

missing values

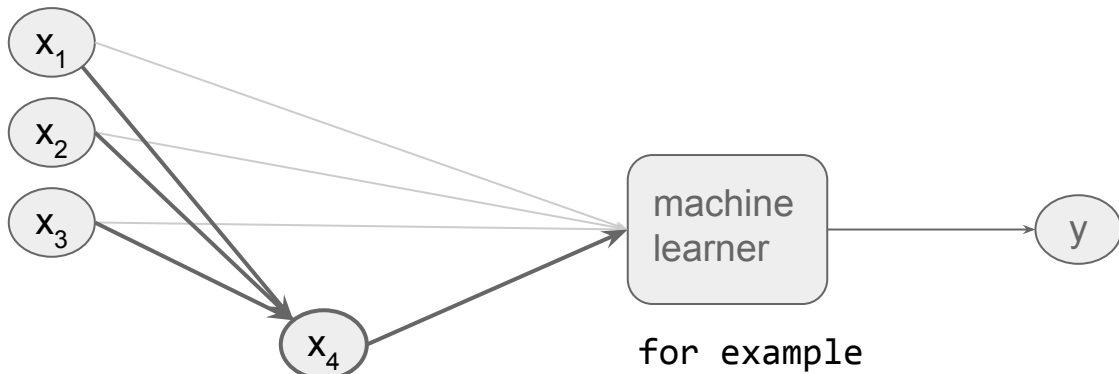
but first: finishing off previous lecture

L1: feature engineering



L2: imputing missing data

many-to-one



put basic features together to construct new features that can capture **relationships** between the basic features – the idea is to augment the feature space with these too

for example

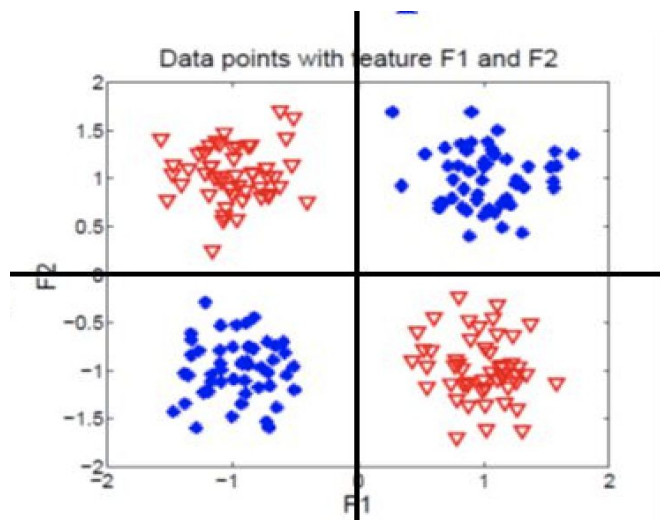
- $x \in \{\text{TRUE}, \text{FALSE}\}$:
 $(x_1 \text{ and } x_2) \text{ or } x_3$
- $x \in \{\text{reals}\}$:
 $(x_1 + x_2) * x_3$

feature construction

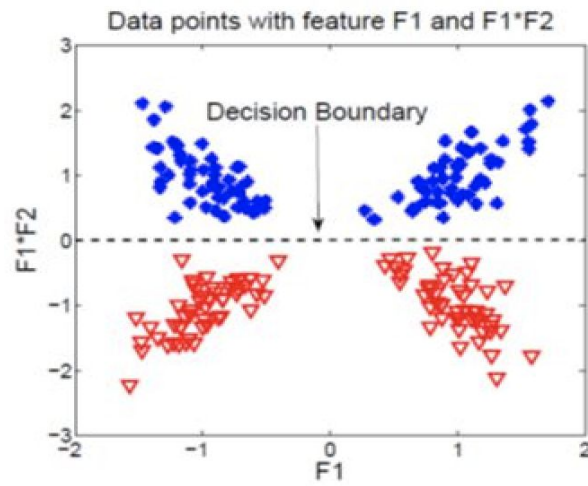
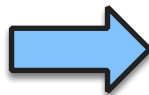
(a.k.a. “many-to-one”)

the quality of such constructed features can drastically affect the learning performance

- -ve: Feature interactions introduce errors and add complexity
- +ve: more meaningful features that lead to more concise and accurate machine learning models



$$F_3 = F_1 * F_2$$



components of a feature construction system

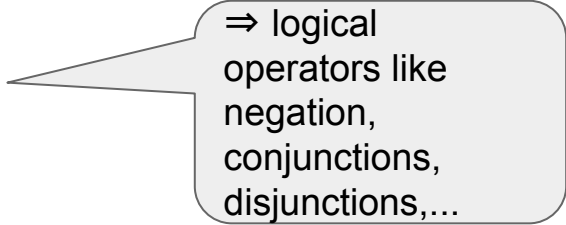
Automatic feature construction systems need the following two components:

- a **search strategy** to search the space of possible functions (of original features)
- an **evaluation mechanism** to measure the goodness of a candidate function

operators for constructing new features

Boolean features:

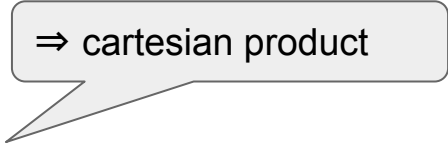
- $x_1, x_2 \in \{\text{TRUE}, \text{FALSE}\}$
 - e.g. $x_3 \leftarrow \text{not}(x_1)$
 - e.g. $x_4 \leftarrow \text{xor}(x_1, x_2)$



⇒ logical operators like negation, conjunctions, disjunctions,...

