Implementing Factor Analysis and PCA in Python

Vitthal Srinivasan
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Assemble a data set of returns from correlated stocks

Use Python to calculate principal components of the financial data

Eliminate low-value principal components using eigenvalues

Relate the principal components to underlying latent factors

PCA in Python

Explain Google’s returns
Yahoo finance

Using returns of correlated stocks

Covariance and Correlation
Correlation matrix signals trouble

Multicollinearity problems

Eigen Decomposition
Python library function

On covariance matrix

Scree Plot
Number of dimensions

Discard low-value dimensions

Principal Components
From eigen vectors

Uncorrelated components

Interpret and Regress
Beta, bonds, sectors

Now regress Google

Demo

Implement Eigen analysis and PCA in Python

Negative Indices in R

goog

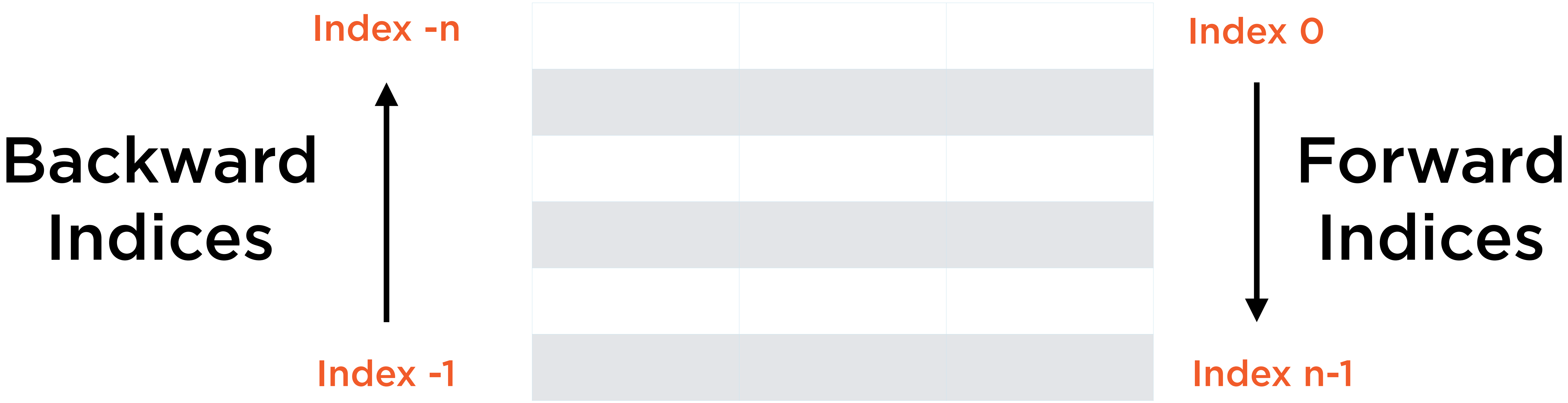
DATE	GOOG. PRICE	NASDAQ. PRICE	
2016-12-01	779	5550	Row 1
2016-11-01	747	5324	
2006-01-01	309	1900	Row nrow(goog)

Exclude

Column 1

```
goog[-nrow(goog),-1]
```

Negative Indices In Python



PCA should always be applied on the covariance matrix of standardised vectors

Standardising Data

$$\begin{bmatrix} X_{11} & & X_{1k} \\ X_{21} & & X_{2k} \\ X_{31} & \dots & X_{3k} \\ \dots & & \dots \\ X_{n1} & & X_{nk} \end{bmatrix}$$

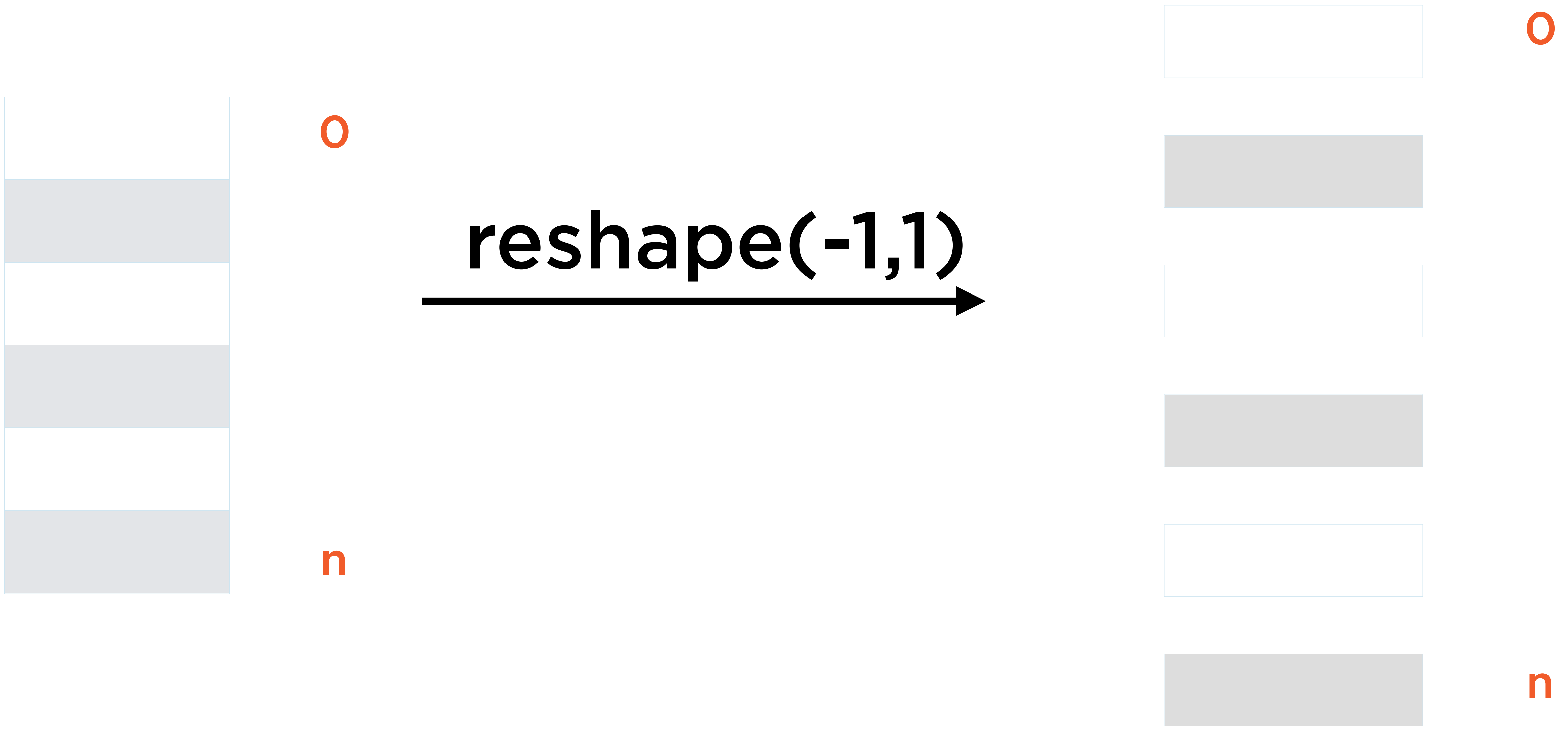
$$\begin{array}{ccc} avg(X_1) & \dots & avg(X_k) \\ stdev(X_1) & \dots & stdev(X_k) \end{array}$$

