-table-sqlite.dat and sqlite3-make-observations-table.sql from Github. Edit the latter to reflect the location of YAEPS.observations-table-sqlite.dat

```
> sqlite3 MLD2025.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .read sqlite3-make-observations-table.sql
sqlite> .tables
Observations  Stars
```

sqlite from Python
or - what you really want to know

sqlite3 in python - an example

```
import sqlite3 as lite;
```

Load what is necessary

The database must be created first!

```
con = lite.connect(database)
```

Connect to database

```
with con:
```

Use **with** to gracefully handle exceptions

```
   # Get a cursor.
   cur = con.cursor()

   # Execute commands
   cur.execute(command)
```

cursors are used to navigate relational databases and are often needed in programatic access

Building the table in python:

```
# Next, we create a connection to the database.
con = lite.connect(database)
with con:

    table = ‘Stars’
    # Create the command to create the table. I use a
    # multiline string to ease readability here.
    command = """CREATE TABLE IF NOT EXISTS {0} (StarID INT,
              FieldID INT, Star varchar(10), ra DOUBLE,
              decl DOUBLE, g FLOAT, r FLOAT,
              UNIQUE(StarID), PRIMARY KEY(StarID),
              FOREIGN KEY(FieldID) REFERENCES Observations(ID))""".format(table)

    # Next, actually execute this command.
    con.execute(command)

    # Now that this is working, let us loop over the table entries
    # and insert these into the table.
    for row in cat:
        command = "INSERT INTO Stars VALUES({0},{1},'{2}',{3},{4},{5},{6})".format(row[0], row[1],
row[2], row[3], row[4], row[5], row[6])
        print command
        con.execute(command)
```

See the GitHub site for the script - now build one for the observations. It is much easier to do this with Pandas!

Using python to query the database:

```
In [1]: import sqlite3 as lite;

In [2]: con = lite.connect('MLD19-python.db')

In [3]: rows = con.execute('SELECT ra, decl FROM Stars')

In [4]: for row in rows:
   ...:     print "Ra={0}  Dec={1}".format(row[0], row[1])
   ...:
Ra=198.8475  Dec=10.5034722
Ra=198.5654167  Dec=11.0231944
Ra=198.9370833  Dec=9.9168889
Ra=199.2516667  Dec=10.3486944
```

As should be clear: The execute statements executes SQL statements in the database and returns a list of results.

Regression

Standard linear regression

In this case we typically have p observables at each of N points (predictors) and want to predict a response variable, y_i at each x_i

A standard way to fit this is to minimise the residual sum of squares (RSS) or N times MSE:

$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the estimate of y_i. Which for linear regression is:

$$\hat{y}_i = \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij}$$

Common formulation - the design matrix

The problem to solve is then often written:

$$Y = \mathrm{M}\boldsymbol{\theta}$$

Where Y is $(y_1, y_2, \ldots, y_N)$ and $\boldsymbol{\theta}=(\theta_1, \theta_2, \ldots, \theta_p)$

M is known as the design matrix as mentioned earlier and is

$$\mathrm{M} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,p} \end{pmatrix}$$

Common formulation - the design matrix

If we also introduce the covariance matrix of uncertainties on Y:

$$\mathsf{C} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{pmatrix} \qquad Y = \mathsf{M}\boldsymbol{\theta}$$

the general solution of the linear regression is given by

$$\boldsymbol{\theta} = \left(\mathsf{M}^T \mathsf{C}^{-1} \mathsf{M}\right)^{-1} \left(\mathsf{M}^T \mathsf{C}^{-1} Y\right)$$

with uncertainties on the parameters given by

$$\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \left(\mathsf{M}^T \mathsf{C}^{-1} \mathsf{M}\right)^{-1}$$

Basis functions

Note that a regression would still be *linear* if we transformed all the predictors with a function:

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

This is known as basis function regression and can for instance be done using BasisFunctionRegression in astroML.linear_model. This transformation is a *pre-processing* step and while `sklearn` does not have a lot of functionality for this, a good introduction and explanation how to implement other basis functions is given in Jake Vanderplas' Data Science Handbook chapter 5 (available online)..

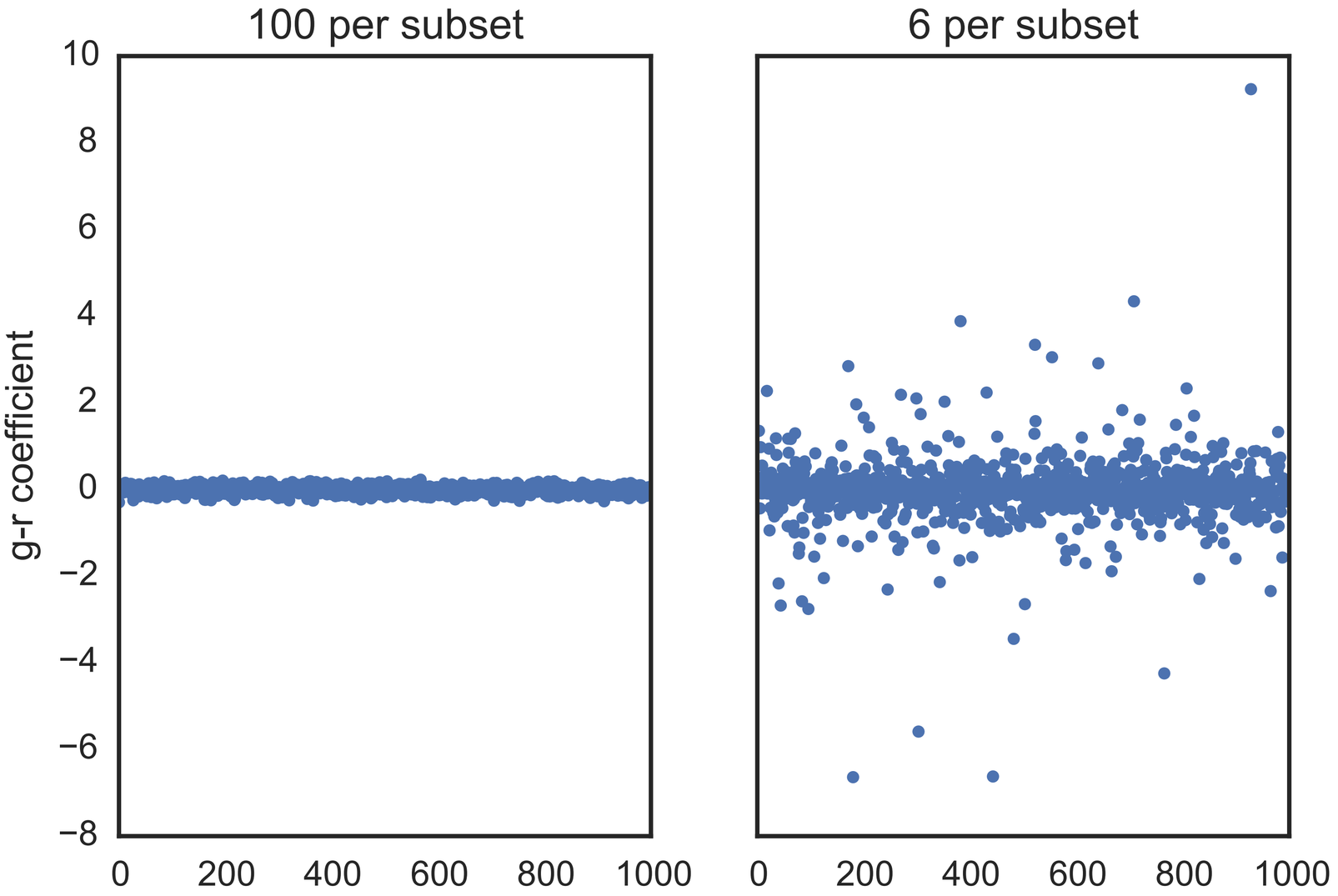
Regularising linear regression - ridge regression

When N is comparable to p, linear regression typically has large variance. To combat this we might want to trade a bit of bias for a lower variance.

As an example we can fit this:

$$T = \theta_0 + \theta_1(u - g) + \theta_2(g - r) + \theta_3(r - i) + \theta_5(i - z)$$

to the sspp dataset in astroML, where T is the effective temperature of the stars and the colours should be obvious.



Fits of 1000 random subsets with a linear regressor and showing just one coefficient.

Regularising linear regression - ridge regression

When N is comparable to p, linear regression typically has large variance. To combat this we might want to trade a bit of bias for a lower variance.

We do this by **regularising** the solution and minimise:

$$\mathrm{RSS} + \lambda f(\theta)$$

$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

The function $f(\theta)$ here would normally limit the size or number elements in θ.

This introduces a **regularisation parameter**: λ

We **determine** the value of the **regularisation** parameter using **cross-validation**

Regularising linear regression - ridge regression

The simplest(?) way to **regularise** the solution is to minimise:

$$\text{RSS} + \lambda \sum_i \theta_i^2$$

$$\text{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

Here we limit the *size* of the parameter vector θ.

When applied to linear regression, this leads to **ridge regression**.

Ridge regression - how to

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.05, normalize=True)

result = model.fit(M, T/1e4)
Tpred = model.predict(M)

res.coef_    # Coefficients of the fit.
```

Note: alpha = λ in my (and others') notation.

Very similar to LinearRegression - but there is something we should be aware of

Ridge regression - normalisation

Ridge regression can also be seen to want to minimise

$$\sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{p} \theta_j x_{ij} \right)^2$$

subject to

$$\sum_{j=1}^{p} \theta_j^2 \leq s$$

Obviously that length will depend on the **units** of x_{ij}. It is therefore common to "**whiten**" or **standardize** x by dividing by its standard deviation (ideally robustly).

Standardising - various ways to do this

Standardising variables means typically making them have a standard deviation of one and often a mean of zero (we do not want that here)

Manual approach:

```
scaler = StandardScaler(with_mean=False)
Ms = scaler.fit_transform(M)
```

We run the fit. If we want to scale the parameters back to the original M, we need to divide them by `scaler.scale_` which is the normalising scale.

Standardising - various ways to do this

Standardising variables means typically making them have a standard deviation of one and often a mean of zero (we do not want that here)

Manual approach:

```
scaler = StandardScaler(with_mean=False)
Ms = scaler.fit_transform(M)

fit = Ridge(alpha=alpha, fit_intercept=True)
res = model.fit(Ms, T/1e4)
# Scaling back:
res.coef_ = res.coef_/scaler.scale_
```

Standardising - various ways to do this

Standardising variables means typically making them have a standard deviation of one and often a mean of zero (we do not want that here)

Pipeline approach:

```
B = make_pipeline(
    StandardScaler(with_mean=False),
    Ridge(alpha=0.1, fit_intercept=True))

res = B[1].fit(B[0].fit_transform(M), T/1e4)

# If we want to scale back
res.coef_ = res.coef_/B[0].scale_
```

In the notebook you will see more details on this and how this can be improved

Ridge regression - degrees of freedom

There are of course p parameters, but because of the constraints in ridge regression, the effective number of degrees of freedom is not p - rather it is a smaller number depending on λ.

As far as I know this is not available through sklearn, but you can calculate it from the SVD of X:

$$X = UDV^T$$

If the diagonal entries in D are d_j, the d.o.f. is:

$$\mathrm{df} = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$$

Regularising linear regression - LASSO

Ridge regression minimises the L^2 norm of the coefficients. The Lasso (Least Absolute Shrinkage and Selection Operator) minimises the L^1 norm.

The minimisation is now of

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} |\theta_j|$$

Lasso regression - how to

```
from sklearn.linear_model import Lasso
model = Lasso(alpha=alpha, fit_intercept=True)

Ms = scaler.fit_transform(M)
result = model.fit(Ms, T/1e4)
Tpred = model.predict(Ms)