Nominal/Categorical features:

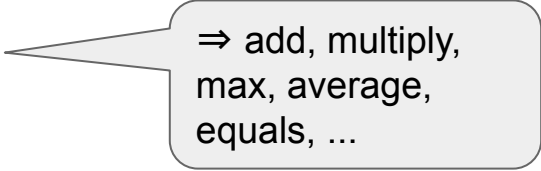
- $x_1 \in \{\text{dog}, \text{cat}\}, x_2 \in \{\text{rain}, \text{snow}\} \dots$
 - e.g. $x_3 \in \{(\text{dog}, \text{rain}), (\text{dog}, \text{snow}), (\text{cat}, \text{rain}), (\text{cat}, \text{snow})\}$



⇒ cartesian product

Numerical features:

- $x_1 \in \{\text{ints}\}, x_2 \in \{\text{reals}\}, \dots$
 - $x_4 \leftarrow \text{min}(x_1, \text{mean}(x_2, x_3))$



⇒ add, multiply, max, average, equals, ...

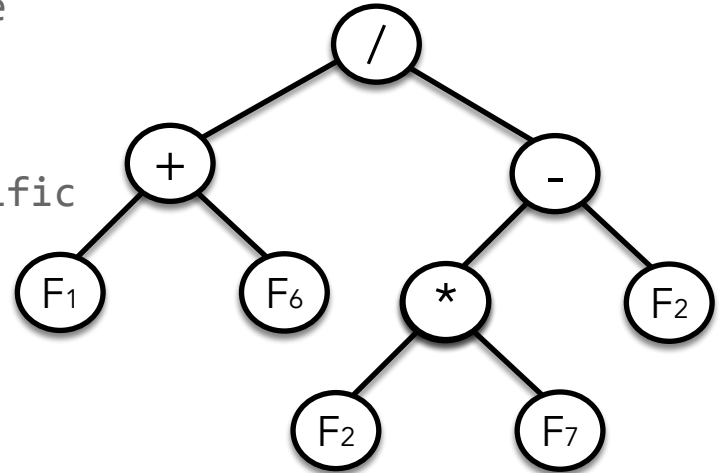
construct new features automatically with Genetic Programming?

Genetic Programming is a **flexible** way to make mathematical and logical functions

but warning: there **isn't much structural (topological) information** in the search space of possible functions


→ yes it's possible,
but be prepared to spend a lot of compute time trying out dud trees...

→ Also: just evolving good answers to specific problems is not a recipe for insight. Insight comes from having theories about the world (a.k.a. “a model”) and testing it.



Evaluating new features

- Not all constructed features are useful or good!...
- Apply the usual feature selection techniques to remove redundant and irrelevant features



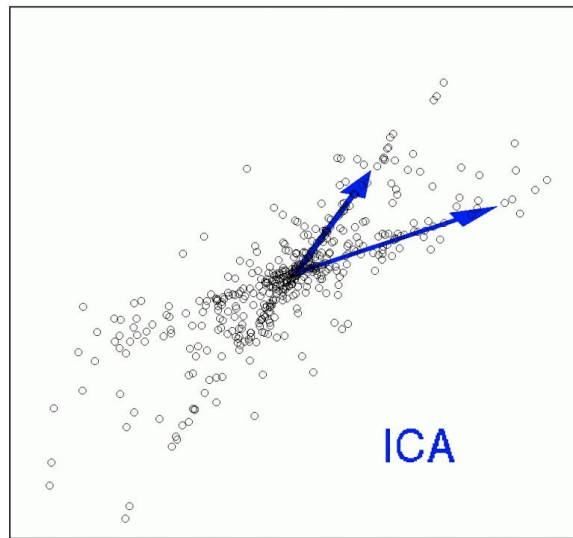
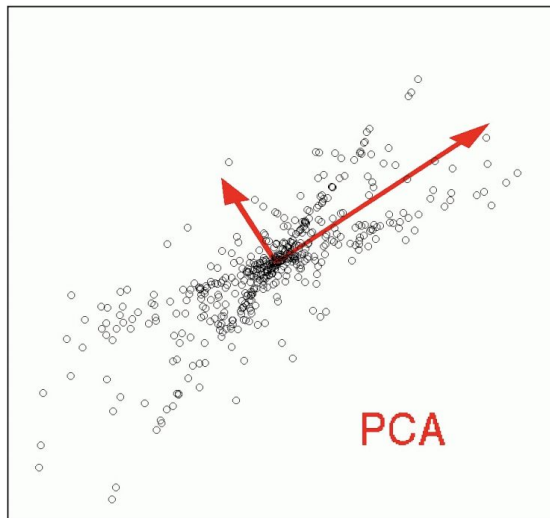
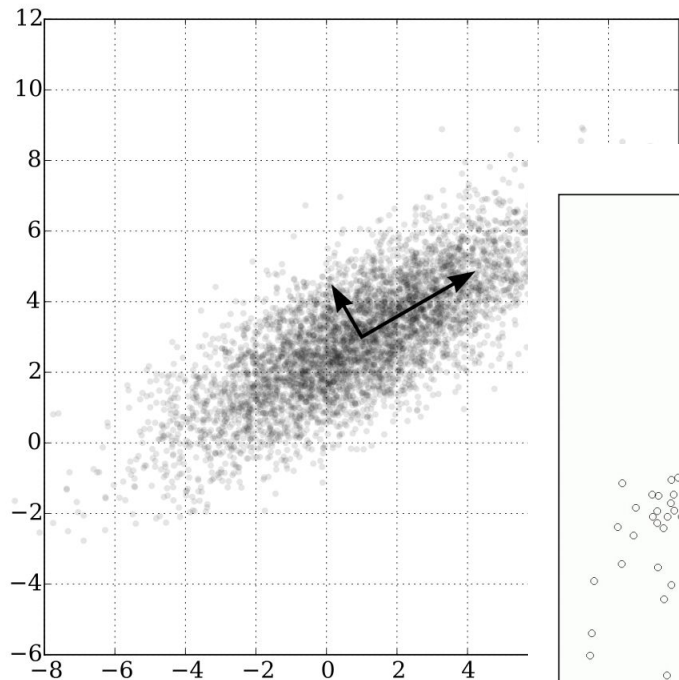
comment: it is *not necessarily the case* that “redundant” features should be culled.