Standardising Data

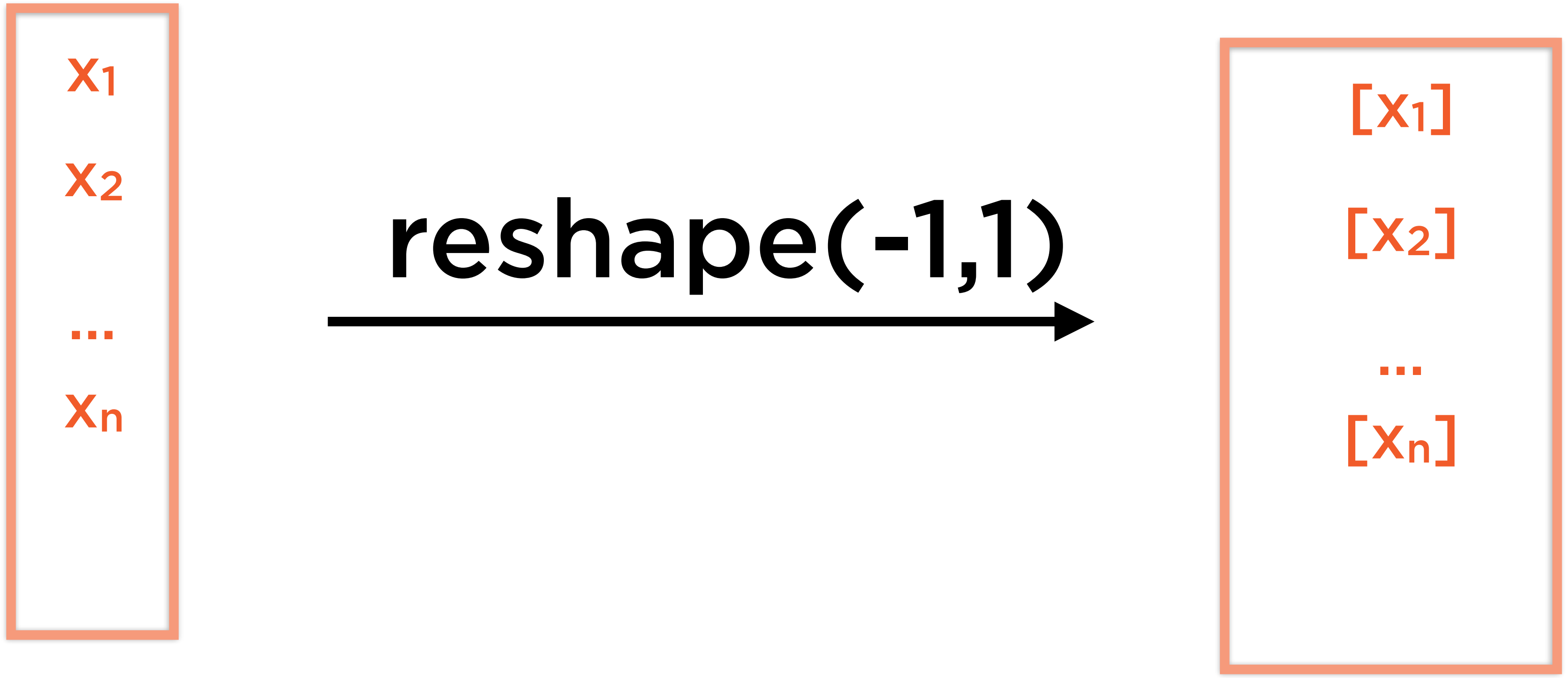
$$
\begin{bmatrix}
\dfrac{x_{11} - avg(X_1)}{stdev(X_1)} & \cdots & \dfrac{x_{1k} - avg(X_k)}{stdev(X_k)} \\
\cdots & & \cdots \\
\dfrac{x_{n1} - avg(X_1)}{stdev(X_1)} & & \dfrac{x_{nk} - avg(X_k)}{stdev(X_k)}
\end{bmatrix}
$$

