DIMENSIONALITY REDUCTION

FOR LATER
DOWNLOAD THE
“Exercise - Dimensionality Reduction.zip”
NOTEBOOK & DATASET
FROM THE COURSE SITE

CORE INTUITION

Most datasets have many dimensions

Hard to see (and think about) differences/groupings when $|\text{dimensions}| > 2$

WHAT ARE OUR OPTIONS?

Feature Elimination/Selection

Pick just a few dimensions to show?

(Can do this manually, but plenty of automated techniques too)

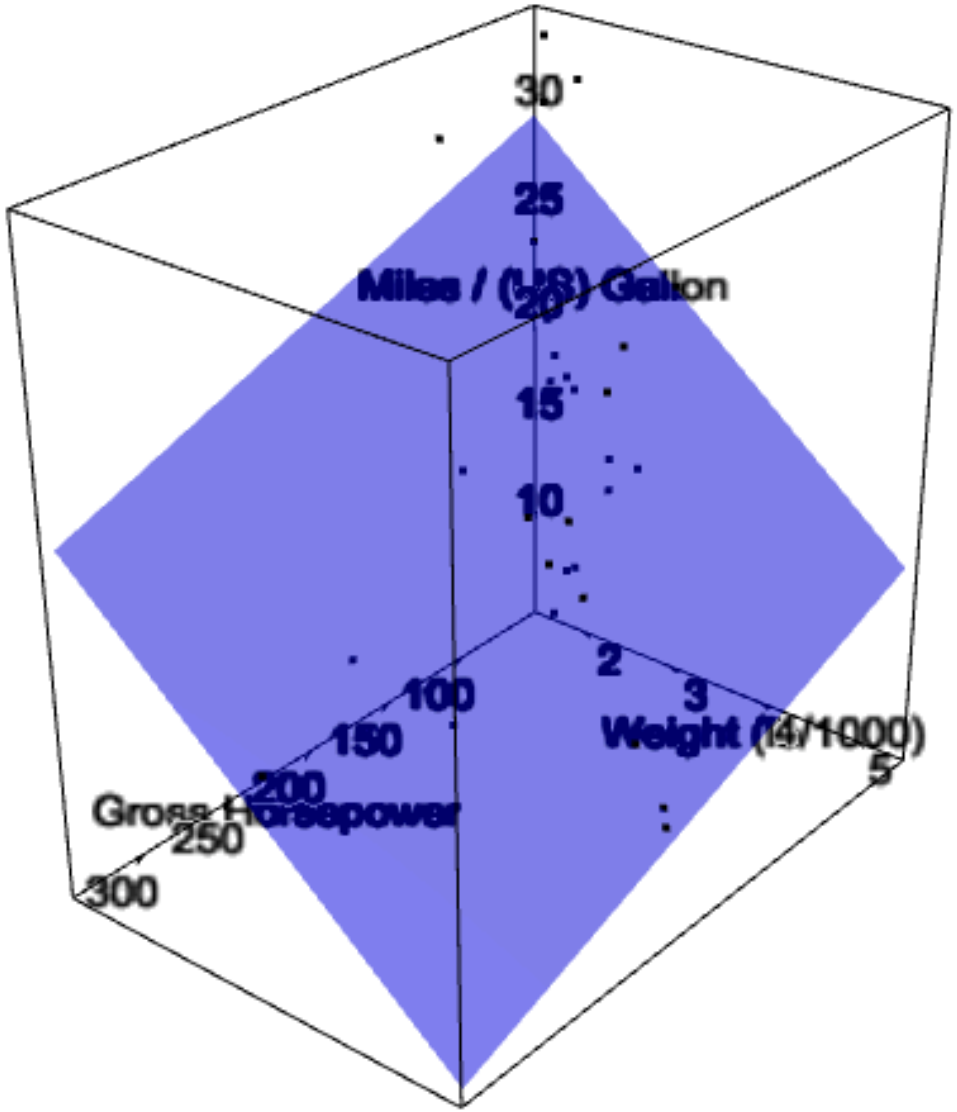
Try to find a way to show them…

3D CAN ALREADY BE HARD

Occlusion, depth ambiguity, etc.

4D, 5D, … nD are even harder.

How can we **see** clusters, fitted planes, etc. in high-dimensional spaces?



http://shiny.stat.calpoly.edu/3d_regression/

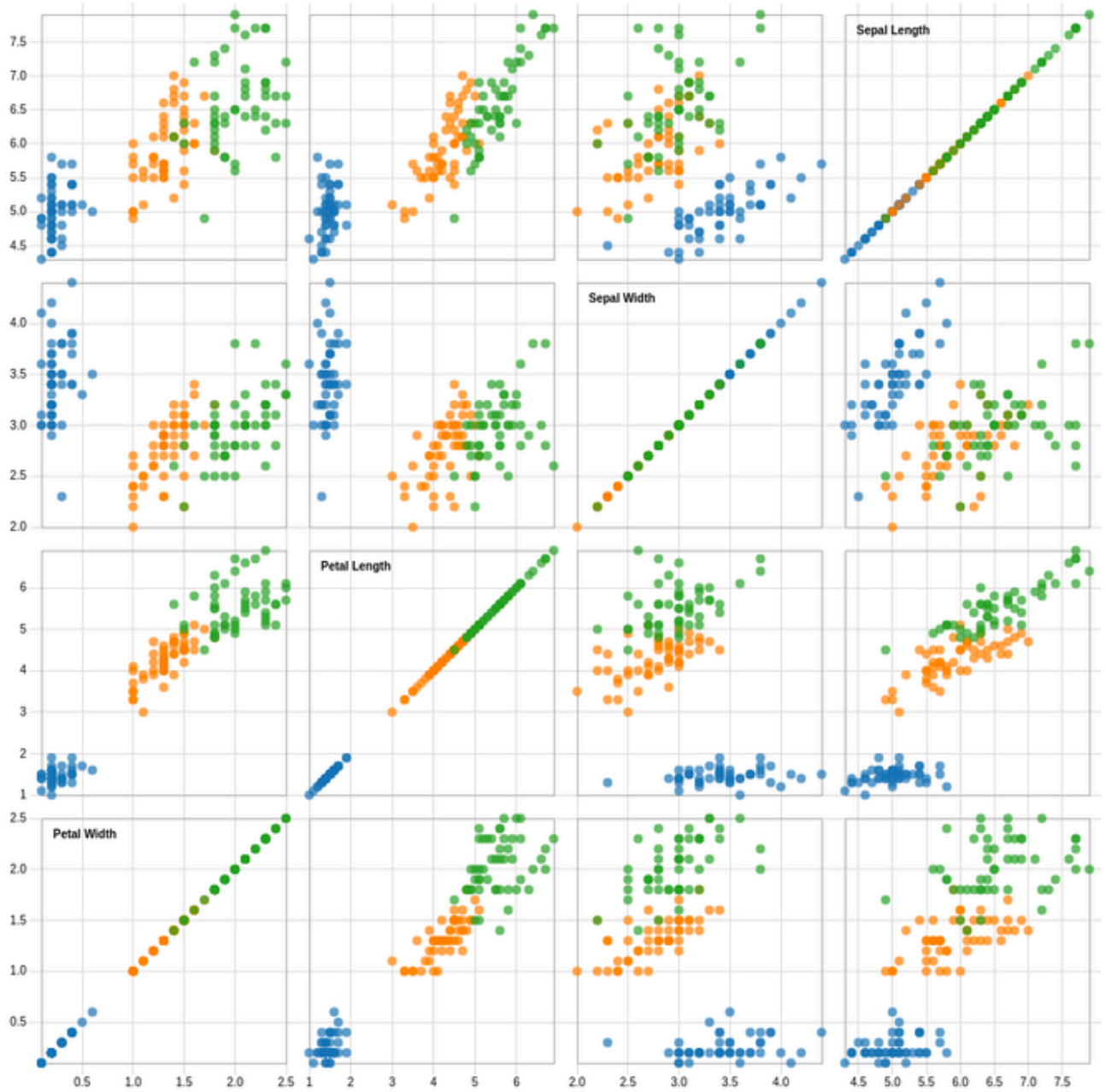
MANY 2D MULTIPLES

PRO

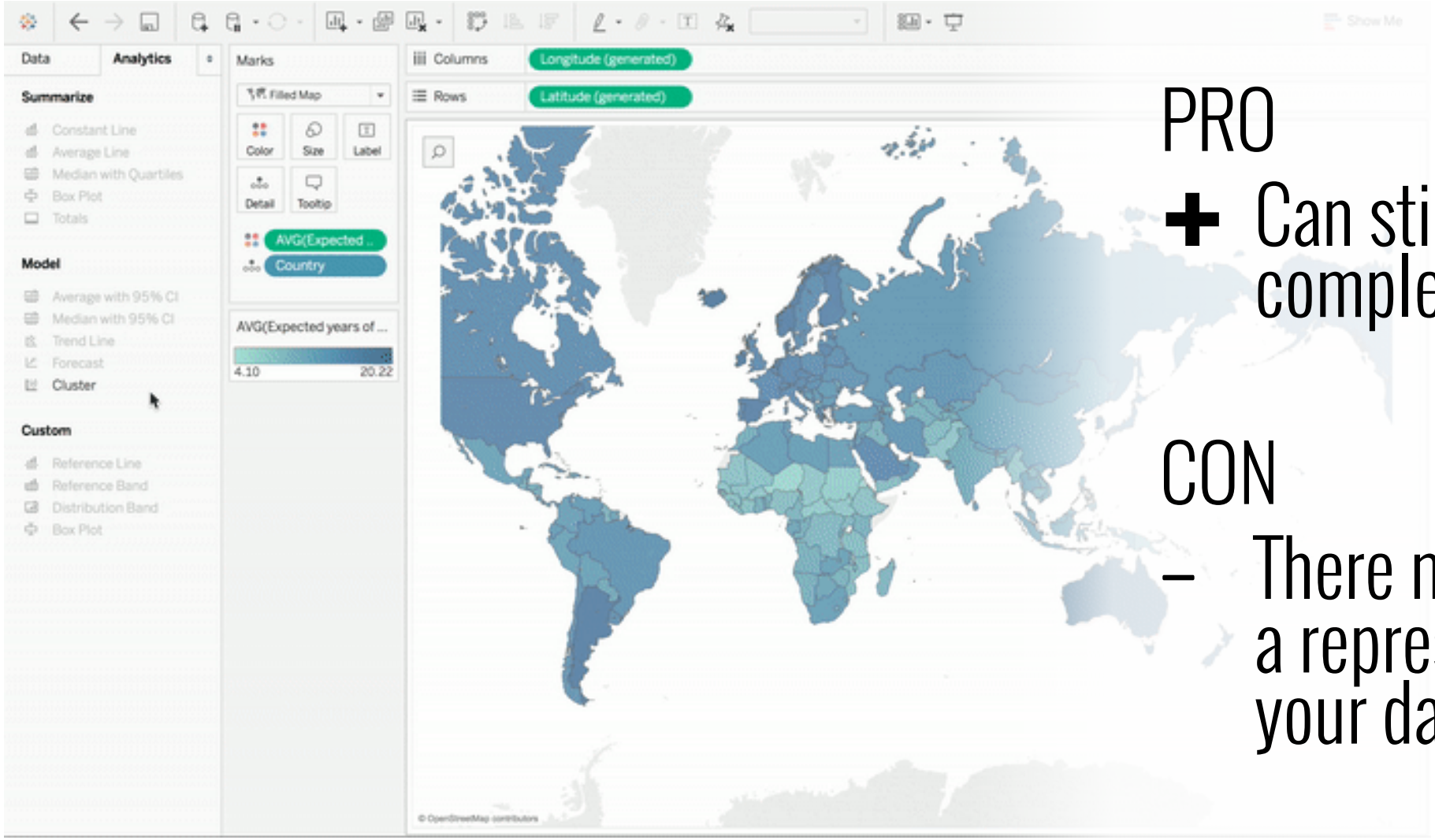
+ Straightforward extension of existing 2D methods

CON

– Hard to mentally fuse
– Hard to see **n**D groupings
– Scales badly



PICK A REPRESENTATION YOU CAN REASON ABOUT



PRO

\+ Can still support complex reasoning

CON

– There might not be such a representation for your dataset

ALTERNATIVE - PROJECT N-DIMENSIONS BACK TO 2D

(OR SOME OTHER SMALL NUMBER)