Imagine having three noisy sensors that all measure the same quantity. You would be better off keeping all 3 (to average them to fight the noise...) than to discard two as being “redundant”

Some specific “many to many” feature methods:

PCA (Principal Components Analysis)

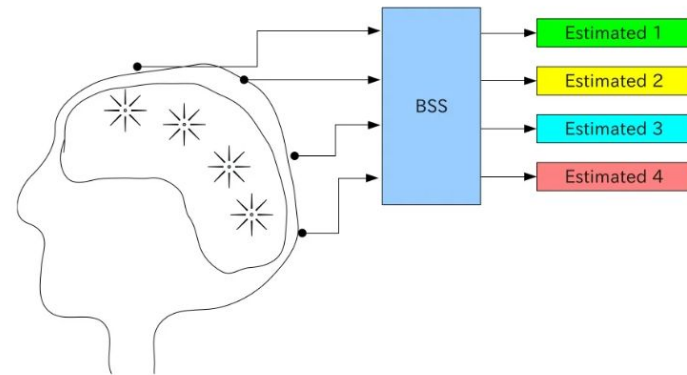
- PCA
- ICA
- Kernel PCA
- autoencoders



ICA = Independent Components Analysis

(a.k.a. “Blind Source Separation”, solving the “cocktail party problem”)

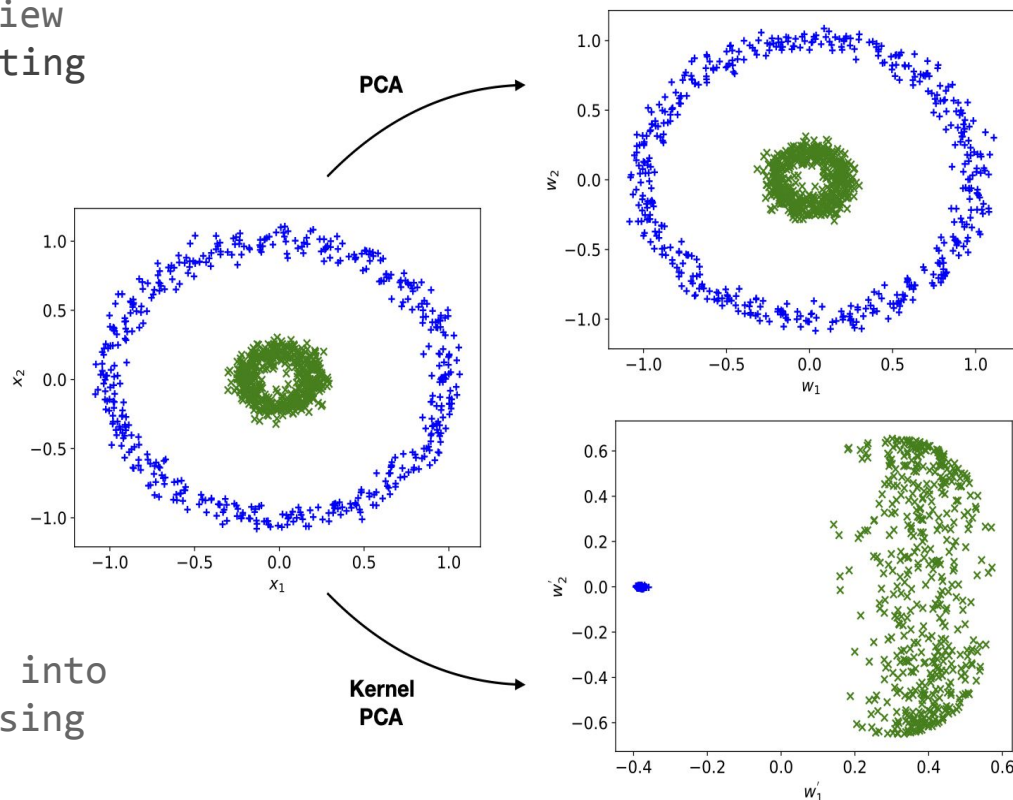
- PCA and ICA both create new components that are **linear combinations** of the originals
- PCA removes correlation, but doesn't usually find those independent components that might actually underly the data
- ICA assumes the observed data arose as a linear combination of some independent components (that are tragically unknown). ICA then aims to *recover* those components from data set X.
- ICA finds components that are as **statistically independent** as possible



Q: why might it be good to find truly *independent* components?

kernel PCA

- combines a specific mathematical view of PCA with **kernel** functions, creating a **nonlinear extension** of PCA

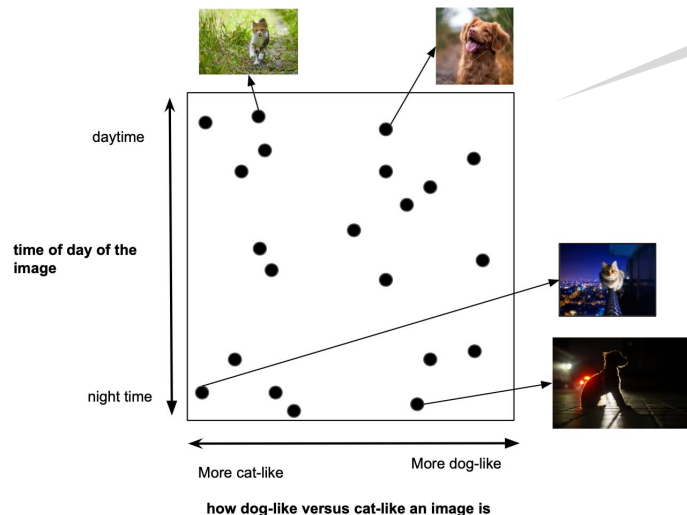


- Perform PCA on data that has (effectively) been transformed into a high-dimensional setting – “blessing of dimensionality” (!)

