LECTURE 21

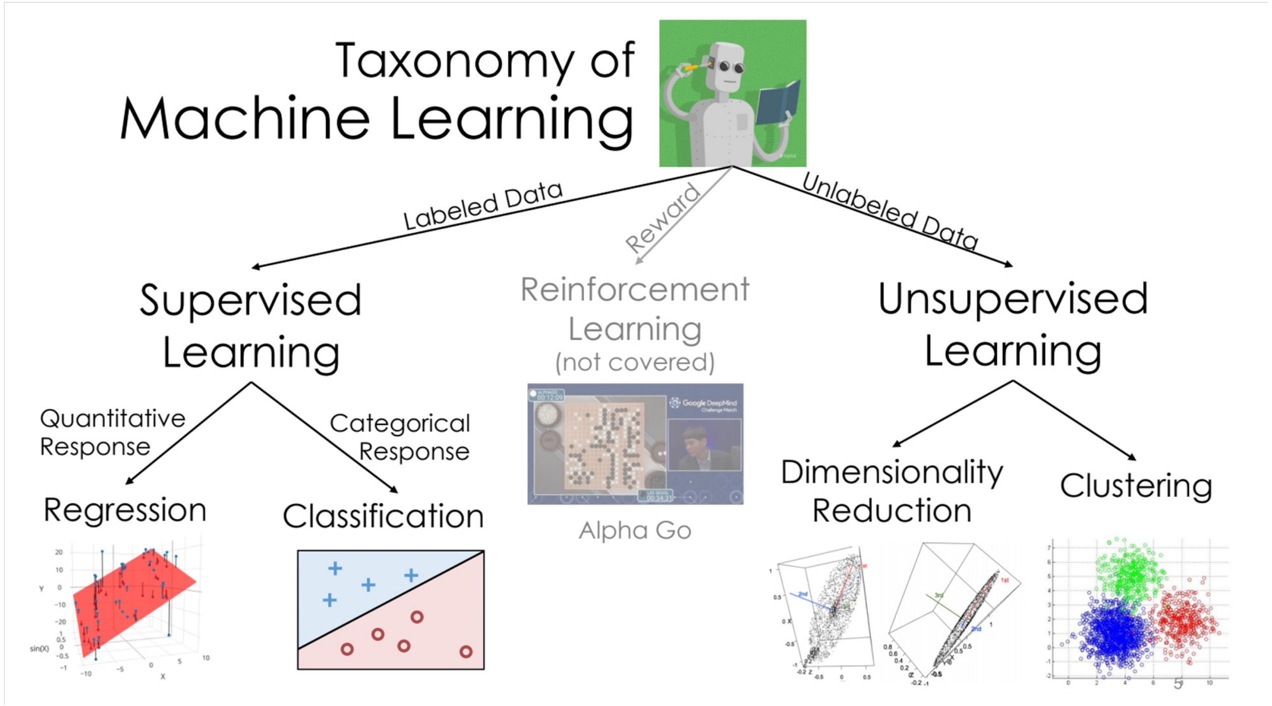
PCA

PCA: An alternate technique for EDA and feature generation.

Machine learning taxonomy

Regression and Classification are both forms of **supervised learning**.

Logistic regression, the topic of this lecture, is mostly used for **classification**, even though it has “regression” in the name.



Dimensionality and Rank of Data

- **Dimensionality and Rank of Data**
- Matrix as Transformation
- Principle Component Analysis
- PCA with SVD
- Data Variances and Centering
- Deriving PCA as Error Minimization

Dimensionality of the Column Space

Suppose we have a dataset of:

- **N** observations (datapoints)
- **d** attributes (features).

In Linear Algebra:

- **N points/row vectors** in a **d**-D space, OR
- **d** column vectors in an **N**-D space.

Dimension of the column space of A is the **rank** of matrix A.

Height (in)	Weight (lbs)
65.8	113.0
71.5	136.5
69.4	153.0

2 dimensions

Height (in)	Weight (lbs)	Age
65.8	113.0	17
71.5	136.5	21
69.4	153.0	18

3 dimensions

Intrinsic Dimension of Data

Suppose we have a dataset of:

- **N** observations (datapoints)
- **d** attributes (features).

In Linear Algebra:

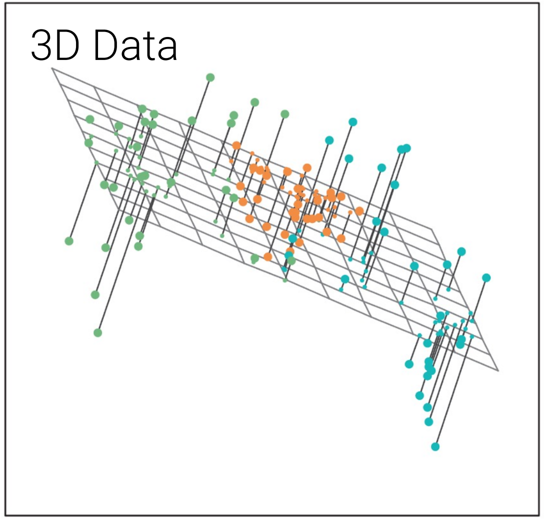
- **N points/row vectors** in a **d**-D space, OR
- **d** column vectors in an **N**-D space.

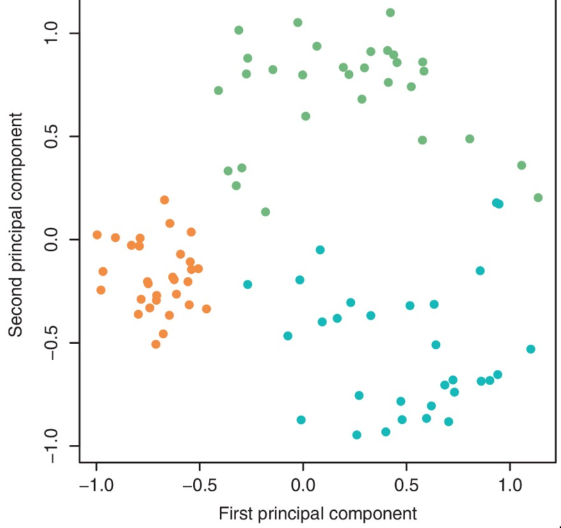
Intrinsic Dimension of a dataset is the **minimal set of dimensions** needed to approximately represent the data.

Example:

- 3D Dataset
- Mostly describe by position on the 2D-plane.

Intrinsic Dimension ≃ 2





Dimensionality of the Column Space

Suppose we have a dataset of:

- **N** observations (datapoints)
- **d** attributes (features).

In Linear Algebra:

- **N points/row vectors** in a **d**-D space, OR
- **d** column vectors in an **N**-D space.

Intrinsic Dimension of a dataset is the **minimal set of dimensions** needed to approximately represent the data.

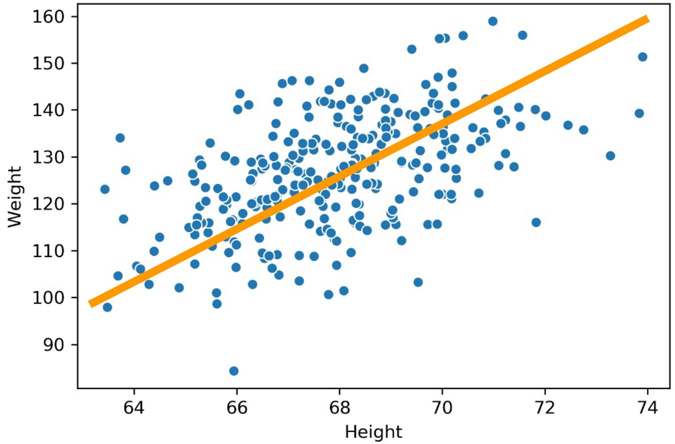
Example:

- "Somewhat" described by position on the 1D-plane (line)

dimension of the column space of A is the **rank** of matrix A.