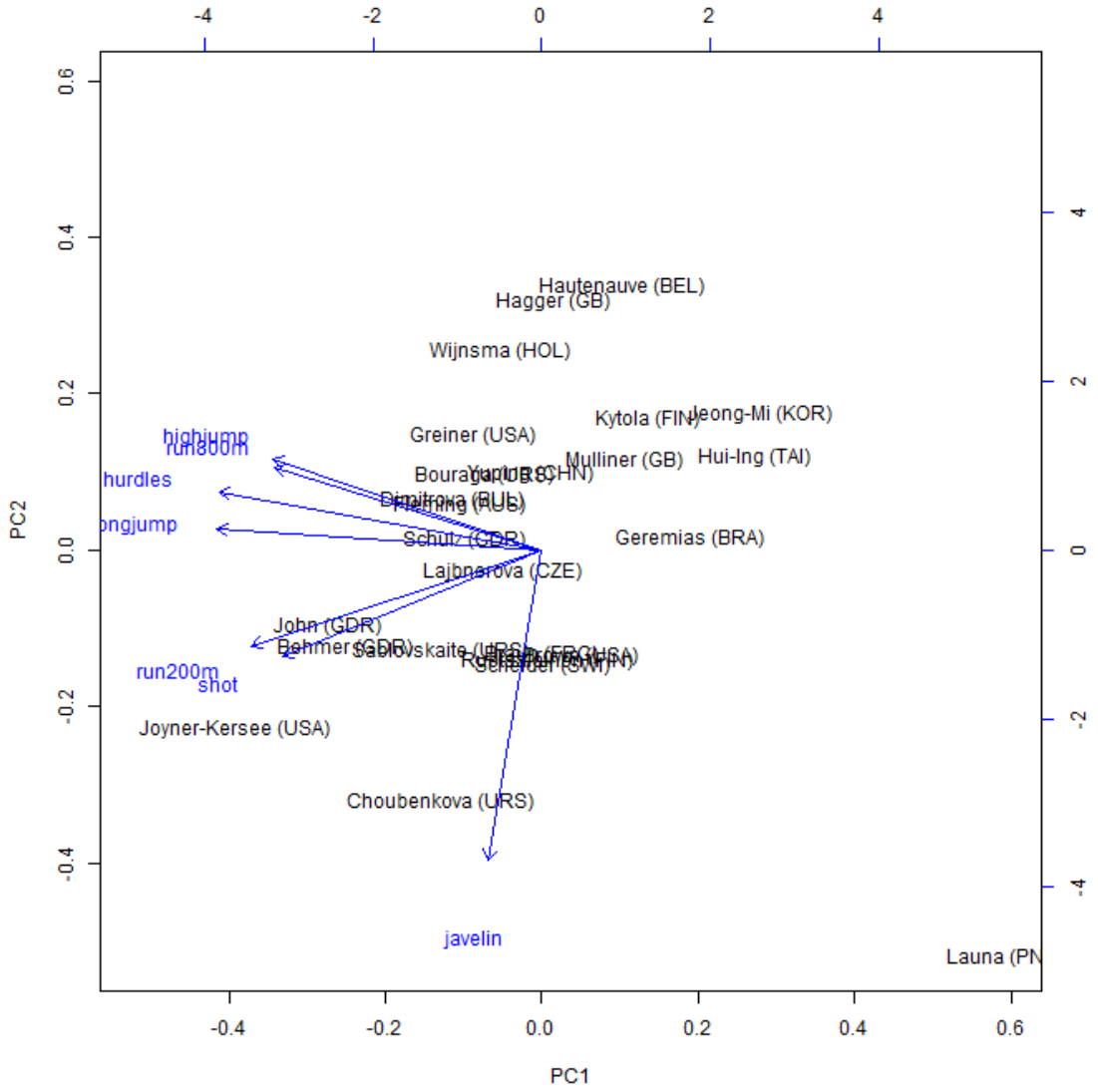
PRO

✚ Can see and compare groupings, distances etc.

CON

– Results depend on **how** you project

– Reasoning about new 2D dimensions can be difficult



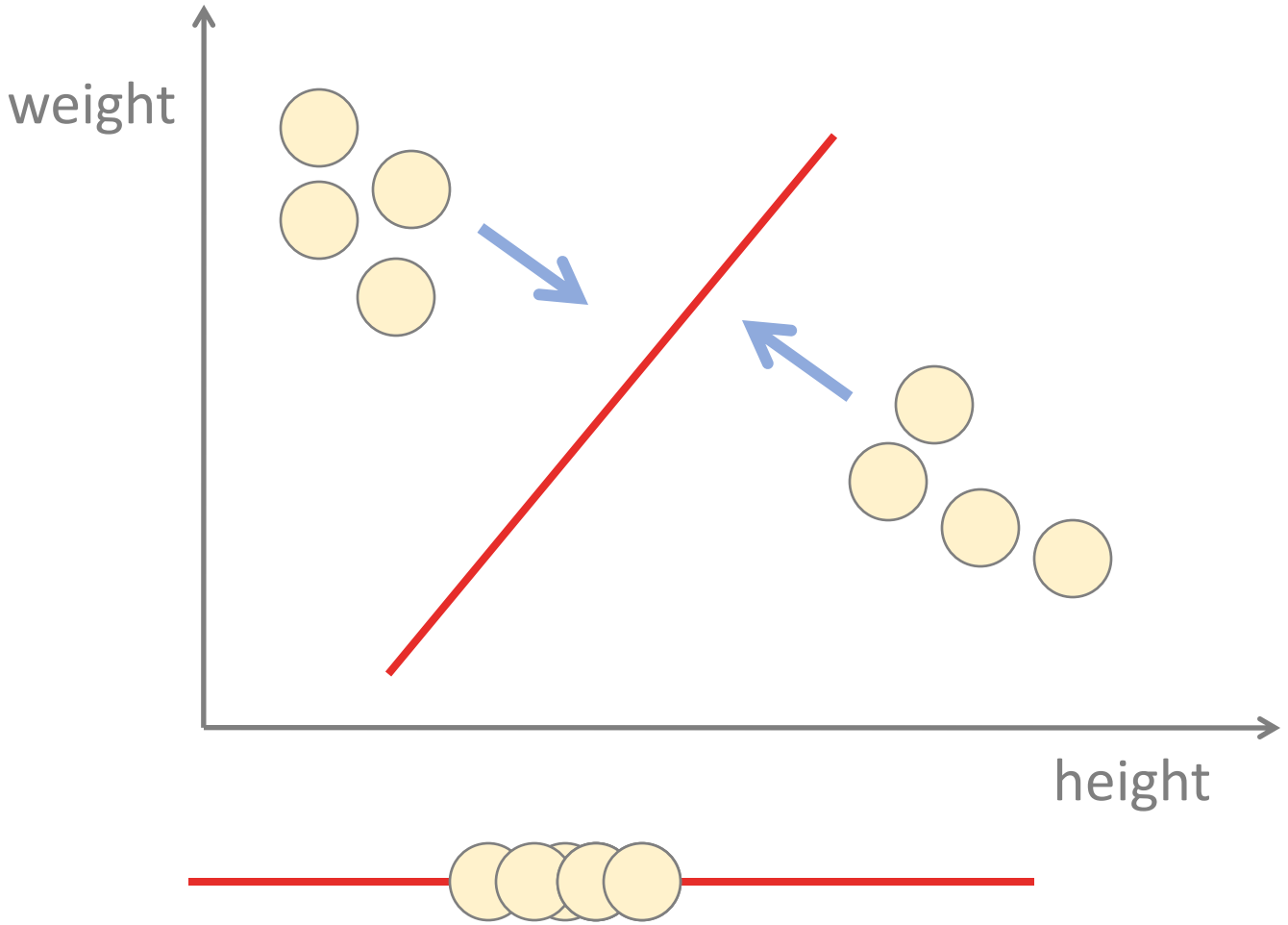
TODAY - A FEW APPROACHES

Principal Components Analysis (PCA)

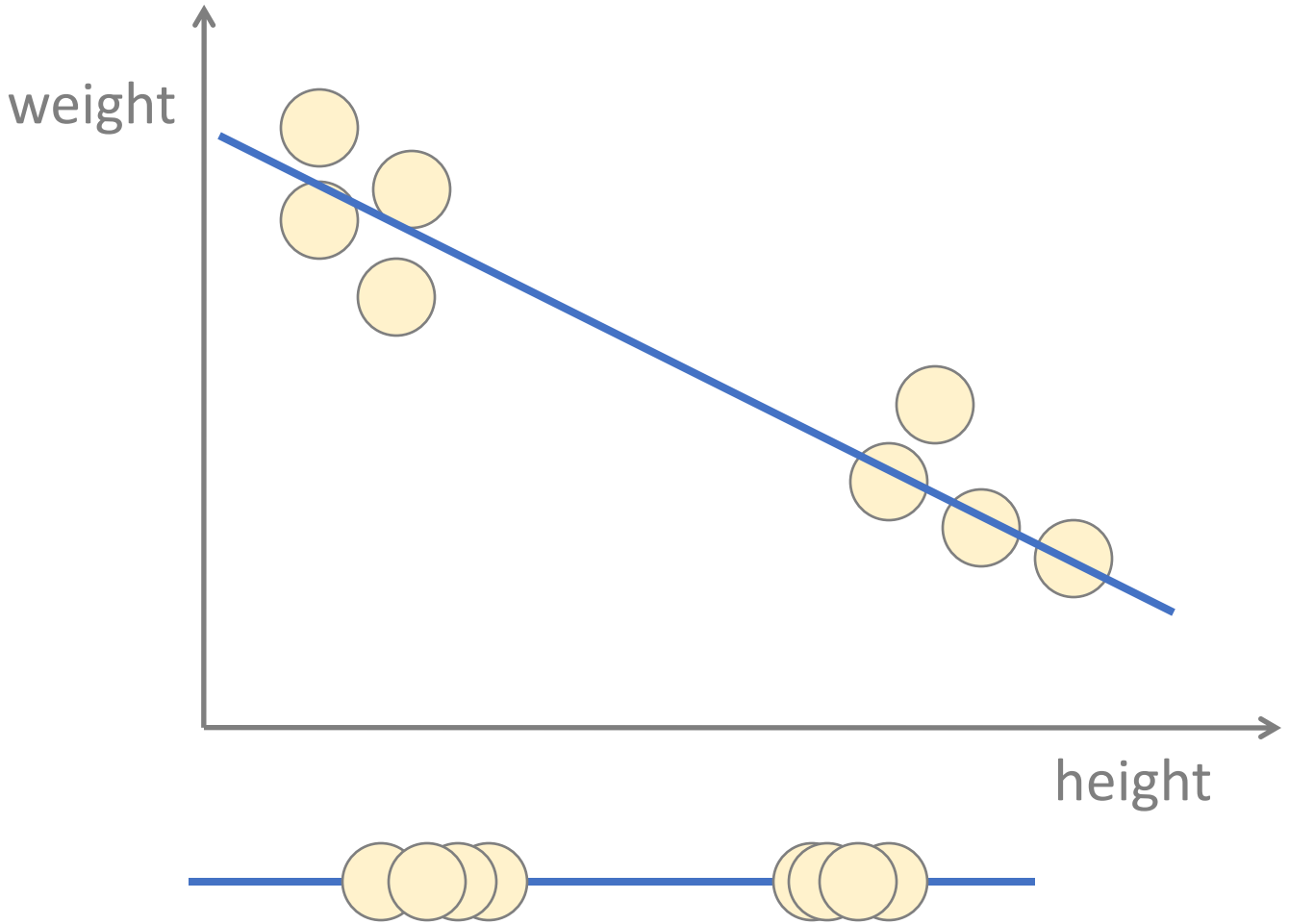
T-distributed Stochastic Neighbor Embedding (T-SNE)

(Many more exist)

Some slides from Eytan Adar & Collins-Thompson



Goal: draw a line and project the data so that things that are far in 2D are far in 1D



Goal: draw a line and project the data so that things that are far in 2D are far in 1D

DIMENSIONALITY REDUCTION

Combine values from many dimensions into just a few

(preferably while maximizing the discriminability of points)

$$nD \rightarrow 2D/3D$$

DIMENSIONALITY REDUCTION

Optimization problem

Computes the **pair-wise similarity/dissimilarity** of each piece of (multi-dimensional) data

Tries to **project the n-dimensional data to 2 or 3 dimensions**

- Preserves similarity as well as possible in the reduced space

CONSIDER A HEPTATHLETE

Athlete

100m sprint

200m run

110m hurdles

Long jump

High jump

Javelin

Shotput

ARE ALL OF THESE SCORES INDEPENDENT?