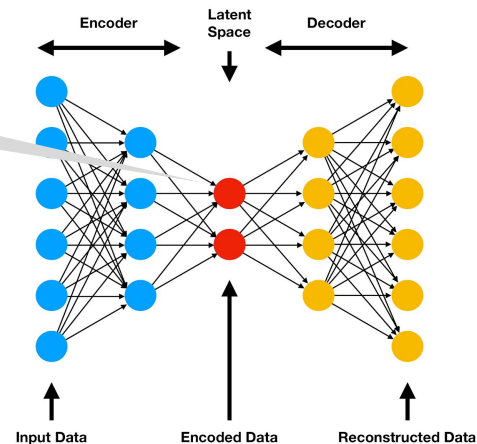
auto-encoders

auto-encoders are a type of neural network, trained to “squash” data through a **bottleneck layer** with just a few neurons – i.e. they do non-linear **dimensionality reduction**

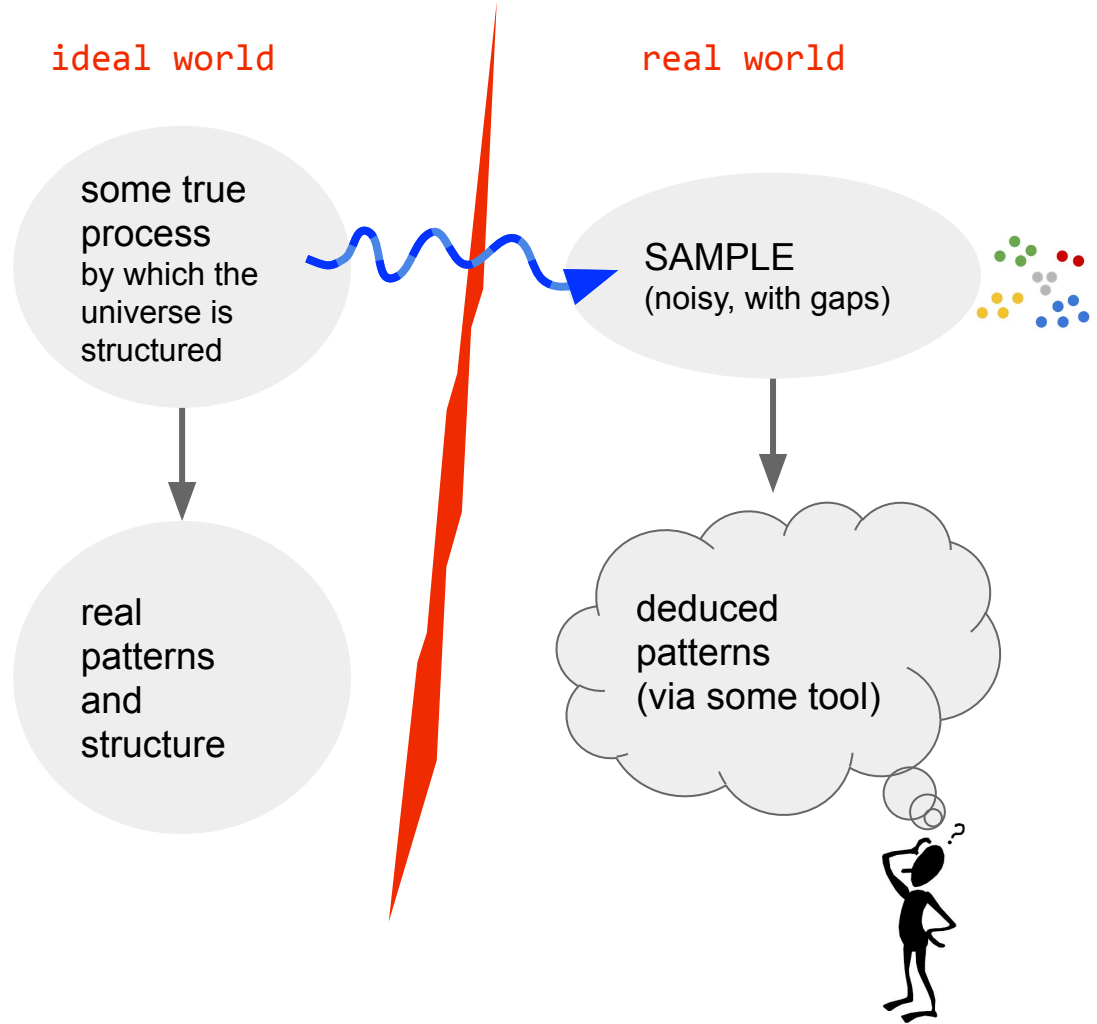
An Oversimplified Example of a Cat/Dog Image Latent Space



often referred to as a “**latent space**”




the problem of
finding true
patterns from
sample data



Missing values

many possible reasons, e.g:

- high cost involved in measuring variables,
- failure of sensors,
- reluctance of respondents to answer some q,
- an ill-designed survey

1. Real-world datasets often contain many missing values! 
2. Most machine learning algorithms *require* all the values, for a given training example.

You can see the problem!