Each column of the standardised data has mean 0 and variance 1

Reshaping in NumPy



Reshaping in NumPy



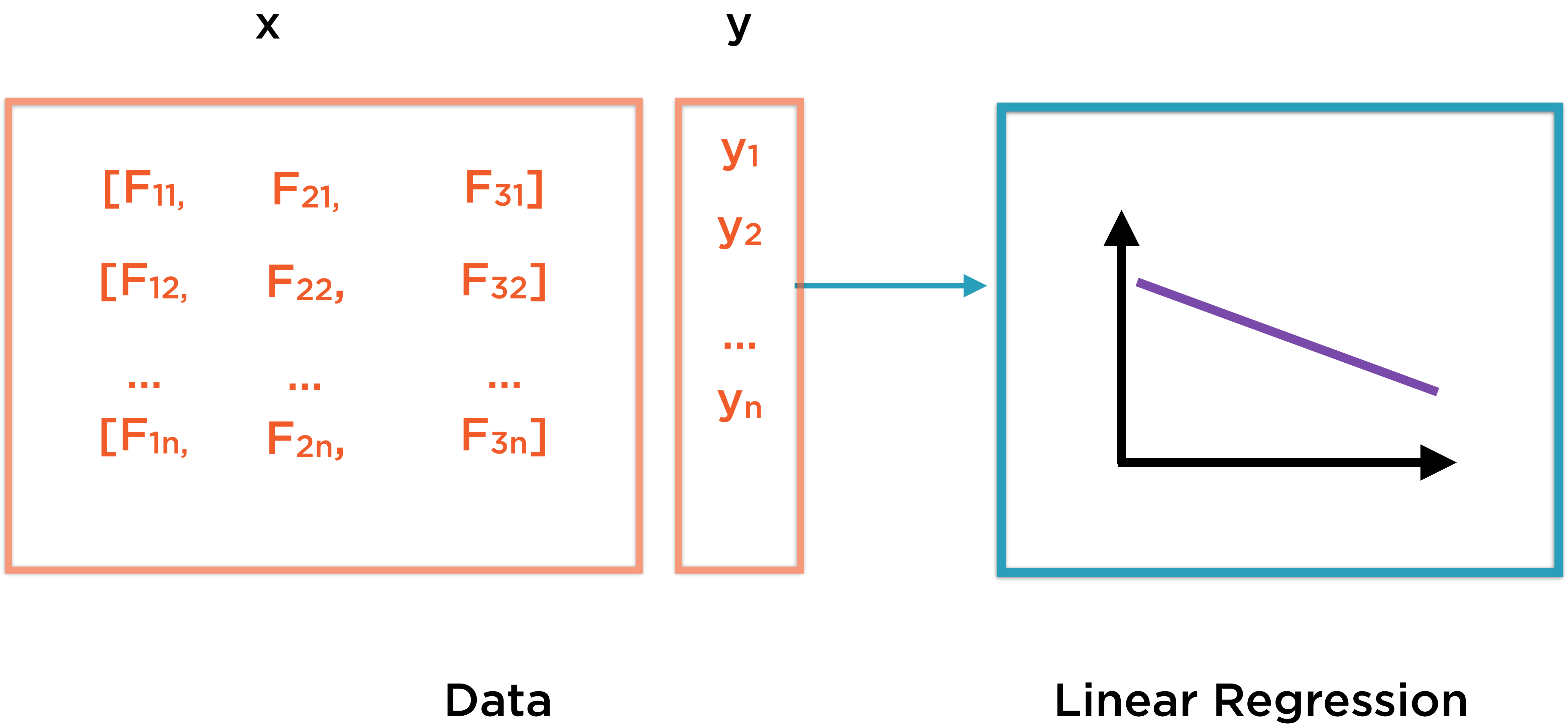
Rearranging Numbers in Python

$PCA_1 \rightarrow [\quad F_{11} \quad F_{12} \quad \ldots \quad]$

$PCA_2 \rightarrow [\quad F_{21} \quad F_{22} \quad \ldots \quad]$

$PCA_3 \rightarrow [\quad F_{31} \quad F_{32} \quad \ldots \quad]$

Rearranging Numbers in Python



Rearranging Numbers in Python

$PCA_1.T$ → $[F_{11}]$ $[F_{12}]$ ... $[F_{1n}]$

$PCA_2.T$ → $[F_{21}]$ $[F_{22}]$ ... $[F_{2n}]$

$PCA_3.T$ → $[F_{31}]$ $[F_{32}]$ ... $[F_{3n}]$

Rearranging Numbers in Python

$PCA_1.T$

$[F_{11}]$

$[F_{12}]$

...

$[F_{1n}]$

$PCA_2.T$

$[F_{21}]$

$[F_{22}]$

...

$[F_{2n}]$

$PCA_3.T$

$[F_{31}]$

$[F_{32}]$

...