Athlete

Probably not..

100m sprint

200m run

110m hurdles

Long jump

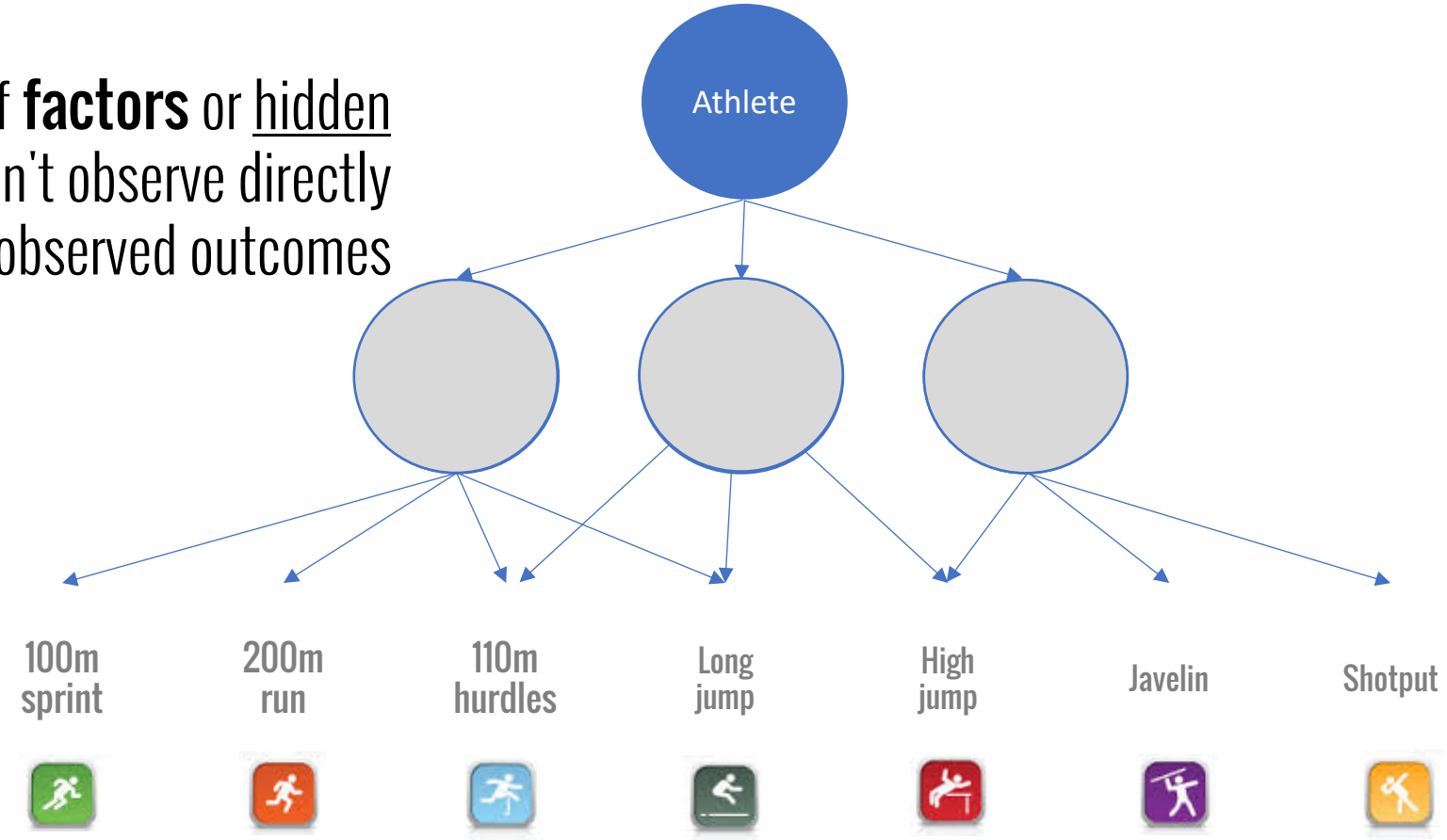
High jump

Javelin

Shotput

ARE ALL OF THESE SCORES INDEPENDENT?

Might be a set of **factors** or hidden variables we don't observe directly but influence the observed outcomes



FACTOR ANALYSIS

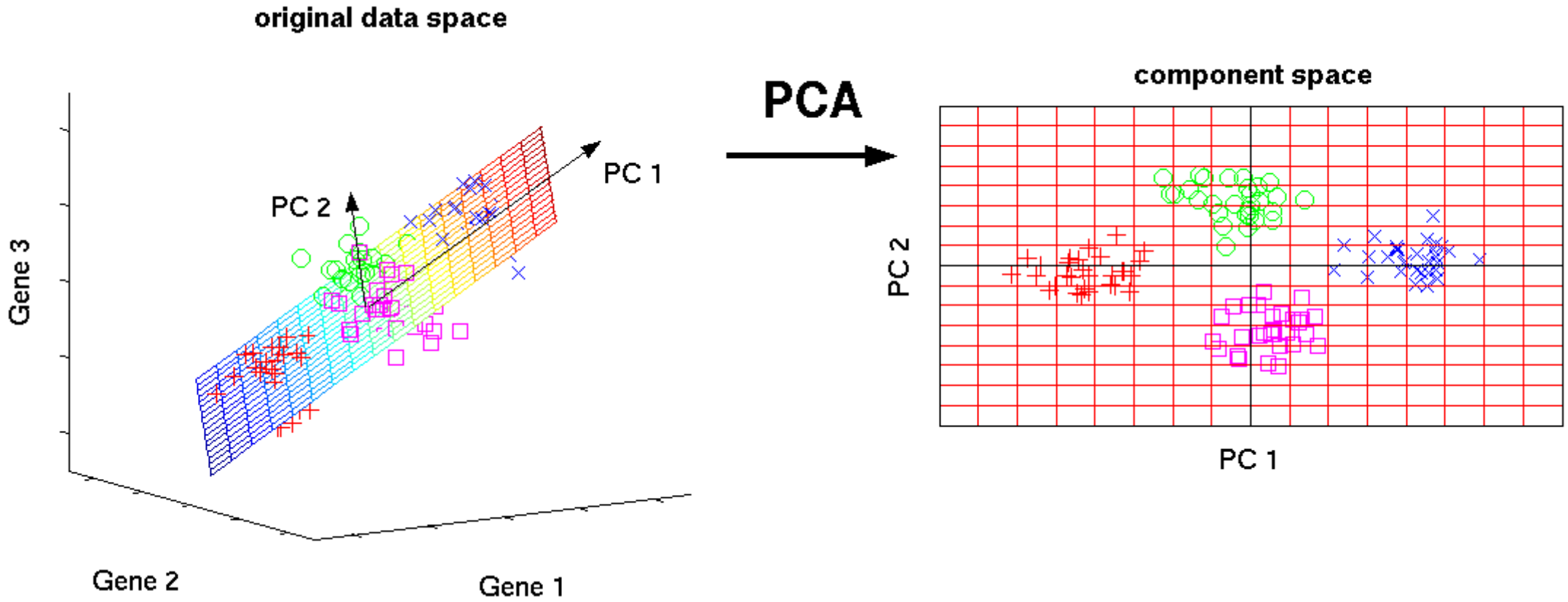
Goal: "explain" many observed variables in terms of a much smaller number of unobserved variables (factors).

Sometimes these hidden factors represent real-world relationships... but often they're just a **simplifying assumption**.

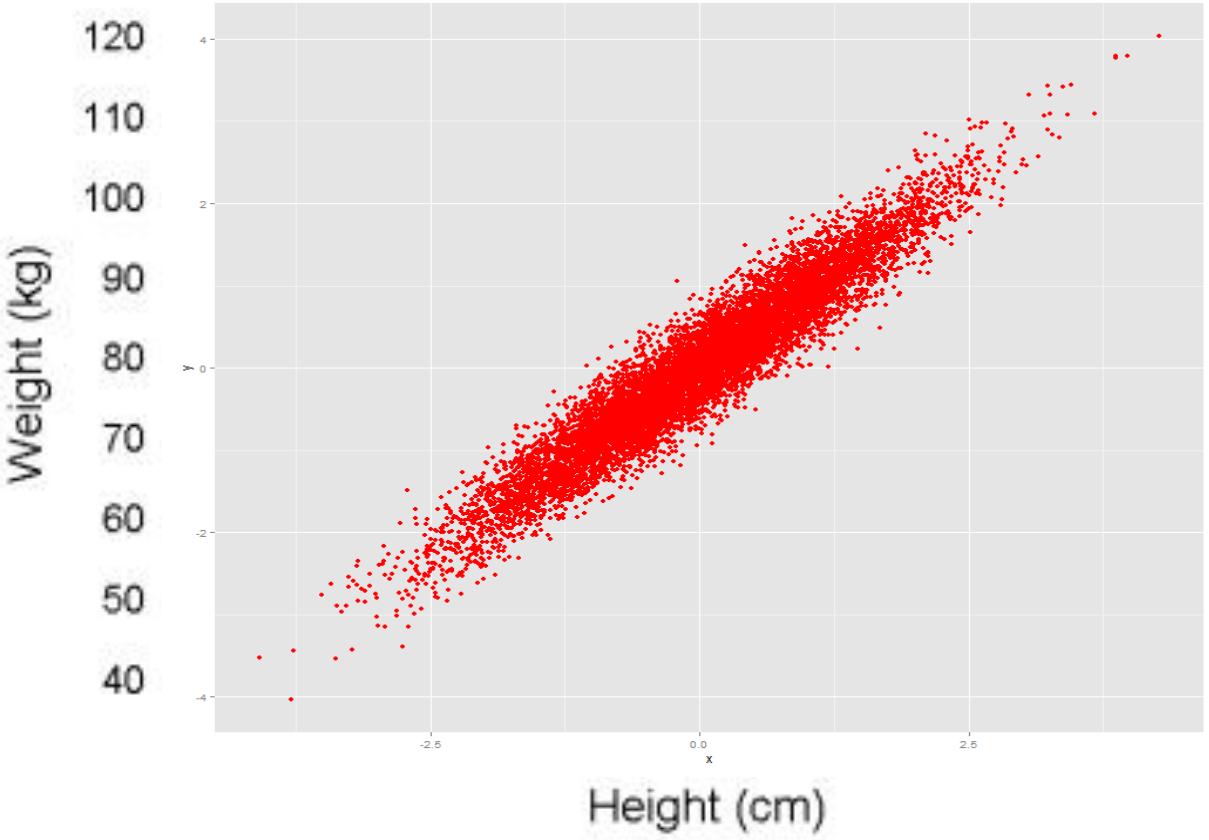
PRINCIPLE COMPONENTS ANALYSIS (PCA)