4 solutions:

- Remove the missing data **instances** from the dataset, or
- Remove the worst offending **attributes** from the dataset, or
- Use **imputation** methods to substitute in “fake” values, or
- Use a learner that is **able to cope** with missing values

Truly

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

ideal world

real world

Recorded

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	?	No
2	?	?	100K	No
3	No	Single	70K	No
4	Yes	Married	?	No
5	No	?	95K	Yes
6	?	Married	60K	No
7	Yes	Divorced	220K	No
8	No	?	?	Yes
9	?	Married	75K	No
10	No	Single	90K	Yes

- Missing data can also be in the form of features, or labels (just one class, or...?) or both

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	?	?	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	?	95K	Yes
6	?	Married	60K	No
7	Yes	Divorced	220K	No
8	No	?	85K	Yes
9	?	Married	75K	No
10	No	Single	90K	Yes

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	?
2	No	Married	100K	?
3	No	Single	70K	No
4	Yes	Married	120K	?
5	No	Divorced	95K	Yes
6	No	Married	60K	?
7	Yes	Divorced	220K	No
8	No	Single	85K	?
9	No	Married	75K	No
10	No	Single	90K	Yes

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	?	No
2	?	?	100K	?
3	No	Single	?	No
4	Yes	Married	120K	?
5	No	?	95K	Yes
6	?	Married	60K	?
7	Yes	Divorced	?	No
8	No	Single	85K	?
9	No	?	75K	No
10	?	Single	?	Yes

Option 1: just eliminate entire observations

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	?	Married	?	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	?	Divorced	?	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	?	?	75K	No
10	No	Single	90K	Yes



Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
10	No	Single	90K	Yes

This seems extreme! It wastes lots of precious, expensive, data.

But let's just say we do this. **Is it even “okay” ?**

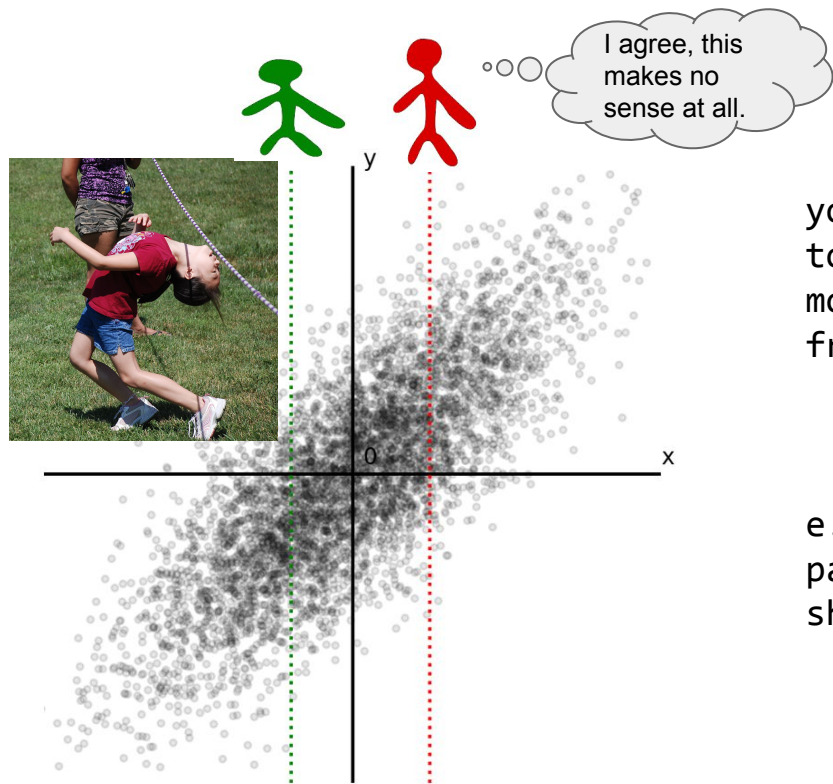
(e.g. suppose we knew *high values of income* were more likely to be missing...)

there are different kinds of missing!

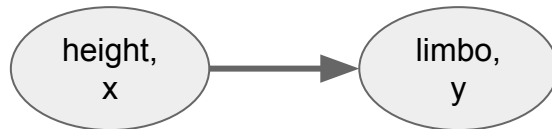
- MCAR missing, completely at random
- MAR missing at random
- MNAR missing, not at random

a toy example

x is height, y is ability to limbo (relative to their average values)



you're using this data to train a regression model to predict y, from x

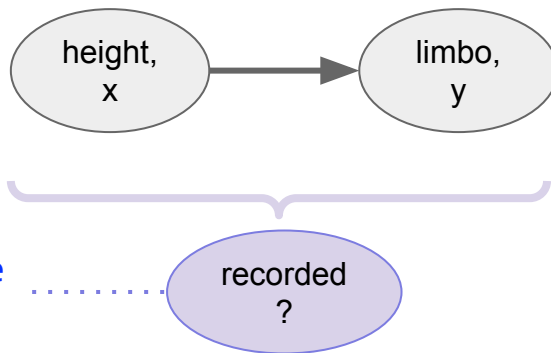
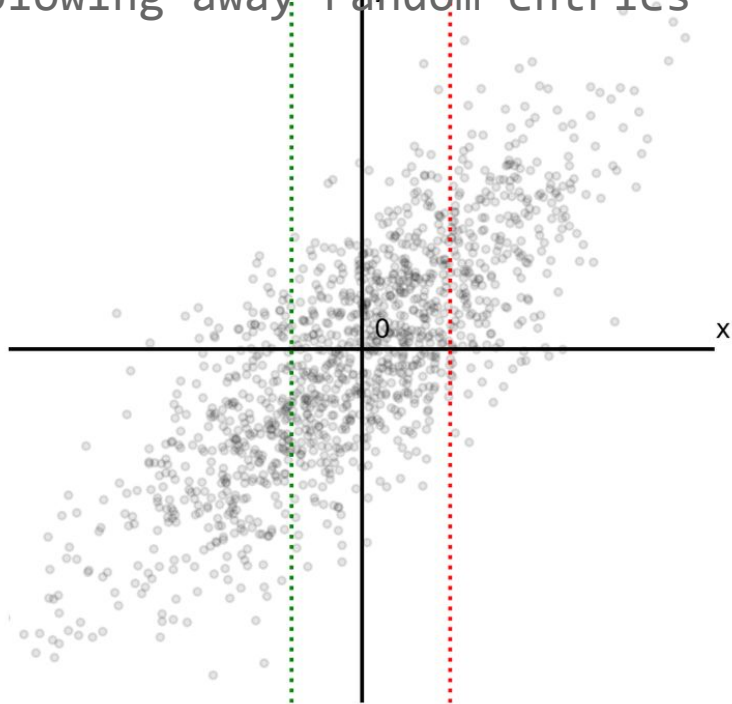
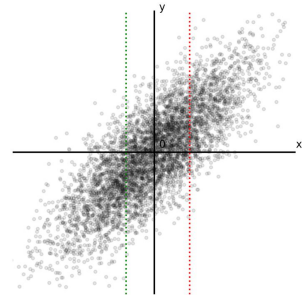


e.g. consider two particular cases, shown as and

Missing Completely At Random (MCAR)