result.coef_/scaler.scale_  # Coefficients of the fit.
```

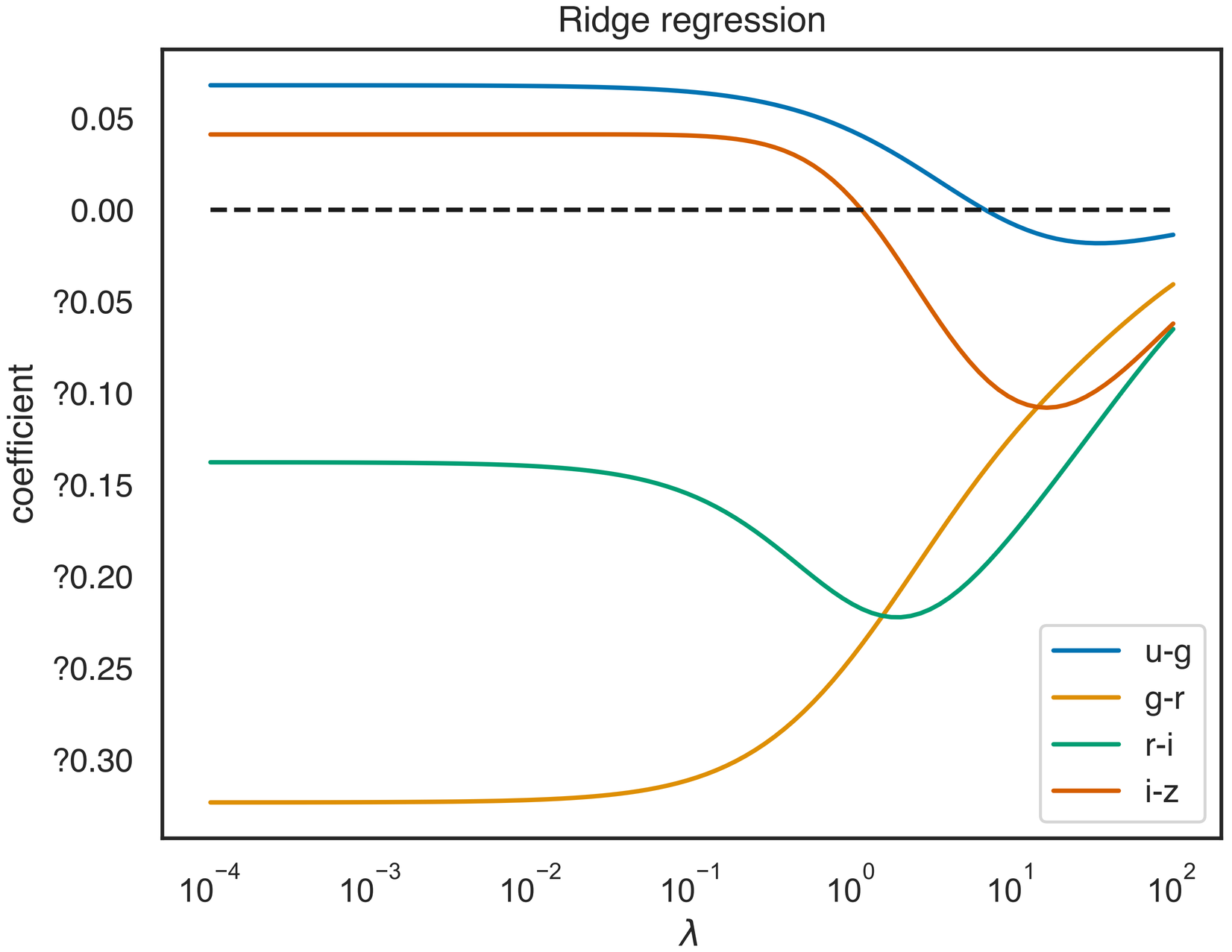
So just like ridge regression - but the result is somewhat different

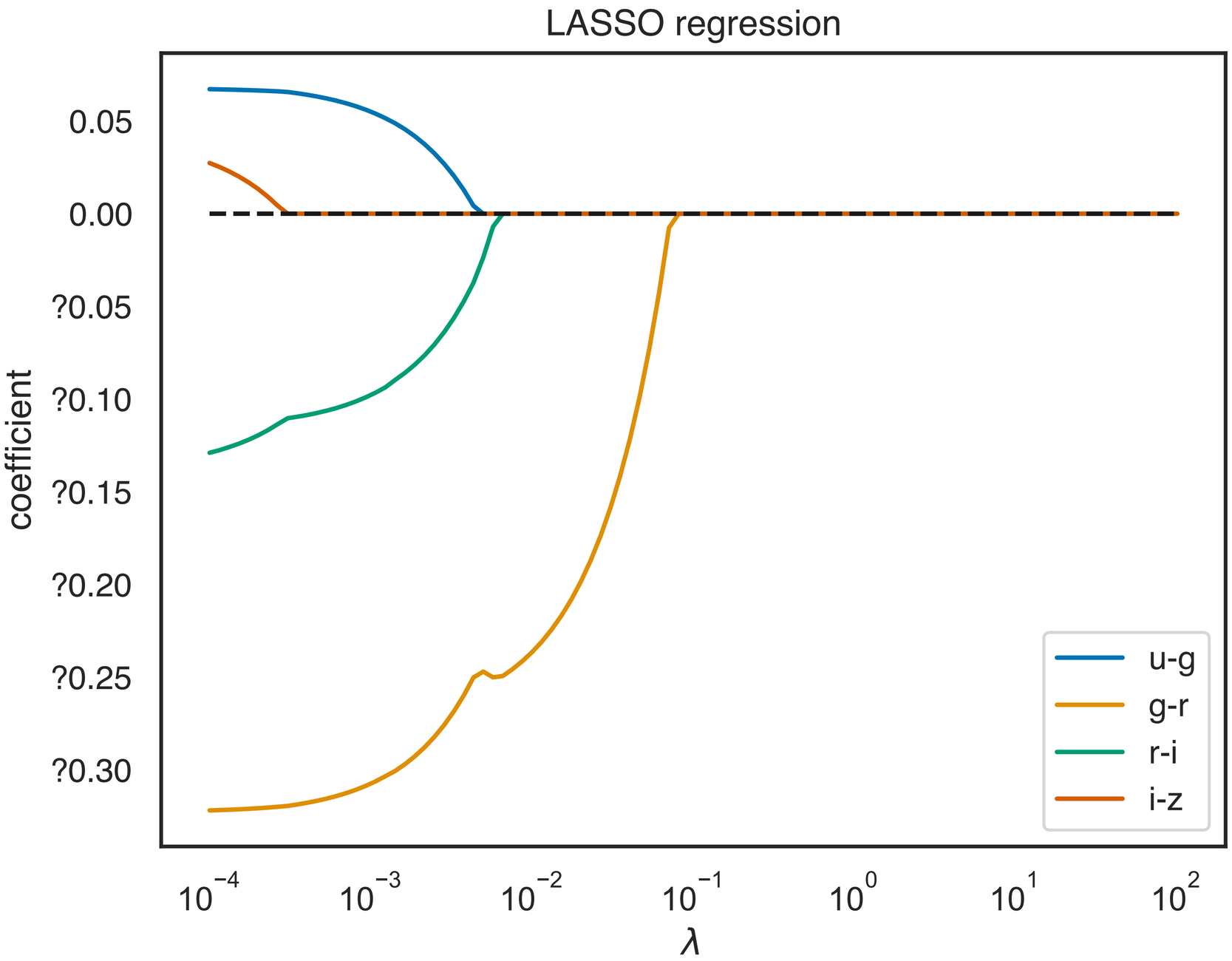
Let’s switch to Colab!

Notebook: MLD2025-03-regression

Direct link

The sensitivity to λ:




LASSO regression
0.05
0.00
?0.05
?0.10
?0.15
?0.20
?0.25
?0.30
coefficient
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
λ
u-g
g-r
r-i
i-z

Variable selection with the LASSO

So some coefficients end up being set to zero!

This fact means that the lasso performs ***variable selection***.

This feature of the lasso can be phrased to say that it returns **sparse models**.

Basically it can be interpreted to say which variables (predictors) are most important for predicting the output.

Subset selection

The most rigorous selection of variables is what is called ***subset selection***. This basically considers all possible combinations of parameters:

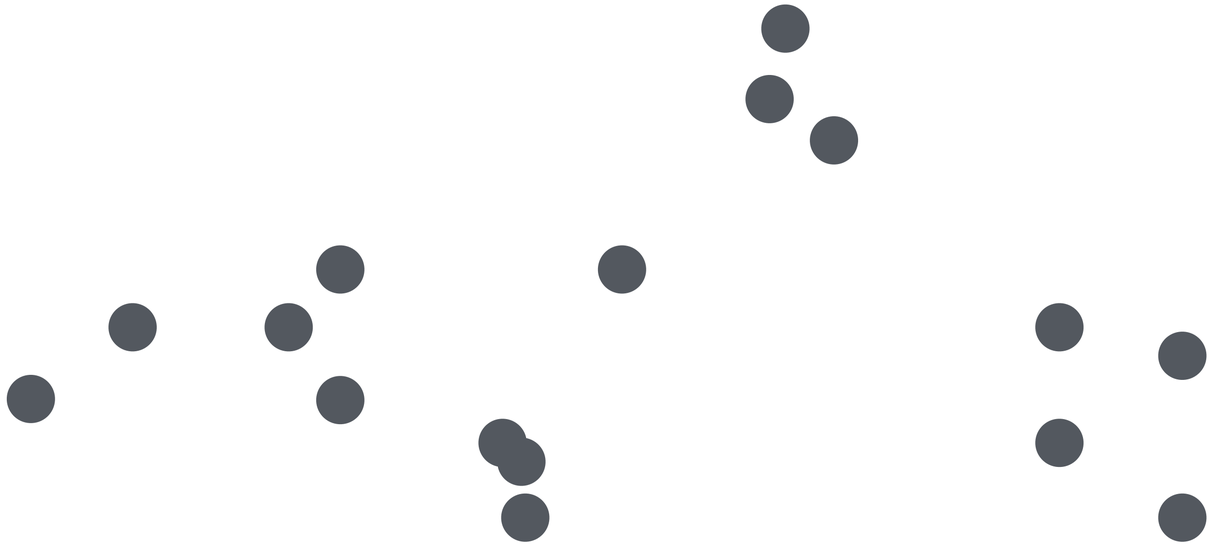
1. Set M_0 to be null model with no predictors
2. for k=1,2...p
   - Fit all $\binom{p}{k}$ models that contain exactly k predictors
   - Pick the model among these that has the lowest RSS and call this M_k.
3. Select the best model among these M by CV or similar.

No ready made routine for this - but similar tools are available in sklearn.feature_selection.

Regression -
keeping track of local properties

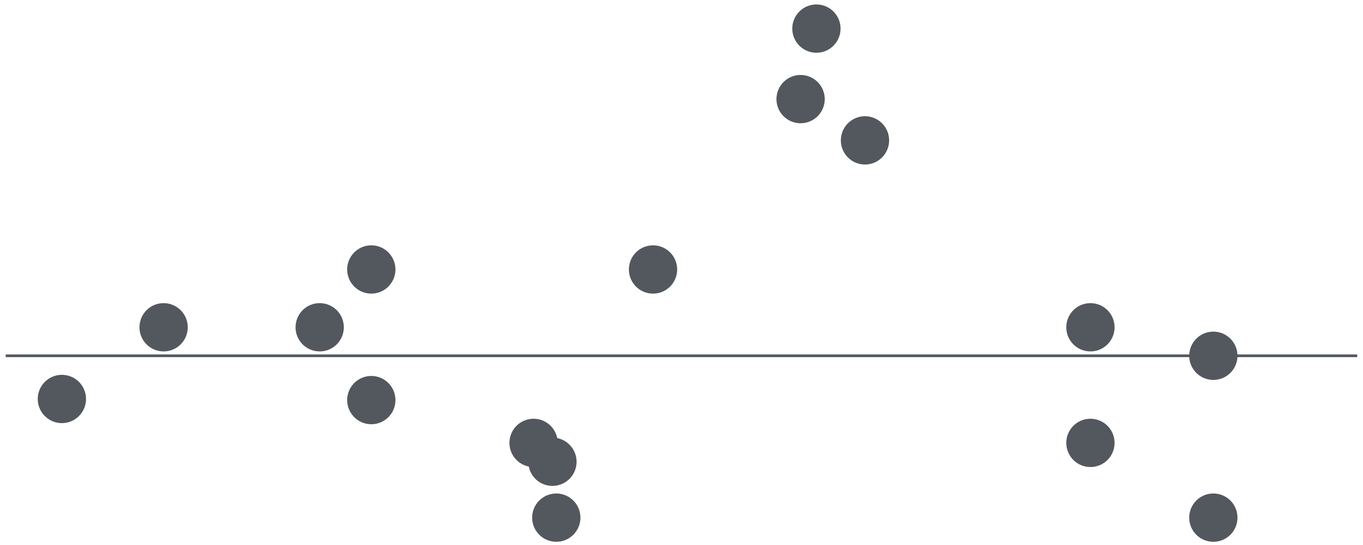
The broad idea

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



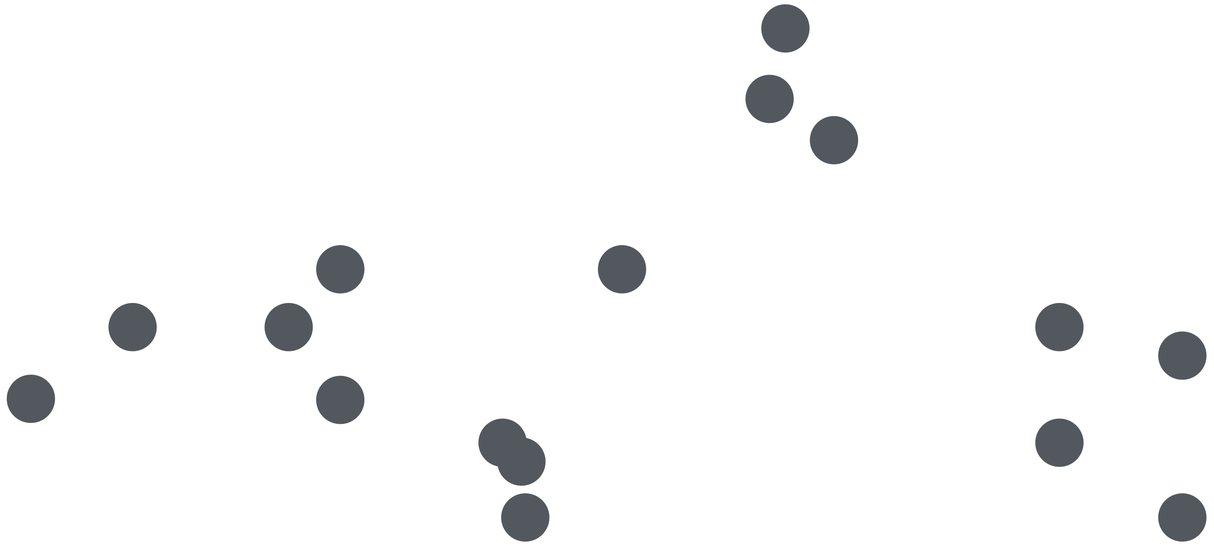
The broad idea

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



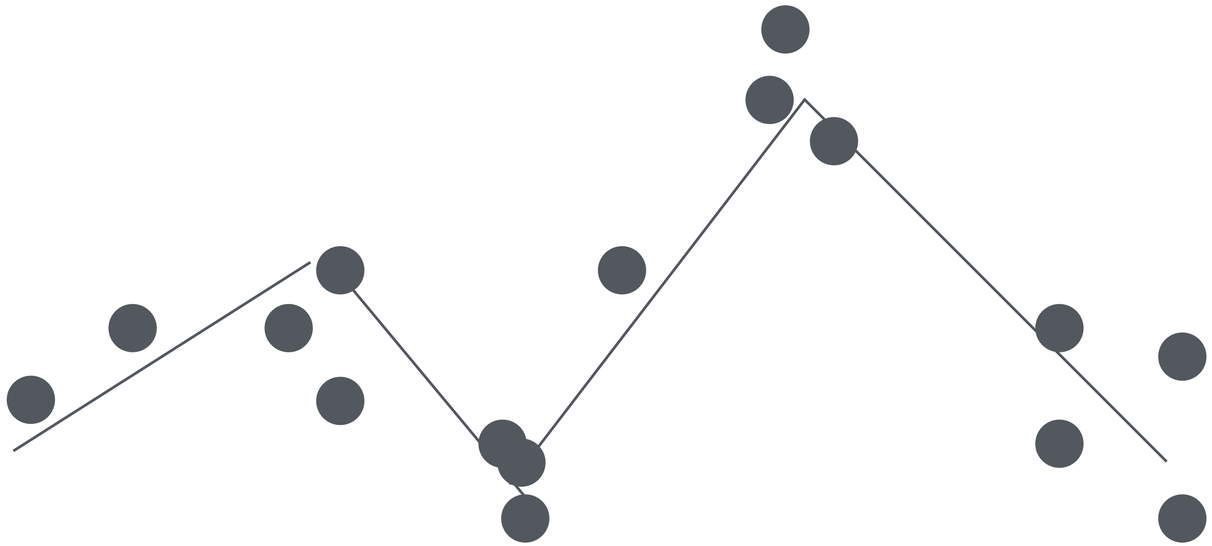
The broad idea

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

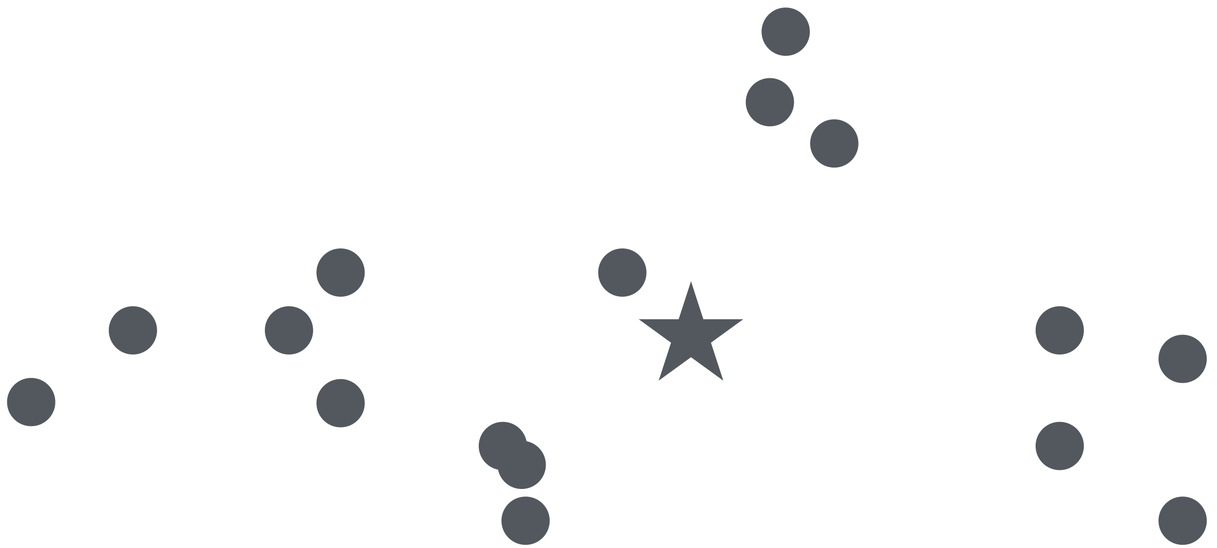
Calculate a weighted linear regression at each point

Gaussian process regression

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

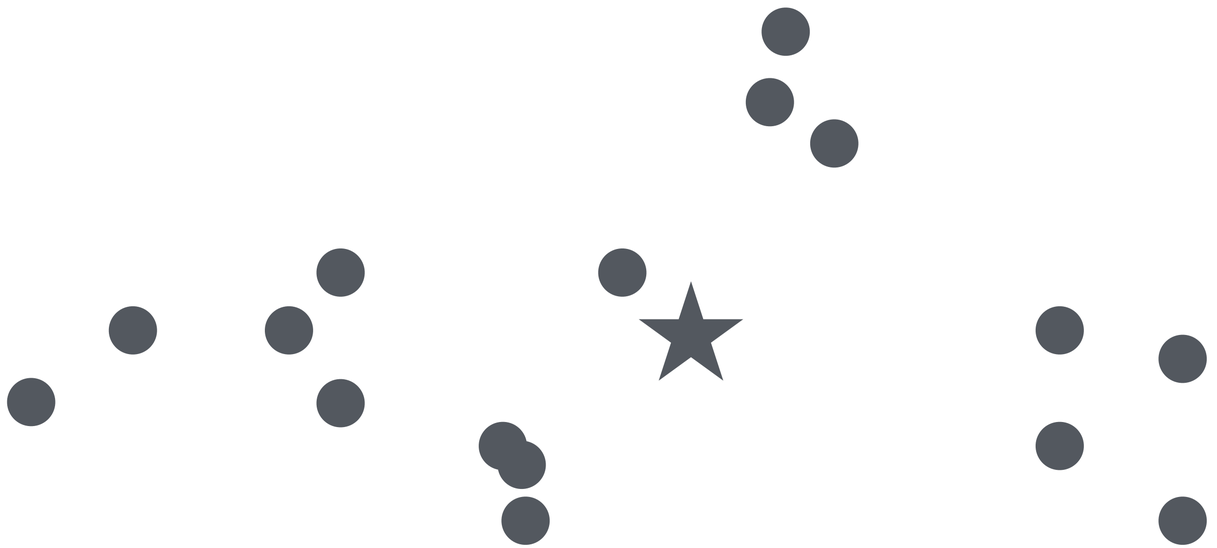
Nearest neighbour regression

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



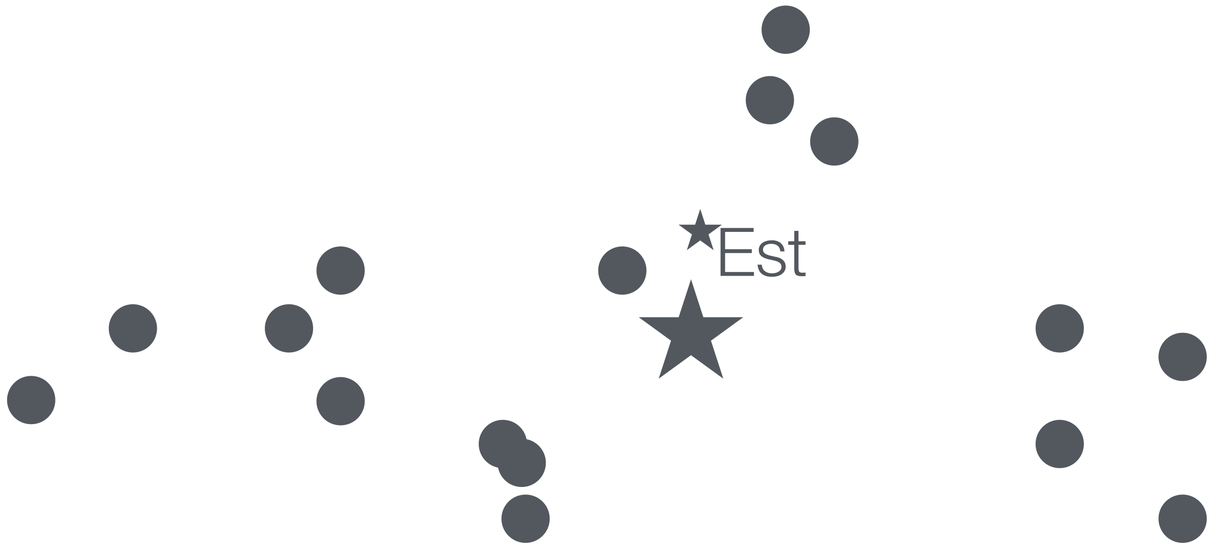
Nearest neighbour regression

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



Nearest neighbour regression

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

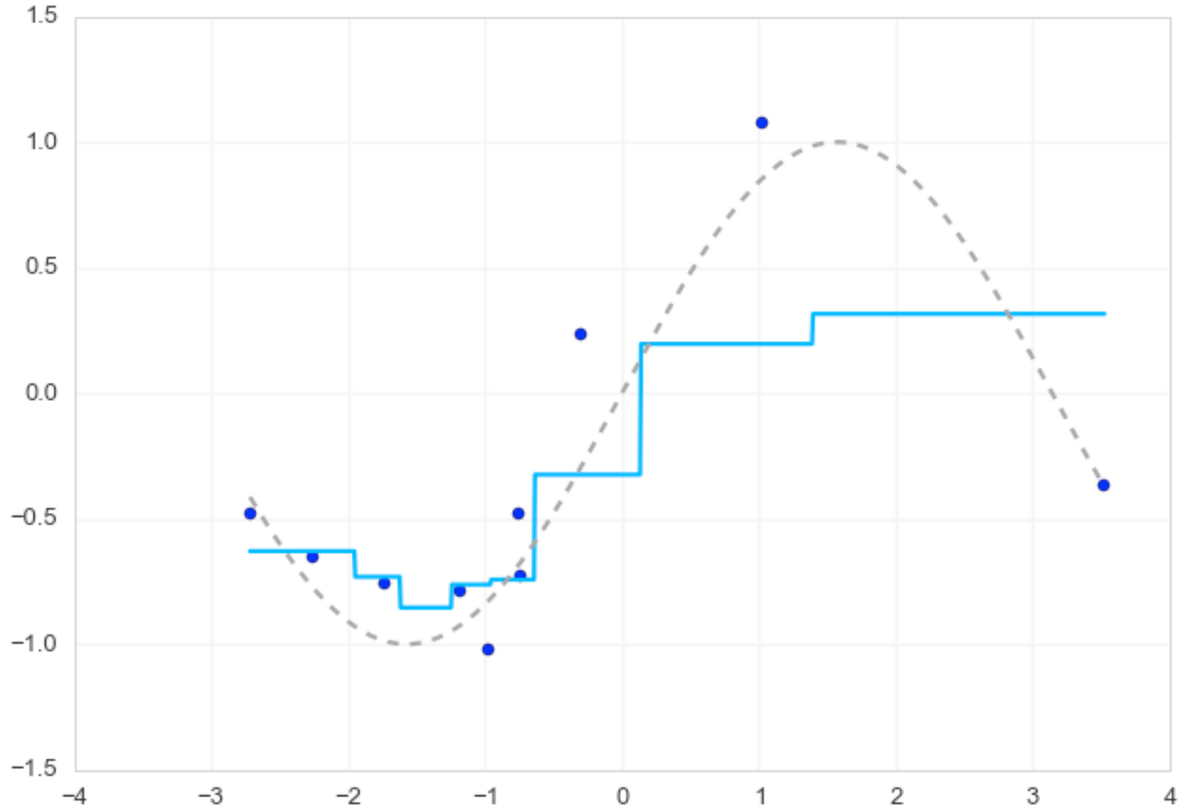


Nearest neighbour regression

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