Project k-dimensional cloud onto a small-dimensional (here 2D) surface

Finds the most "informative" projection

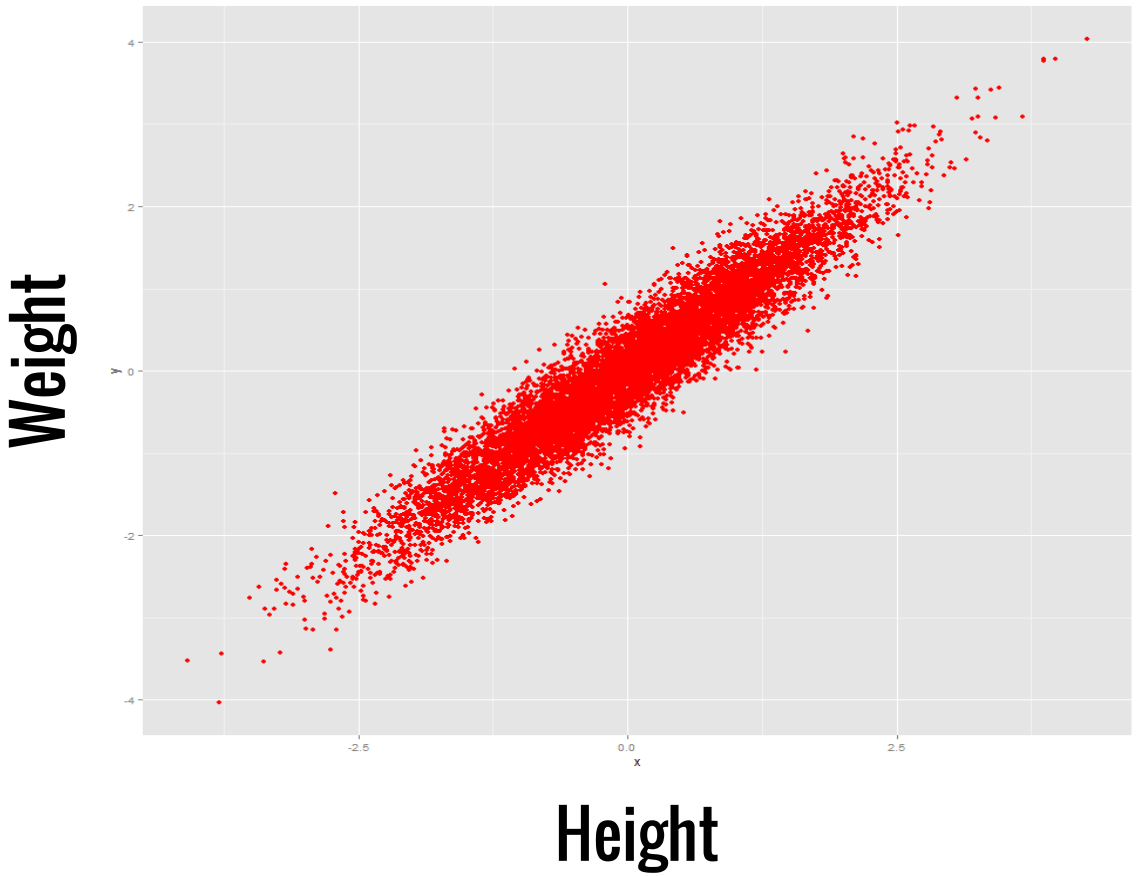


A SIMPLE EXAMPLE



Human weight and height have a 2D distribution (not quite Gaussian but we'll assume they are)

Source: http://www.amstat.org/publications/jse/v11n2/datasets.heinz.html

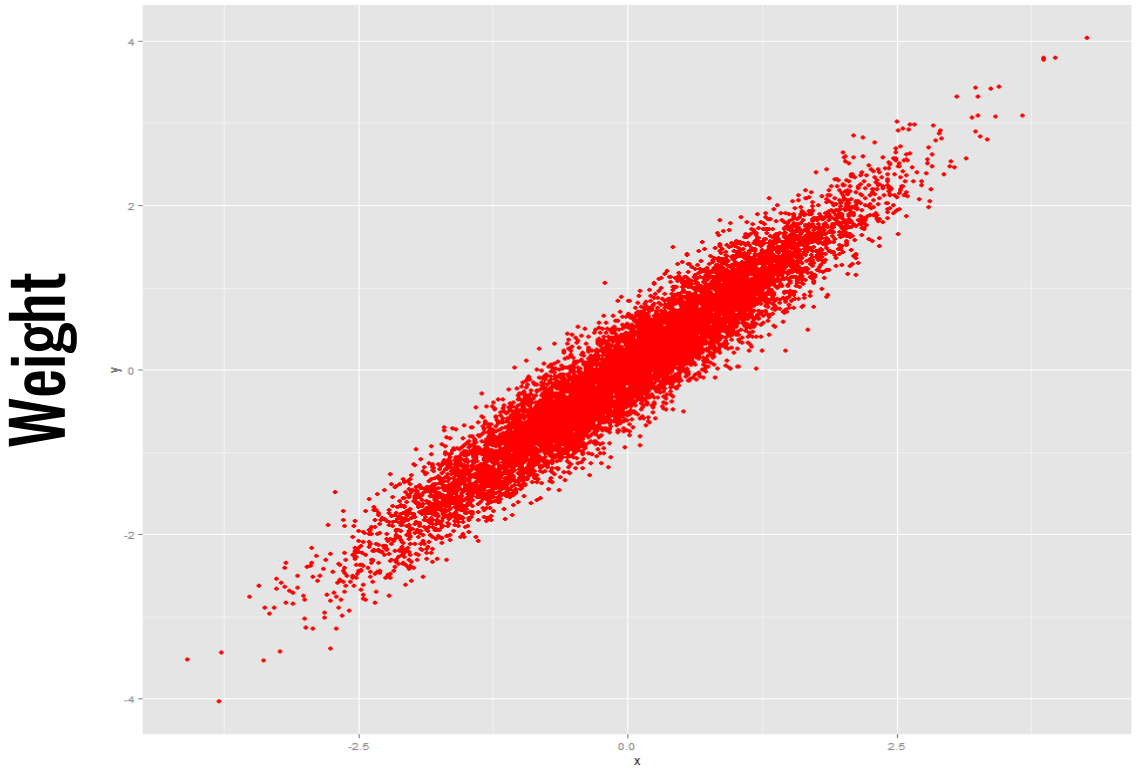


Each point is 2D (x,y)
→ needs 2 numbers

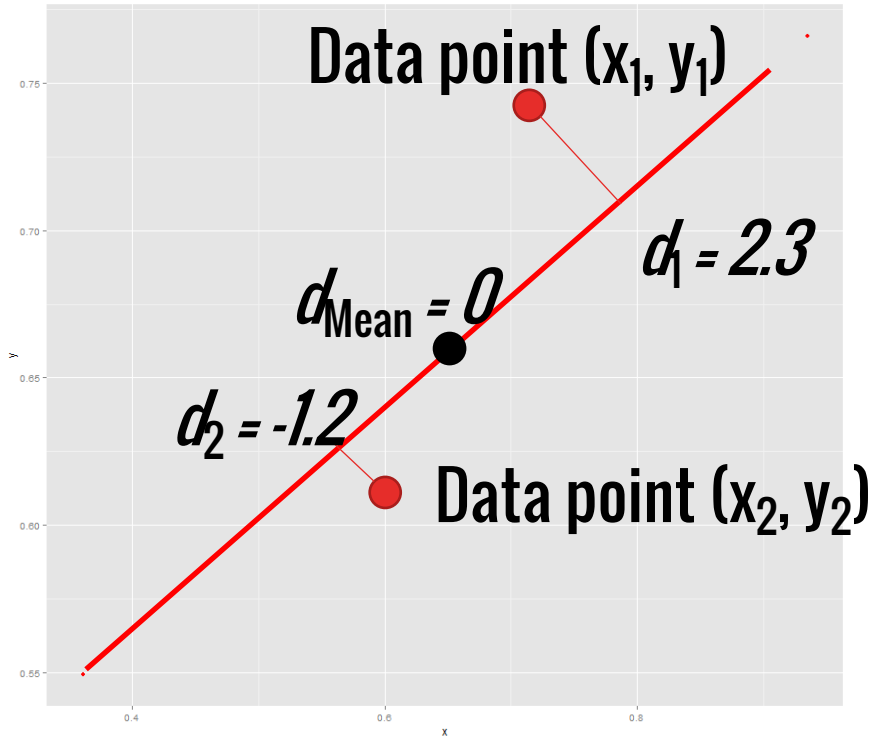
Say you only have space for 1 number

What number would you store to best approximate a point's position?

Idea: Collapse the cloud to 1 dimension, then approximate a point (x_i, y_i) by its projected position d_i onto the line.



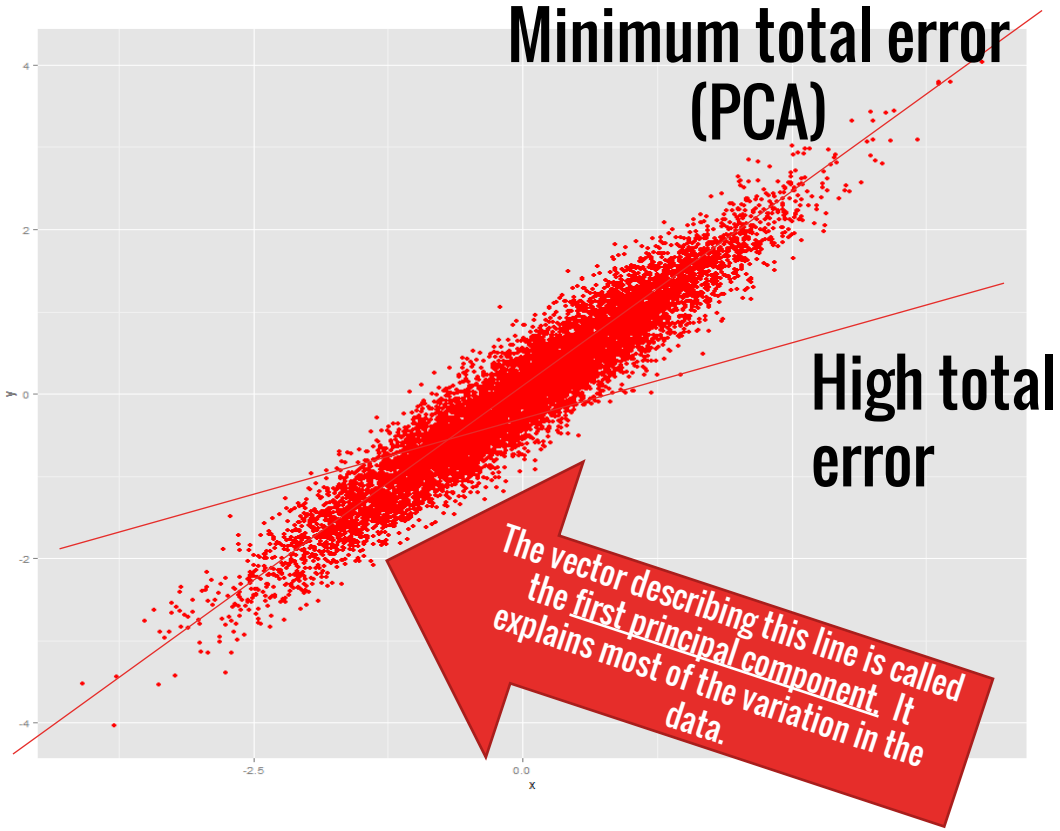
2-dimensional cloud

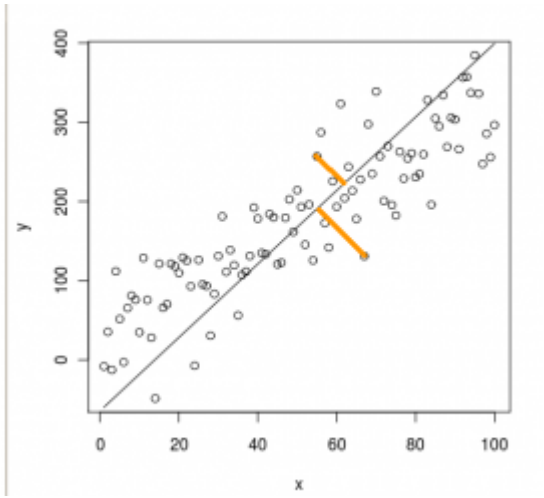


A 1-dimensional approximation to the cloud (using a linear model)

Fit the 'best' linear approximation

(our first "Principle Component")