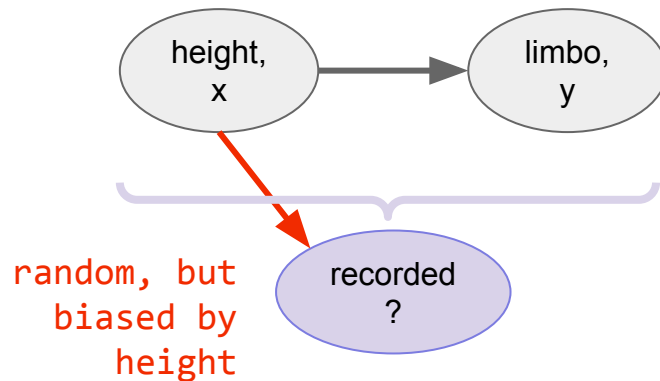
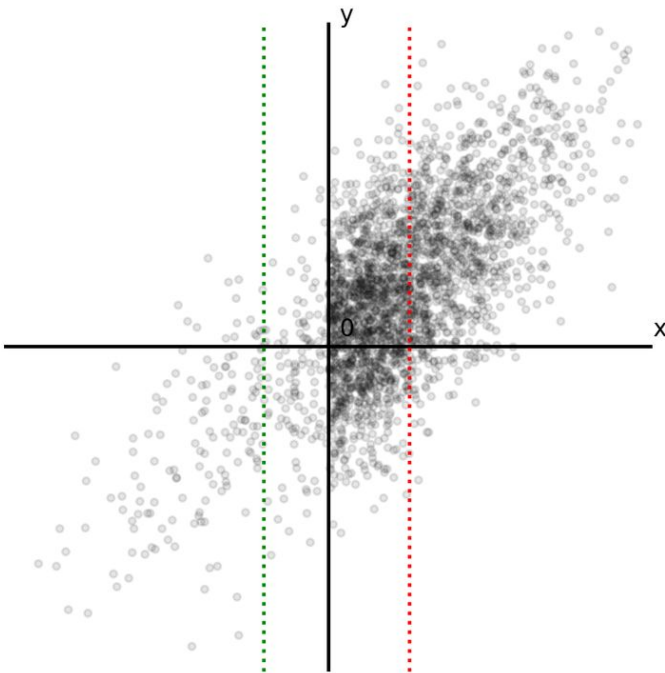
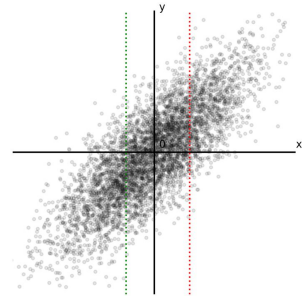
E.g.

I blasted my spreadsheet with a shotgun,
blowing away random entries

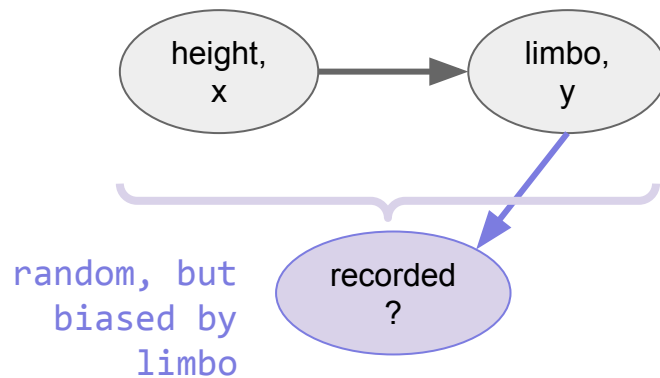
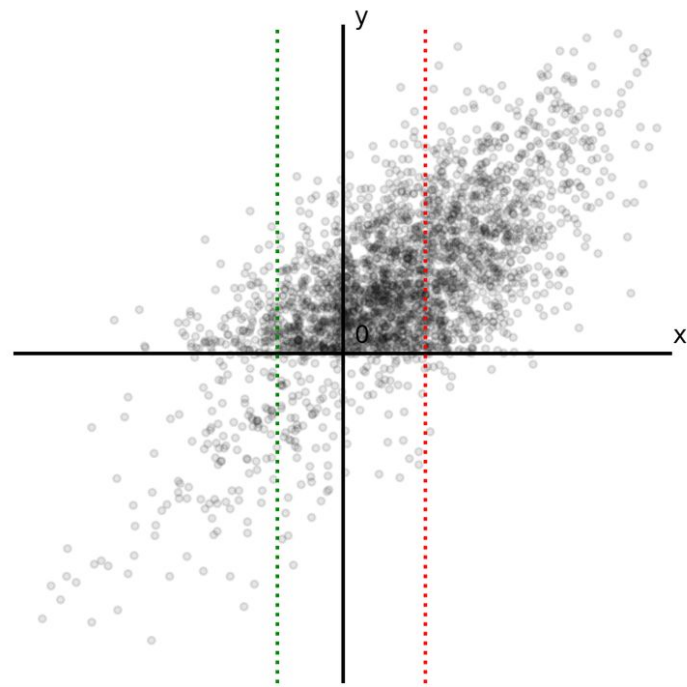
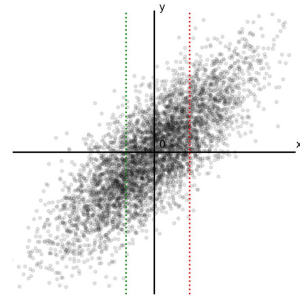