```
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 $p(x)$

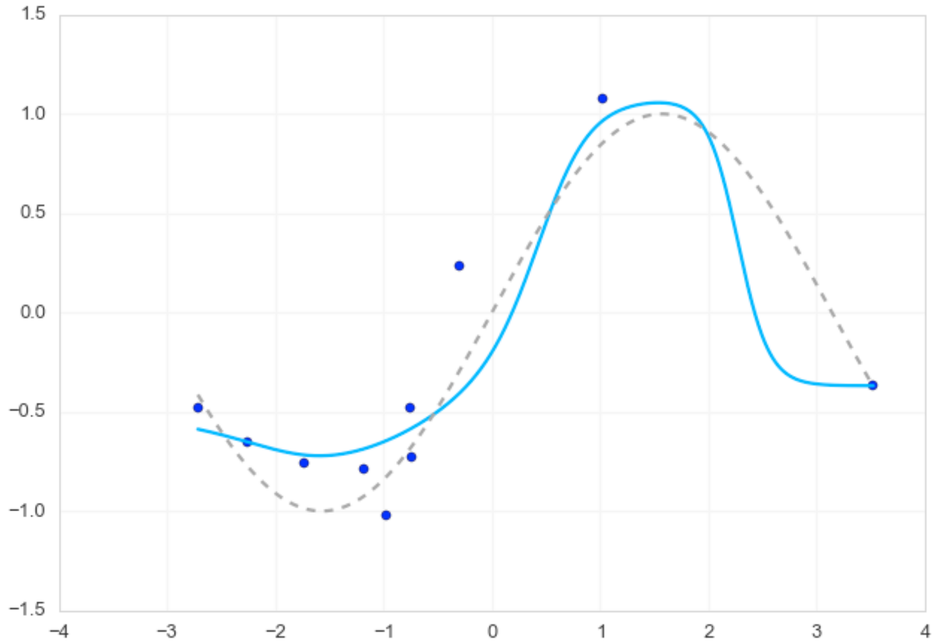
Kernel regression

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)
```

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:

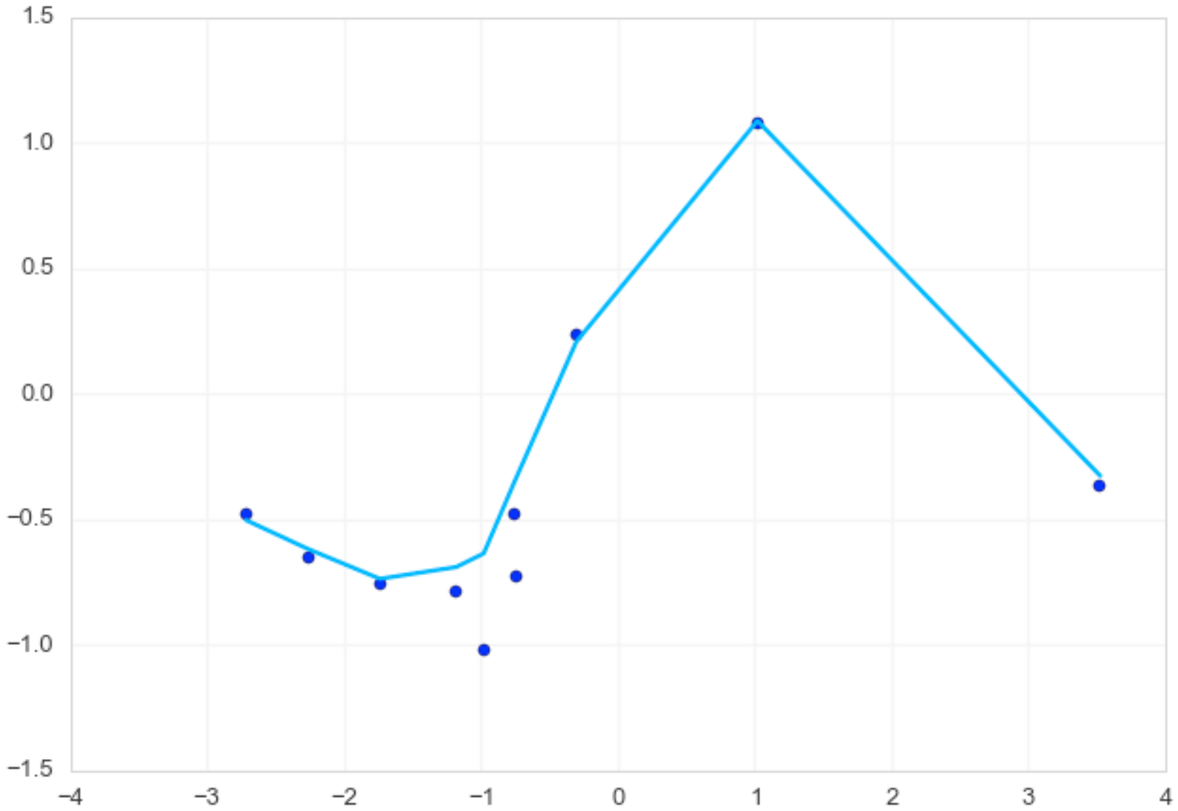
`statsmodels.nonparametric.smoothers_lowess.lowess`

`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])
```



Visualising data

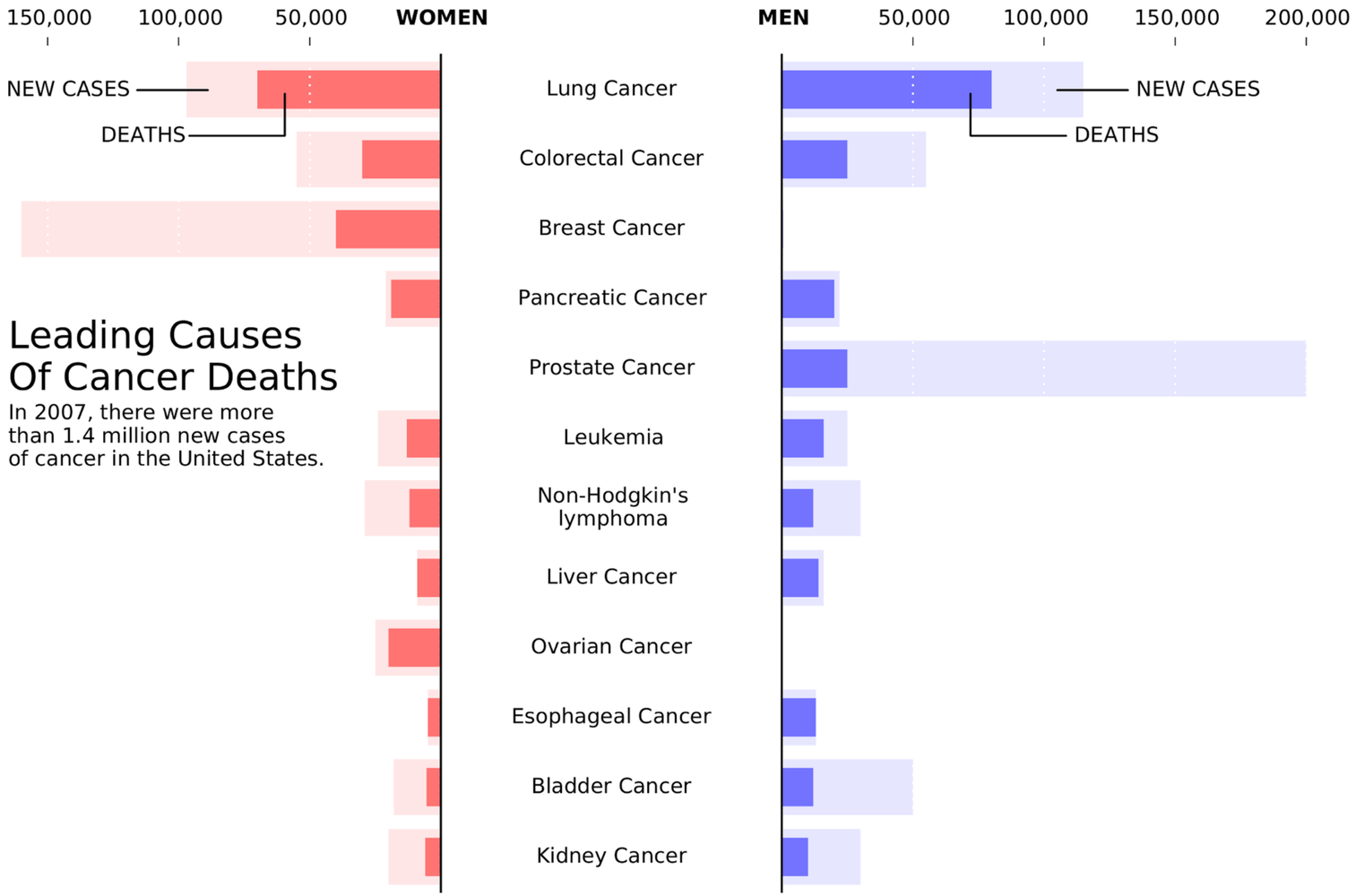
Making good scientific plots

- 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

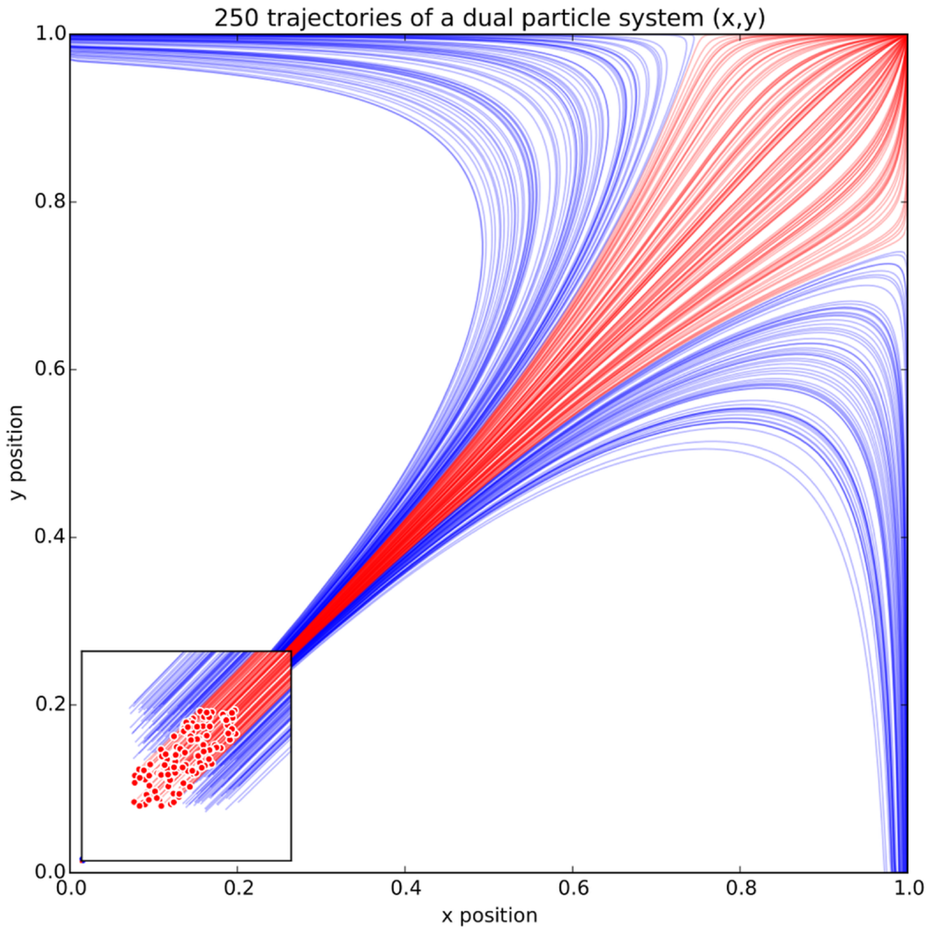
Adapting to the audience

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



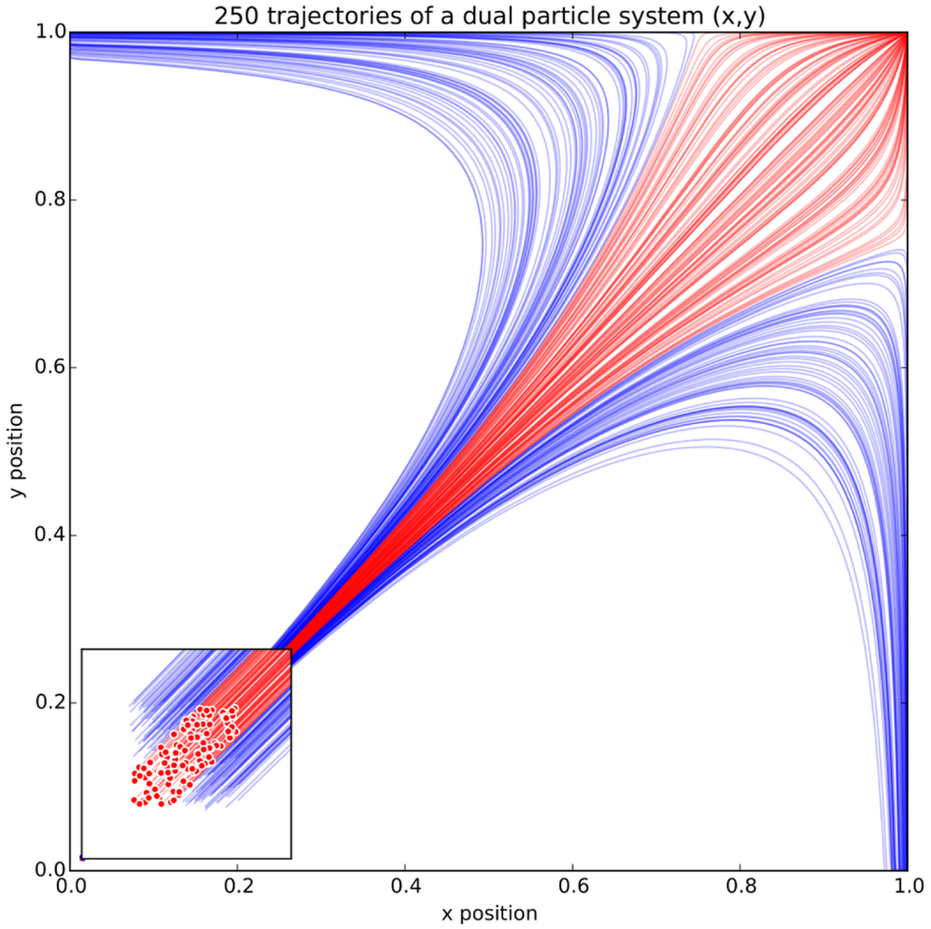
Adapting to the situation

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



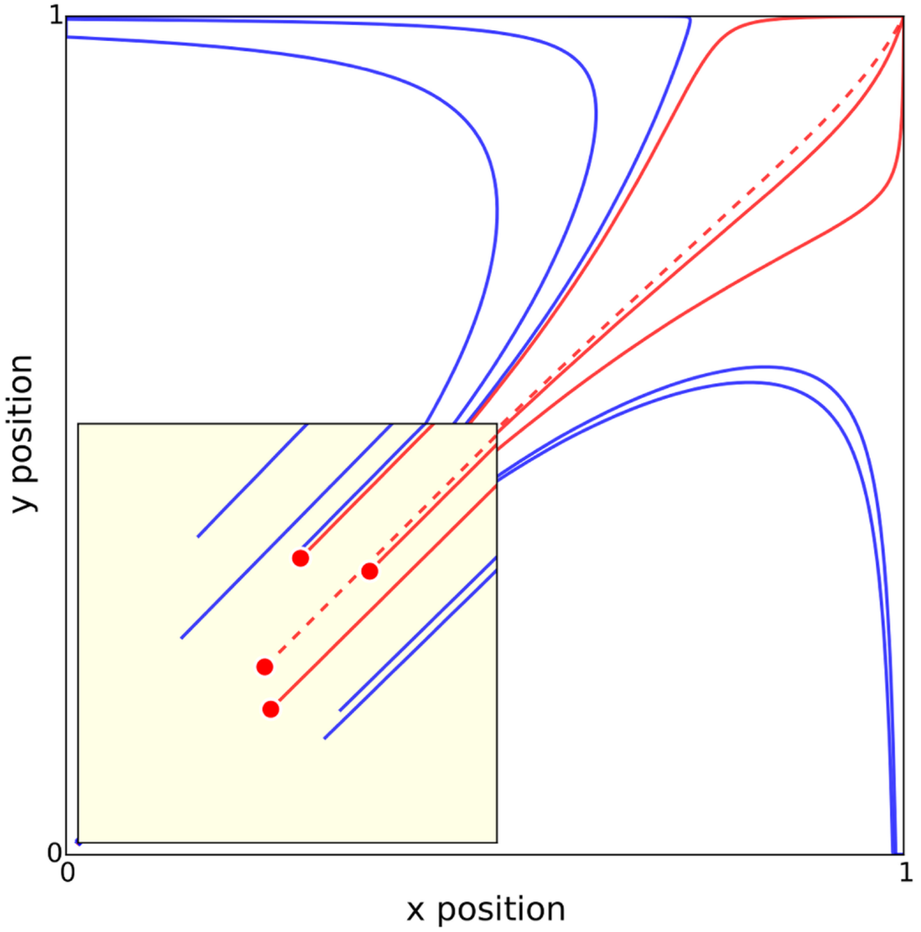
Scientific paper

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”

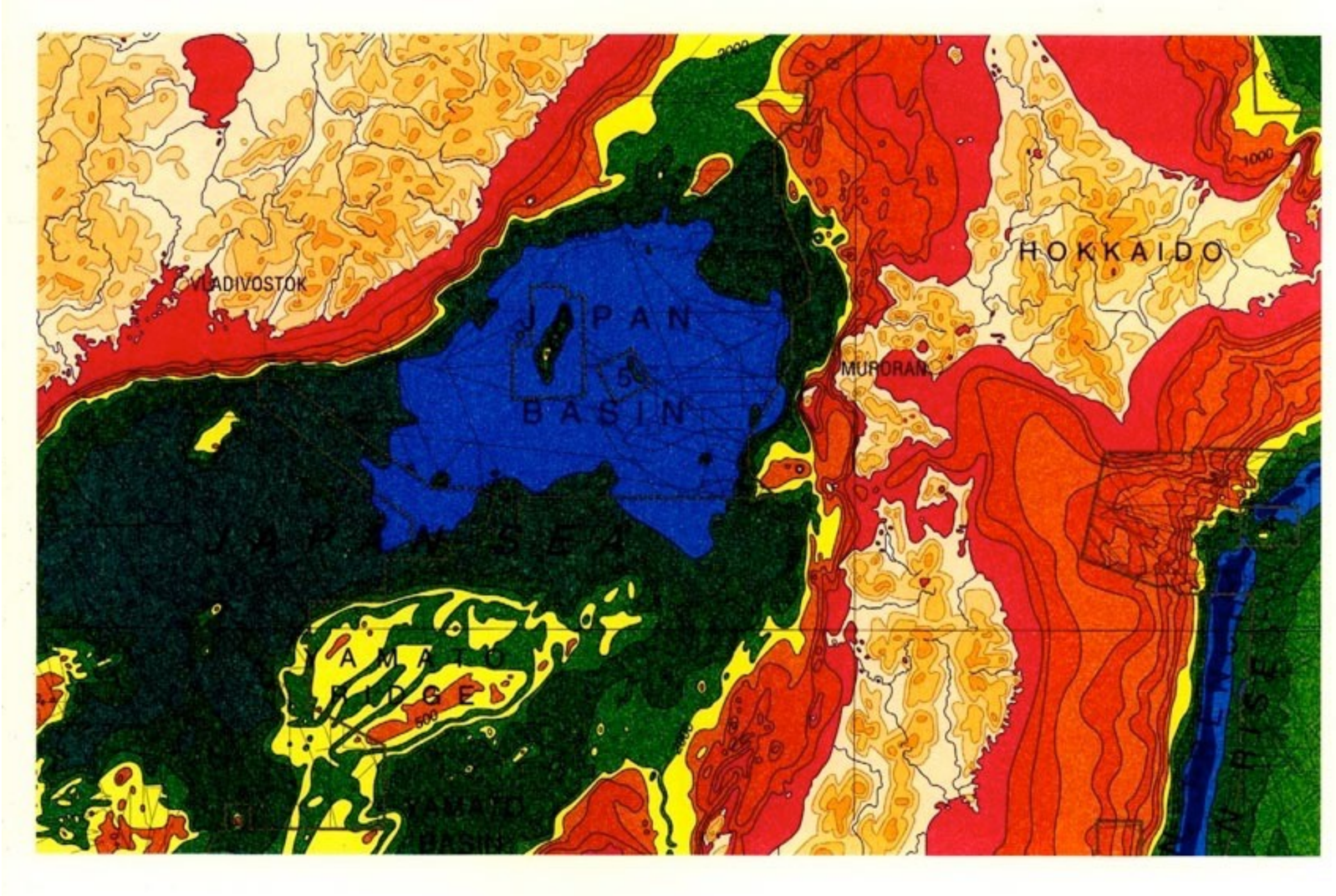
Colour choice

The “Smallest effective difference”

From Edward Tufte (1997), “Visual explanations”