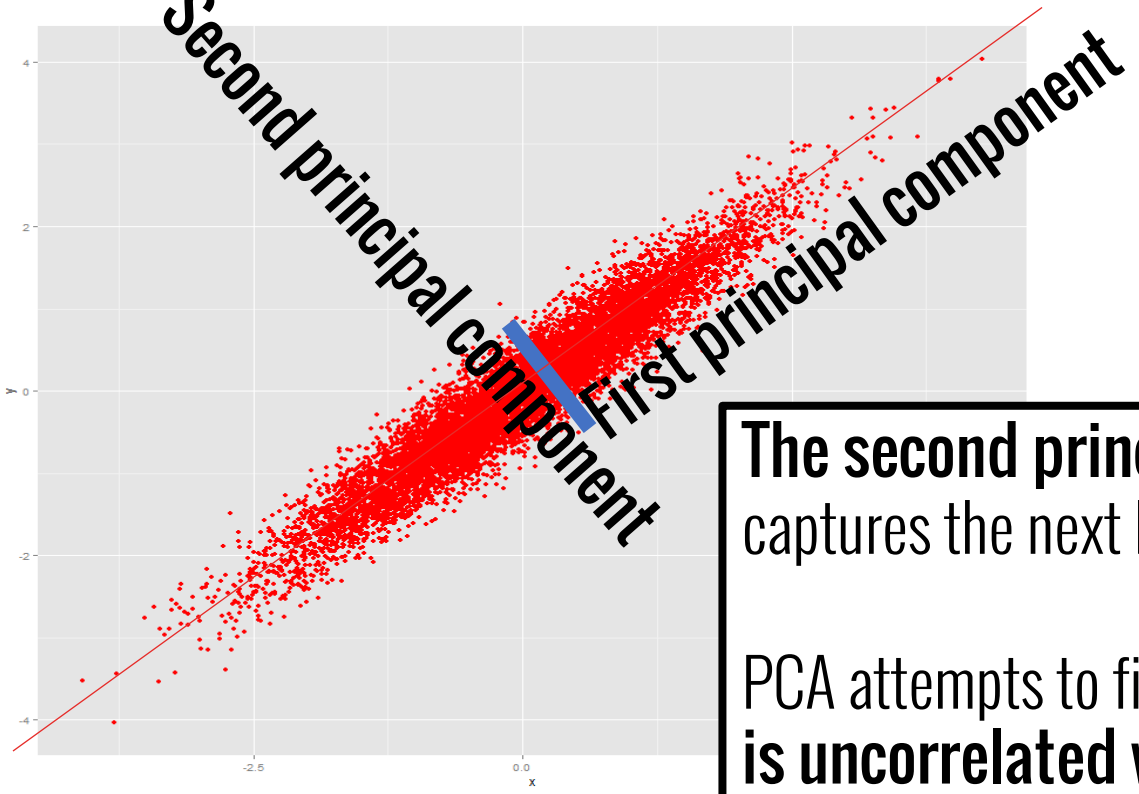


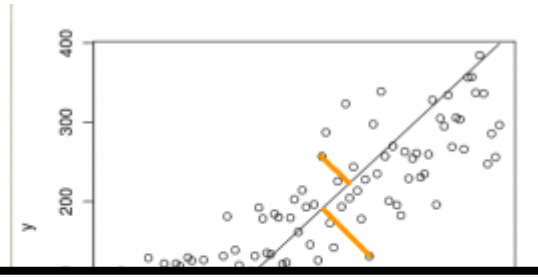


PCA finds the unique model line that minimizes error orthogonal (perpendicular) to the model line.

Source: http://www.cerebralmastication.com/2010/09/principal-component-analysis-pca-vs-ordinary-least-squares-ols-a-visual-explination/

We can repeat this process to improve the approximation. This will give us the **second principal component.**





The second principal component (blue)
captures the next highest orthogonal direction of variance.

PCA attempts to find the next PC that
is uncorrelated with previous PCs.

The principal component vectors have an origin that is the mean (centroid) of the cloud.

Source: http://www.cerebralmastication.com/2010/09/principal-component-analysis-pca-vs-ordinary-least-squares-ols-a-visual-explination/

SCALING/STANDARDIZATION IS IMPORTANT FOR PCA

PCA is sensitive to the **scaling of the variables**

(like k-means and many other algorithms...)

Variables in different units *must* be scaled (e.g. temperature vs mass).

Variables in the same units but **wildly different variances** *should* be scaled.

TYPICAL PRE-PROCESSING STEPS FOR PCA

Mean subtraction (or"mean centering")

- Ensures that the first principal component describes the direction of maximum variance
- Subtract mean from every point

Variable scaling

- Divide each variable by its standard deviation

Do both

HEPTATHLON DATASET

Scores in 7 events: Seoul 1988 Olympics

```
> heptathlon
                  hurdles highjump  shot run200m longjump javelin run800m score
Joyner-Kersee (USA)   12.69     1.86 15.80   22.56     7.27   45.66  128.51  7291
John (GDR)            12.85     1.80 16.23   23.65     6.71   42.56  126.12  6897
Behmer (GDR)          13.20     1.83 14.20   23.10     6.68   44.54  124.20  6858
Sablovskaite (URS)    13.61     1.80 15.23   23.92     6.25   42.78  132.24  6540
Choubenkova (URS)     13.51     1.74 14.76   23.93     6.32   47.46  127.90  6540
Schulz (GDR)          13.75     1.83 13.50   24.65     6.33   42.82  125.79  6411
Fleming (AUS)         13.38     1.80 12.88   23.59     6.37   40.28  132.54  6351
Greiner (USA)         13.55     1.80 14.13   24.48     6.47   38.00  133.65  6297
Lajbnerova (CZE)      13.63     1.83 14.28   24.86     6.11   42.20  136.05  6252
Bouraga (URS)         13.25     1.77 12.62   23.59     6.28   39.06  134.74  6252
Wijnsma (HOL)         13.75     1.86 13.01   25.03     6.34   37.86  131.49  6205
Dimitrova (BUL)       13.24     1.80 12.38   23.59     6.37   40.28  132.54  6171
Scheider (SWI)        13.85     1.86 11.58   24.87     6.05   47.50  134.93  6137
Braun (FRG)           13.71     1.83 13.16   24.78     6.12   44.58  142.82  6109
```

STEP 1: TRANSFORM THE DATA

Seven events have **very different variances**

- Standard deviation for the 800m is 8.29 (sec)
- For high jump it's only 0.078 (m)
- If unscaled scores, the 800m results will have a disproportionate effect.

Also: Some results are measured in seconds (lower numbers better), others in scores, or meters (higher numbers better)

Normalization is important

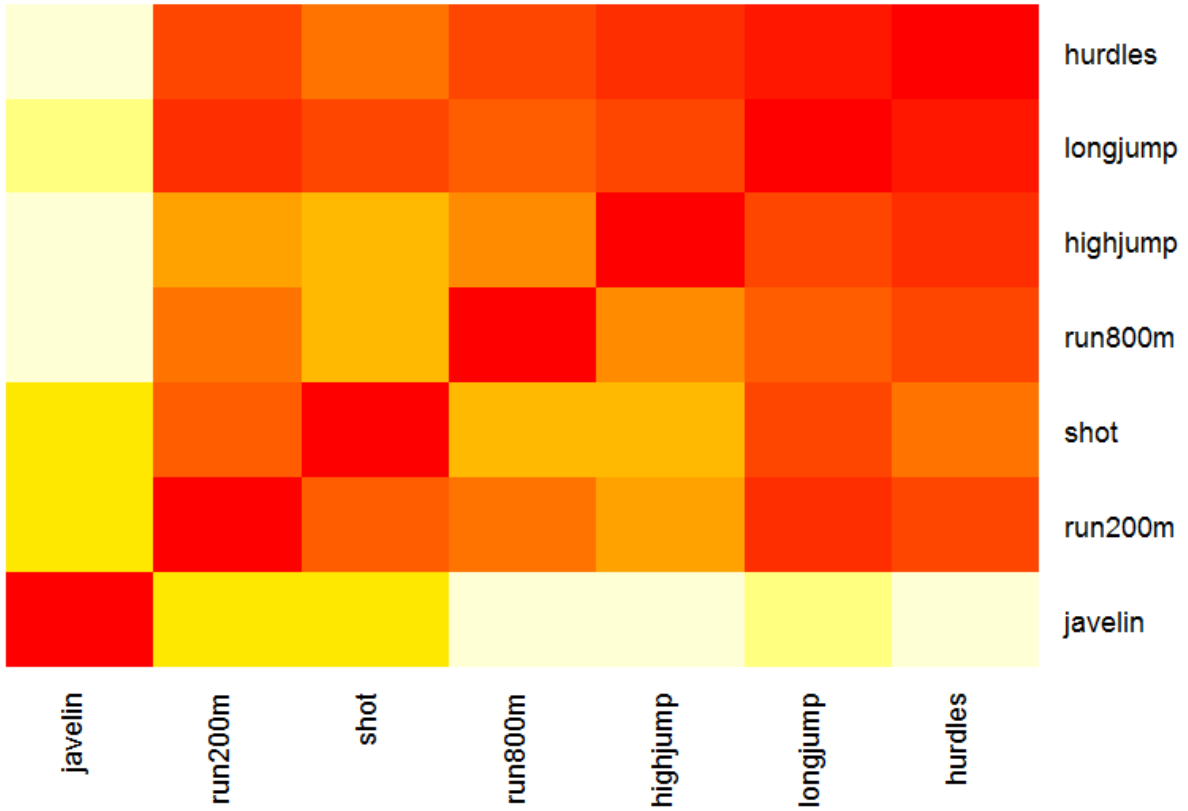
Clean it up!

STEP 2: CHECK FOR CORRELATIONS

	hurdles	highjump	shot	run200m	longjump	javelin	run800m
Hurdles	1.00	0.81	0.65	0.77	0.91	0.01	0.78
Highjump	0.81	1.00	0.44	0.49	0.78	0.00	0.59
shot	0.65	0.44	1.00	0.68	0.74	0.27	0.42
Run200m	0.77	0.49	0.68	1.00	0.82	0.33	0.62
Longjump	0.91	0.78	0.74	0.82	1.00	0.07	0.70
javelin	0.01	0.00	0.27	0.33	0.07	1.00	-0.02
run800m	0.78	0.59	0.42	0.62	0.70	-0.02	1.00

Can you see any structure in the correlation matrix?

Are there groups of related events (variables)?



Could we find a small set of core athletic abilities (factors) that mostly explain the structure of correlated results for a large set of event scores (variables)?

Is there a "running factor" that would lead to good results across multiple running events?

A "jumping factor" ?

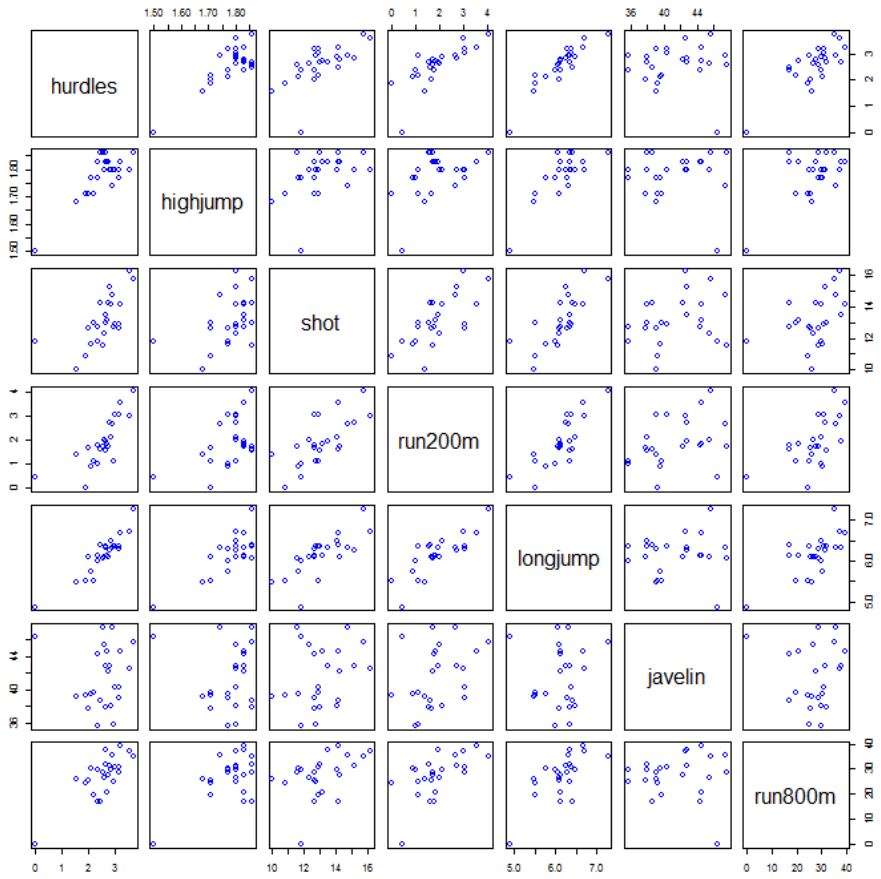
A "throwing factor" (arm strength) ?