Missing At Random (MAR)

note this name is really confusing. A better name for MAR would have been be “Missing, conditionally at random”, but that would screw with the acronyms 🙄

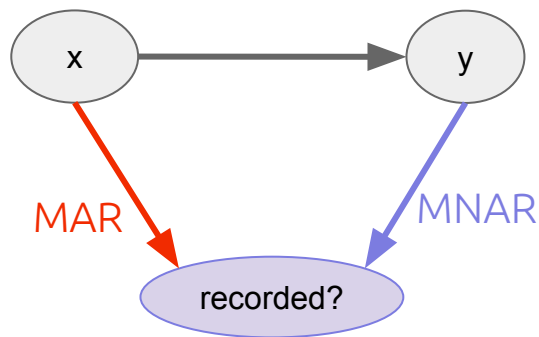


Missing Not At Random (MNAR)

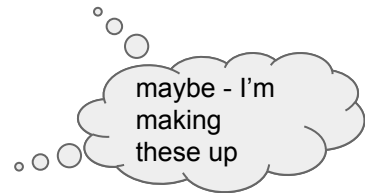


e.g.

MCAR
has no linkage



1. Half the survey forms weren't delivered or got lost, due to random beetle infestations
2. Males are less likely to fill in a depression survey, but this has nothing to do with their level of depression, after accounting for maleness
3. Depressed people are less likely to fill in a survey about depression
(similarly, happy people are less likely to fill in a survey about a grievance)



Q. would you say these are cases of MCAR, MAR, or MNAR?

.... tuesday's lecture got to this point....

conclusions?

So: simply avoiding data that has missing values isn't ideal

- reason 1? ● wastes valuable data - what if *every* row had something missing?!
- reason 2? ● could be MNAR, in which case results will be “biased” (ie. wrong)

but what else can we do?

We could “impute” data – replace missing with actual values.

But what values?!

Option 2: eliminate entire attributes

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	?	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	?	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	?	Married	75K	No
10	No	Single	90K	Yes



<i>Tid</i>		Marital Status	Taxable Income	Cheat
1		Single	125K	No
2		Married	100K	No
3		Single	70K	No
4		Married	120K	No
5		Divorced	95K	Yes
6		Married	60K	No
7		Divorced	220K	No
8		Single	85K	Yes
9		Married	75K	No
10		Single	90K	Yes

Seems a bit drastic

Option 3: Imputation

- Mean imputation
- Mode imputation
- Hot deck imputation
- other ideas...
- Multiple imputation
- MICE (used in sklearn)

impute with Mean, or Mode

- **Mean**: for continuous attributes
 - Fill in with average complete values
- **Mode**: for categorical attributes
 - Fill in with the most frequent value

Pros and cons:

- (+) Doesn't changes the mean/mode of attributes
- (-) Under-represents the variability in the data

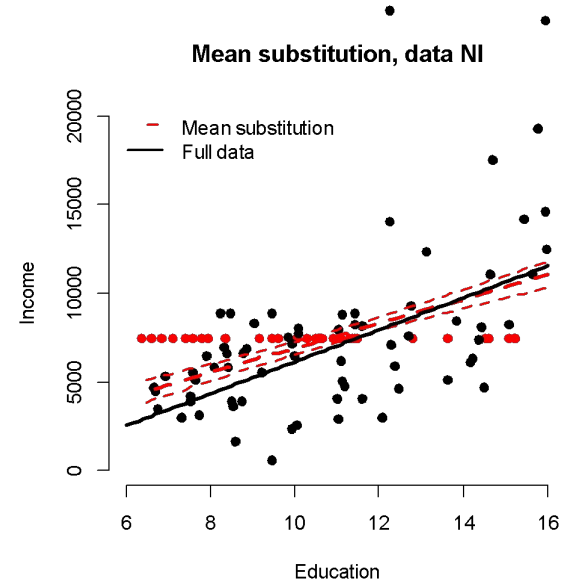
pretty bad imputation: Mean or Mode

Tid	Refund	Marital Status	Taxable Income	Cheat
1	?	Single	125K	No
2	No	?	?	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	?	Yes
6	No	?	60K	No
7	Yes	Divorced	220K	No
8	?	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes
No		Single	105K	

most common/
mean value

Tid	Refund	Marital Status	Taxable Income	Cheat
1	No	Single	125K	No
2	No	Single	105K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	105K	Yes
6	No	Single	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

sklearn's [SimpleImputer](#) has options



“hot deck” imputation

For each record containing missing values:

Find the **most similar record**, then fill missing values with the corresponding values from that!

Pros and cons

(+) Imputes realistic values

(-) what do you think?

Imputation by nearest neighbor?

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	?	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	?	Divorced	95K	Yes
6	?	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

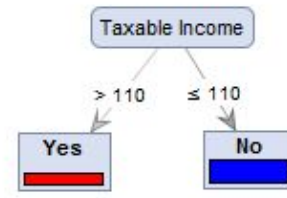


<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

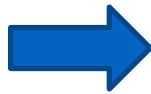


<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	No	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Imputation using a decision rule?



<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	?	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	?	Divorced	95K	Yes
6	?	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Imputation using Regression?

For each attribute containing missing values:

- Divide data into **two parts**:
 - the records that are complete
 - the records having missing values
- Use the **complete data to estimate a regression** model between the attribute and others, and use that to **predict/suggest missing values** in the attribute

Pros and cons

- (+) Maintains relationships between attributes
- (-) Makes an assumption about regression model

Multiple Imputation

idea: to avoid bias caused by **one** imputation method, do several!