Colour choice

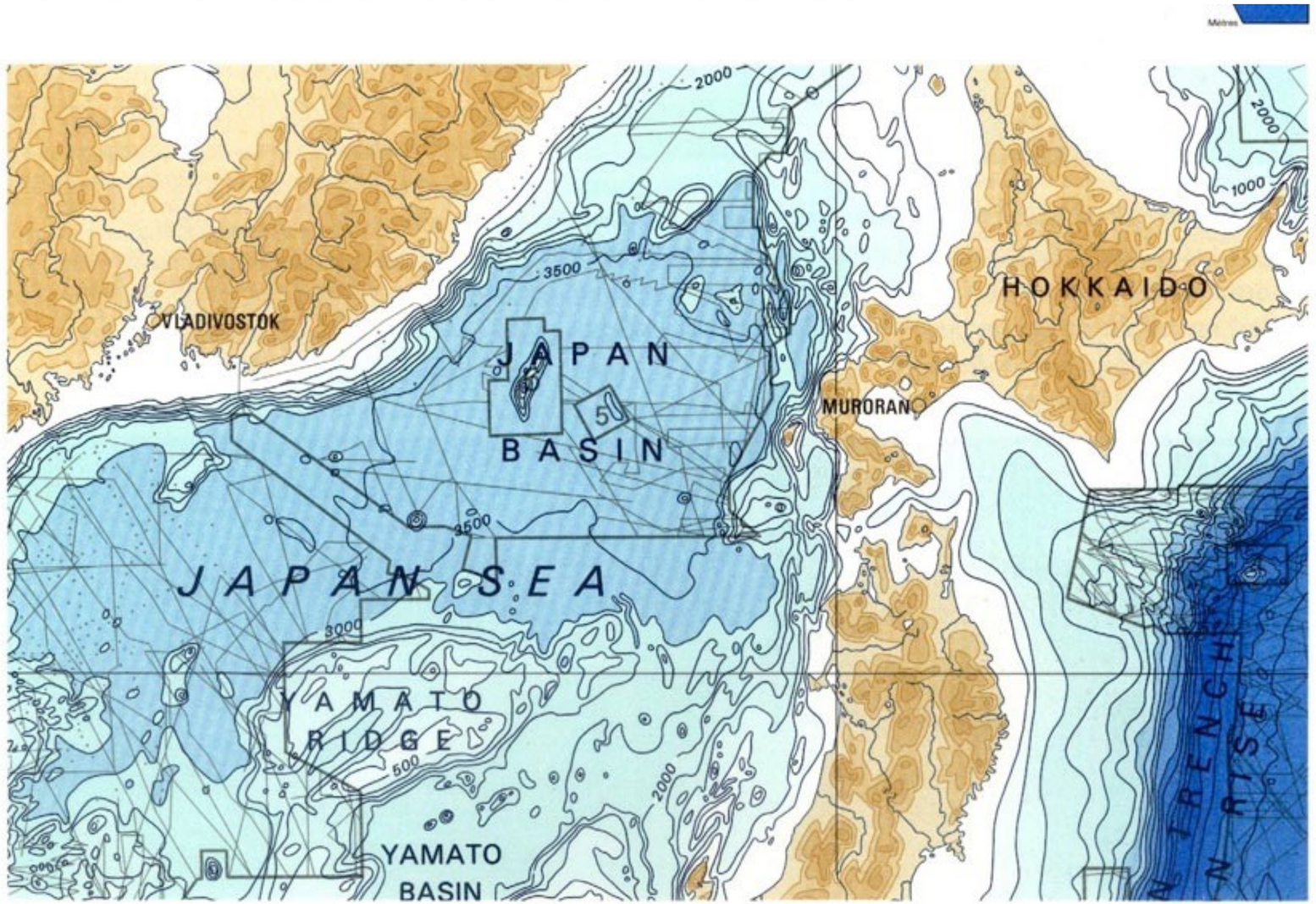
The “Smallest effective difference”



From Edward Tufte (1997), “Visual explanations”

Colour choice

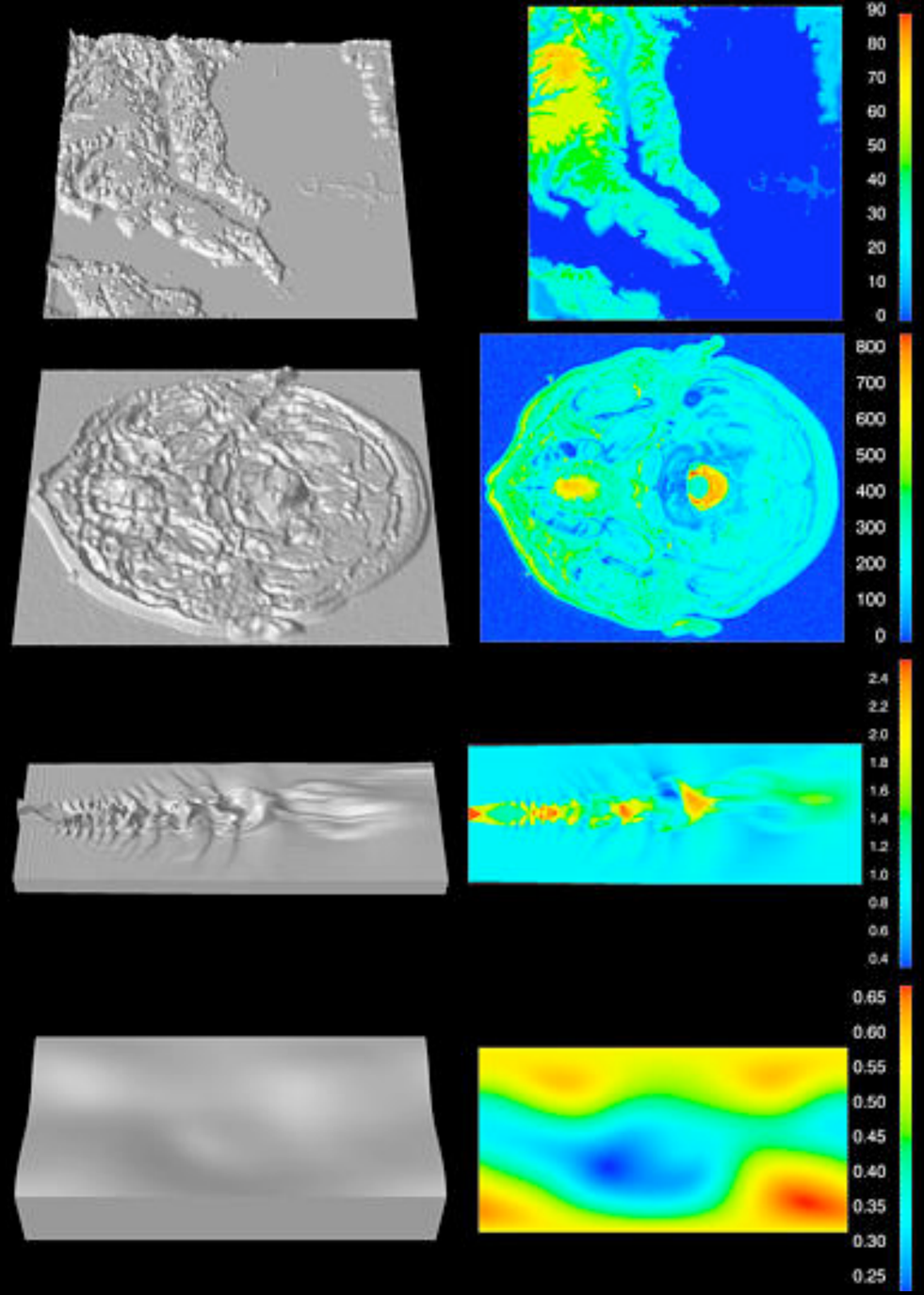
The “Smallest effective difference”



From Edward Tufte (1997), “Visual explanations”

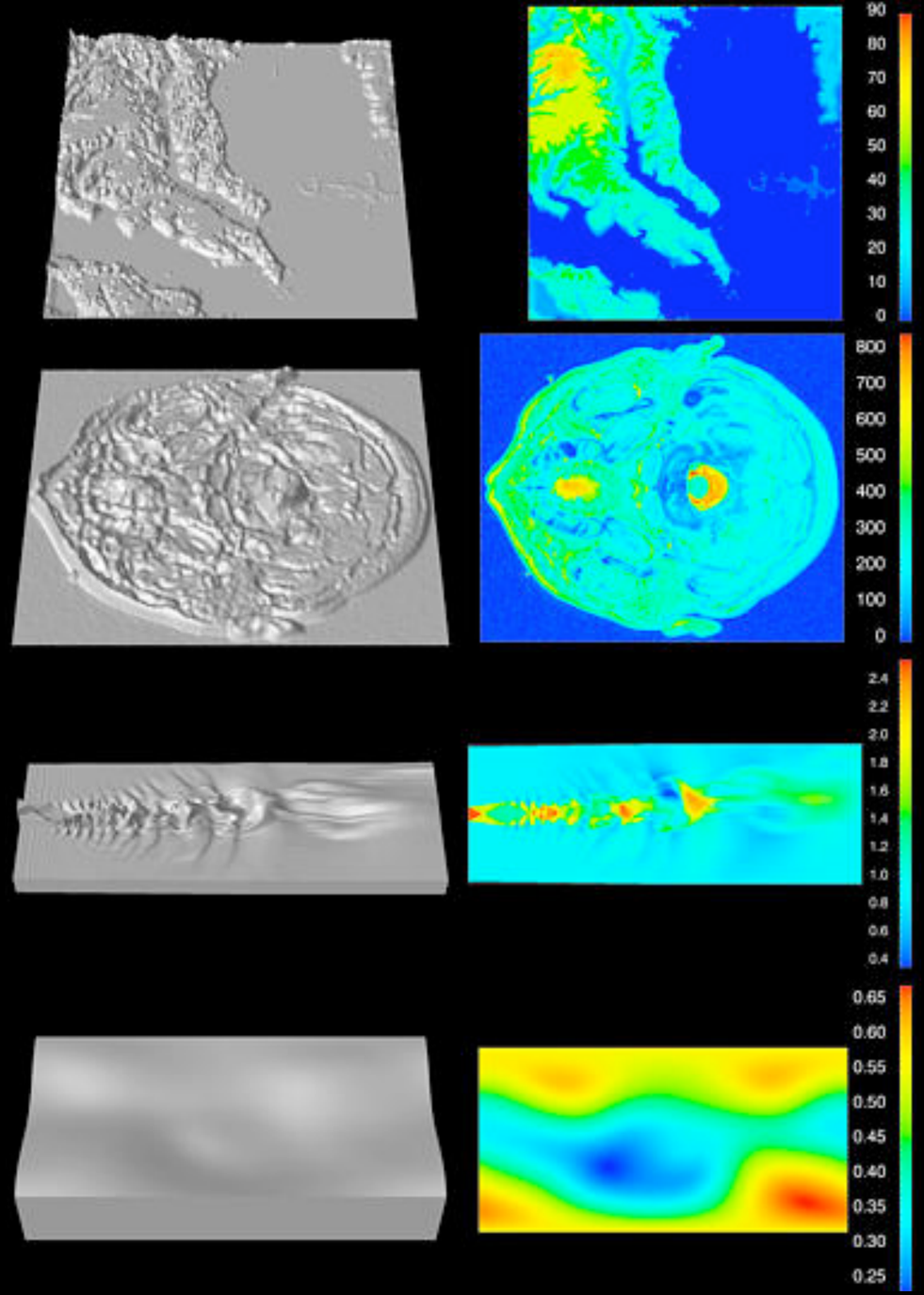
The issue of colour & 2D plots

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

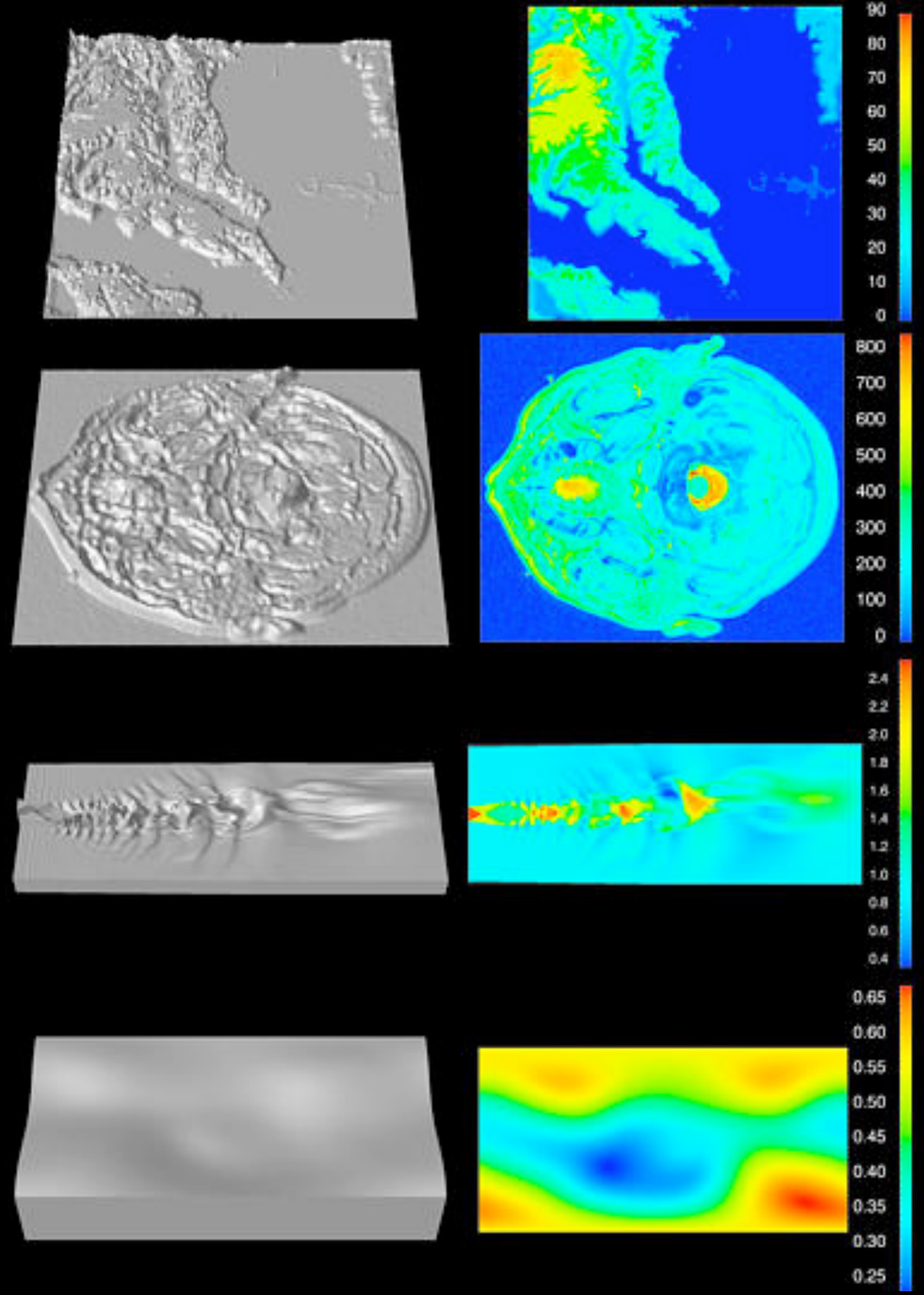
For more details: http://www.research.ibm.com/people/l/lloydt/color/color.HTM



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

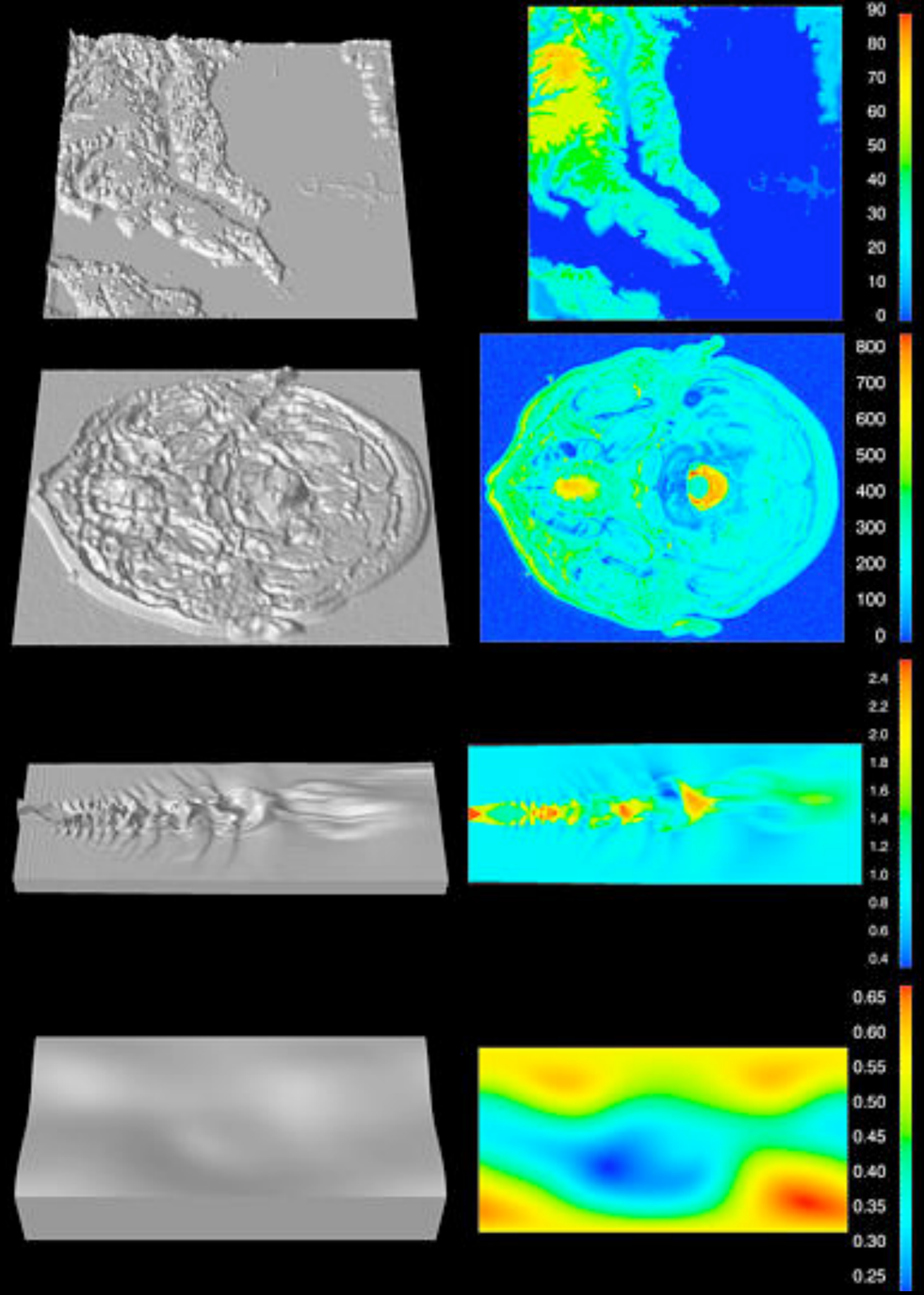


Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

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



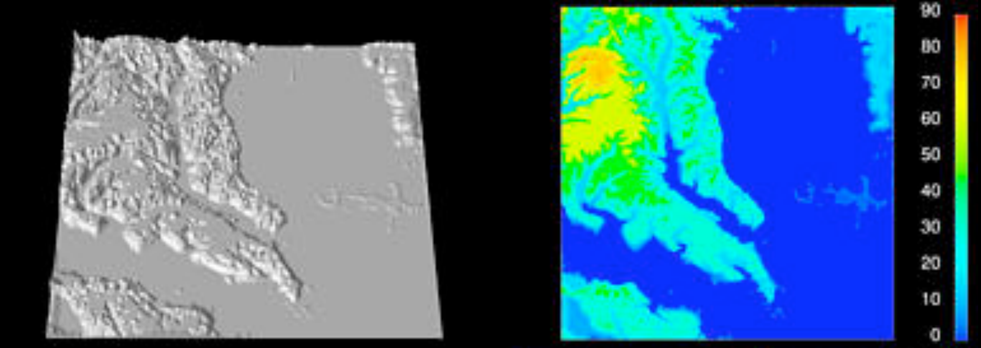
Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

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

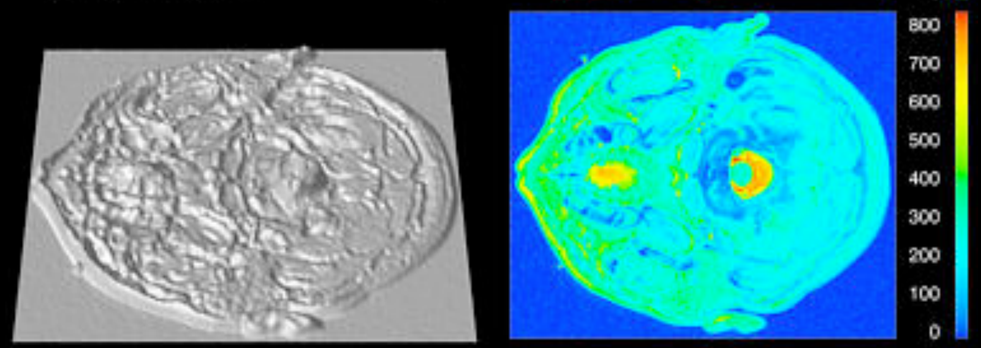
Bad Habits

Turbulent flow from a jet engine.