Endurance?

Seven variables is not a large number

- PCA comes into its own in larger data sets

Scatterplots confirm what we observed in the correlation matrix



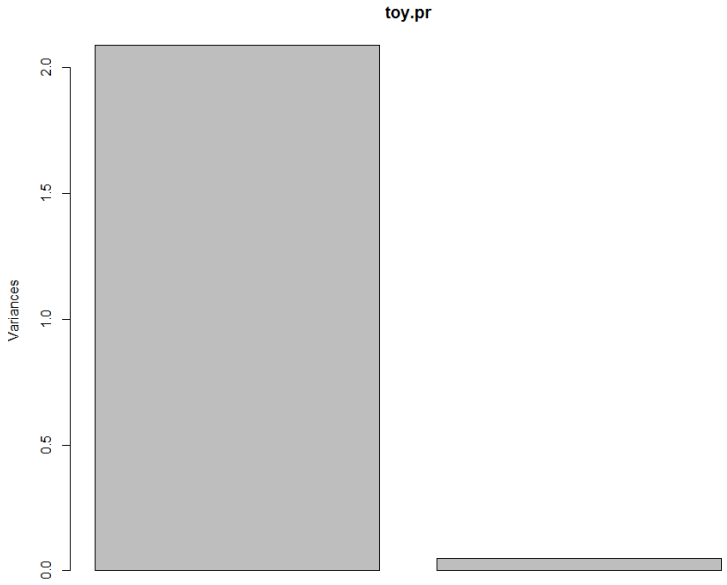

```
pairs(heptathlon[,c(1:7)],col="blue")
```

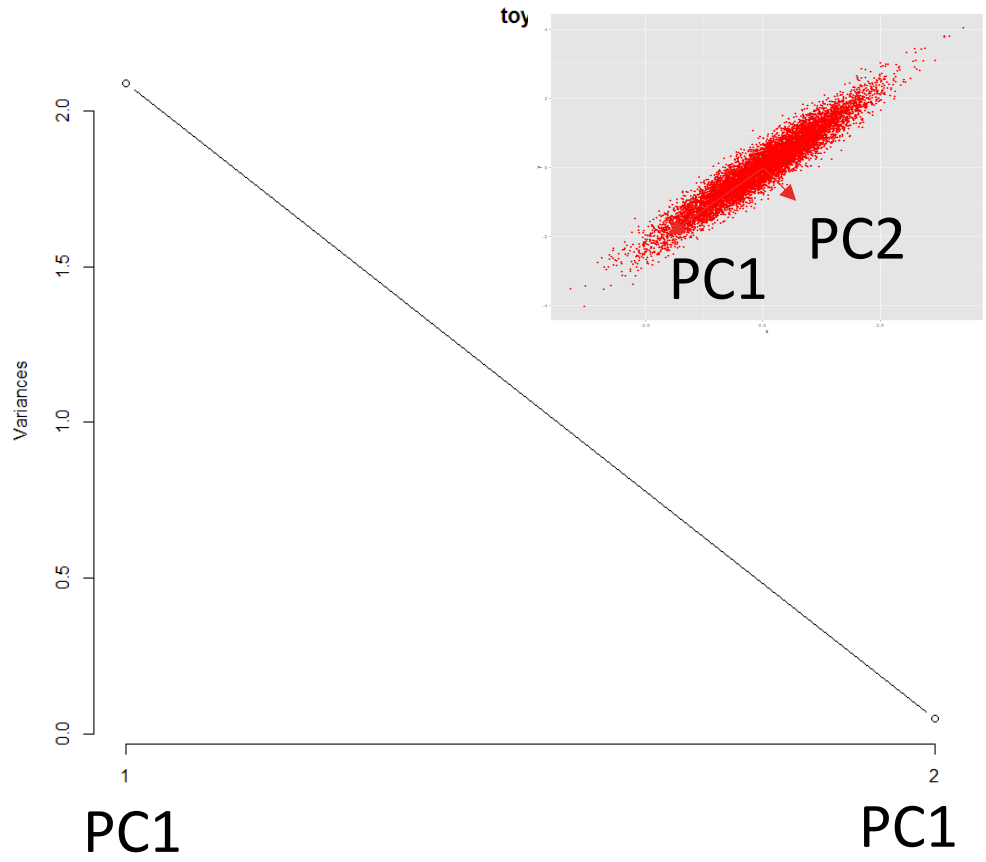
STEP 3: RUN PCA

```
> print(hepPCA)
Standard deviations:
[1] 2.1119364 1.0928497 0.7218131 0.6761411 0.4952441 0.2701029 0.2213617

Rotation:
                PC1         PC2         PC3         PC4         PC5         PC6         PC7
hurdles  -0.4528710  0.15792058 -0.04514996  0.02653873 -0.09494792 -0.78334101  0.38024707
highjump -0.3771992  0.24807386 -0.36777902  0.67999172  0.01879888  0.09939981 -0.43393114
shot     -0.3630725 -0.28940743  0.67618919  0.12431725  0.51165201 -0.05085983 -0.21762491
run200m  -0.4078950 -0.26038545  0.08359211 -0.36106580 -0.64983404  0.02495639 -0.45338483
longjump -0.4562318  0.05587394  0.13931653  0.11129249 -0.18429810  0.59020972  0.61206388
javelin  -0.0754090 -0.84169212 -0.47156016  0.12079924  0.13510669 -0.02724076  0.17294667
run800m  -0.3749594  0.22448984 -0.39585671 -0.60341130  0.50432116  0.15555520 -0.09830963

> summary(hepPCA)
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6    PC7
Standard deviation     2.1119 1.0928 0.72181 0.67614 0.49524 0.27010 0.2214
Proportion of Variance 0.6372 0.1706 0.07443 0.06531 0.03504 0.01042 0.0070
Cumulative Proportion  0.6372 0.8078 0.88223 0.94754 0.98258 0.99300 1.0000
```

SCREEPLOTS

show how much variance is explained by each principal component

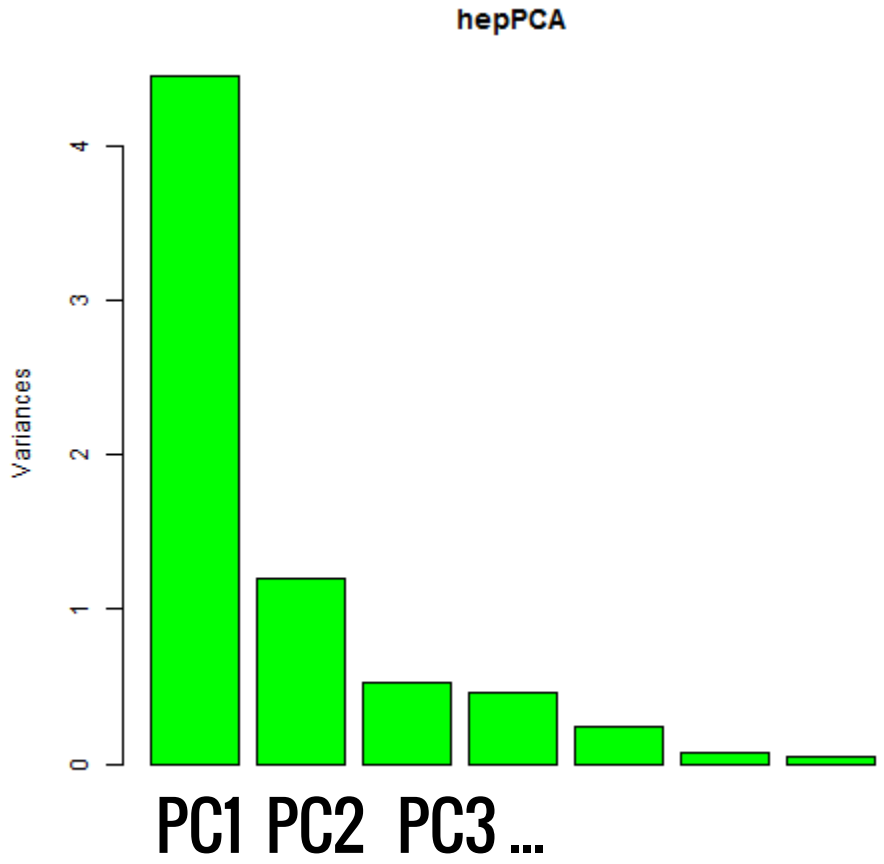




How to pick the "right" number of factors?

Use a screeplot of PCA components
to find the components that explain most of the variance. (Much like to the “elbow” method.)

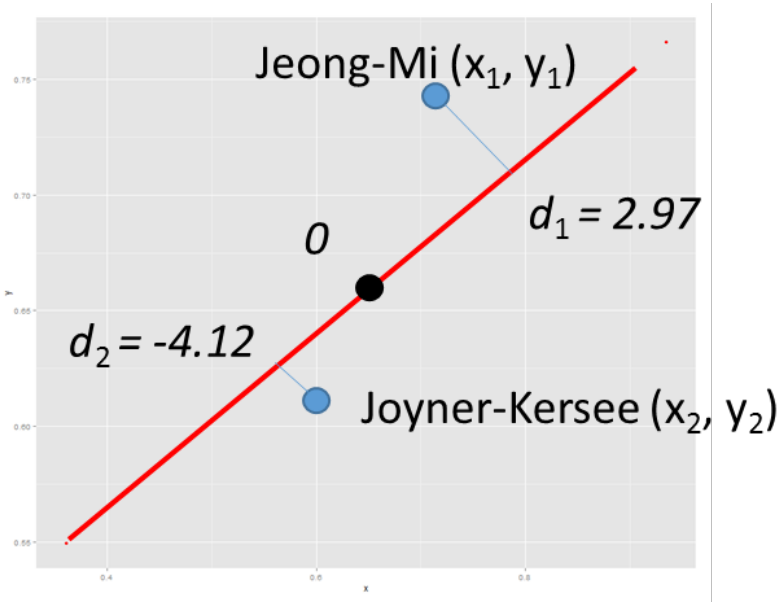
This one suggests **the first PC captures most of the interesting variation in the data.**
Implies that a single factor (skill?) may "explain" performance across multiple events



Can test this: Project each athlete onto PC1 …

Compare to ranking based on original data:

	hurdles	highjump	shot	run200m	longjump	javelin	run800m	score
Joyner-Kersee (USA)	3.73	1.86	15.80	4.05	7.27	45.66	34.92	7291
John (GDR)	3.57	1.80	16.23	2.96	6.71	42.56	37.31	6897
Behmer (GDR)	3.22	1.83	14.20	3.51	6.68	44.54	39.23	6858
Sablovskaite (URS)	2.81	1.80	15.23	2.69	6.25	42.78	31.19	6540
Choubenkova (URS)	2.91	1.74	14.76	2.68	6.32	47.46	35.53	6540
Schulz (GDR)	2.67	1.83	13.50	1.96	6.33	42.82	37.64	6411
Fleming (AUS)	3.04	1.80	12.88	3.02	6.37	40.28	30.89	6351
Greiner (USA)	2.87	1.80	14.13	2.13	6.47	38.00	29.78	6297
Lajbnerova (CZE)	2.79	1.83	14.28	1.75	6.11	42.20	27.38	6252
Bouraga (URS)	3.17	1.77	12.62	3.02	6.28	39.06	28.69	6252
Wijnsma (HOL)	2.67	1.86	13.01	1.58	6.34	37.86	31.94	6205
Dimitrova (BUL)	3.18	1.80	12.88	3.02	6.37	40.28	30.89	6171
Scheider (SWI)	2.57	1.86	11.58	1.74	6.05	47.50	28.50	6137
Braun (FRG)	2.71	1.83	13.16	1.83	6.12	44.58	20.61	6109