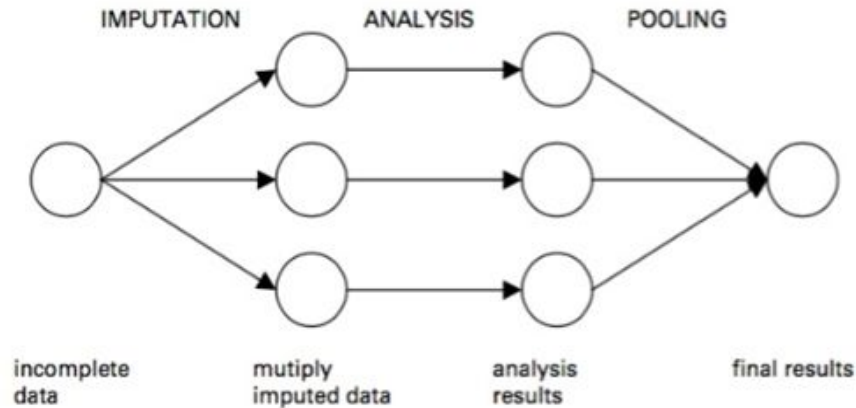
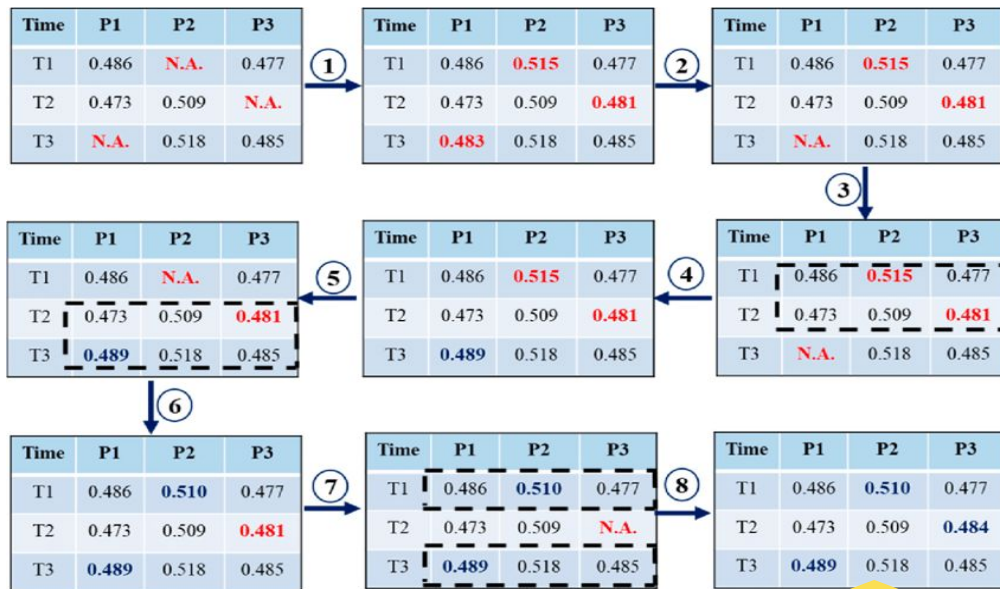


Figure : Schematic representation of the steps in multiple imputation. The process starts with an incomplete data set (on the left side), which is imputed m times ($m=3$ here) thus creating m completed data sets. Each complete data set is analyzed by using standard complete-data software, thus resulting in m analysis results. Finally, these m results are pooled into one final result that adequately reflects the amount of uncertainty in the estimates.

MICE (“Multivariate Imputation by Chained Equations” 😞)

a version of multiple imputation that apparently works quite well and is implemented in sklearn:

1. Fill all missing values with random values (?!?)
2. Regress each attribute that contains missing values on other attributes, in a chain => 1 imputed dataset



sklearn's [IterativeImputer](#) does this, which is steps 1 and 2

3. Repeat whole procedure N times to generate N imputed datasets
4. Average N imputed datasets => final imputed dataset

Option 4: use a learning algorithm that can cope with missing values

- Missing values can sometimes be taken into account during the learning process of acquiring knowledge
- For example:
 - Clustering algorithms: similarity between the objects can be calculated using only the attributes that do not have missing values.
 - Classification: Gradient Boosting, kNN, and Random Forests



scikit-learn resources

<https://scikit-learn.org/stable/modules/impute.html>



Install Use

Prev Up Next

scikit-learn 1.1.2
[Other versions](#)

Please [cite us](#) if you use the software.

6.4. Imputation of missing values

- 6.4.1. Univariate vs. Multivariate Imputation
- 6.4.2. Univariate feature imputation
- 6.4.3. Multivariate feature imputation
- 6.4.4. References
- 6.4.5. Nearest neighbors imputation
- 6.4.6. Marking imputed values
- 6.4.7. Estimators that handle NaN values

6.4. Imputation of missing values

For various reasons, many real world datasets contain missing values, often encoded as blanks, NaNs or other placeholders. Such datasets however are incompatible with scikit-learn estimators which assume that all values in an array are numerical, and that all have and hold meaning. A basic strategy to use incomplete datasets is to discard entire rows and/or columns containing missing values. However, this comes at the price of losing data which may be valuable (even though incomplete). A better strategy is to impute the missing values, i.e., to infer them from the known part of the data. See the glossary entry on [imputation](#).

6.4.1. Univariate vs. Multivariate Imputation

One type of imputation algorithm is univariate, which imputes values in the i -th feature dimension using only non-missing values in that feature dimension (e.g. `impute.SimpleImputer`). By contrast, multivariate imputation algorithms use the entire set of available feature dimensions to estimate the missing values (e.g. `impute.IterativeImputer`).