Example: 2 dimensions

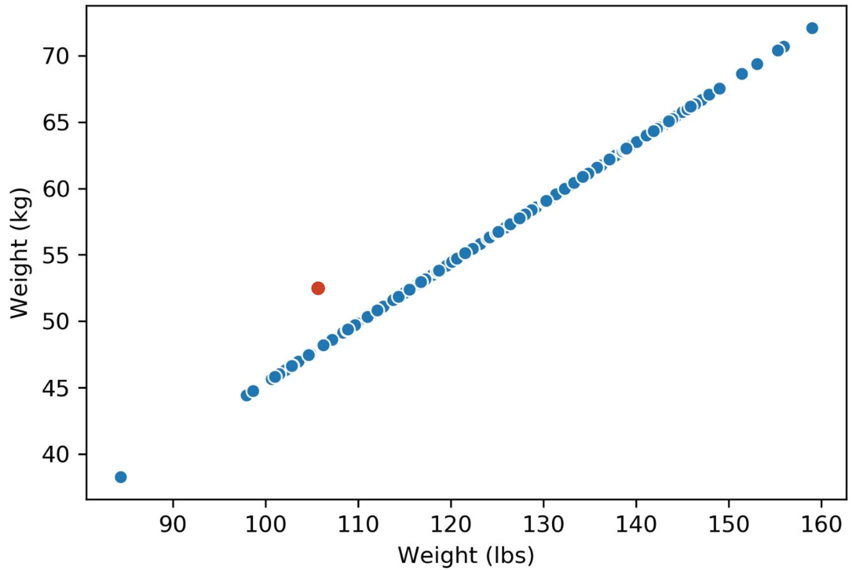
Height (in)	Weight (lbs)
65.8	113.0
71.5	136.5
69.4	153.0



Dimensionality - what does it mean…?

Note that in the dataset below, I've added one **outlier** point to the dataset

- Just this one outlier is enough to change the **rank** of the matrix to 2.
- But the data is still ***approximately*** **1-dimensional**!

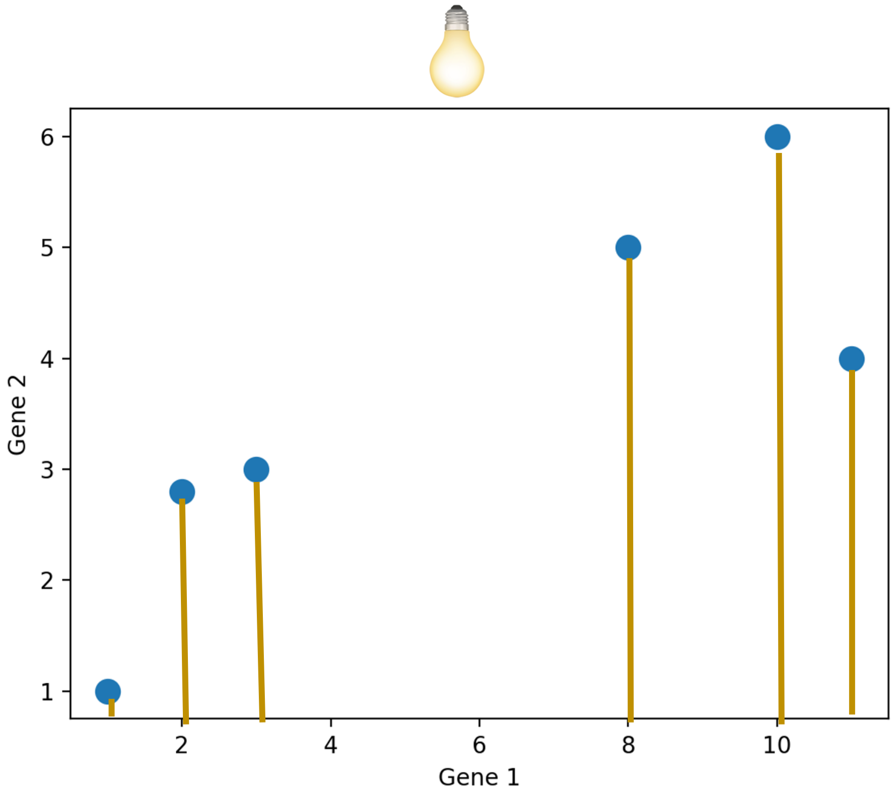


Intrinsic Dimension of a dataset is the **minimal set of dimensions** needed to approximately represent the data.

Dimensionality reduction is generally an **approximation of the original data**. This is achieved through matrix factorization.

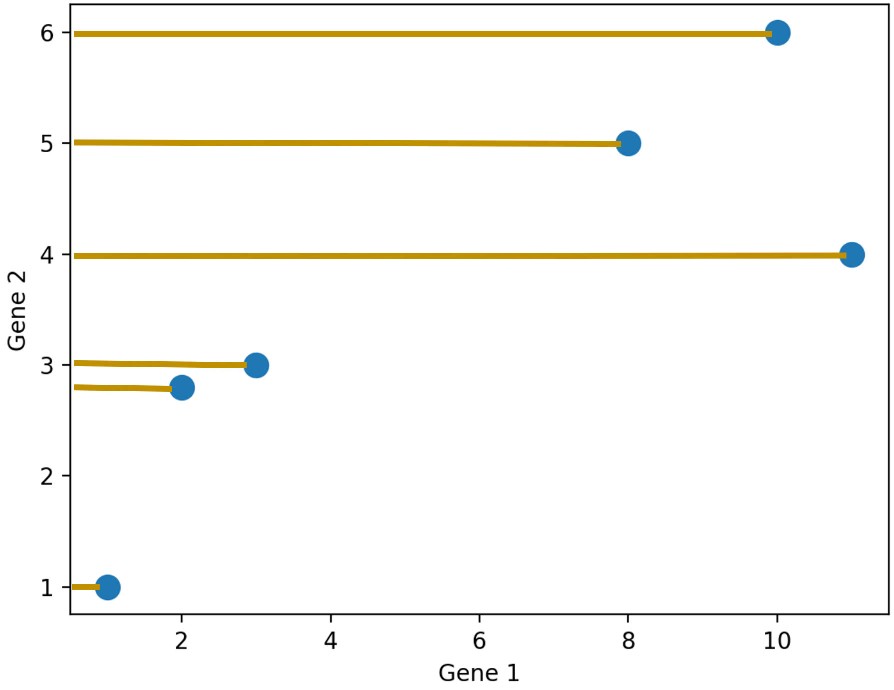
Reducing the Dimensions of Gene Data

How do we project a dataset onto a lower dimension? There are many ways to do it.



Reducing the Dimensions of Gene Data

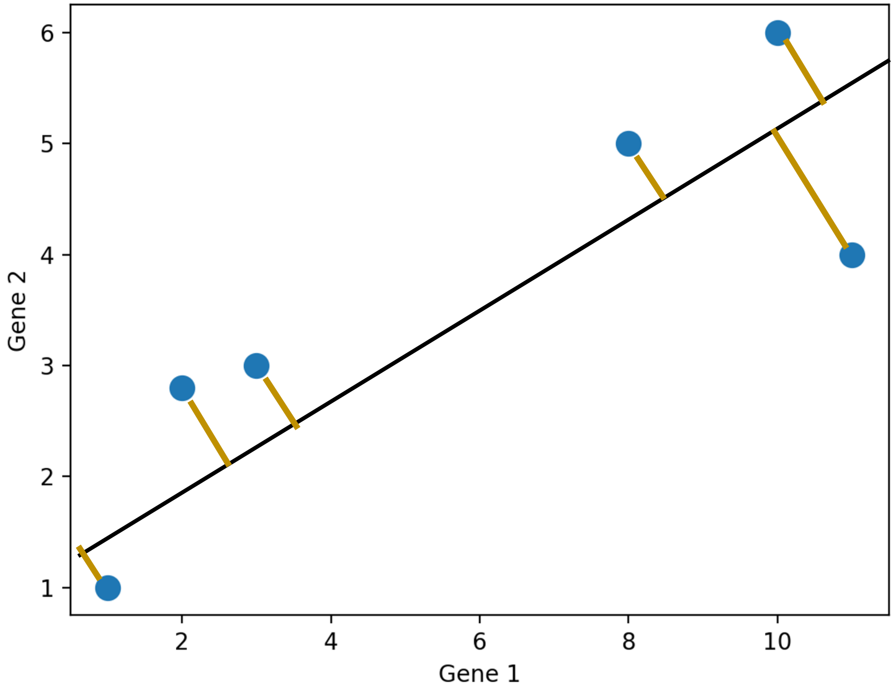
How do we project a dataset onto a lower dimension? There are many ways to do it.



Reducing the Dimensions of Gene Data

How do we project a dataset onto a lower dimension? There are many ways to do it.

How do we know which projection to choose?



In general, we want the projection that is the **best approximation** for the original data.

In other words, we want the projection that capture the **most variance** of the original data.