$[F_{3n}]$

Rearranging Numbers in Python

zip($PCA_1.T$, $PCA_2.T$, $PCA_3.T$)

(	$[F_{11}]$	$[F_{21}]$	$[F_{31}]$	)
(	$[F_{12}]$	$[F_{22}]$	$[F_{32}]$	)
	...	...	...	
(	$[F_{1n}]$	$[F_{2n}]$	$[F_{3n}]$	)

Rearranging Numbers in Python

$$zip(PCA_1.T,\ PCA_2.T,\ PCA_3.T)$$

$$
\begin{array}{lllll}
(& [F_{11}] & [F_{21}] & [F_{31}] &) \\
(& [F_{12}] & [F_{22}] & [F_{32}] &) \\
 & \ldots & \ldots & \ldots & \\
(& [F_{1n}] & [F_{2n}] & [F_{3n}] &)
\end{array}
$$

Rearranging Numbers in Python

$$\text{zip}(\ PCA_1.T\ ,\ \ PCA_2.T,\ \ PCA_3.T\ )\ .\text{reshape}(-1,3)$$

$$\begin{array}{lll}
[F_{11}, & F_{21}, & F_{31}] \\
[F_{12}, & F_{22}, & F_{32}] \\
\ldots & \ldots & \ldots \\
[F_{1n}, & F_{2n}, & F_{3n}]
\end{array}$$

Rearranging Numbers in Python

$zip(PCA_1.T,\ PCA_2.T,\ PCA_3.T)$ **.reshape(-1,3)**

$[F_{11},$	$F_{21},$	$F_{31}]$
$[F_{12},$	$F_{22},$	$F_{32}]$
...	...	...
$[F_{1n},$	$F_{2n},$	$F_{3n}]$

Principal Components Analysis

$[\ X_1\ X_2\ X_3 \dots X_k\ ]$ →

Eigenvalue Decomposition

→

Principal Components:

$[\ F_1\ F_2\ F_3 \dots F_k\ ]$ n rows

k columns

Eigenvectors:

$[\ v_1\ v_2\ v_3 \dots v_k\ ]$ k rows

k columns

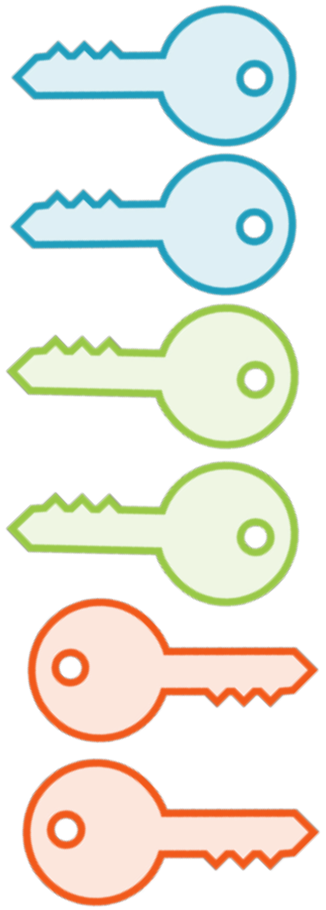
Eigenvalues:

$[\ e_1\ e_2\ e_3 \dots e_k\ ]$ 1 row

k columns

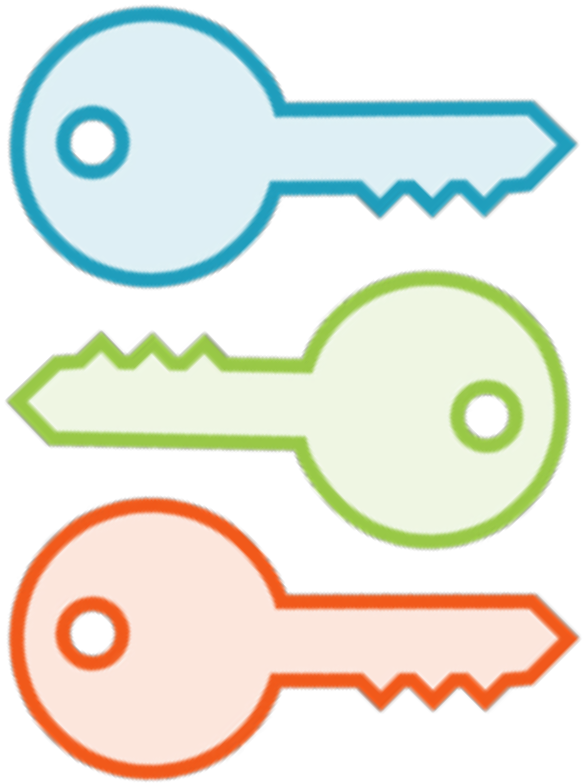
Keeping things simple is quite complicated