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

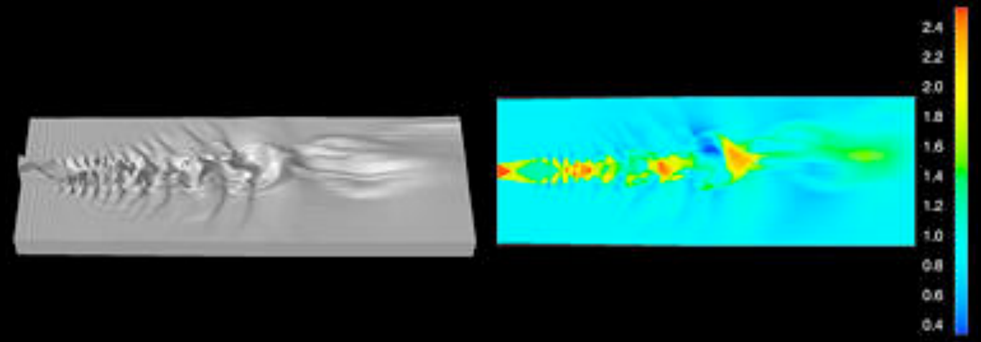


Chesapeake Bay - note the artificial structure at higher altitude when it is actually quite gradual

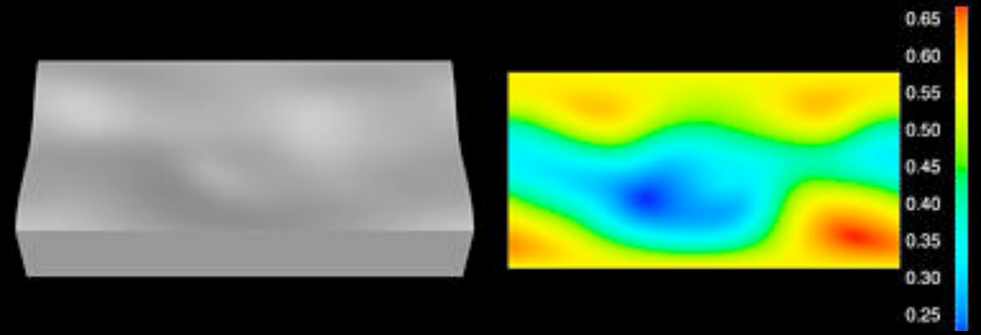


An slice of an MRI scan of a human brain. Washing out of detail and artificial structure.

Bad Habits

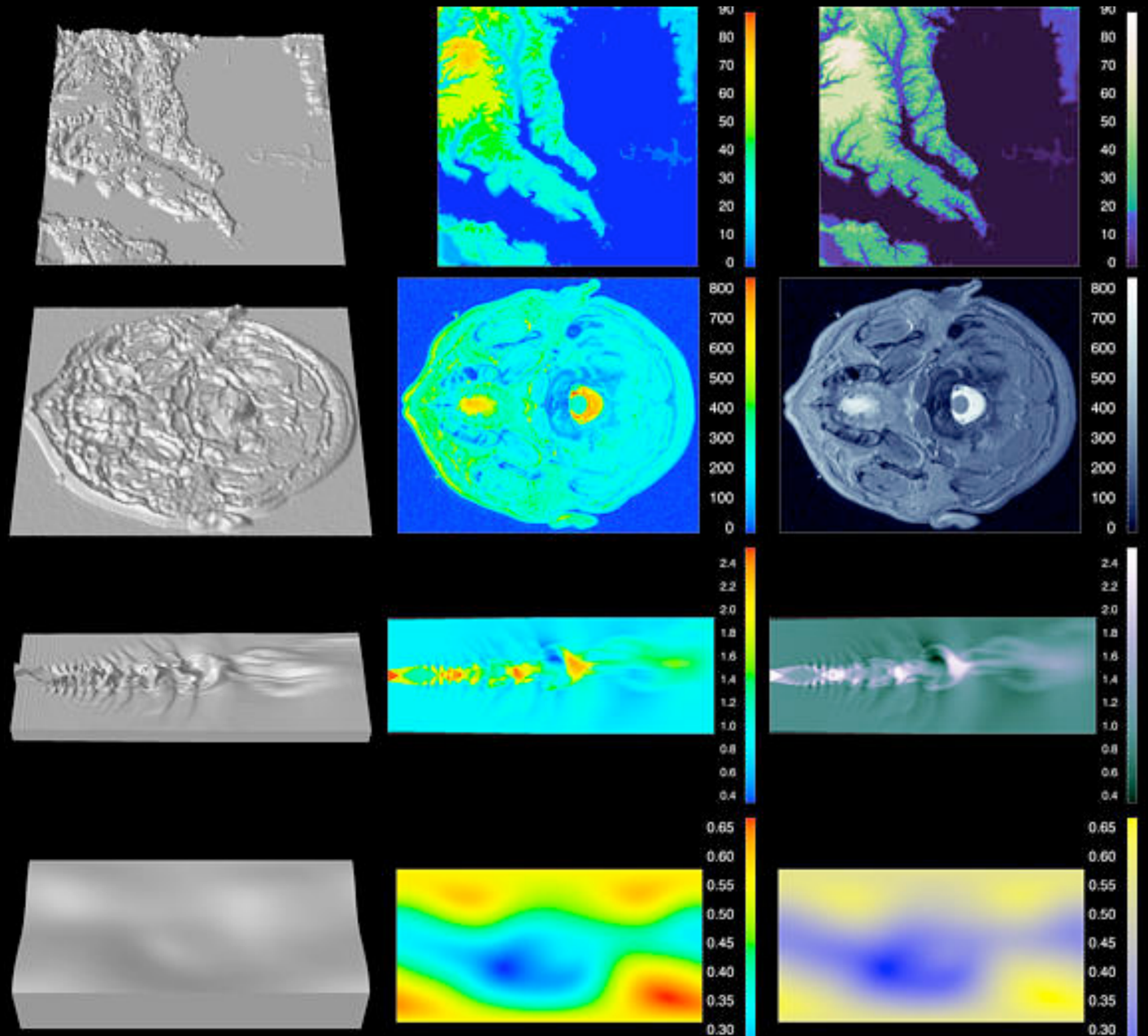


Turbulent flow from a jet engine.



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

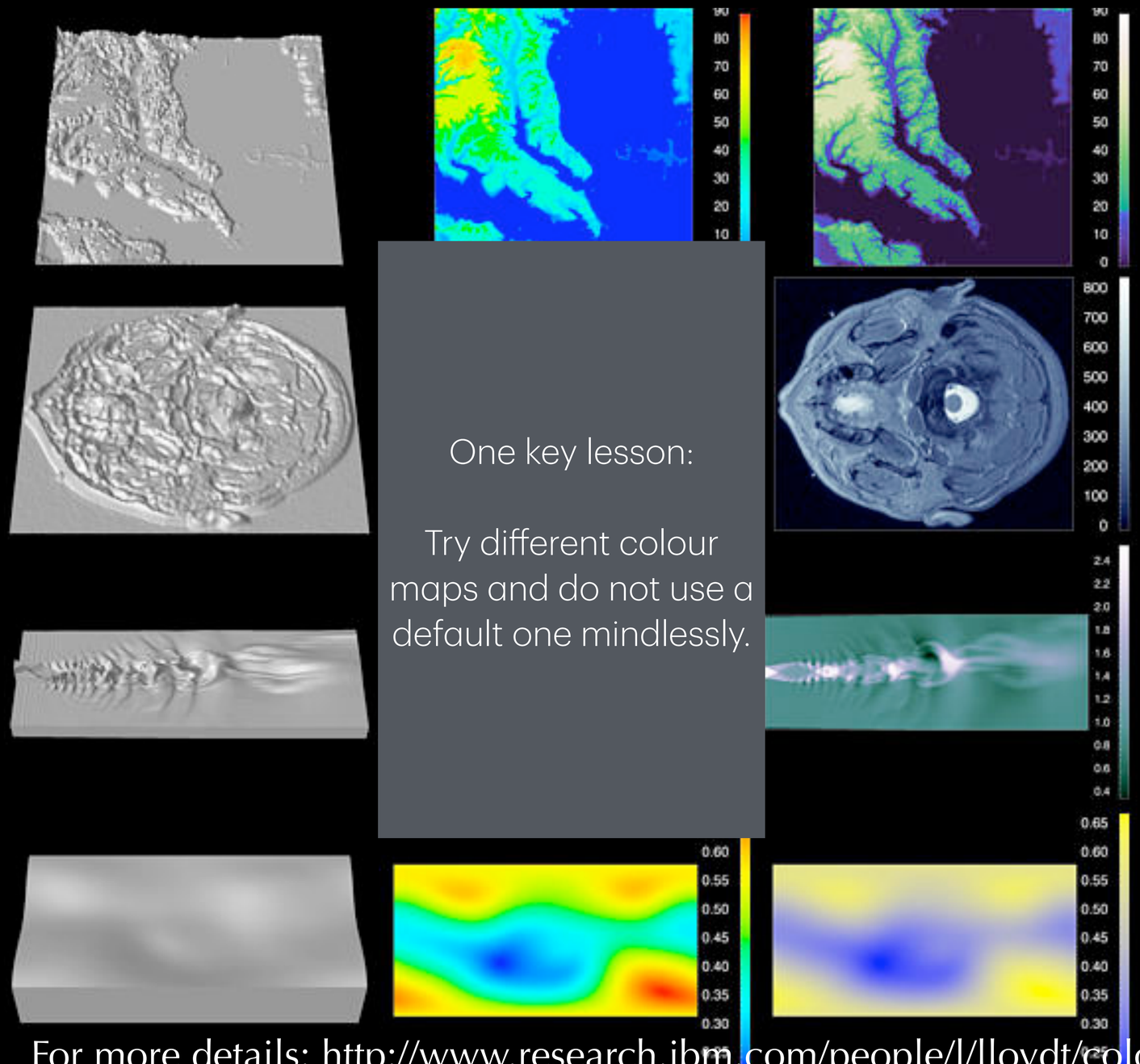


A natural zero

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:

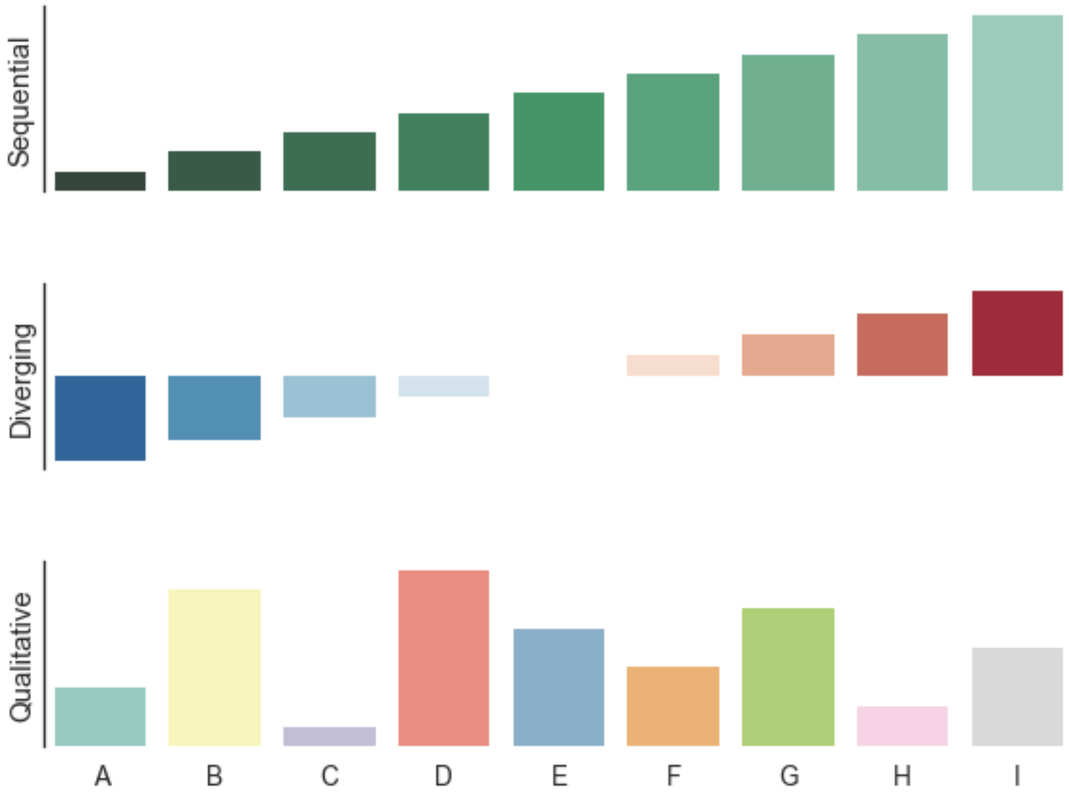
Try different colour maps and do not use a default one mindlessly.

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')
```

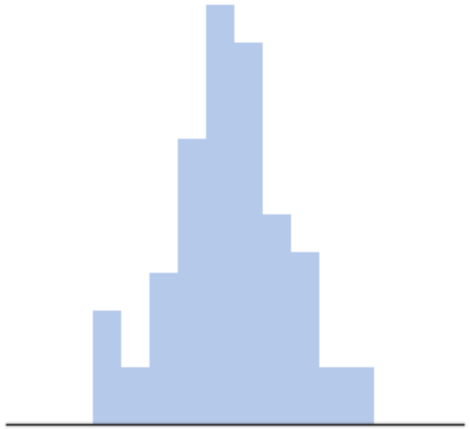
Paul Tol has a very useful page:

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
```



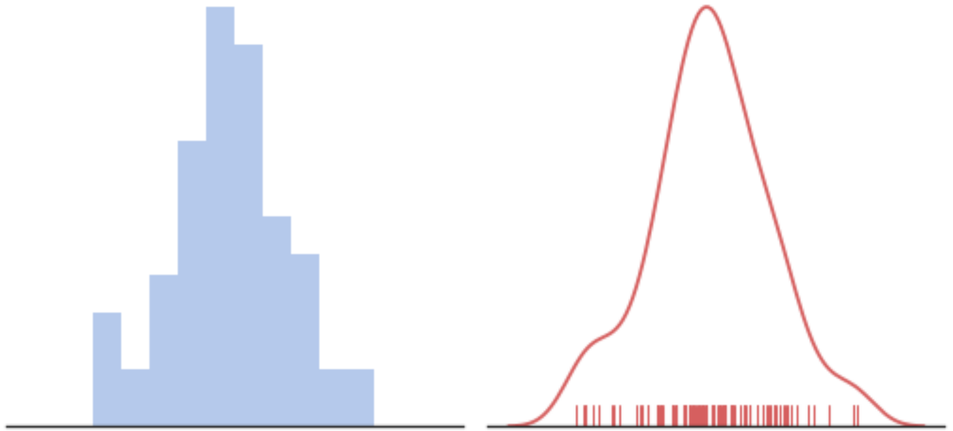
```
sns.distplot(x, kde=False)

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
```



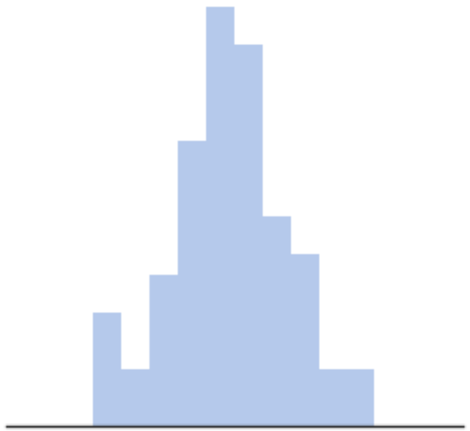
```
sns.distplot(x, kde=True,
rug=True)
```

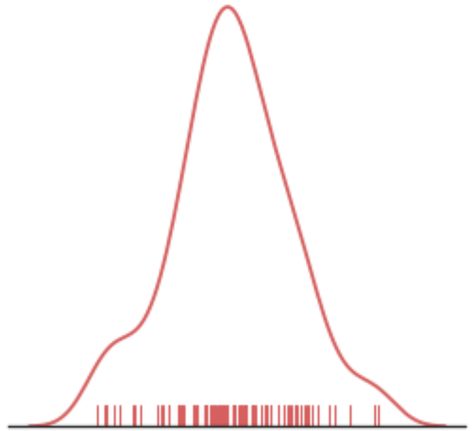
Histograms are convenient but bin the data and are not differentiable - we'll look at kernel density plots later. Rug plots complement histograms and kernel density plots by showing the data.

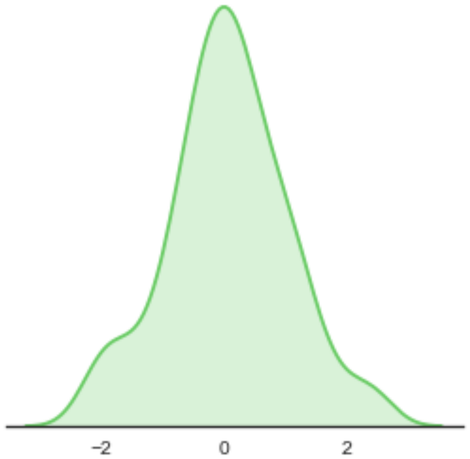
http://seaborn.pydata.org/examples/distplot_options.html

Various ways to show 1D distributions

```
import matplotlib.pyplot as plt

import seaborn as sns
```







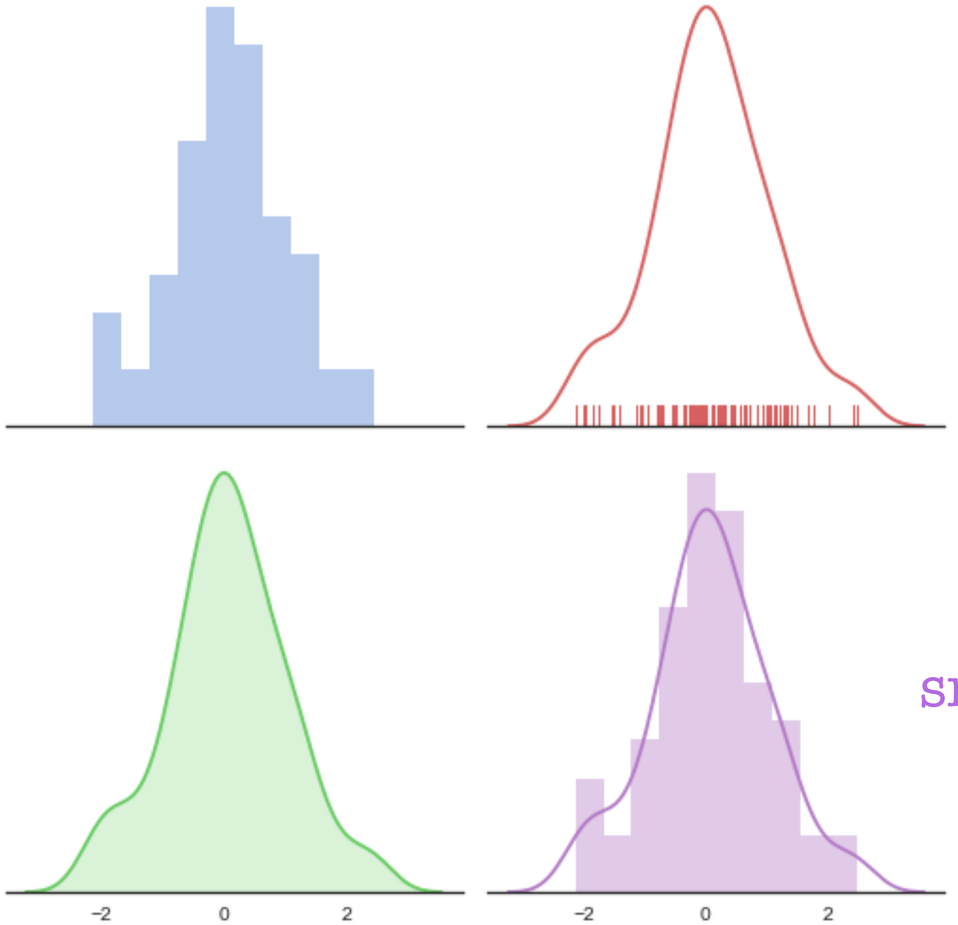

```
sns.distplot(x, hist=False)
```

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
```



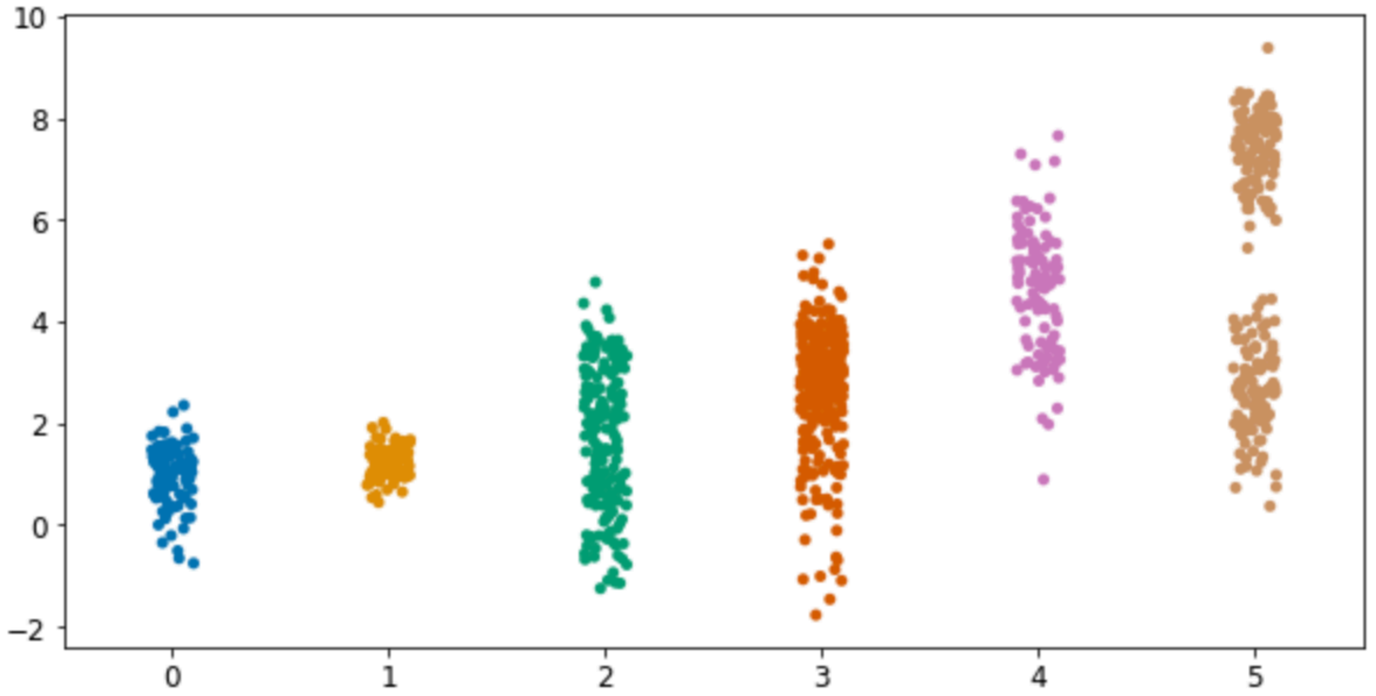