Matrix Decomposition (Factorization)

- Dimensionality and Rank of Data
- **Matrix as Transformation**
- Principle Component Analysis
- PCA with SVD
- Data Variances and Centering
- Deriving PCA as Error Minimization

Dimensionality Reduction as Matrix Factorization

Original Dataset

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75
630	31	2.58
124	24	2

≈

Reduced Dimension Dataset

Age (days)	Height (in)
182	28
399	30
725	33
630	31
124	24

*

	?	

One **linear** technique to dimensionality reduction is via **matrix decomposition**, which is closely tied to **matrix multiplication**.

Dimensionality Reduction as Matrix Factorization

Original Dataset

Reduced Dimension Dataset

$$\underbrace{\mathbf{X}}_{n \times d} \approx \underbrace{\mathbf{Z}}_{n \times k} * \underbrace{\mathbf{W}}_{k \times d}$$

Today we will develop a procedure to **decompose our data matrix X** into a **lower dimensions matrix Z** that when multiplied by **W approximately recovers the original data**.

Interpreting Matrix multiplication

Consider the matrix multiplication example below.

- Each **row** of the **fruits matrix** represents one bowl of fruit.
  - First bowl: 2 apples, 2 lemons, 2 melons.
- Each **column** of the **dollars matrix** represents the cost of fruit at a store.
  - First store: 2 dollars for an apple, 1 dollar for a lemon, 4 dollars for a melon.
- **Output** is the cost of each bowl at each store.

Number of Fruits

	# Apples	# Lemons	# Melons
Bowl 1	2	2	2
Bowl 2	5	8	0

X

Fruit Costs

	Store 1	Store 2
$/Apple	2	1
$/Lemon	1	1
$/Melon	4	1

`matmul`

=

Output

	Store 1	Store 2
Bowl 1	14	6
Bowl 2	18	13

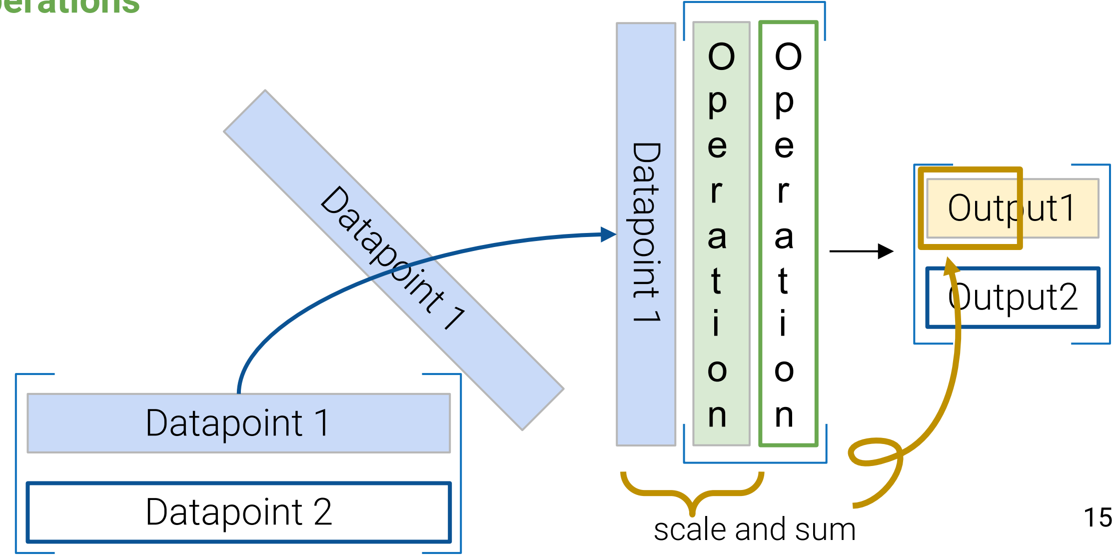
Two ways to **interpret** matrix multiplication:
1. Linear operations per datapoint
2. Column transformation.

Multiplication View 1/2: Right matrix is Linear Operations

$$\underbrace{\begin{bmatrix} 2 & 2 & 2 \\ 5 & 8 & 0 \end{bmatrix}}_{\text{data}} \times \underbrace{\begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 4 & 1 \end{bmatrix}}_{\text{operations}} = \begin{bmatrix} 14 & 6 \\ 18 & 13 \end{bmatrix}$$

View 1: Perform multiple linear operations on data.

- We use this view when building linear models.



Multiplication View 2/2: Right Matrix Transforms Features

2	2	2
5	8	0

Original columns

X

2	1
1	1
4	1

transformation

=

14	6
18	13

New column

View 1: Perform multiple linear operations on data.

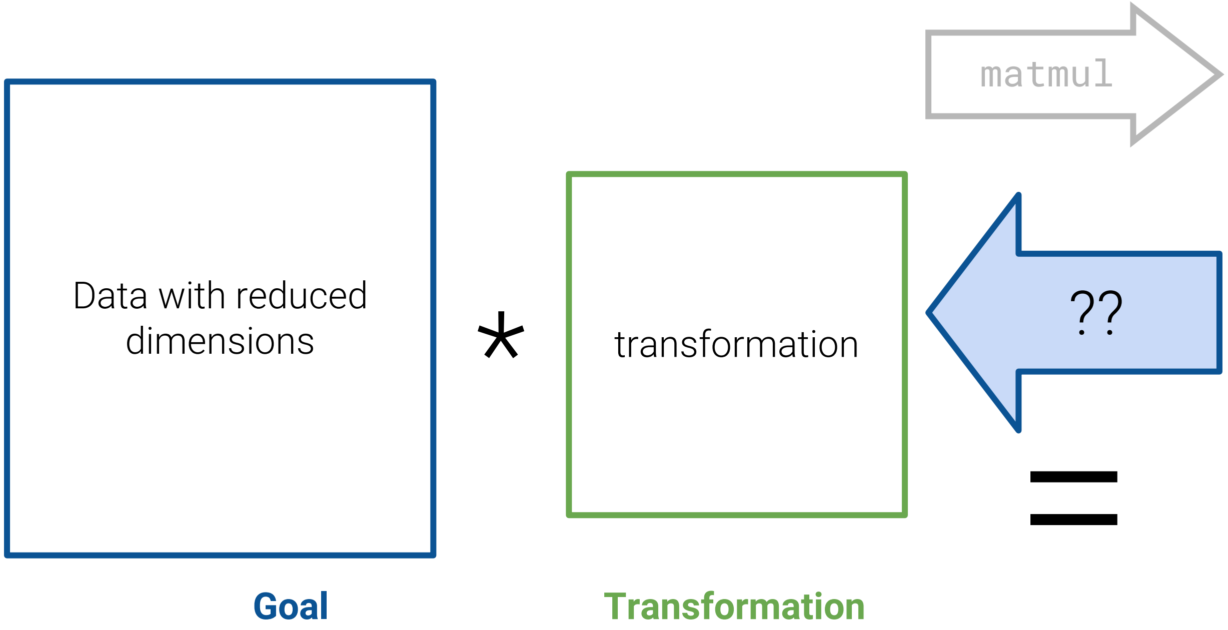
- We use this view when building linear models.

View 2: Multiplication is a column transformation.

$$\begin{bmatrix} 2 & 2 & 2 \\ 5 & 8 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

$$= \boxed{2} \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \boxed{1} \begin{bmatrix} 2 \\ 8 \end{bmatrix} + \boxed{4} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 14 \\ 18 \end{bmatrix}$$

Matrix Decomposition as a Means of Dimensionality Reduction



Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

Matrix Decomposition (Matrix Factorization)

Matrix decomposition (a.k.a. Matrix **Factorization**) is the opposite of matrix multiplication, i.e. taking a matrix and decomposing it into two separate matrices.

- Just like with real numbers, there are **infinitely** many such decompositions.
  - 9.9 = 1.1 * 9 = 3.3 * 3.3 = 1 * 9.9 = …
- Matrix sizes aren't even unique…

Some example factorizations:

3x2

182	28
399	30
725	33
Age	Height (in)

*

1	0	0
0	1	1/12

2x3

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

Goal

Transformation

Matrix Decomposition: Infinite Ways

Matrix decomposition (a.k.a. Matrix **Factorization**) is the opposite of matrix multiplication, i.e. taking a matrix and decomposing it into two separate matrices.

- Just like with real numbers, there are **infinitely** many such decompositions.
  - 9.9 = 1.1 * 9 = 3.3 * 3.3 = 1 * 9.9 = …
- Matrix sizes aren't even unique…

3x2

182	28
399	30
725	33

X

1	0	0
0	1	1/12

2x3

3x3

182	28	2.33
399	30	2.5
725	33	2.75

X

1	0	0
0	1	0
0	0	1

3x3

Age (days)	Height (in)	Height (ft)
182	28	2.33
399	30	2.5
725	33	2.75

What are possible matrix factorizations? Select all that apply.