Similar, yet Different



Regression

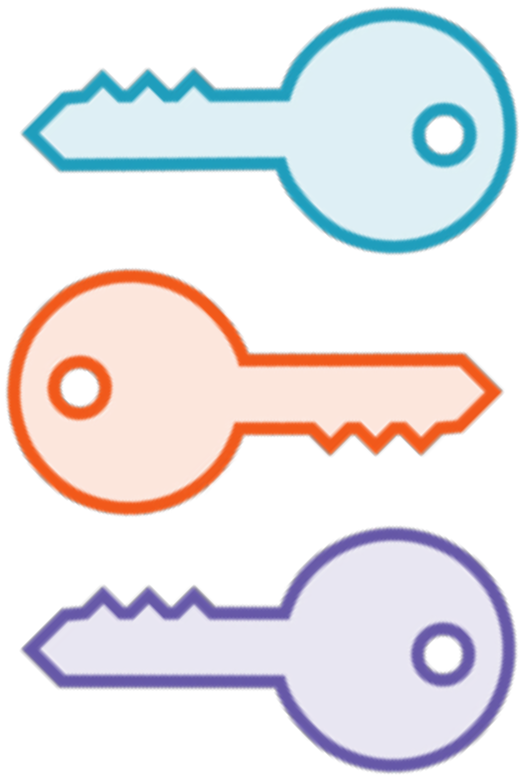
Connect the dots



Factor Analysis

Cut through the clutter

Regression



Causes

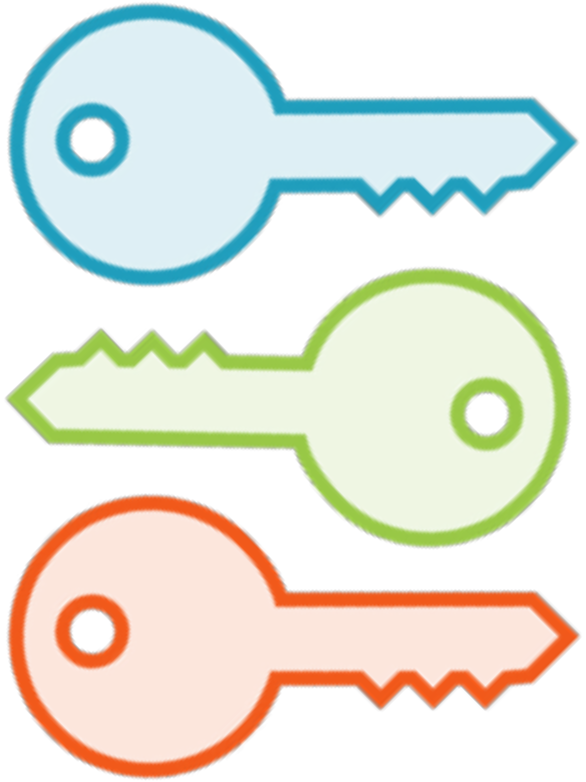
Independent variables

Effect