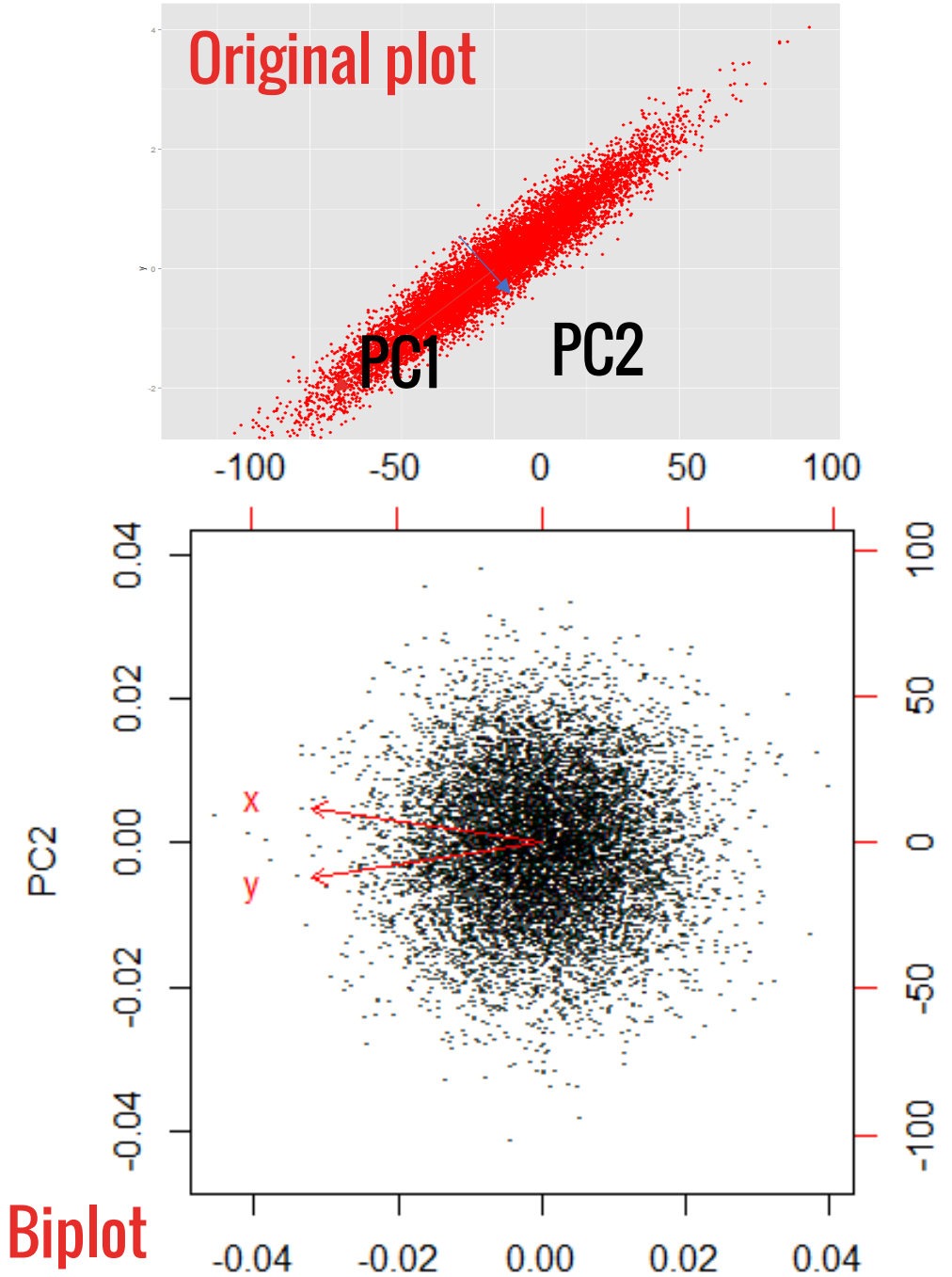
The PCA-based score captures the actual score ranking well.

VISUALIZING PCA RESULTS WITH **BIPLOTS**

Re-plot the data using the Principle Components as your X and Y.

Plot the values **as points** and the original data dimensions **as vectors**

The angle between the vectors for any two variables reflects their actual pairwise correlation

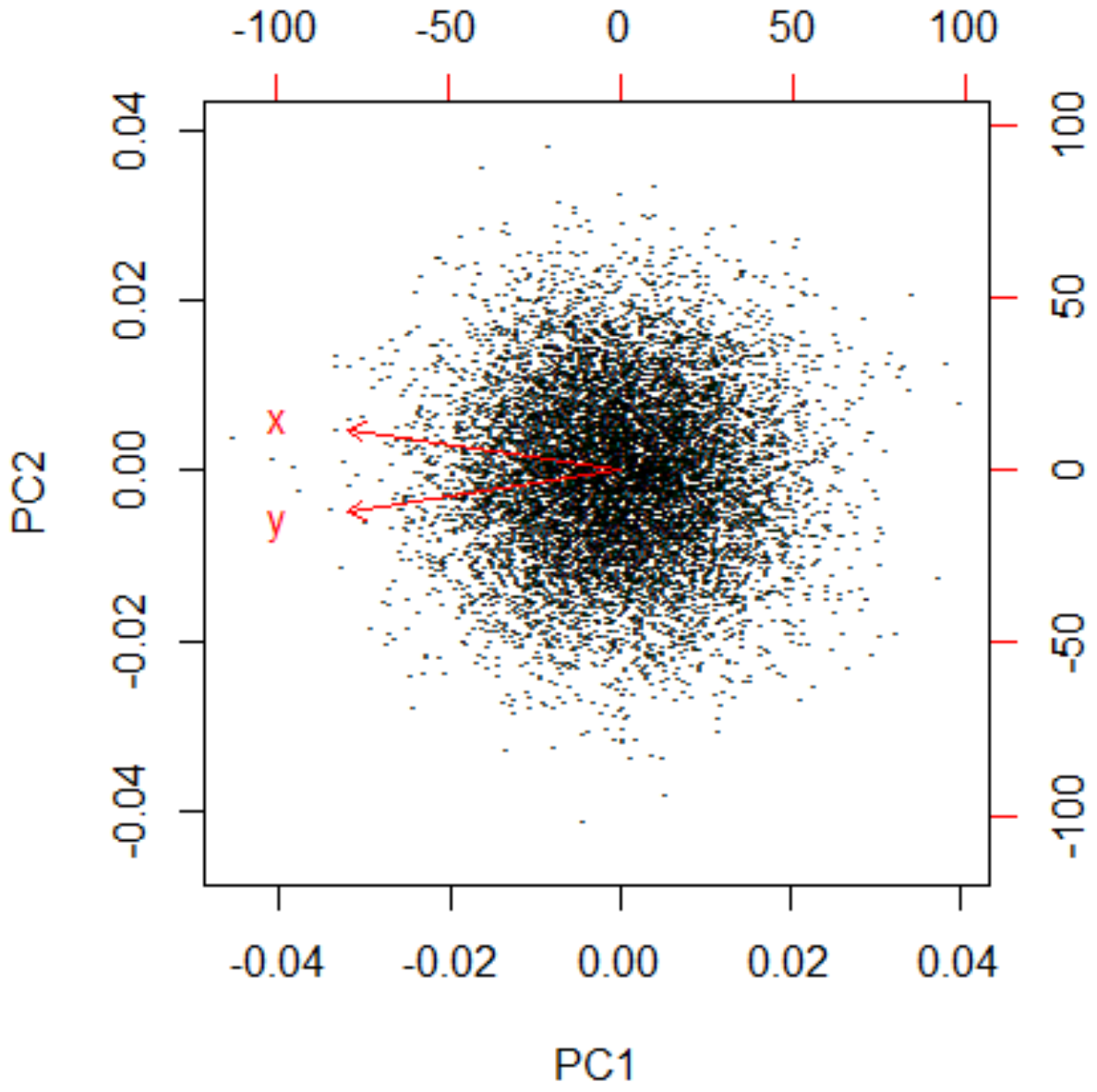


READING BIPLOTS

Close points= Observations with similar component projections

Close Vectors =
Variables that are correlated

Observations whose points project furthest in the direction of a variable have the most of whatever the variable measures.

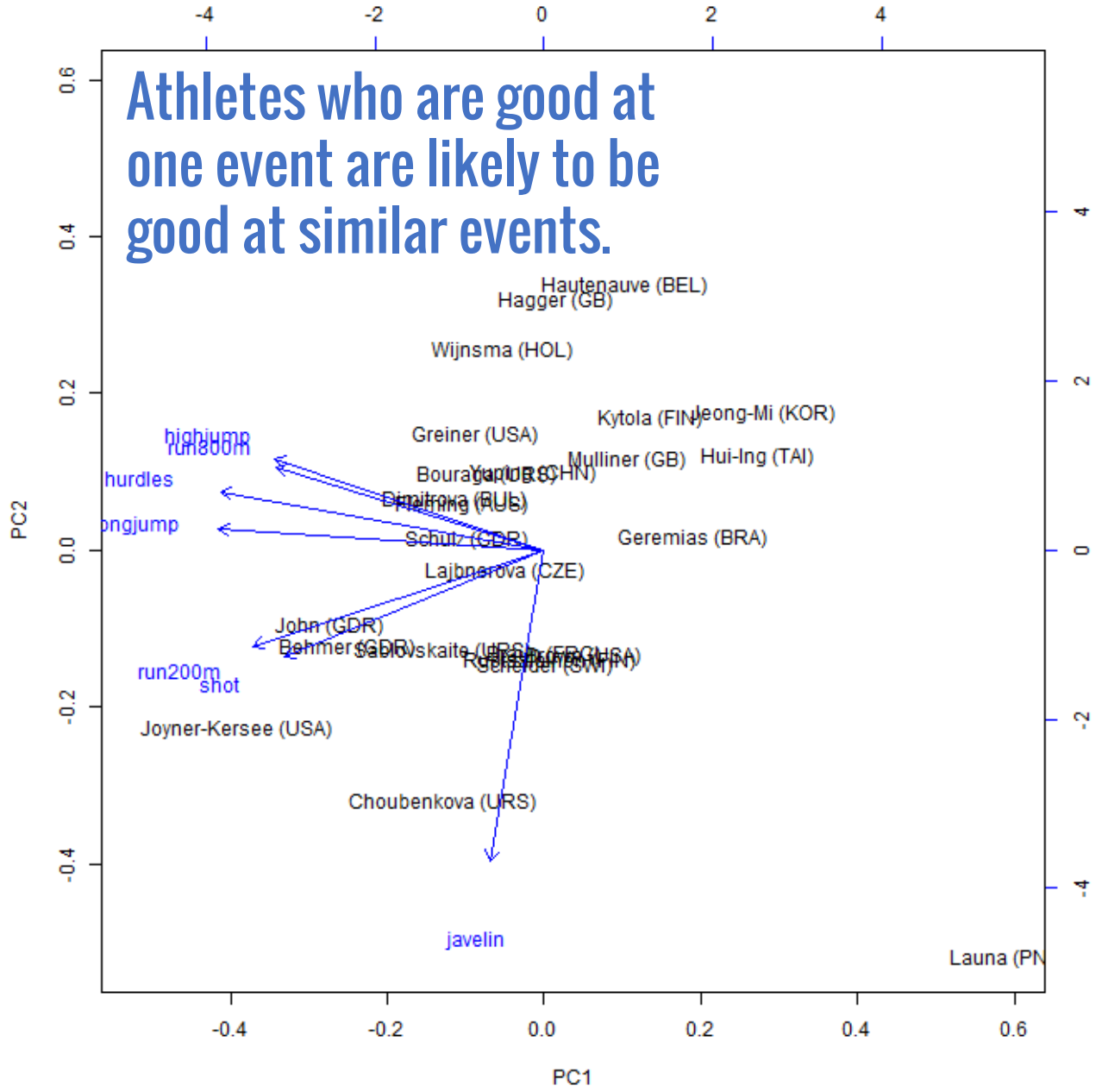


READING BIPLOTS

Close points= Observations with similar component projections

Close Vectors =
Variables that are correlated

Observations whose points project furthest in the direction of a variable have the most of whatever the variable measures.