A. (3x2) x (2x3)
B. (3x3) x (3x3)
C. (3x1) x (1x3)
D. (3x4) x (4x3)
E. Something else

Matrix Decomposition: Limited by Rank

3x4

182	28	2.33	0
399	30	2.5	0
725	33	2.75	0

X

1	0	0
0	1	0
0	0	1
99	31	17

4x3

Fine, but defeats the point of dimension **reduction**…

Age	Height	Height

In practice we usually construct decompositions < rank of the original matrix!

They provide **approximate reconstructions** of the original matrix.

How do we **automatically** choose a reasonable matrix decomposition?

What are possible matrix factorizations? Select all that apply.

✅ (3x2) x (2x3) ✅ **C.** (3x1) x (1x3) **E.** Something else

✅ (3x3) x (3x3) ✅ **D.** (3x4) x (4x3)

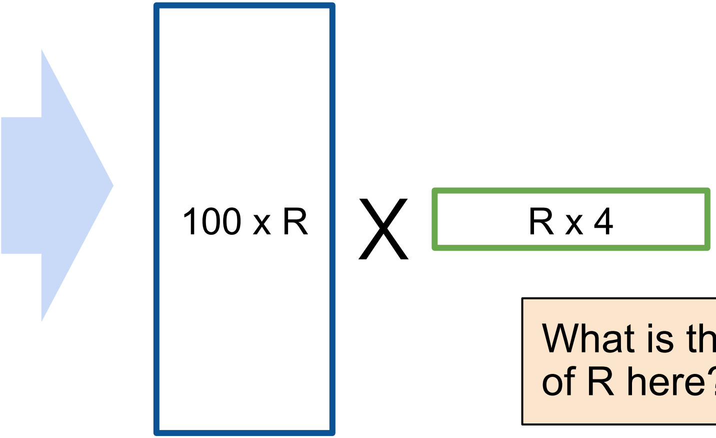
Automatic factorization

Possible goal: Find a procedure to **automatically** factorize a rank R matrix into an R dimensional representation times some transformation matrix.

- **Lower dimensional representation** avoids redundant features.
- Imagine a 1000 dimensional dataset: If the rank is only 5, it's much easier to do EDA after this mystery procedure.

100 x 4

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72



What is the value of R here?

Automatic and Approximate factorization

Possible goal: Find a procedure to **automatically** factorize a rank R matrix into an R dimensional representation times some transformation matrix.

- **Lower dimensional representation** avoids redundant features.
- Imagine a 1000 dimensional dataset: If the rank is only 5, it's much easier to do EDA after this mystery procedure.

What if we wanted a 2-D representation?

- Rank of the 4D matrix is 3, so we can no longer exactly reconstruct the 4-D matrix.

Still, some 2D matrices yield **better approximations** than others. **How well can we do?**

100 x 4

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

≈

100 x 2

...	...

X

2 x 4

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

Goal: Transform observations from high-dimensional data down to **low dimensions** (often 2) through linear transformations.

Related Goal: Low-dimension representation should capture the **variability** of the original data. (to define later)

100 x 4

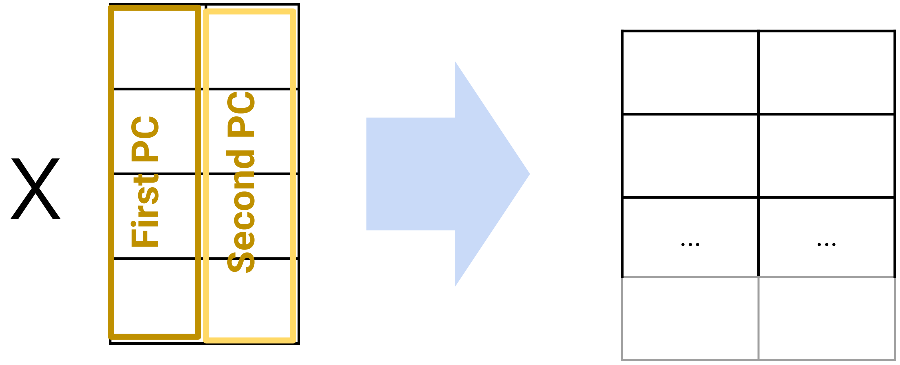
width	length	area	perimeter
20	20	400	80
16	12	192	56
10	10	100	40
...	...	...	...
24	12	288	72

Principal Components (cols)

4x2

Latent Factors (cols)

100 x 2



Transformation Matrix

Why perform PCA?

Goal: Transform observations from high-dimensional data down to **low dimensions** (often 2) through linear transformations.

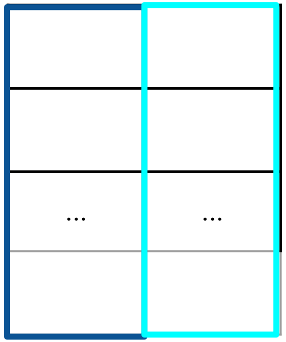
Related Goal: Low-dimension representation should capture the **variability** of the original data.

Exploratory Data Analysis:

- **Visually identify clusters** of similar observations in high dimensions.
- You have reason to believe the **data are inherently low rank,** e.g., There are many attributes but only a few mostly determine the rest through linear associations.
- Some modeling techniques **benefit from decorrelated features**
  - **PCA** will eliminate correlations between features.

Often work with Latent Factors

100 x 2



1

2

Why **two** dimensions?

- Most visualizations are 2-D! Choose the two axes on which to plot datapoints.

Two Equivalent Framings of PCA

There are two equivalent ways to frame PCA:

1. Finding the directions of **maximum variability** in the data
2. Finding the low dimensional (rank) matrix factorization that **best approximates the data**

We will start with the **variance maximization** framing (more common) and then return to the **best approximation** framing (more general).

As you explore more advanced dimensionality reduction techniques, they will often seek to find **“simplified representations”** of data from which we can **still approximately recover the original data**

Capturing Total Variance

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72
7.69	5.35	50.79	338.73

Total Variance: **402.56** = 7.69 5.35 50.79 338.73

Goal of PCA, restated:

Find a linear transformation that creates a low-dimension representation which captures as much of the original data's **total variance** as possible.

Capturing Total Variance, Approach 1

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

Total Variance: **402.56**

Reasonable **Approach 1**:

1. Find variances of each attribute

```
np.var(rectangle,axis=0).sort_values()
```

```
height        5.3475
width         7.6891
perimeter    50.7904
area        338.7316
dtype: float64
```

2. Keep the two attributes with highest variance.

$$\begin{bmatrix} W & L & A & P \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} A & P \end{bmatrix}$$

Low-D Total Variance: **389.52**. *Can we do better?*

Capturing Total Variance: PCA's approach

Reasonable **Approach 1**:

1. Find variances of each attribute

```
np.var(rectangle,axis=0).sort_values()
```

```
height         5.3475
width          7.6891
perimeter     50.7904
area         338.7316
dtype: float64
```

2. Keep the two attributes with highest variance.

W	L	A	P

X

0	0
0	0
1	0
0	1

A	P

Total Variance: **389.52**.

Approach 2: PCA

It turns out that the 2-D approximation that captures the most variance is the following:

-26.4	0.163
17.0	-2.18
...	...
11.8	-1.61
389.62	7.53

These **latent factors** (feature columns) were constructed by a **linear combinations of features** (using PCA).

Total Variance: **397.15.**

Principal Component Analysis: A Procedural View

1. **Center the data matrix** by subtracting the mean of each attribute column.

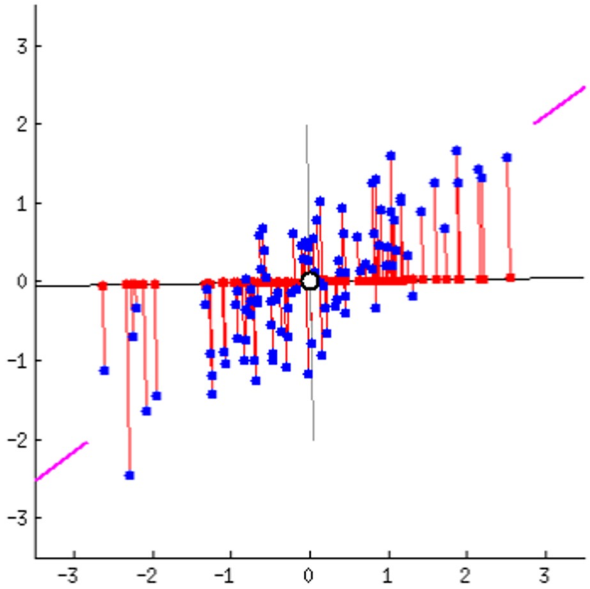
2. To find $\mathbf{v}_i$, the i-th **principal component**:
   - v is a **unit vector** that linearly combines the attributes.
   - v gives a one-dimensional projection of the data.
   - v is chosen to **maximize the variance** along the projection onto v.
   - Choose v such that it is orthogonal to all previous principal components.

k principal components capture the **most variance** of any k-dimensional reduction of the data matrix.