Dependent variable

Factor Analysis

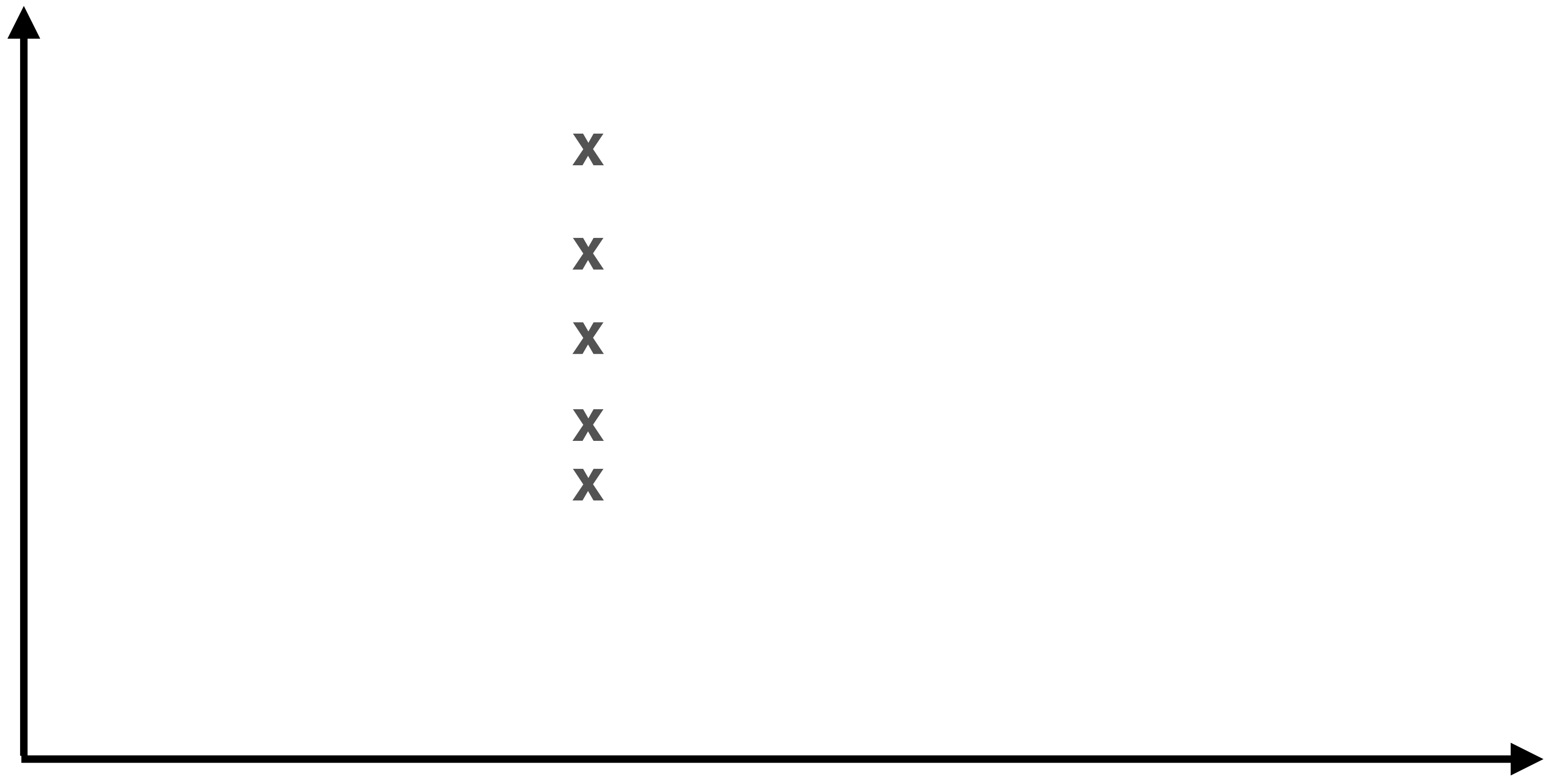
Many Observed Causes



Few Underlying Causes

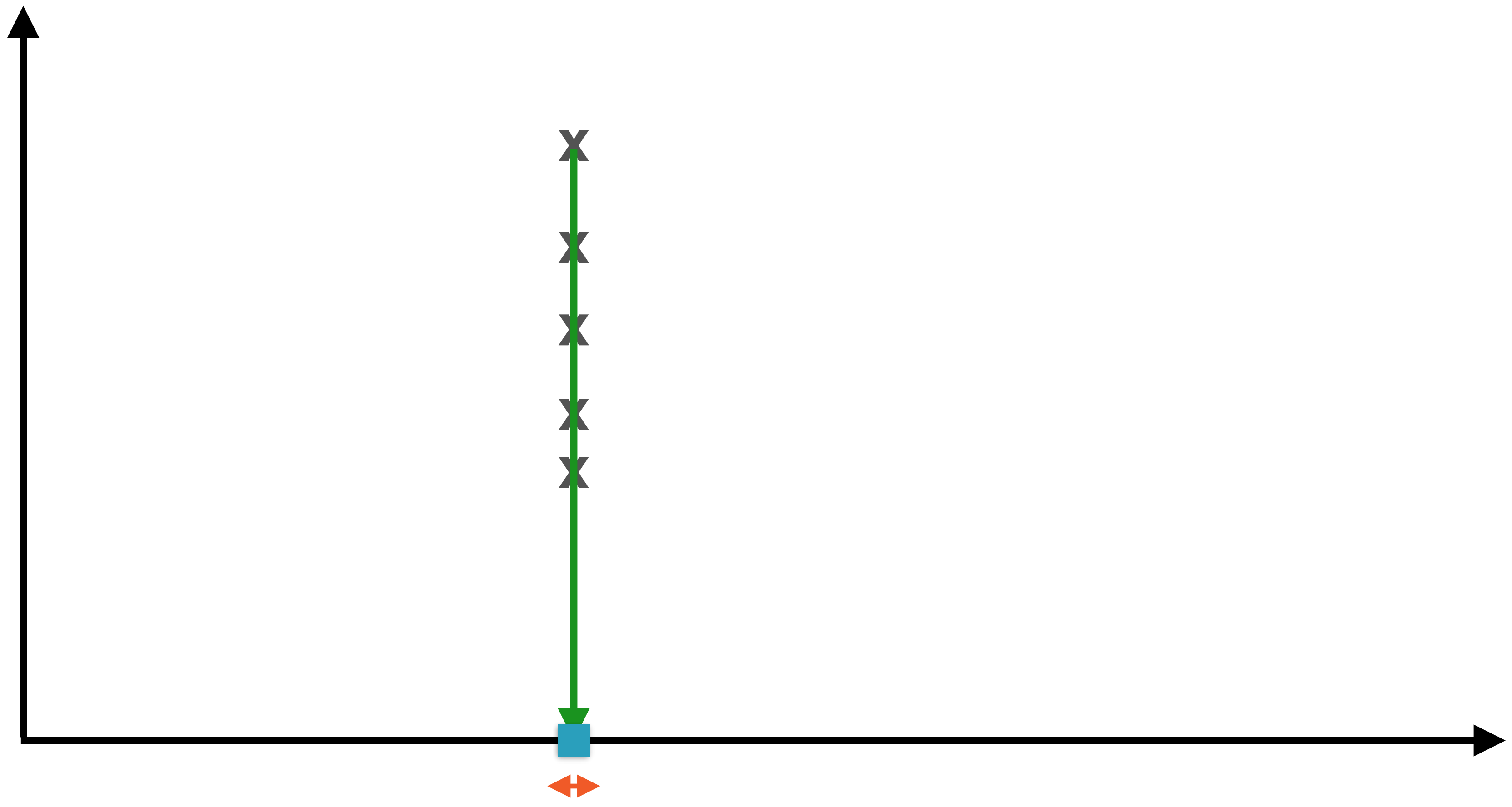
One Effect

A Question of Dimensionality