```
sns.distplot(x)
```

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

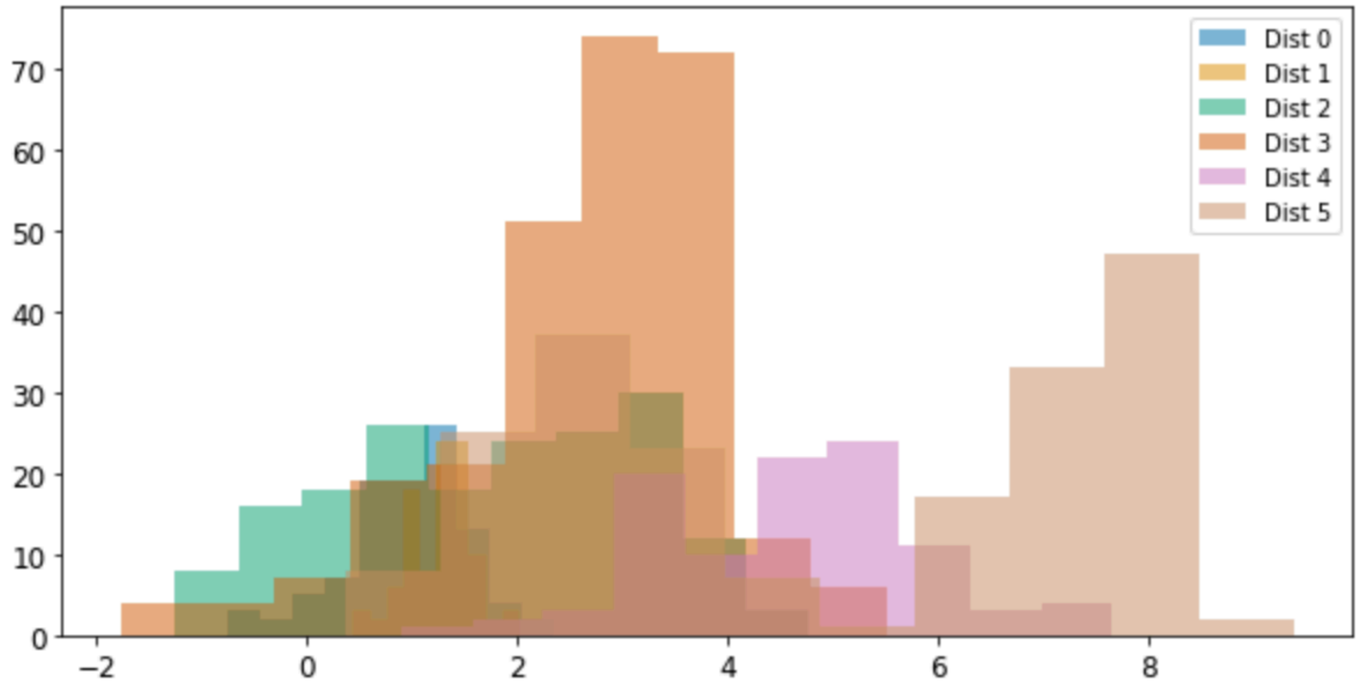
Scenario: 6 classes with measurements - how do we compare them?



See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

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

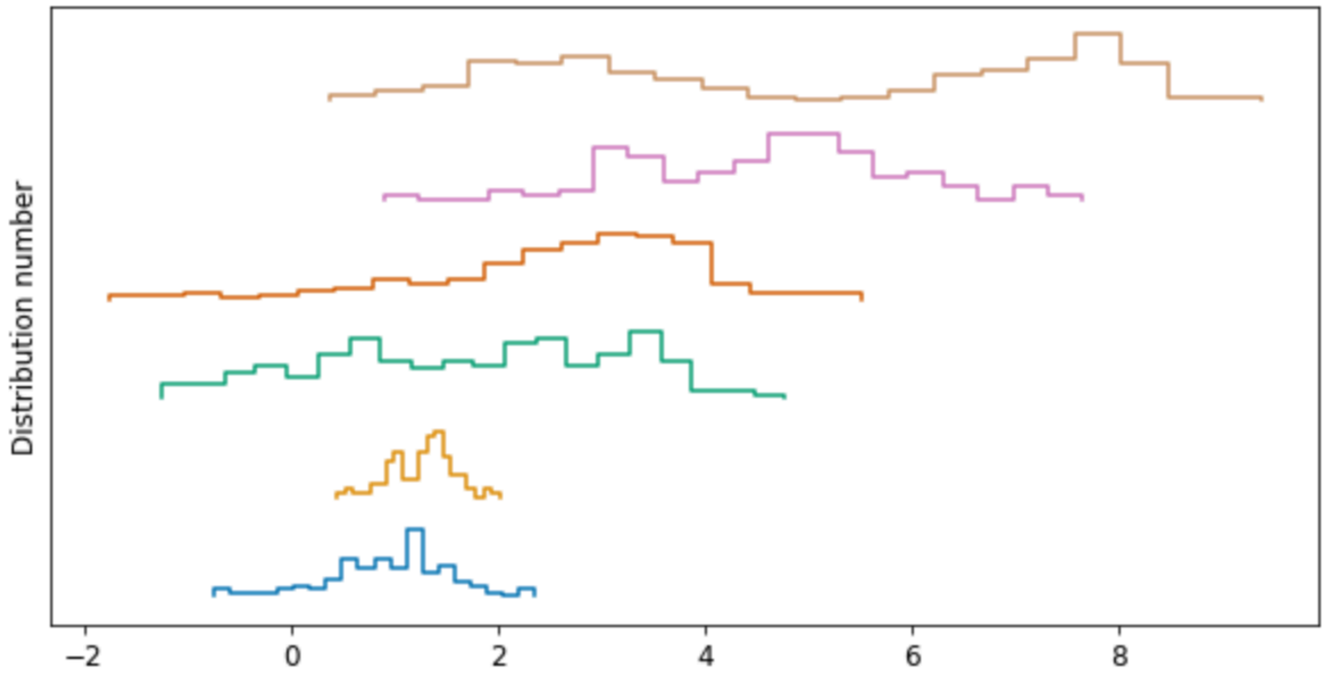


over-plotting histograms does not work well

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

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

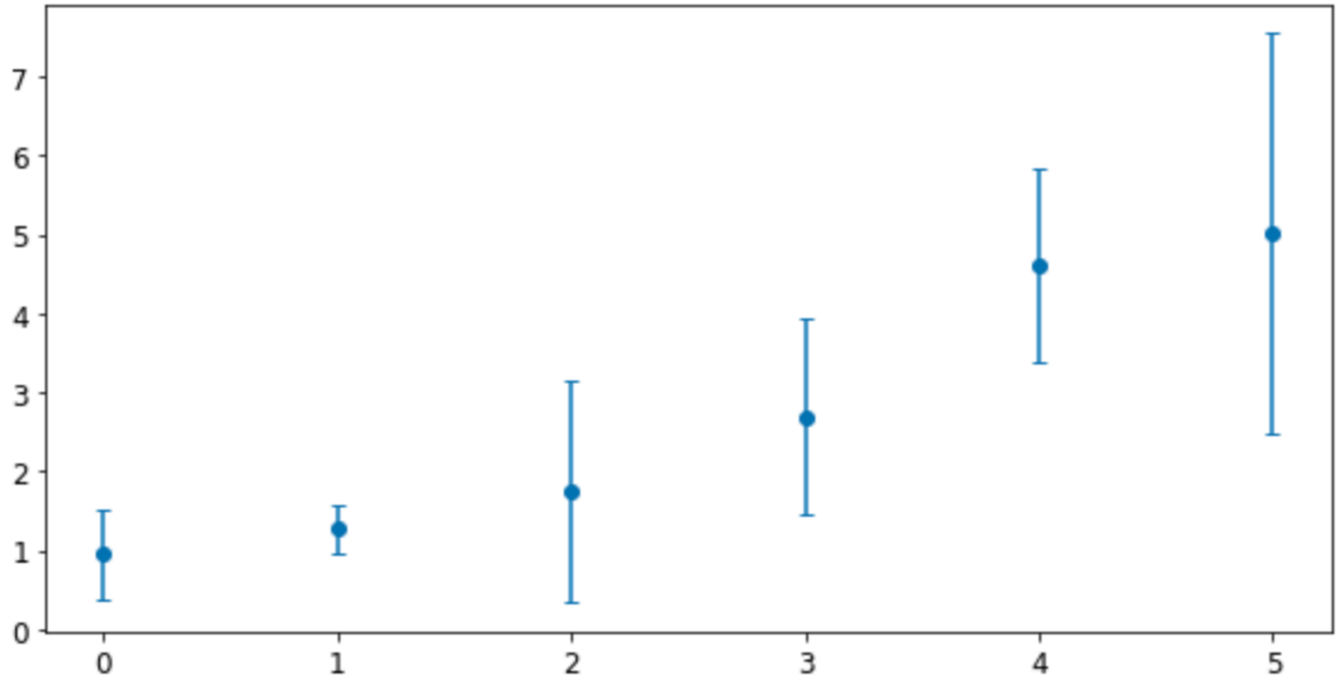


offsetting histograms is a possibility

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

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

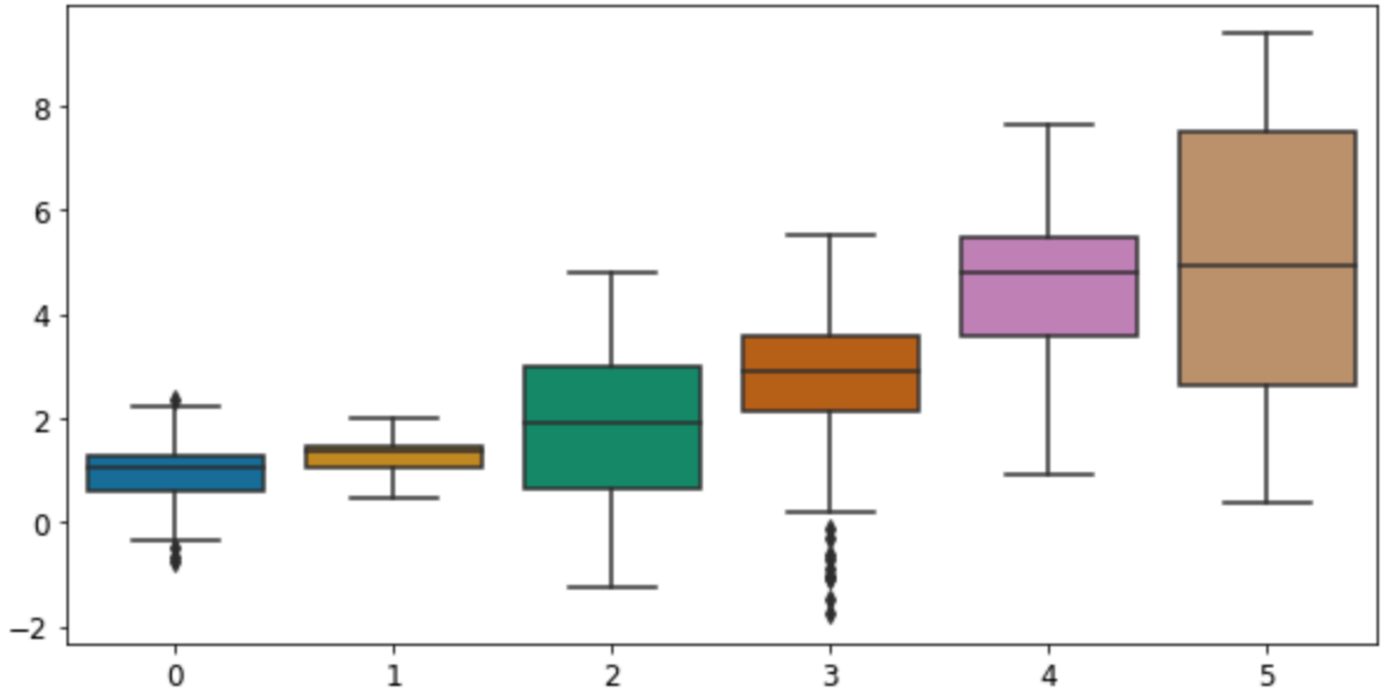


errorbars give some illustration of spread but lacks detail

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

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

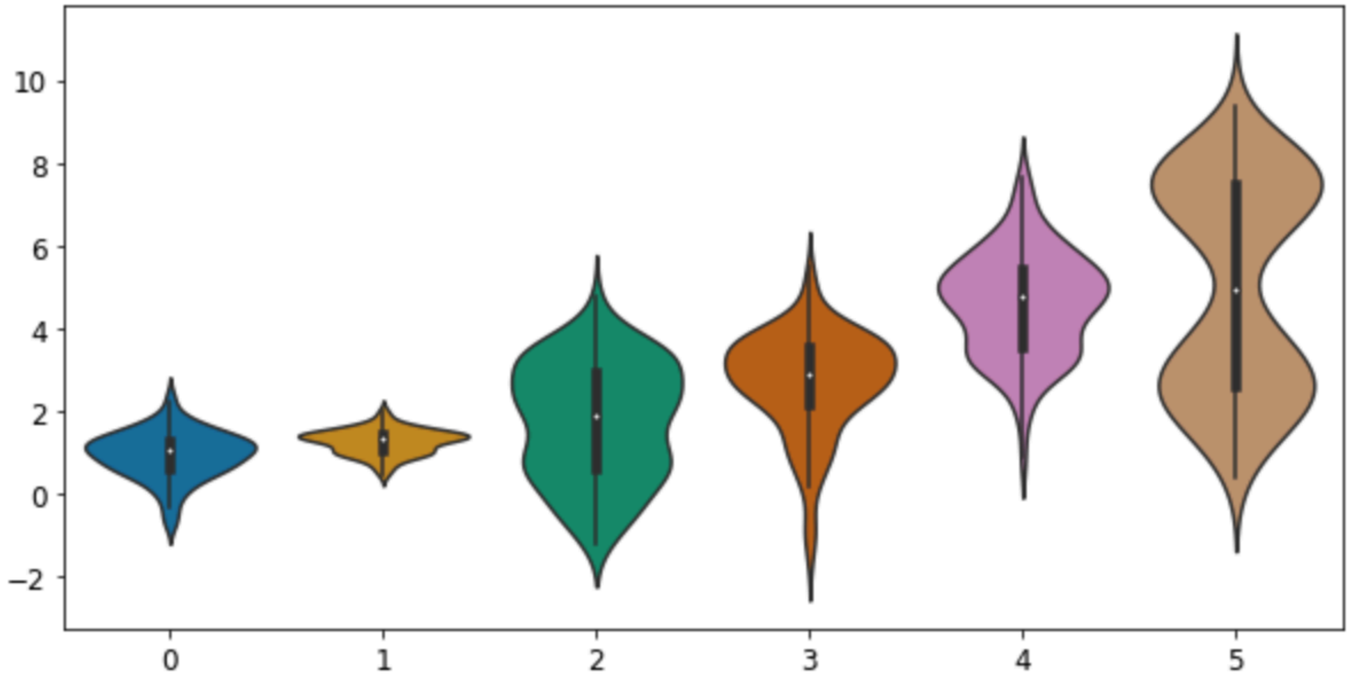


box plots are better - but only for uni-modal data

See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions

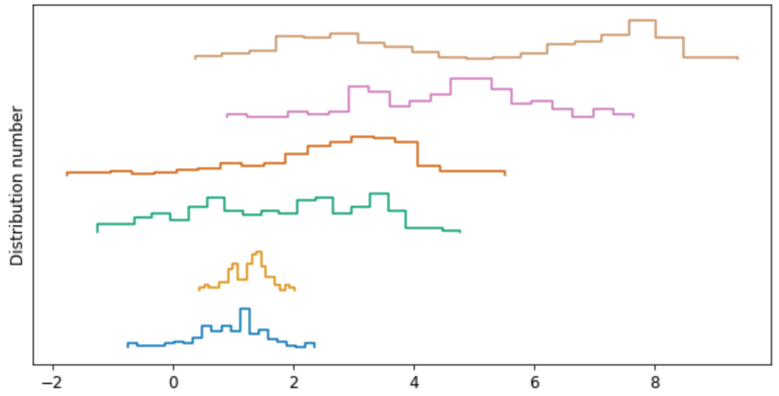
Scenario: 6 classes with measurements - how do we compare them?

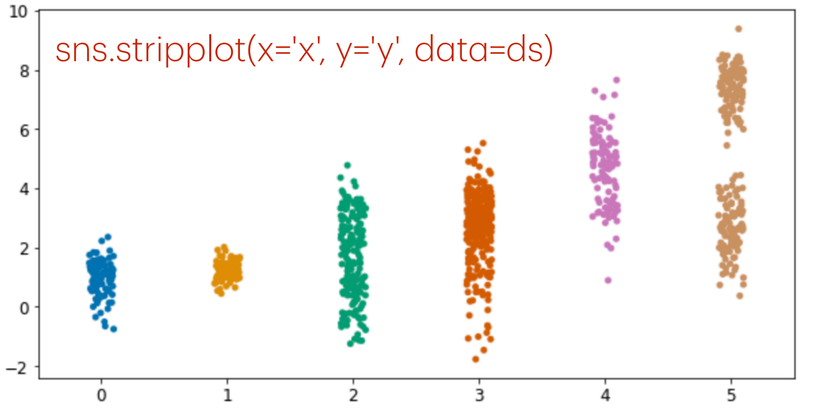


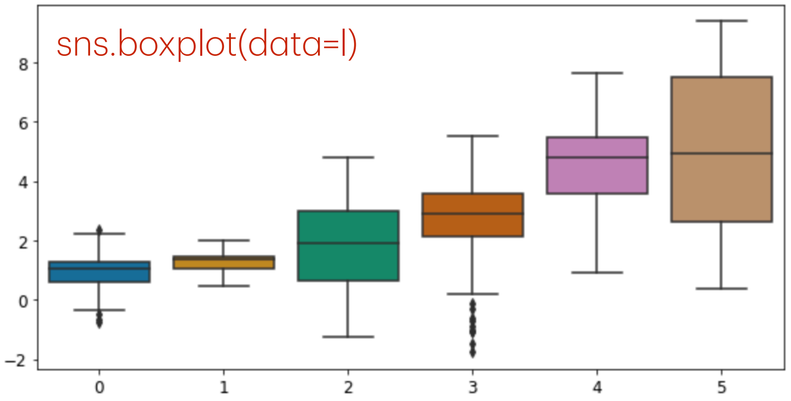
violin plots can be useful for this

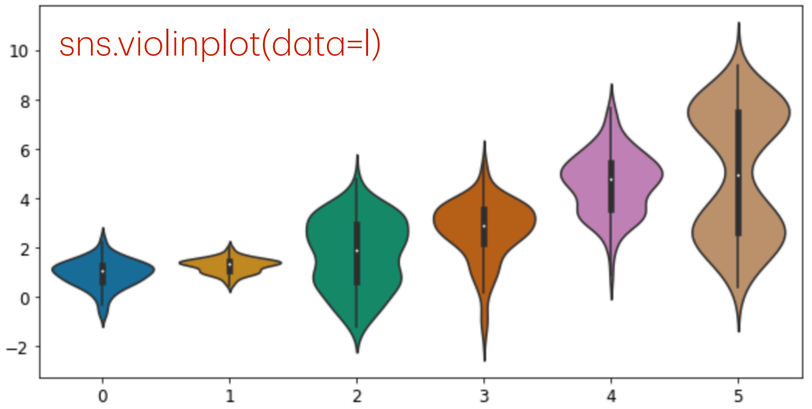
See the Distribution Illustration notebook in the Lecture directory

Comparing multiple 1D distributions



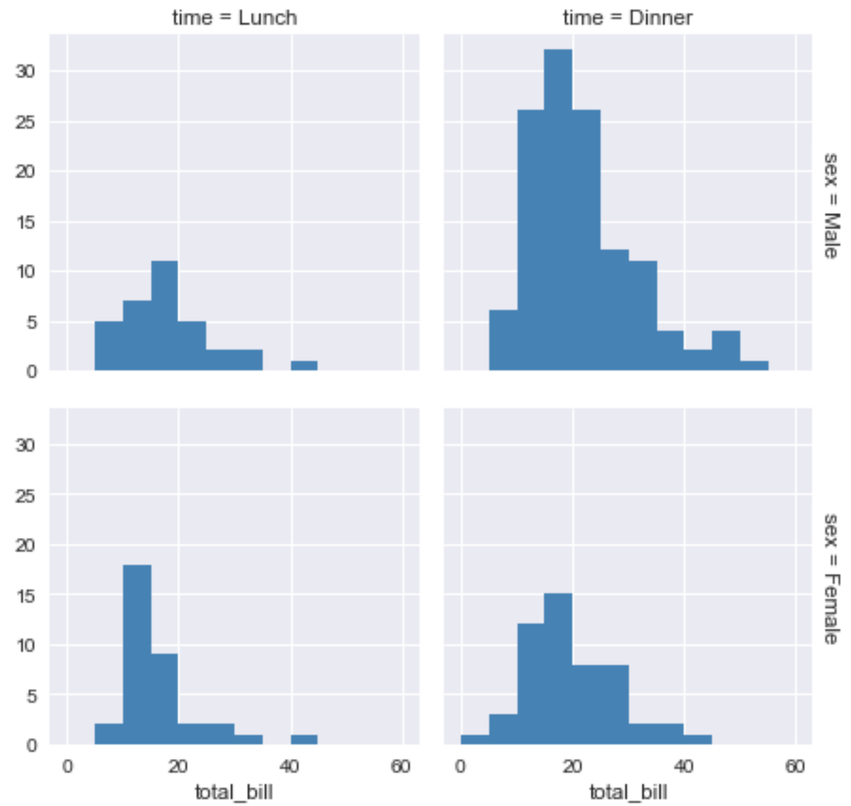






See the Distribution Illustration notebook in the Lecture directory

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

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

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

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

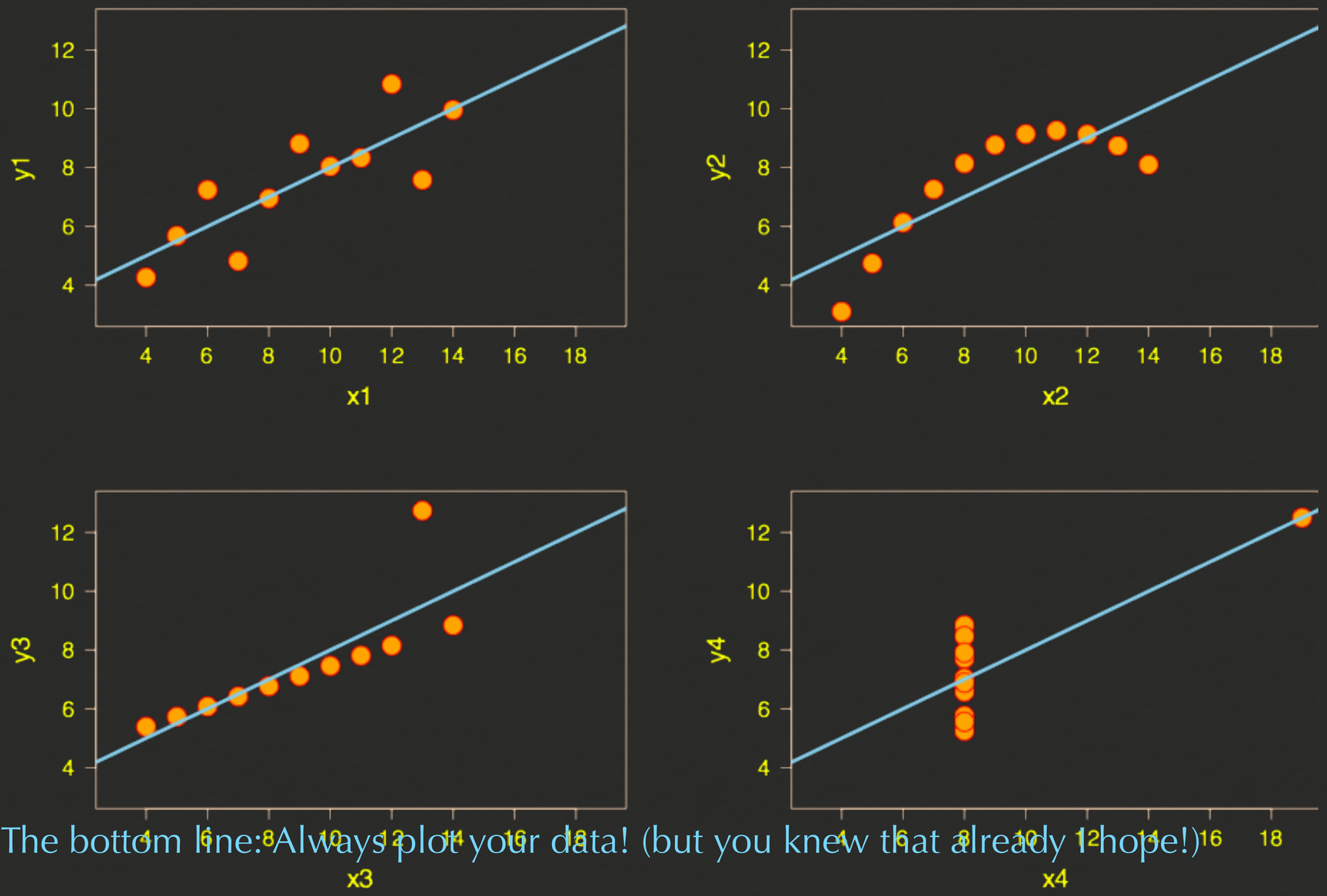
mean(x) = **9**, mean(y) = **7.5**