Principal Component Analysis: If you're curious

1. Center the data matrix by subtracting the mean of each attribute column.

1. To find $\mathbf{v_i}$, the i-th **principal component**:
   - v is a **unit vector** that linearly combines the attributes.
   - v gives a one-dimensional projection of the data.
   - v is chosen to **maximize the variance** along the projection onto v.
   - Choose v such that it is orthogonal to all previous principal components.



k principal components capture the **most variance** of any k-dimensional reduction of the data matrix.

Maximizing variance = **spreading out red dots**
Minimizing error (i.e., projection)

=

making red lines short

From SVD to PCA

1. Center the data matrix by subtracting the mean of each attribute column.

2. To find $\mathbf{v}_i$, the i-th **principal component**:
   - v is a **unit vector** that linearly combines the attributes.
   - v gives a one-dimensional projection of the data.
   - v is chosen to minimize the sum of squared distances between each point and its projection onto v.
   - Choose v such that it is orthogonal to all previous principal components.

Let's now use SVD to get us **principal components**.

Singular Value Decomposition

Singular Value Decomposition

Singular value decomposition (SVD) describes a matrix decomposition into three matrices:

$$X = U S V^T$$

$X \in \mathbb{R}^{n \times d}$

$U \in \mathbb{R}^{n \times d}$

columns of U are orthonormal

Columns of U are

eigenvectors of XX^T

$S \in \mathbb{R}^{d \times d}$

diagonal matrix

of **singular values**, ordered from **largest to smallest**

r non-zero singular values

rank r ≤ d

$V \in \mathbb{R}^{d \times d}$

columns of V are orthonormal

Columns of V are **eigenvectors of X^TX**

There are infinite possible factorizations! SVD chooses a special (but non-unique) one with these properties.

*note 1: assume d < n.

NumPy SVD

```
U, S, Vt = np.linalg.svd(X, full_matrices = False)
```

[documentation]

$\underbrace{\texttt{full\_matrices = False}}_{U \in \mathbb{R}^{n \times d}}$

$$X = U \quad S \quad V^T$$

width	height	area	Perim.
2.97	1.35	24.78	8.64
-3.03	-0.65	-15.22	-7.36
-4.03	-1.65	-20.22	-11.36
3.97	-1.65	3.78	4.64
3.97	3.35	48.78	14.64
-2.03	-3.65	-20.22	-11.36
-1.03	-2.65	-15.22	-7.36
0.97	0.35	6.78	2.64
1.97	-3.65	-16.22	-3.36
2.97	-2.65	-7.22	0.64
...	...	...	...

-0.13	0.01	0.03	-0.21
0.09	-0.08	0.01	0.56
0.12	-0.13	0.09	-0.07
-0.03	0.18	0.01	-0.05
-0.26	-0.09	0.09	-0.06
0.12	-0.05	0.17	-0.05
0.09	0	0.1	-0.08
-0.04	0.01	0	-0.08
0.08	0.18	0.04	-0.05
0.03	0.19	0.02	-0.05
...	...	...	...

197.39			
	27.43		
		23.26	
			0

-0.1	-0.07	-0.93	-0.34
0.67	-0.37	-0.26	0.59
0.31	-0.64	0.26	-0.65
0.67	0.67	0	-0.33

X is therefore **rank 3**.

Principal Components are the Eigenvectors of the Covariance Matrix

Assume we have constructed the Singular Value Decomposition (SVD) of X:

$$X = USV^T$$

Because **X is centered** the **covariance matrix of X** is:

$$\Sigma = X^T X = \left(USV^T\right)^T USV^T = VS^T \underbrace{U^T U}_{\mathbf{I}} SV^T$$
$$= VS^2V^T$$

Right multiplying both sides by V we get:

The columns of V are the **eigenvectors** of the **covariance matrix Σ** and **therefore the <u>Principal Components</u>**

$$\Sigma V = VS^2 \underbrace{V^T V}_{\mathbf{I}} = VS^2$$

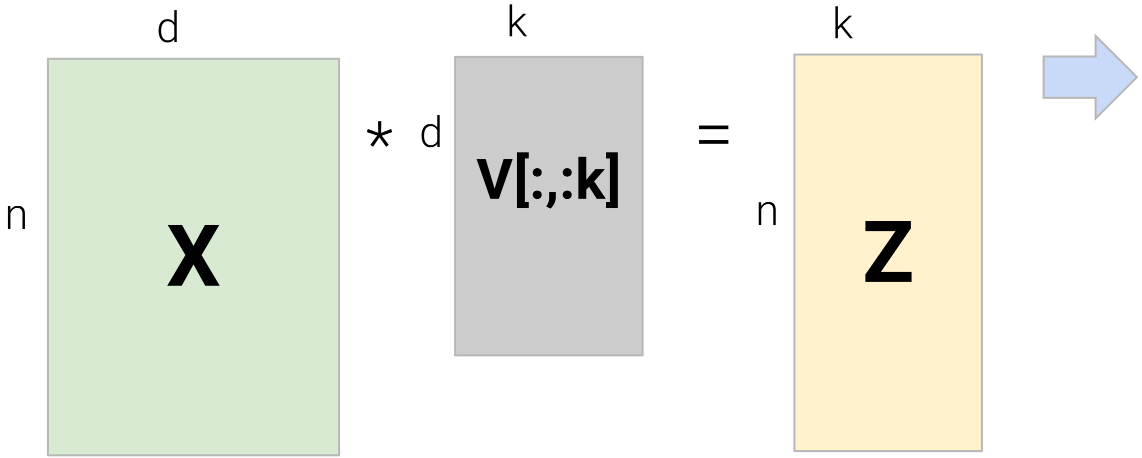
The squared **singular values** are the **eigenvalues** of **Σ**

From SVD to PCA

We have now shown that if we construct the singular value decomposition of X:

$$X = USV^T$$

The **first k** columns of V are the **first k principal components** and we can construct the **latent vector** representation of X by projecting X onto the **principal components**



This gives us a **second way to compute Z**

$$Z = XV = USV^TV$$
$$= US$$

Using only the **first k** columns and **rows** of **U** and **S**

Computing Latent Vectors Using X * V

Constructing a 2 principal component approximation (k=2)

$$X \quad * \quad V \quad = \quad Z$$

width	height	area	Perim.
2.97	1.35	24.78	8.64
-3.03	-0.65	-15.22	-7.36
-4.03	-1.65	-20.22	-11.36
3.97	-1.65	3.78	4.64
3.97	3.35	48.78	14.64
-2.03	-3.65	-20.22	-11.36
…	…	…	…

PC1	PC2		
-0.1	0.67	0.31	0.67
-0.07	-0.37	-0.64	0.67
-0.93	-0.26	0.26	0
-0.34	0.59	-0.65	-0.33

-26.43	0.16
17.05	-2.18
23.25	-3.54
-5.38	5.03
-51.09	-2.59
23.19	-1.45

Computing Latent Vectors Using U * S

Constructing a 2 principal component approximation (k=2)

$$U \quad * \quad S \quad = \quad Z$$

-0.13	0.01	0.03	-0.21
0.09	-0.08	0.01	0.56
0.12	-0.13	0.09	-0.07
-0.03	0.18	0.01	-0.05
-0.26	-0.09	0.09	-0.06
0.12	-0.05	0.17	-0.05
...	...	...	...