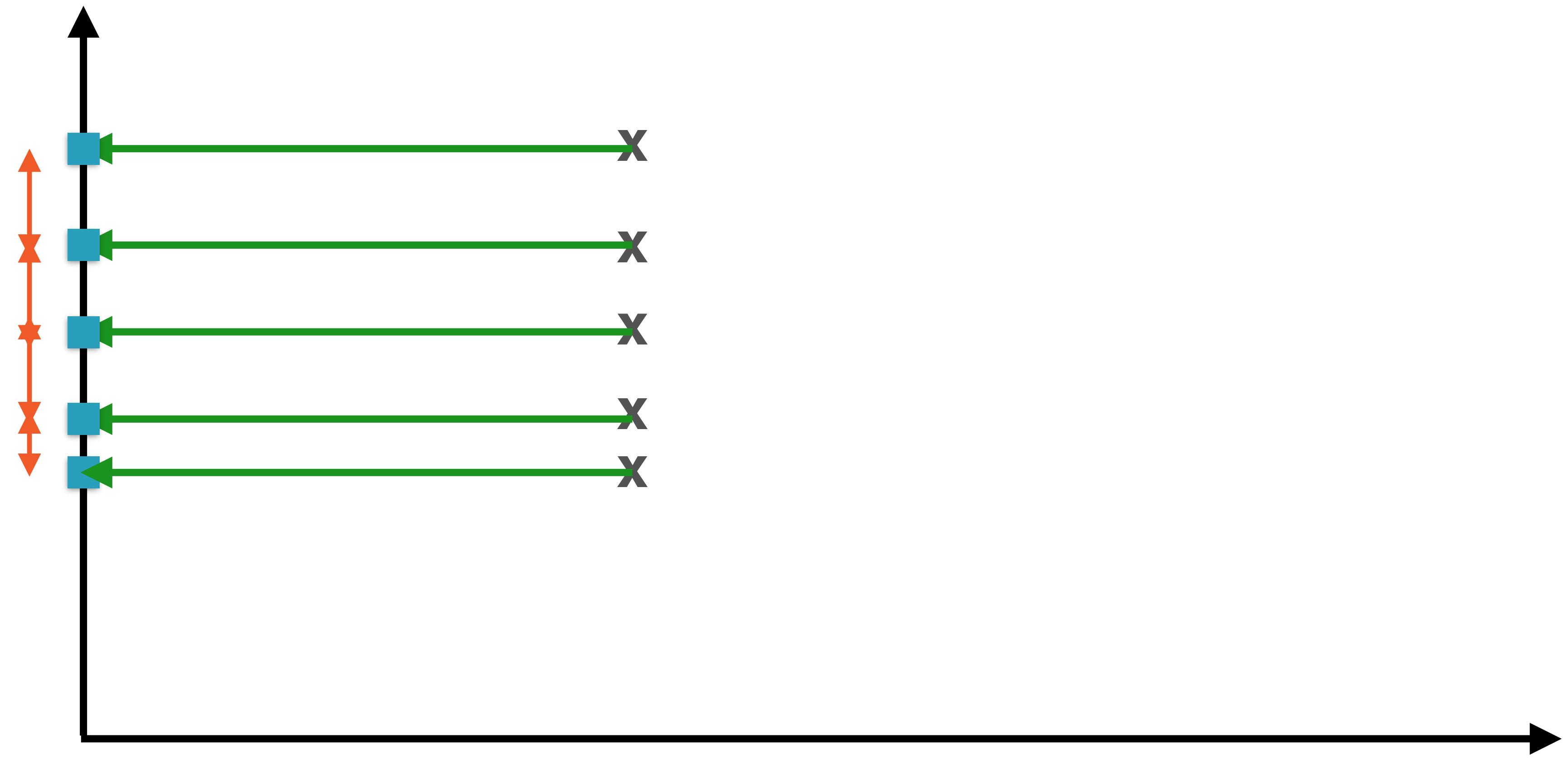
Pop quiz: Do we really need two dimensions to represent this data?

Bad Choice of Dimensions



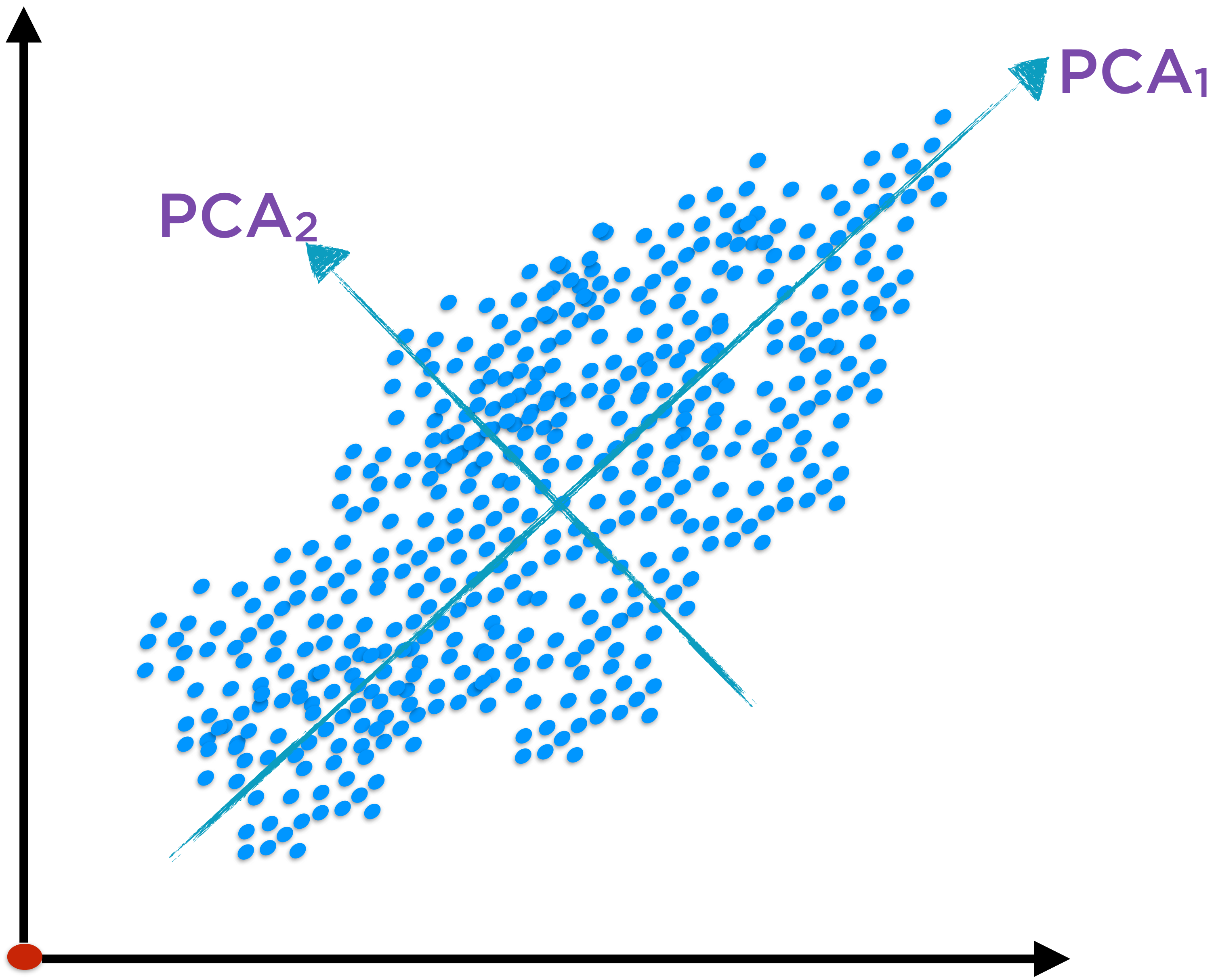
If we choose our axes (dimensions) poorly then we do need two dimensions