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

The correlation coefficient: **0.816**

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

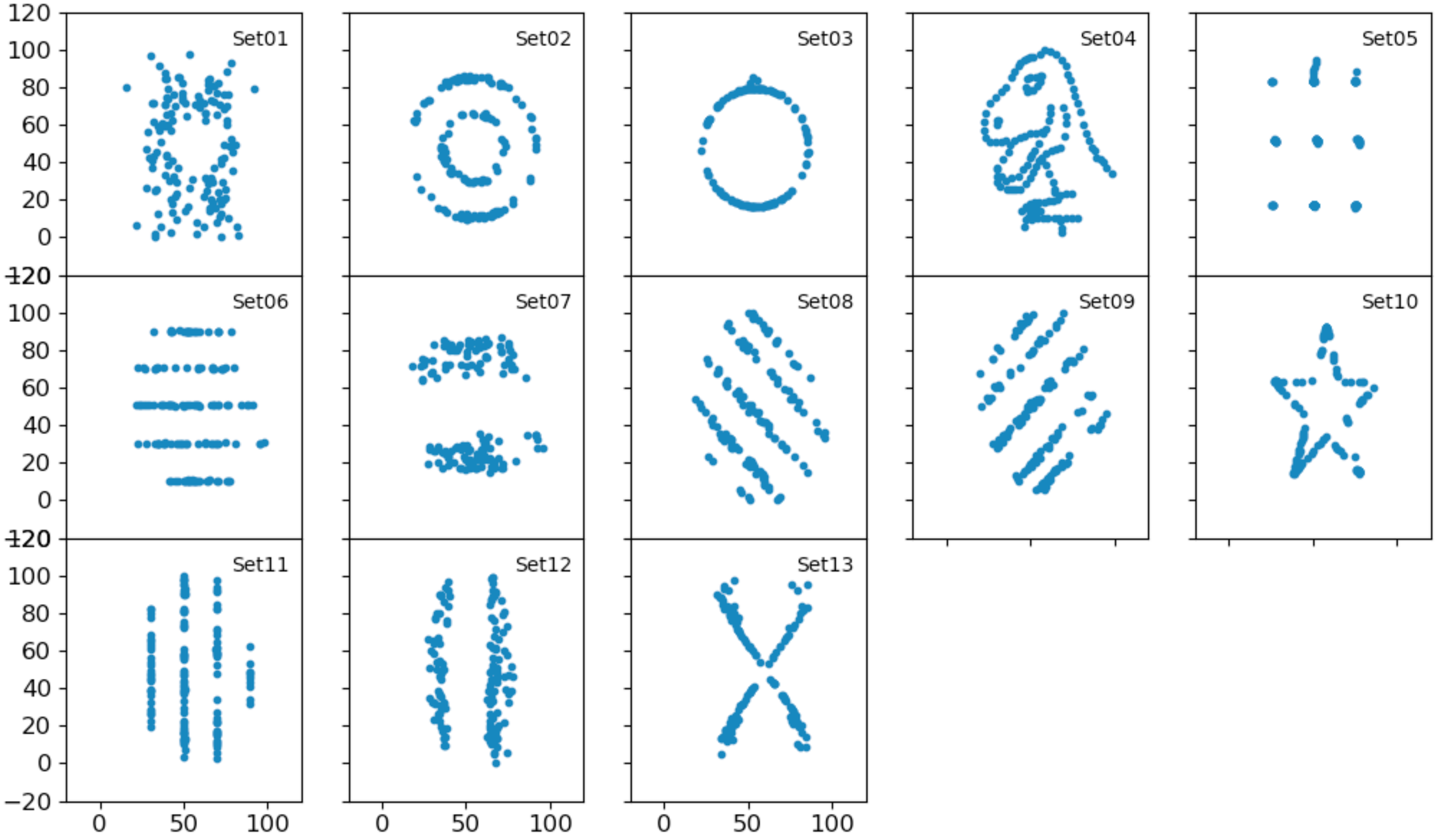
So they seem to be very similar!



The bottom line: Always plot your data! (but you knew that already I hope!)

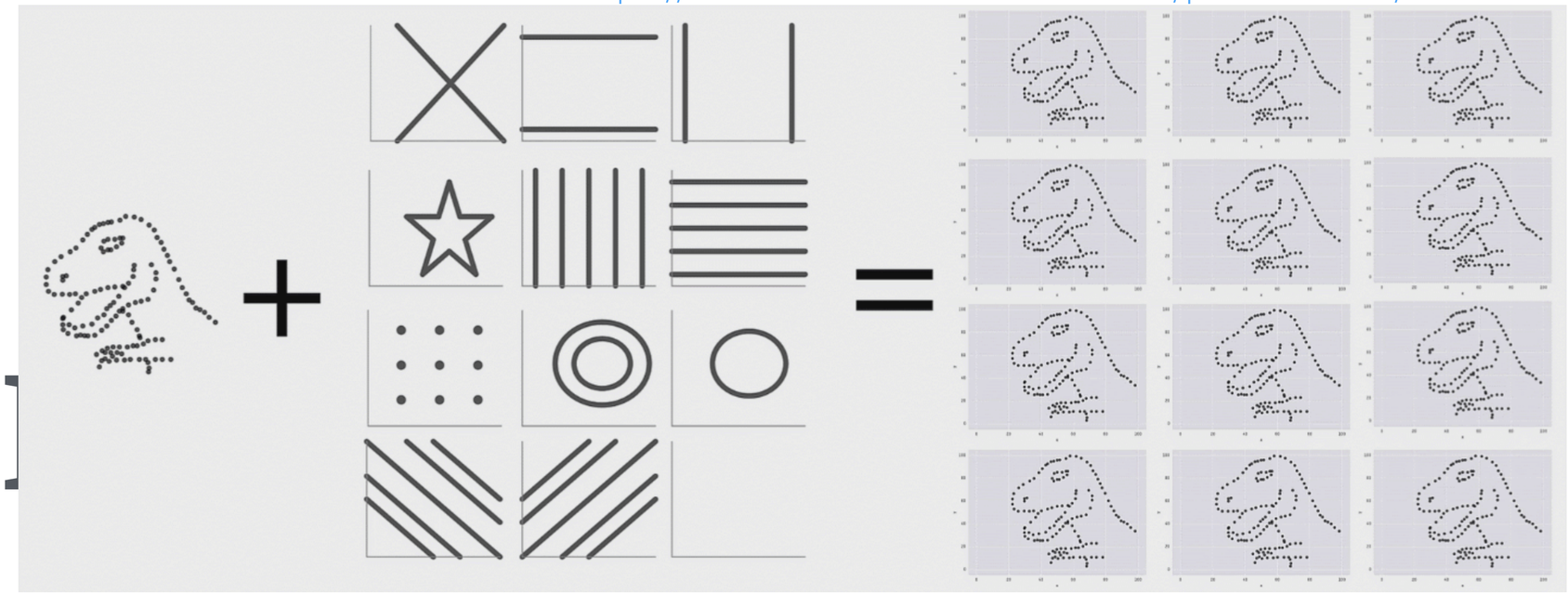
But how would you deal with this in N dimensions?

Vis


Set01
Set02
Set03
Set04
Set05
Set06
Set07
Set08
Set09
Set10
Set11
Set12
Set13
120
100
80
60
40
20
0
−20
0
50
100

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

But

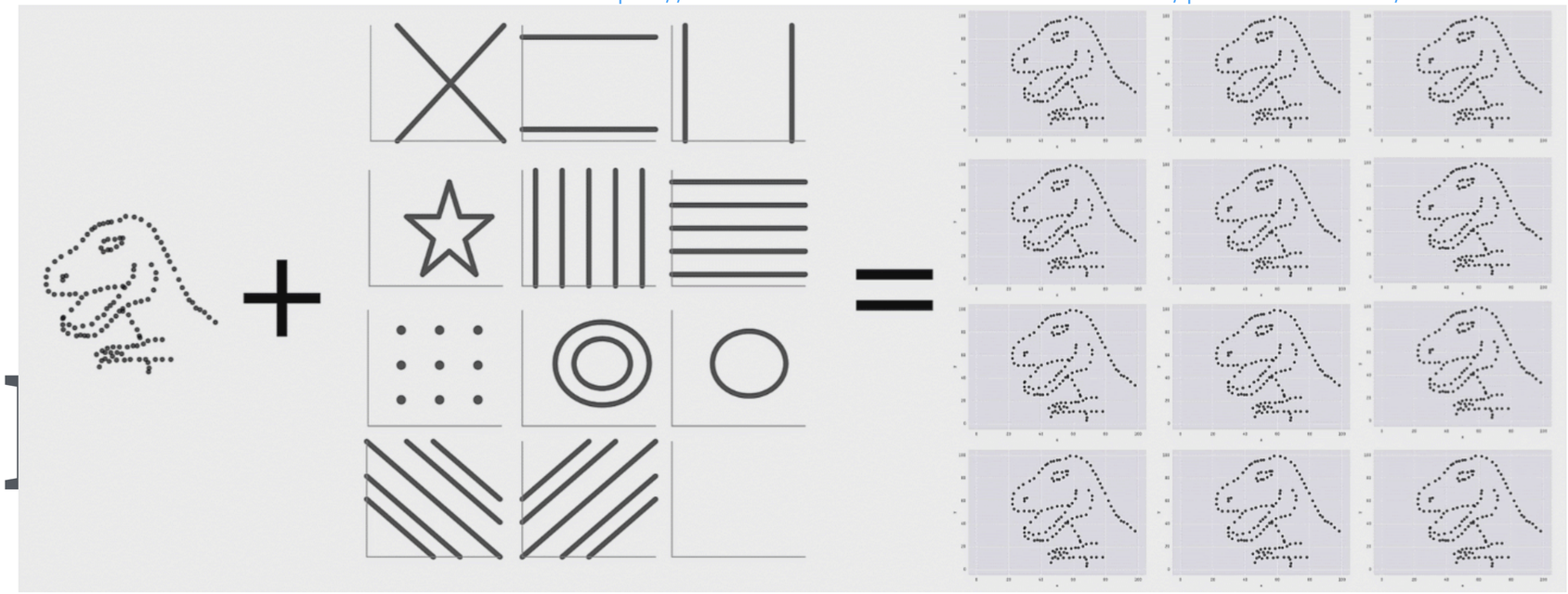


Start with a particular distribution, then shift points slightly making sure the summary statistics is the same and that you head towards a particular target statistic.

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

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

But

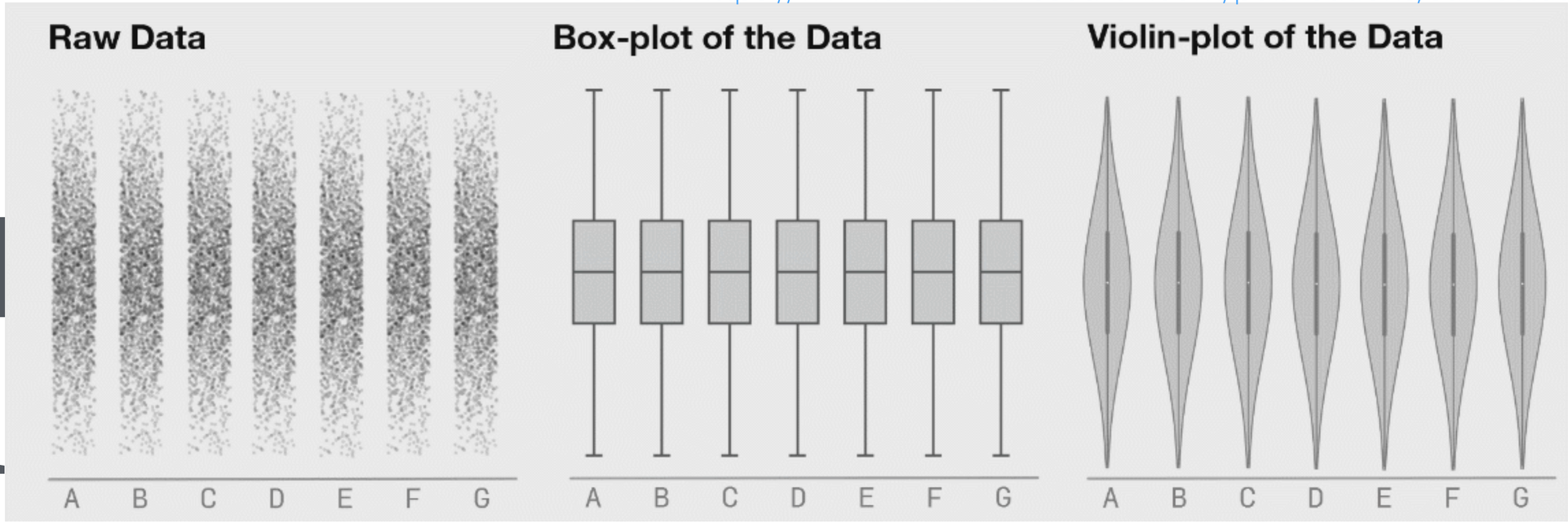


Start with a particular distribution, then shift points slightly making sure the summary statistics is the same and that you head towards a particular target statistic.

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

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

wh
un

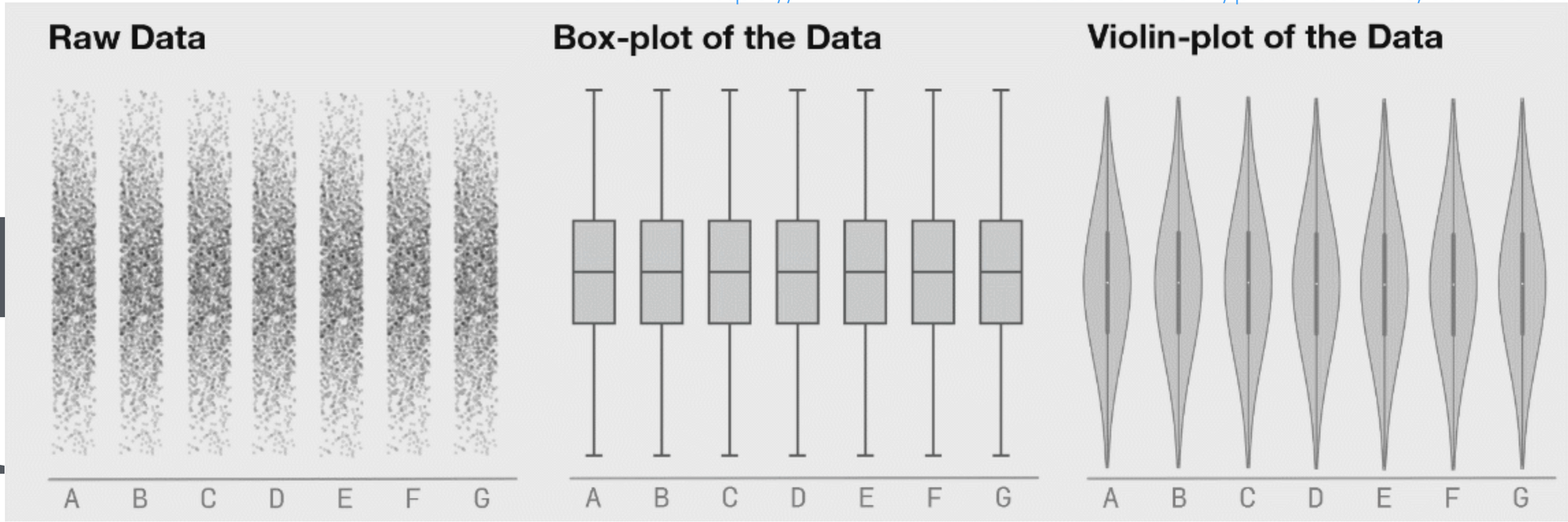


Again - same technique. Read more at

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

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

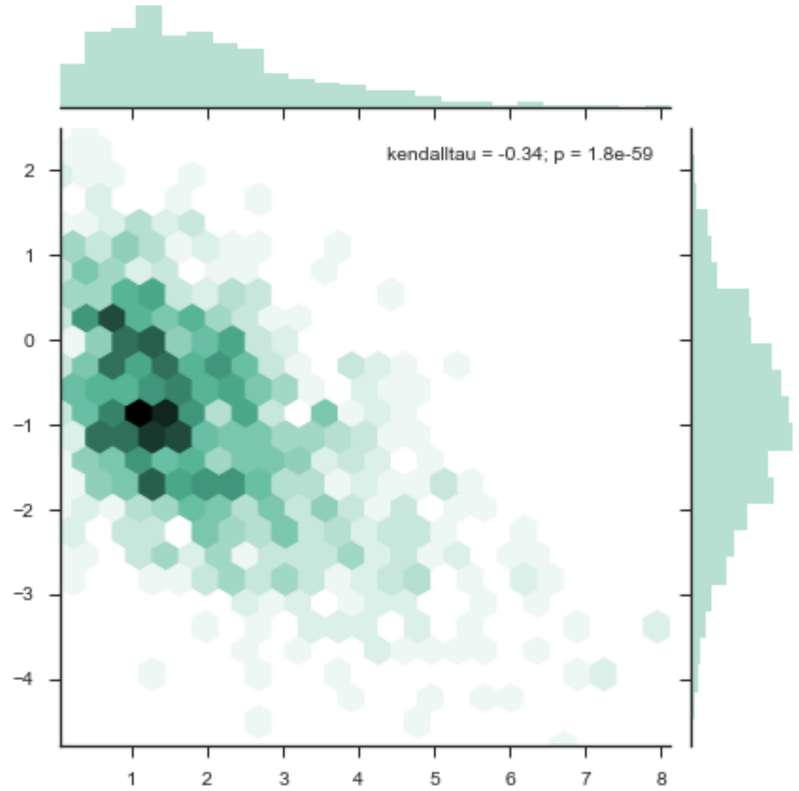
wh
un



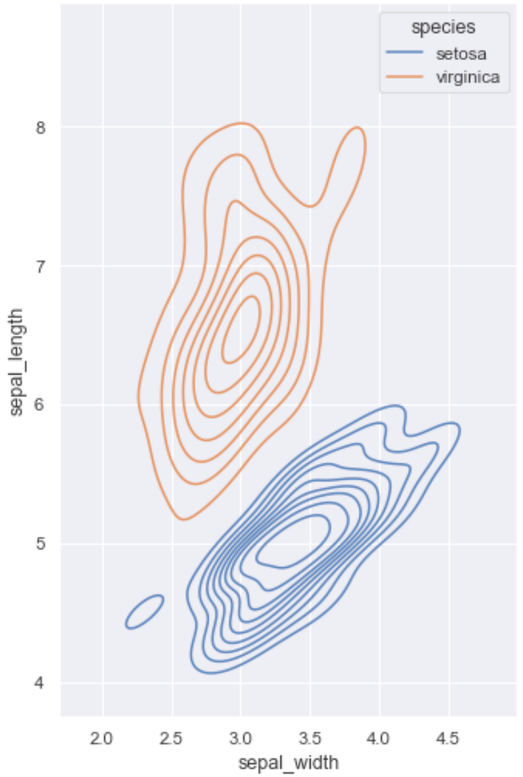
Again - same technique. Read more at

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

Var
dis



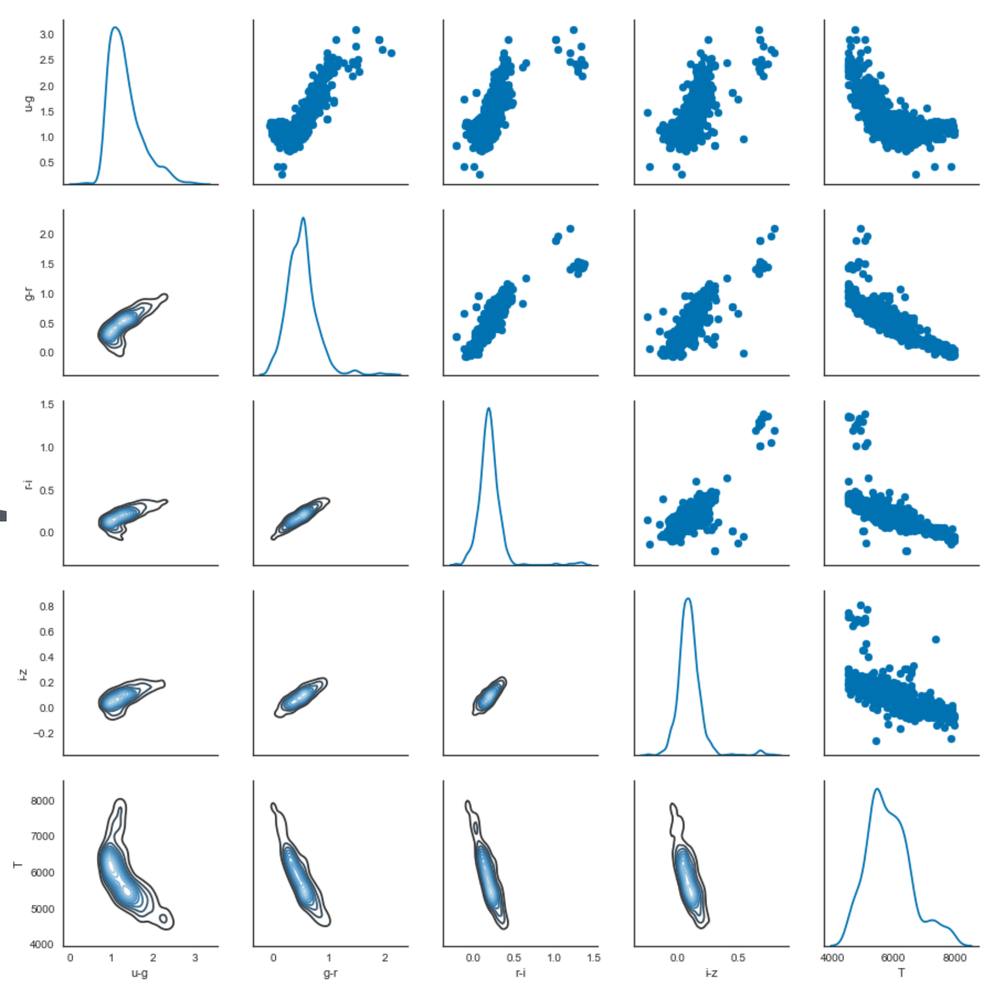
to sh



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

Mul al 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

Further topics in SQL - self-study

Sorting in SQL - ORDER BY

Sorting your output on some variable is simple:

```
SELECT Field, Quality
FROM Observations
ORDER BY Quality
```

What is the first field?

Sorting in SQL - ORDER BY

Sorting your output on some variable is simple:

```
SELECT Field, Quality
FROM Observations
ORDER BY Quality
```

What is the first field?

```
sqlite> SELECT Field, Quality FROM Observations ORDER BY Quality;
StF-051,0.0
StF-045,0.5
StF-048,0.7
StF-043,1.0
StF-044,1.0
StF-046,1.0
StF-047,1.0
```

When you don’t need everything

You want to make a histogram of the magnitude distribution of stars in the SDSS. There are 260,562,744 stars - do you need to download them all?

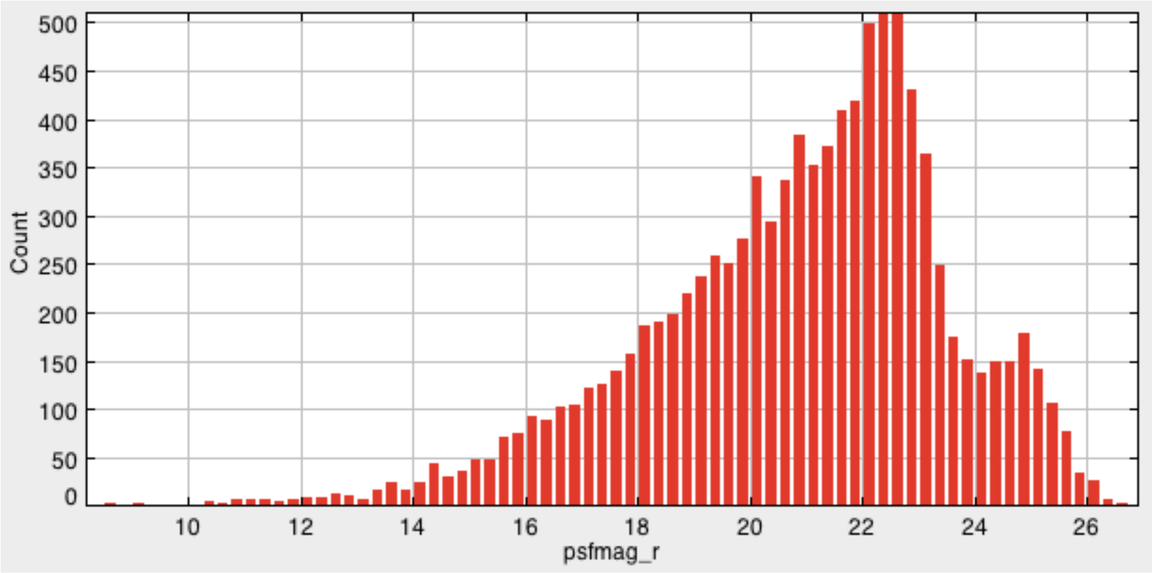
No.

When you don’t need everything

You want to make a histogram of the magnitude distribution of stars in the SDSS. There are 260,562,744 stars - do you need to download them all?

No.

But you can if you want - here are the first 10,000. It can take a lot of time though!



Grouping your output

A better solution is sometimes to let the database server do the work. To do that we need to group our output.

Let us look at a simple case - we want to count the number of stars per field.

FieldID	StarID	Star	Ra	Decl	g	r	
1	1	S1	198.8475	10.5034722	14.5	15.2	2
1	2	S2	198.5654167	11.0231944	15.3	15.4	
3	3	S5	198.9370833	9.9168889	16.4	15.8	1
2	4	S7	199.2516667	10.3486944	14.6	14.1	1

Grouping your output

Counting the number of stars per field.

FieldID	StarID	Star	Ra	Decl	g	r	
1	1	S1	198.8475	10.5034722	14.5	15.2	2
1	2	S2	198.5654167	11.0231944	15.3	15.4	
3	3	S5	198.9370833	9.9168889	16.4	15.8	1
2	4	S7	199.2516667	10.3486944	14.6	14.1	1

The statement we need is: GROUP BY

```
SELECT FieldID, COUNT(*) as NperField
FROM Stars
GROUP BY FieldID
```

Accumulative functions - a side-step

SQL has certain functions that are ‘accumulative’:

```
COUNT
SUM
AVG
MIN
MAX
```

```
SELECT COUNT(*) as Nstars
FROM Stars
```

FieldID	StarID	Star	Ra	Decl	g	r
1	1	S1	198.8475	10.5034722	14.5	15.2
1	2	S2	198.5654167	11.0231944	15.3	15.4
3	3	S5	198.9370833	9.9168889	16.4	15.8
2	4	S7	199.2516667	10.3486944	14.6	14.1

GROUP BY - operating on accumulative functions

On their own these functions work on everything in one bunch - often not what you want. This is the role of **GROUP BY**.

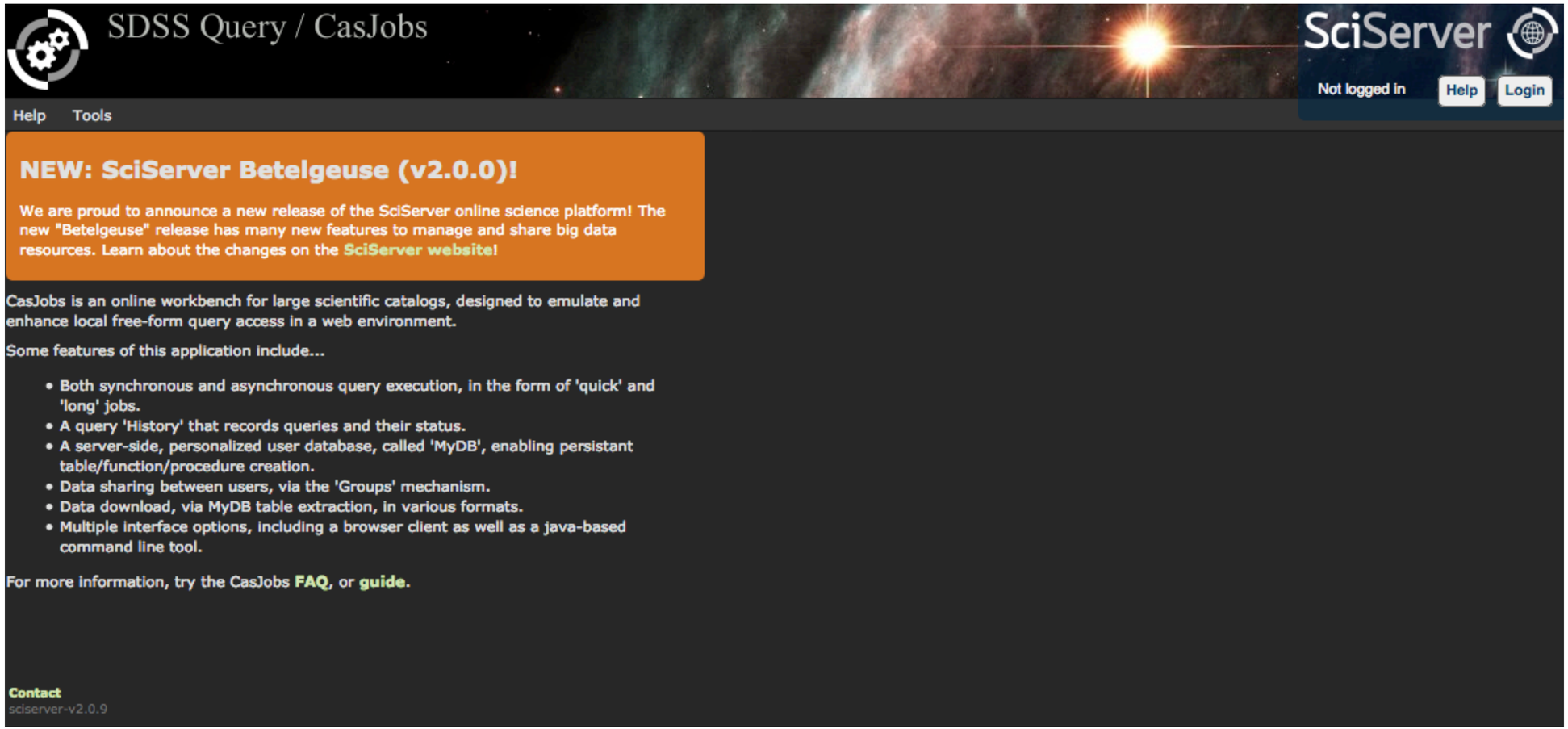
```
select fieldID, avg(r) as 'mean r'
from stars
group by FieldID;
```

Big data easily: SDSS & SQL

This also allows us to get to Kepler, TESS, GALEX and more in the same way.

The SDSS database

http://skyserver.sdss.org/casjobs/default.aspx



Create an account here when you can!

Ordering & Groupings

Counting objects in bins:

```
SELECT .2*(.5+floor(g.r/.2)) as mag,
       count(*) as num
FROM GALAXY as g
GROUP BY .2*(.5+floor(g.r/.2))
ORDER BY mag
```

In general if you want to make C bins per unit for variable x, you use:

$$\frac{1}{C}\left(0.5 + \lfloor Cx \rfloor\right)$$

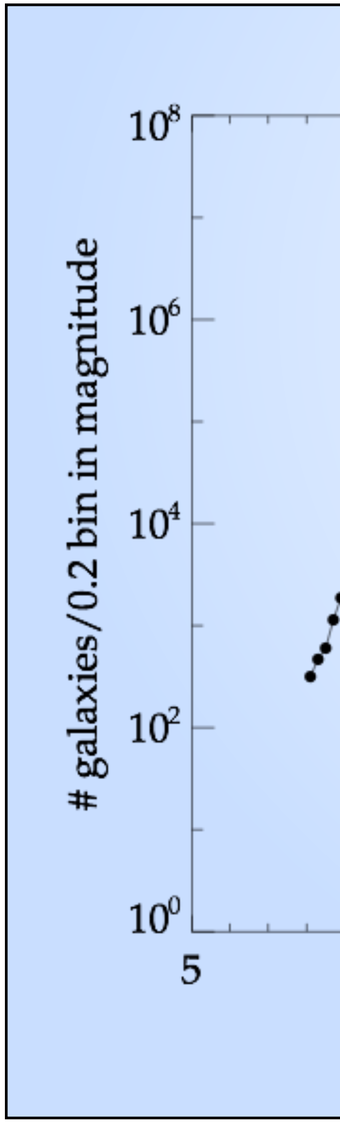
Ordering & Groupings

Counting objects in bins:

```
SELECT .2*(.5+floor(g.r/.2)) as mag,
       count(*) as num
FROM GALAXY as g
GROUP BY .2*(.5+floor(g.r/.2))
ORDER BY mag
```

In general if you want to make C bins per unit for variable x, you use:

$$\frac{1}{C}\left(0.5 + \lfloor Cx \rfloor\right)$$



Creating intermediate tables

The result of a SELECT query is another table - we can therefore use this into another query.

As an example we can use this to create a histogram of the first 10,000 stars in the SDSS Star table:

```
select FLOOR(psfMag_r*10+0.5)/10 as
rmag, COUNT(*) as N
  FROM
    (select TOP 10000 psfMag_r
    FROM Star) smallcat
GROUP BY (FLOOR(psfMag_r*10+0.5))
ORDER BY rmag
```