197.39			
	27.43		
		23.26	
			0

-26.43	0.16
17.05	-2.18
23.25	-3.54
-5.38	5.03
-51.09	-2.59
23.19	-1.45

Recovering the Data

Given Z we can always (**approximately**) recover the **centered* X** by **multiplying by V^T:**

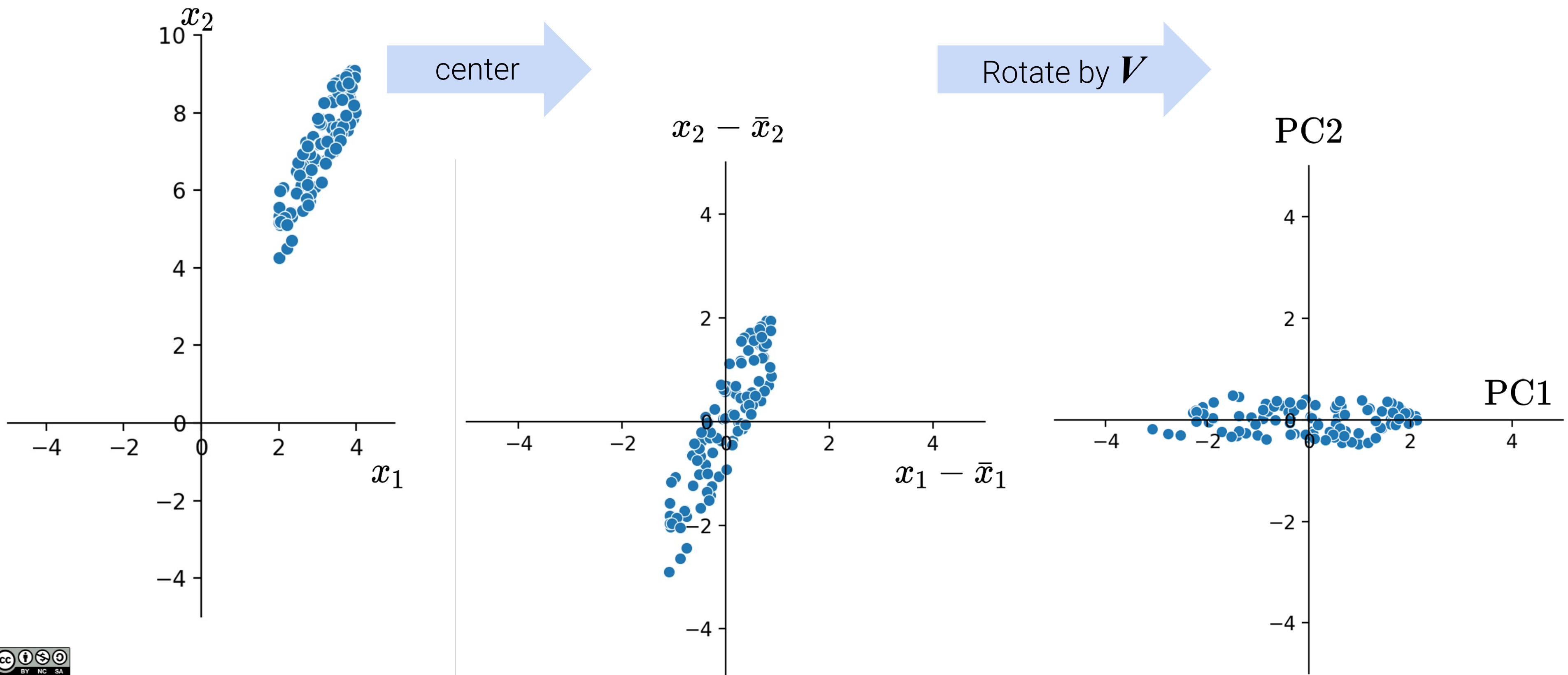
$$ZV^T = XVV^T = USV^T = X$$

If you **choose a k** that is **less than the rank** of X **you will only recover X approximately.**

*To recover the original (uncentered) X we would also need to add back the mean.

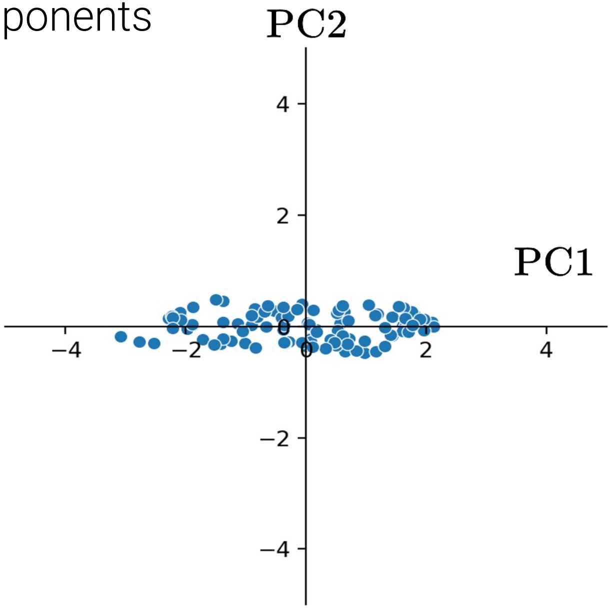
How PCA Transforms Data, Visually

PCA first centers the data matrix, then rotates it such that the direction with the most variation (i.e. the direction that's the most spread-out) is aligned with the x-axis.



Principal Components

- Principal components are all **orthogonal** to each other
  - Why? Recall that the **columns of V are orthonormal**!
- Principal Components are **axis-aligned**
  - If we plot two PCs on a 2D plane, one will lie on the x-axis, the other on the y-axis
- Latent Vectors are **linear combinations** of columns in our data $\boldsymbol{X}$ obtained by **projecting** $\boldsymbol{X}$ onto the principal components



Data Variance and Centering

Capturing Total Variance

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	**length**	**area**	**perimeter**
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

Total Variance: **402.56** = 7.69 5.35 50.79 338.73

Variance and Singular Values

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

Total Variance: **402.56** = 7.69 5.35 50.79 338.73

Formally, the ith singular value tells us the **component score**, i.e., how much of the variance is captured by the ith principal component. n is # of datapoints.

$$i\text{-th component score} = \frac{(i\text{-th singular value})^2}{n}$$

197.4	0	0	0
0	27.43	0	0
0	0	23.26	0
0	0	0	0

→ $197.39^2/100 = 389.63$

→ $27.43^2/100 = 7.52$

→ $23.26^2 / 100 = 5.41$

Variance and Singular Values

We define the **total variance** of a data matrix as the sum of variances of attributes.

width	length	area	perimeter
20	20	400	80
16	12	192	56
...	...	...	...
24	12	288	72

Total Variance: **402.56** = 7.69 5.35 50.79 338.73

Formally, the ith singular value tells us the **component score**, i.e., how much of the variance is captured by the ith principal component. N is # of datapoints.

$$\text{i-th component score} = \frac{(\text{ith singular value})^2}{n}$$

197.4	0	0	0
0	27.43	0	0
0	0	23.26	0
0	0	0	0

Variance captured by **PC1**

→ $197.39^2/100$ = **389.63**

→ $27.43^2/100 = 7.52$

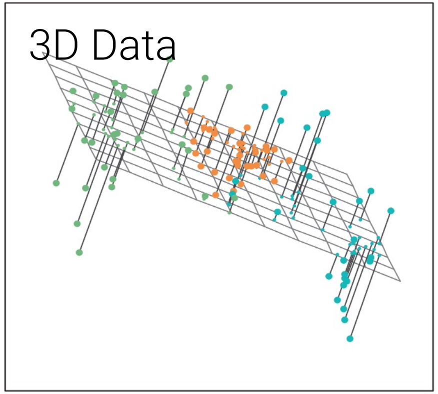
→ $23.26^2 / 100 = 5.41$

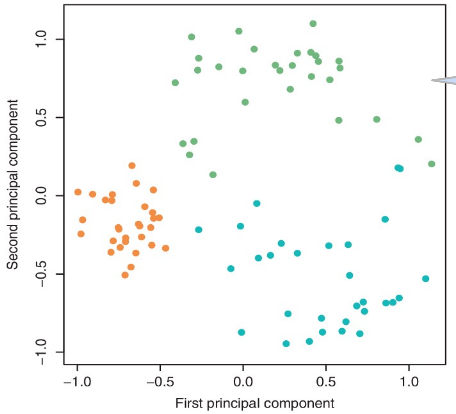
Sum = **402.56**. ✅

PCA Plot

We often construct a scatter plot of the data projected onto the **first two principal components**. This is often called a **PCA plot**.

- PCA plots allow us to visually assess similarities between our data points and if there are any clusters in our dataset.





PCA Plot

If the first two singular values are large and all others are small, **then two dimensions are enough to describe most of what distinguishes one observation from another.** If not, then a **PCA plot** is omitting lots of information.

PCA Plot

We often construct a scatter plot of the data projected onto the **first two principal components**. This is often called a **PCA plot**.

- PCA plots allow us to visually assess similarities between our data points and if there are any clusters in our dataset.