Good Choice of Dimensions



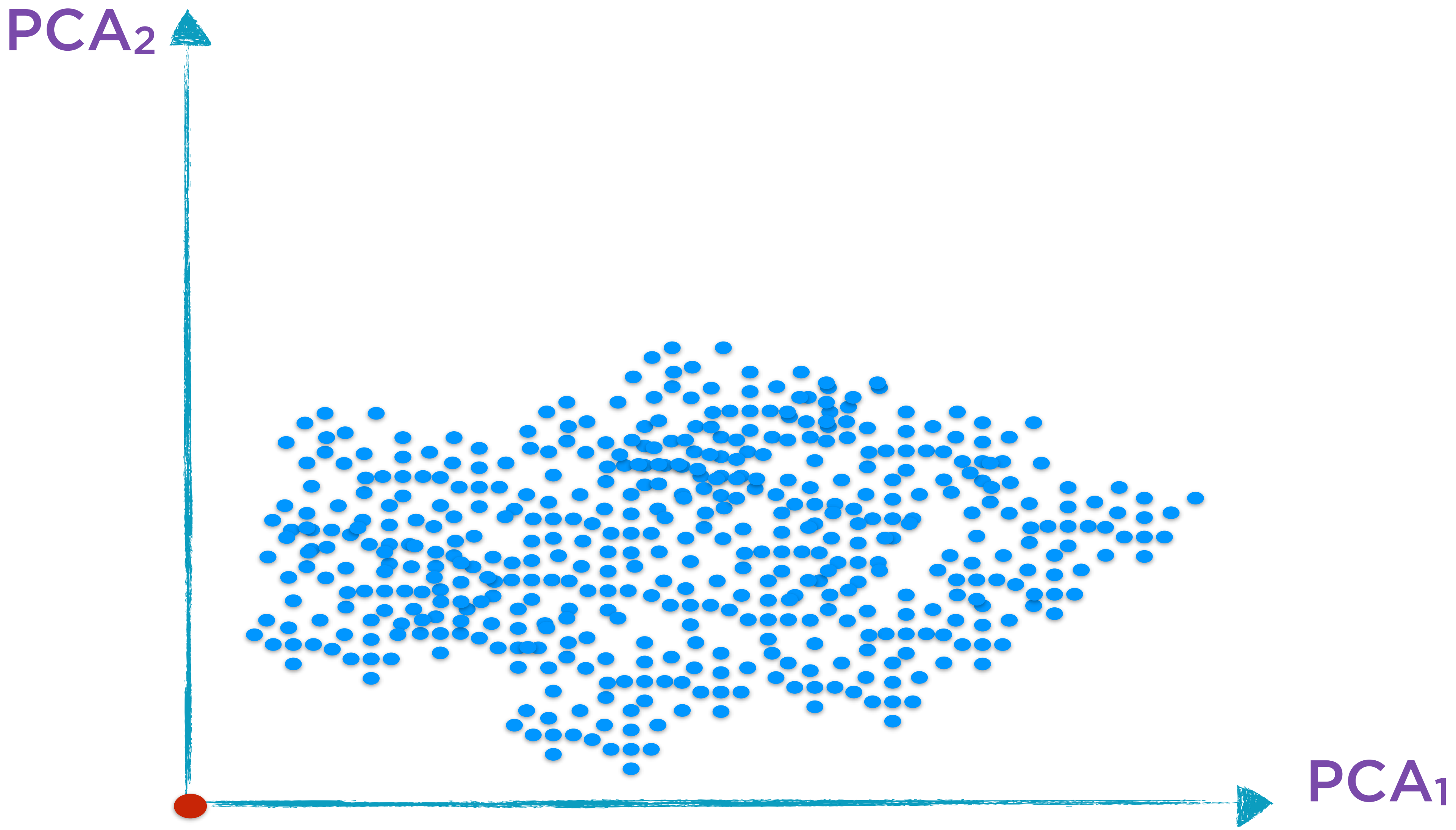
If we choose our axes (dimensions) well then one dimension is sufficient

Intuition Behind PCA



In general, there are as many principal components as there are dimensions in the original data

Intuition Behind PCA



Re-orient the data along these new axes

Summary

Python has powerful libraries for PCA and eigen analysis

PCA of equity returns reveals three important principal components