PCA USING PYTHON

Scaling

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
df_scaled = scale.fit_transform(my_dataframe)
```

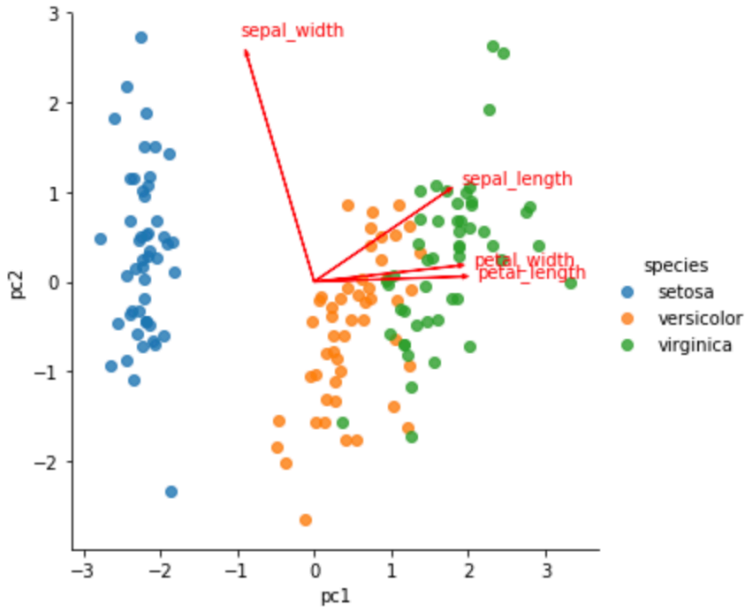
PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
principle_components = pca.fit_transform(df_scaled)
```

PCA WITH PYTHON

Plotting

(with original feature vectors)




```
# Plot using the Principle Components as Axes
sns.lmplot('pc1', 'pc2', df_pca, hue='species', fit_reg=False)

# set the maximum variance of the first two PCs
# this will be the end point of the arrow of each **original feature**
xvector = pca.components_[0]
yvector = pca.components_[1]

# value of the first two PCs, set the x, y axis boundary
xs = pca.transform(df_scaled)[:,0]
ys = pca.transform(df_scaled)[:,1]

# arrows project features (columns from csv) as vectors onto PC axes
for i in range(len(xvector)):
    plt.arrow(0, 0, xvector[i]*max(xs), yvector[i]*max(ys),
              color='r', width=0.005, head_width=0.05)
    plt.text(xvector[i]*max(xs)*1.1, yvector[i]*max(ys)*1.1,
             list(df_iris.columns.values)[i], color='r')
```

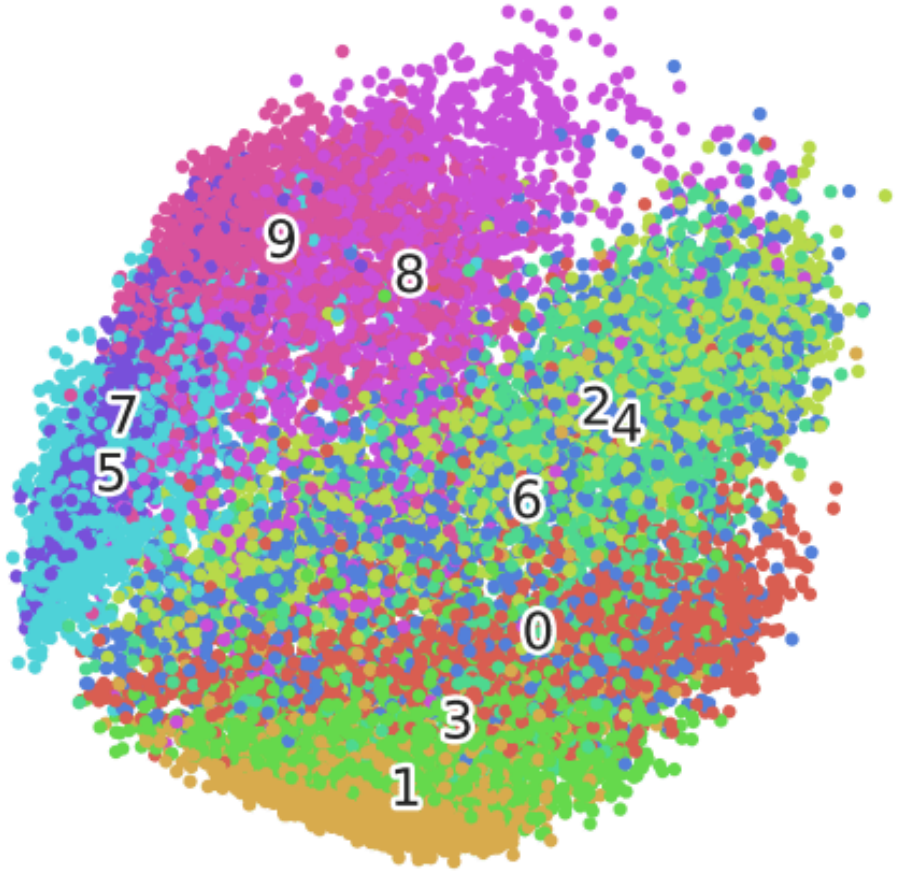
T-SNE (T-distributed Stochastic Neighbor Embedding)

A popular modern non-linear approach for creating **embeddings** for high-dimensional datasets.

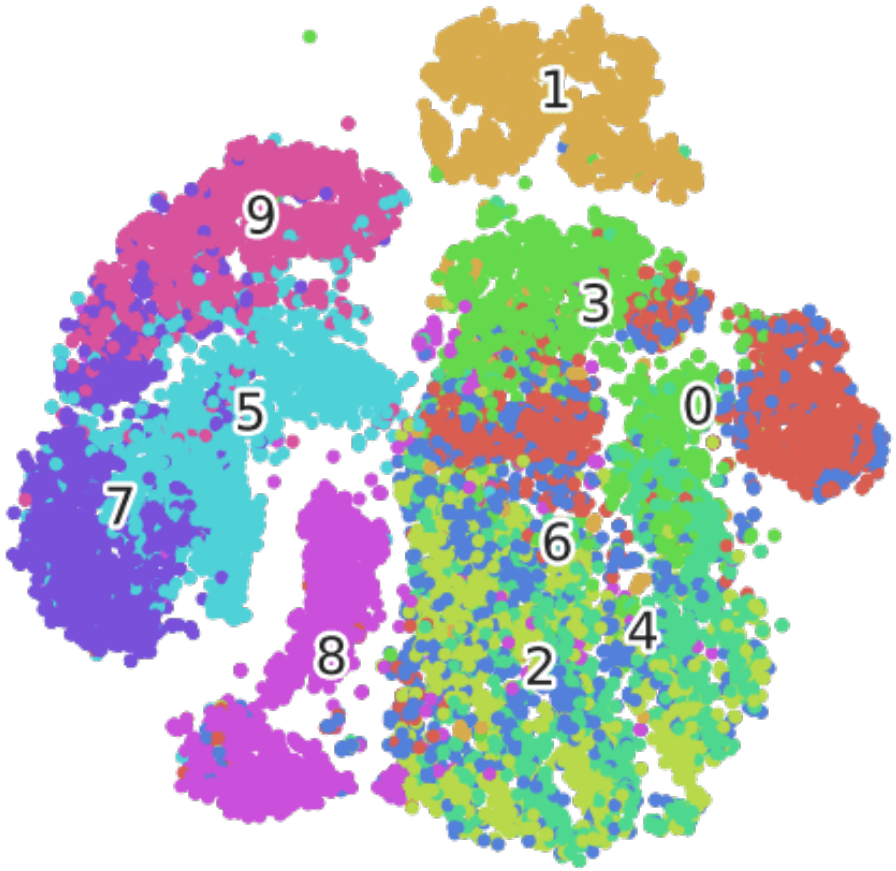
Adapts to underlying data and **performs different transformations on different regions**.

You should be aware of this ...

but probably shouldn't use it until you're ready.



PCA



T-SNE

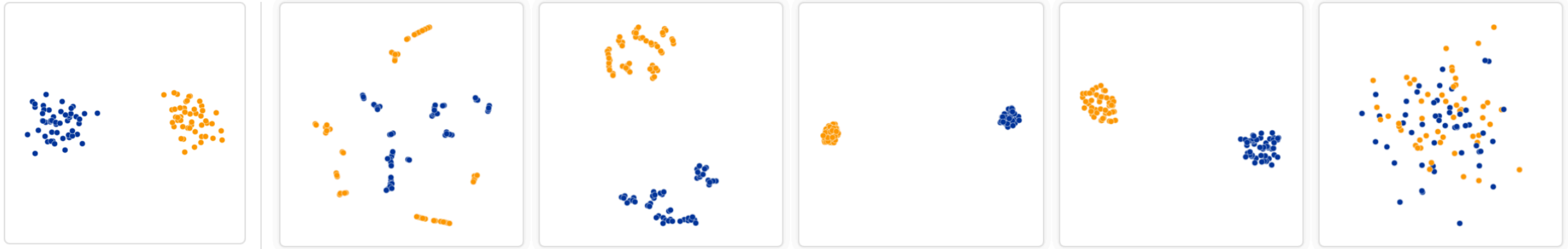
Uses local relationships between points to create a low-dimensional mapping that can capture non-linear structure.

CAN REVEAL STRUCTURE

IN REALLY COMPLEX HIGH-DIMENSIONAL DATASETS

BUT...

- DOESN'T PRODUCE A CONSISTENT PROJECTION
- REALLY SENSITIVE TO PARAMETER CHANGES



Wattenber, Viégas, Johnson (2016)

EASY TO MISINTERPRET

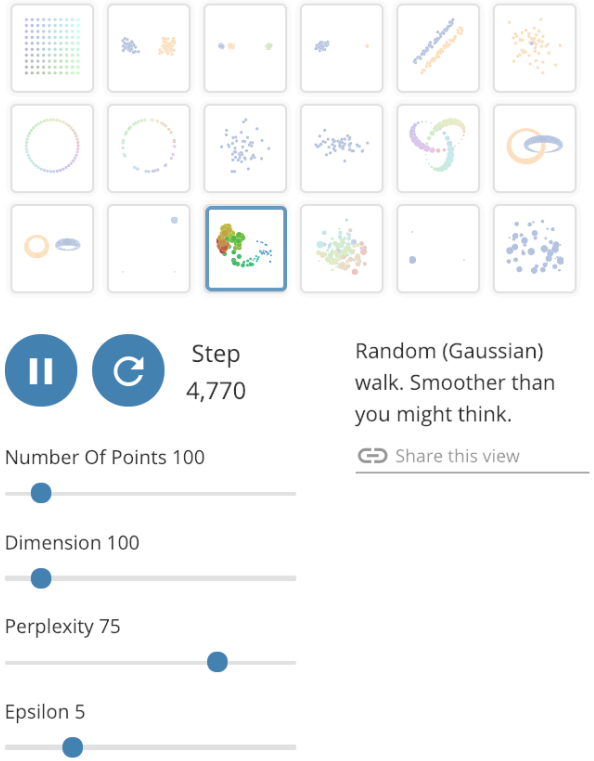
WHERE YOU STOP CAN GIVE YOU DIFFERENT RESULTS!

MULTIPLE RUNS CAN GIVE YOU DIFFERENT RESULTS!

CLUSTER SIZES AND DISTANCES **MIGHT NOT MEAN ANYTHING**

How to Use t-SNE Effectively

Although extremely useful for visualizing high-dimensional data, t-SNE plots can sometimes be mysterious or misleading. By exploring how it behaves in simple cases, we can learn to use it more effectively.



MARTIN WATTENBERG
Google Brain

FERNANDA VIÉGAS
Google Brain

IAN JOHNSON
Google Cloud

Oct. 13
2016

Citation:
Wattenberg, et al., 2016

https://distill.pub/2016/misread-tsne/

USING DIMENSIONALITY REDUCTION **WITH CLUSTERING/CLASSIFICATION**

Should you use dimensionality reduction **before** or **after**?
... it depends.

If you reduce before:

+ Compresses data and can make clustering/classification more efficient (especially for approaches like k-NN).
+ Clusters and labels often easier to interpret.
– Throws away valuable data that might help you cluster/classify.