If the first two singular values are large and all others are small, **then two dimensions are enough to describe most of what distinguishes one observation from another.** If not, then a **PCA plot** is omitting lots of information.

How do we compute an array of **variance ratios**, where each element is the **fraction** that each principal component contributes to total data variance?

```
u, s, vt = np.linalg.svd(X, full_matrices = False)
```

A. `s / n        # n is len(X), num features`
B. `s ** 2 / n`
C. `s / sum(s)`
D. `s**2 / sum(s**2)`
E. `Something else`

Variance Ratios

$$\text{i-th component score} = \frac{(\text{i-th singular value})^2}{n}$$

$$X = USV^T$$

$$\text{total variance} = \text{sum of all the component scores} = \sum_{i=1}^{k} \frac{s_i^2}{N}$$

$$\text{variance ratio of principal component j} = \frac{\text{component score j}}{\text{total variance}} = \frac{s_j^2/N}{\sum_{i=1}^{k} s_i^2/N}$$

```
= s**2 / sum(s**2)
```

```
u, s, vt = np.linalg.svd(X, full_matrices = False)
```

How do we compute an array of **variance ratios**, where each element is the **fraction** that each principal component contributes to total data variance?

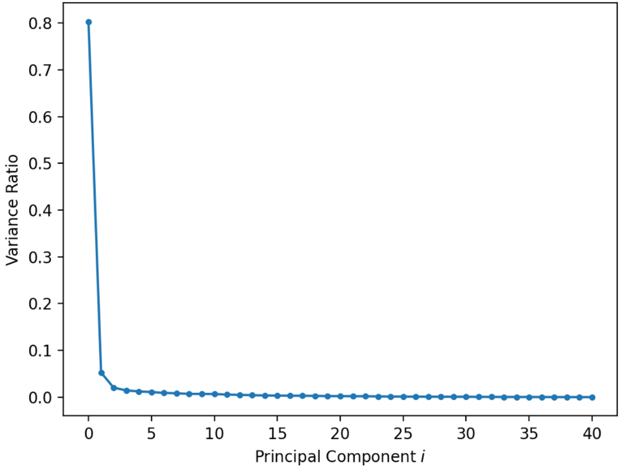
A. `s / n        # n is len(X), num features`
B. `s ** 2 / n`
C. `s / sum(s)`
D. `s**2 / sum(s**2)`
E. Something else

🤔

Scree Plot

If the first two singular values are large and all others are small, then **two dimensions are enough** to describe most of what distinguishes one observation from another. If not, then a PCA scatter plot is omitting lots of information.

A **scree plot** shows the variance ratio captured by each principal component, largest first.



Scree [wikipedia]

Deriving PCA as Error Minimization

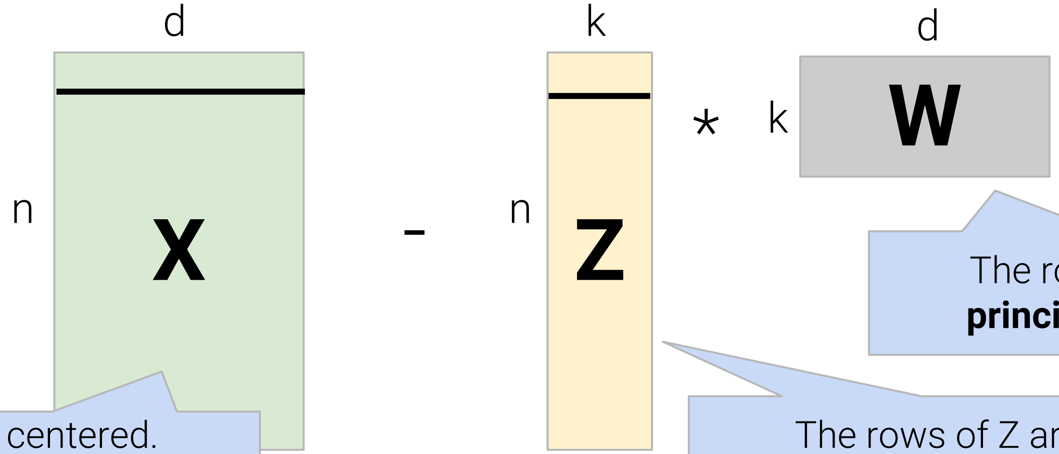
Derive PCA using Loss Minimization

Goal: Minimize the **reconstruction loss** for our **matrix factorization model**:

$$L(Z,W) = \frac{1}{n}\sum_{i=1}^{n}\|X_i - Z_i W\|^2$$

Row Vector

Row Vector



The rows of W are the **principal components**

Assume centered. (subtracted the mean)

The rows of Z are the **latent vectors** (used for EDA)

Derive PCA using Loss Minimization

Goal: Minimize the **reconstruction loss** for our **matrix factorization model**:

$$
\begin{aligned}
L(Z, W) &= \frac{1}{n} \sum_{i=1}^{n} \|X_i - Z_i W\|^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(X_i - Z_i W\right) \left(X_i - Z_i W\right)^T
\end{aligned}
$$

Row Vector

Column Vector

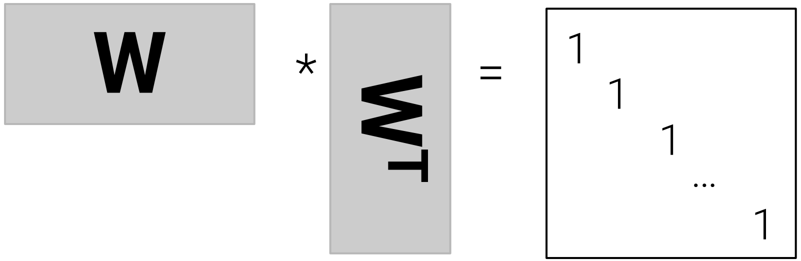
Derive PCA using Loss Minimization

Goal: Minimize the reconstruction loss for our matrix factorization model:

$$L(Z, W) = \frac{1}{n} \sum_{i=1}^{n} (X_i - Z_i W)(X_i - Z_i W)^T$$

Recall there are many solutions so **we constrain our model** to:

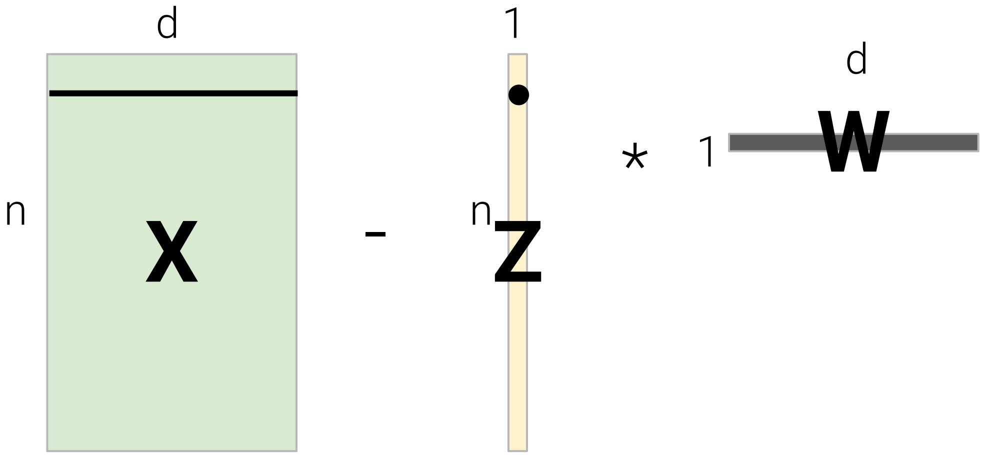
- **W is a row-orthonormal matrix (i.e., $WW^T=I$)** where the rows of W are our Principal Components.



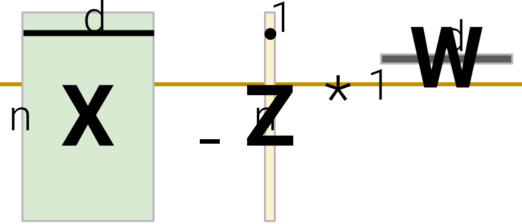
Simplified Derivation: consider (k=1)

Let consider the situation when k=1:

$$L(z, w) = \frac{1}{n} \sum_{i=1}^{n} (X_i - z_i w)(X_i - z_i w)^T$$



Simplified Derivation: Differentiating wrt z



Let consider the situation when k=1:

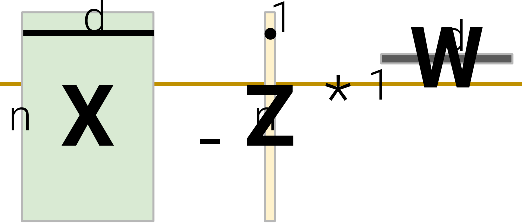
$$L(z,w) = \frac{1}{n}\sum_{i=1}^{n}(X_i - z_i w)(X_i - z_i w)^T$$

Expanding the loss:

$$L(z,w) = \frac{1}{n}\sum_{i=1}^{n}\left(\underbrace{X_i X_i^T}_{\text{Constant (ignore)}} - 2z_i X_i w^T + z_i^2 \underbrace{ww^T}_{=1 \text{ by orthonormality}}\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(-2z_i X_i w^T + z_i^2\right)$$

Simplified Derivation: Substituting soln for z



Substituting the solution for z: $z_i = X_i w^T$

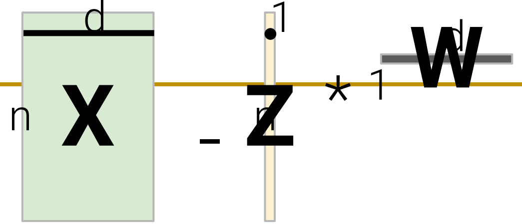
$$L(z, w) = \frac{1}{n} \sum_{i=1}^{n} \left(-2 z_i X_i w^T + z_i^2 \right)$$

$$L(z = Xw^T, w) = \frac{1}{n} \sum_{i=1}^{n} \left(-2 X_i w^T X_i w^T + \left(X_i w^T \right)^2 \right)$$

Algebra: $$= \frac{1}{n} \sum_{i=1}^{n} \left(-X_i w^T X_i w^T \right) = \frac{1}{n} \sum_{i=1}^{n} \left(-w X_i^T X_i w^T \right)$$

Definition of Cov (Σ): $$= -w \frac{1}{n} \sum_{i=1}^{n} \left(X_i^T X_i \right) w^T = -w \Sigma w^T$$

Simplified Derivation: Solving for z



Let consider the situation when k=1:

$$L(z,w) = \frac{1}{n}\sum_{i=1}^{n}\left(-2z_iX_iw^T + z_i^2\right)$$

Taking the derivative with respect to z_i:

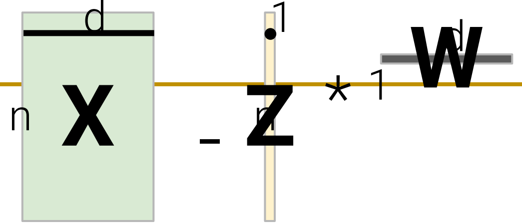
$$\frac{\partial}{\partial z_i}L(z,w) = \frac{1}{n}\left(-2X_iw^T + 2z_i\right)$$

Setting the derivative equal to 0 and solving for z_i:

$$z_i = X_iw^T$$

We can compute z by **projecting onto w**

Simplified Derivation: Substituting soln for z



Substituting the solution for z: $z_i = X_i w^T$

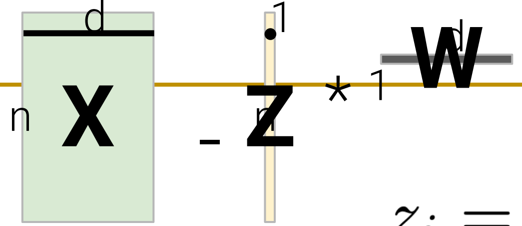
$$L(z, w) = \frac{1}{n}\sum_{i=1}^{n}\left(-2 z_i X_i w^T + z_i^2\right)$$

$$L(z = Xw^T, w) = \frac{1}{n}\sum_{i=1}^{n}\left(-2 X_i w^T X_i w^T + \left(X_i w^T\right)^2\right)$$

Algebra: $$= \frac{1}{n}\sum_{i=1}^{n}\left(-X_i w^T X_i w^T\right) = \frac{1}{n}\sum_{i=1}^{n}\left(-w X_i^T X_i w^T\right)$$

Definition of Cov (Σ): $$= -w\frac{1}{n}\sum_{i=1}^{n}\left(X_i^T X_i\right) w^T = -w\Sigma w^T$$

Simplified Derivation: Substituting soln for z



$$z_i = X_i w^T$$

Minimize the loss with respect to w:

$$L(w) = -w\Sigma w^T$$

Make **w really big** (toward infinity) … but we have the **orthonormality constraint ww^T=1**

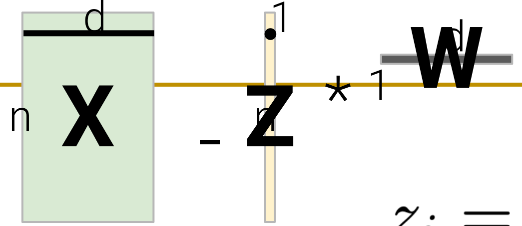
Use **Lagrange multiplier λ** to introduce the constraint **ww^T=1** to our optimization problem:

$$L(w, \lambda) = -w\Sigma w^T + \lambda \left(ww^T - 1 \right)$$

Take **derivative with respect to w:**

$$\frac{\partial}{\partial w} \left(-w\Sigma w^T + \lambda \left(ww^T - 1 \right) \right) = -2\Sigma w^T + 2\lambda w^T$$

Simplified Derivation: Substituting soln for z



$$z_i = X_i w^T$$

Use Lagrange multiplier λ to introduce the constraint (**ww^T=1**)

$$L(w, \lambda) = -w\Sigma w^T + \lambda \left(w w^T - 1 \right)$$

Take derivative with respect to w

$$\frac{\partial}{\partial w} \left(-w\Sigma w^T + \lambda \left(w w^T - 1 \right) \right) = -2\Sigma w^T + 2\lambda w^T$$

Setting equal to zero: $-2\Sigma w^T + 2\lambda w^T = 0$

$$\Sigma w^T = \lambda w^T$$

This implies that:

1. w is a **unitary eigenvector** of the **covariance matrix** and
2. the **error is minimized** when w is the eigenvector **with the largest eigenvalue** λ

Extending the Derivation to the Second PC (Bonus)

We can extend the derivation inductively to the next principal component:

$$\frac{\partial}{\partial w_2} L(w_2, \lambda_2, \lambda_{12}) = -w_2 \Sigma w_2^T + \lambda_2 \left(w_2 w_2^T - 1\right) + \underbrace{\lambda_{12} \left(w_1 w_2^T - 0\right)}_{\textbf{Orthogonality Constraint}}$$

Taking the derivative with respect to w_2:

$$\frac{\partial}{\partial w_2} L(w_2, \lambda_2, \lambda_{12}) = -2\Sigma w_2^T + 2\lambda_2 w_2^T + \lambda_{12} w_1^T$$

Set equal to 0 and left multiply by w_1:

$$-2\underbrace{\underbrace{w_1 \Sigma}_{\lambda w_1} w_2^T}_{0} + 2\lambda_2 \underbrace{w_1 w_2^T}_{0} + \lambda_{12} \underbrace{w_1 w_1^T}_{1} = 0$$

$$\Rightarrow \lambda_{12} = 0$$

Extending the Derivation to the Second PC (Bonus)

We can extend the derivation inductively to the next principal component:

$$\frac{\partial}{\partial w_2} L(w_2, \lambda_2, \lambda_{12}) = -w_2 \Sigma w_2^T + \lambda_2 \left(w_2 w_2^T - 1\right) + \underbrace{\lambda_{12} \left(w_1 w_2^T - 0\right)}_{\textbf{Orthogonality Constraint}}$$

Taking the derivative with respect to w_2:

$$\frac{\partial}{\partial w_2} L(w_2, \lambda_2, \lambda_{12}) = \boxed{-2\Sigma w_2^T + 2\lambda_2 w_2^T} + \cancel{\lambda_{12} w_1^T}$$

Set equal to 0 and left multiply by w_1:

$$-2\underbrace{\underbrace{w_1 \Sigma}_{\lambda w_1} w_2^T}_{0} + 2\lambda_2 \underbrace{w_1 w_2^T}_{0} + \lambda_{12} \underbrace{w_1 w_1^T}_{1} = 0$$

$$\Rightarrow \lambda_{12} = 0$$

Take Away from the Optimization Framing

The **principal components** are the **eigenvectors** with the **largest eigenvalues** of the **covariance matrix.**

- These are the directions of **maximum variance** in the data

We can construct the **latent factors (the Z matrix)** by projecting the **centered data X** onto the **principal component** vectors: