1039867

slido

# Join at slido.com #1039867

Click **Present with Slido** or install our Chrome extension to display joining instructions for participants while presenting.

1039867

LECTURE 24

# Principal Component Analysis I

A New Tool for EDA using Singular Value Decomposition

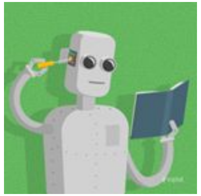

**Data 100/Data 200, Spring 2024 @ UC Berkeley**

Narges Norouzi and Joseph E. Gonzalez



Content credit: Acknowledgments

1039867



Labeled Data

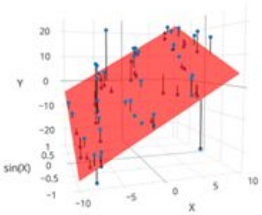
Supervised Learning

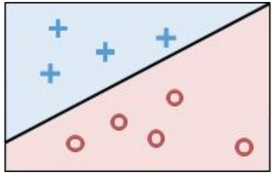
Quantitative Response

Categorical Response

Regression

Classification





Data 8: Nearest Neighbors
Earlier: Logistic Regression

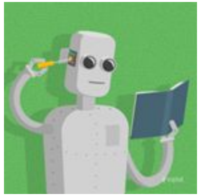
In “Supervised Learning”:

Goal is to create a function that maps inputs to outputs.

- Model is learned from **example input and output pairs.** Each pair consists of:
  - **Input vector (features)**
  - **Output value (label)**.
- **Regression**: Output value is quantitative.
- **Classification**: Output value is categorical.

# Taxonomy of Machine Learning

1039867



Labeled Data

Unlabeled Data

Supervised Learning

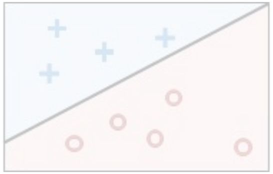
Quantitative Response

Categorical Response

Regression

Classification





Data 8: Nearest Neighbors
Earlier: Logistic Regression

Unsupervised Learning

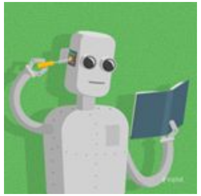
In “Unsupervised Learning”:

Goal is to identify patterns in **unlabeled** data.

- We have **features** but **no labels**
  - Sometimes we may have labels, but we choose to ignore them.

# Taxonomy of Machine Learning

1039867



Labeled Data

Unlabeled Data

Supervised Learning

Unsupervised Learning

Quantitative Response

Categorical Response

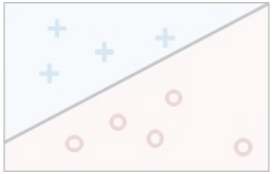
Regression

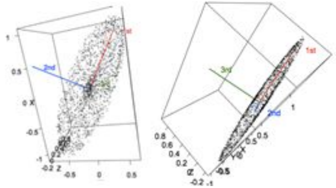
Classification

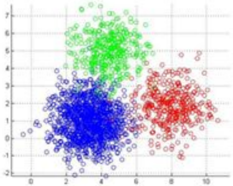
**Dimensionality Reduction**

Clustering









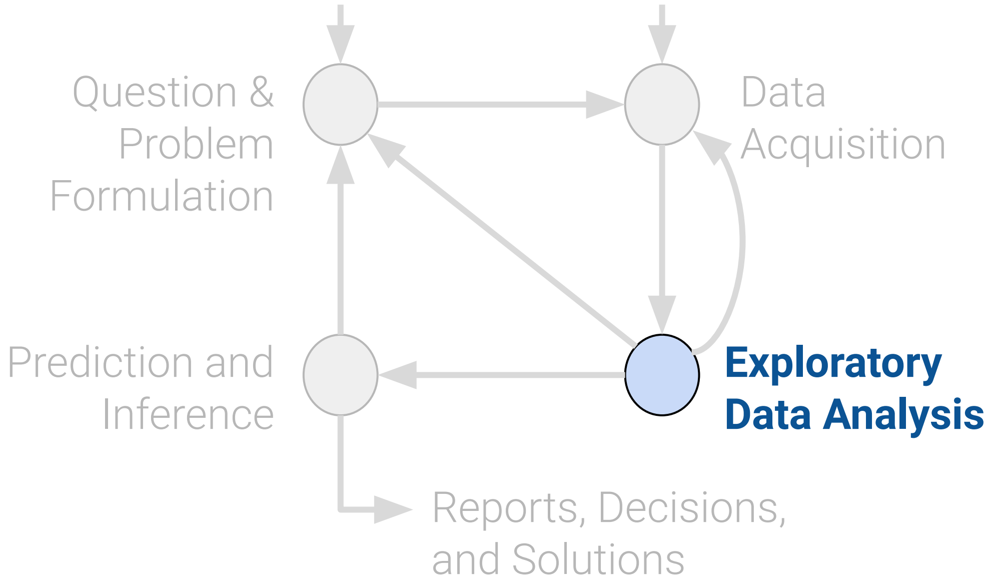
Data 8: Nearest Neighbors
Earlier: Logistic Regression

Today with **Principal Component Analysis (PCA)**

Infer **Quantitative Labels**

# PCA: A Technique for High Dimensional EDA and Featurization






SVD
PCA

today

PCA II
Applications

**Principal Component Analysis (PCA)** is a linear technique for dimensionality reduction.

PCA relies on a linear algebra algorithm called **Singular Value Decomposition**.

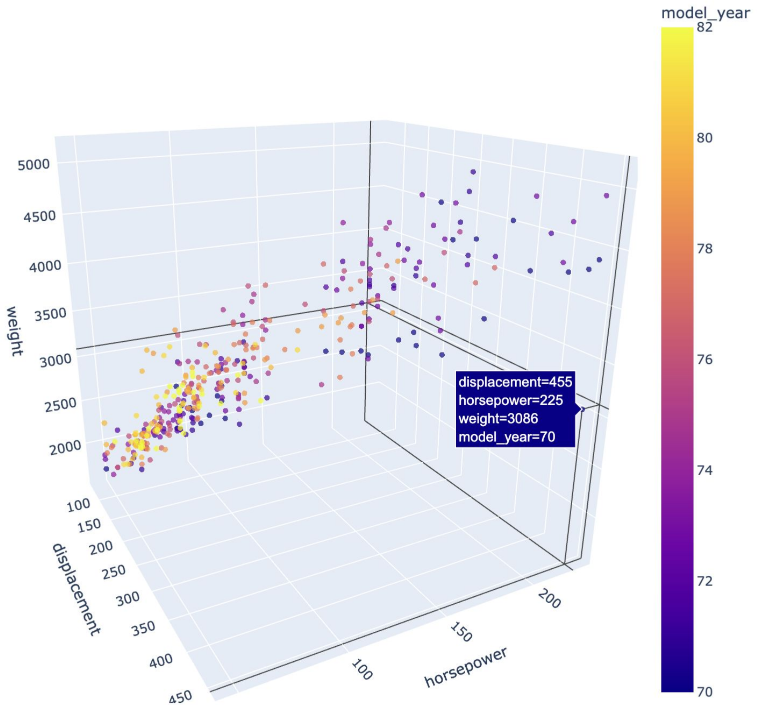
# Today's Roadmap

Lecture 24, Data 100 Spring 2024

# Visualization Revisited

Lecture 24, Data 100 Spring 2024

# Demo (MPG Visualization)

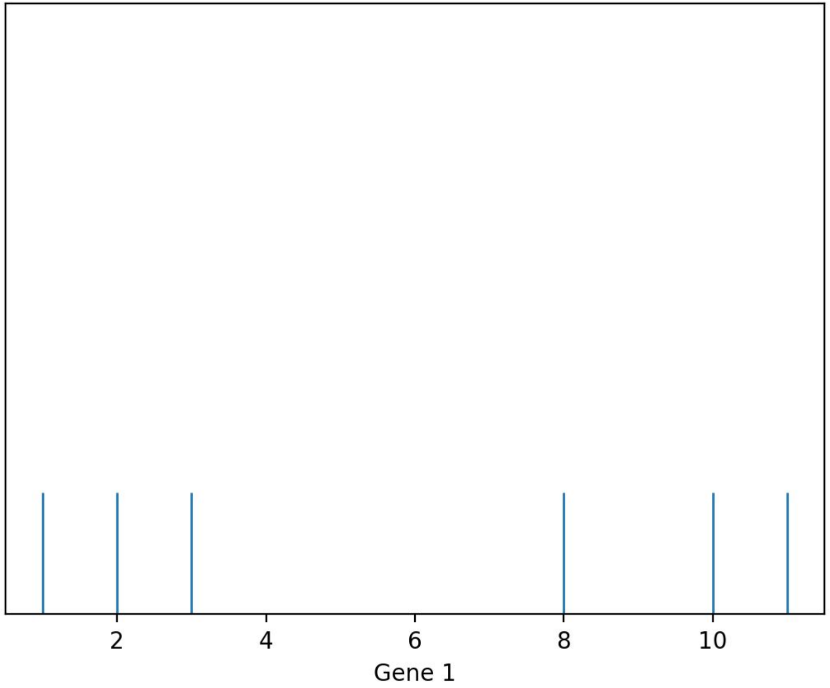


# Visualizing Gene Data

1039867

Visualization can help us identify clusters in our dataset.

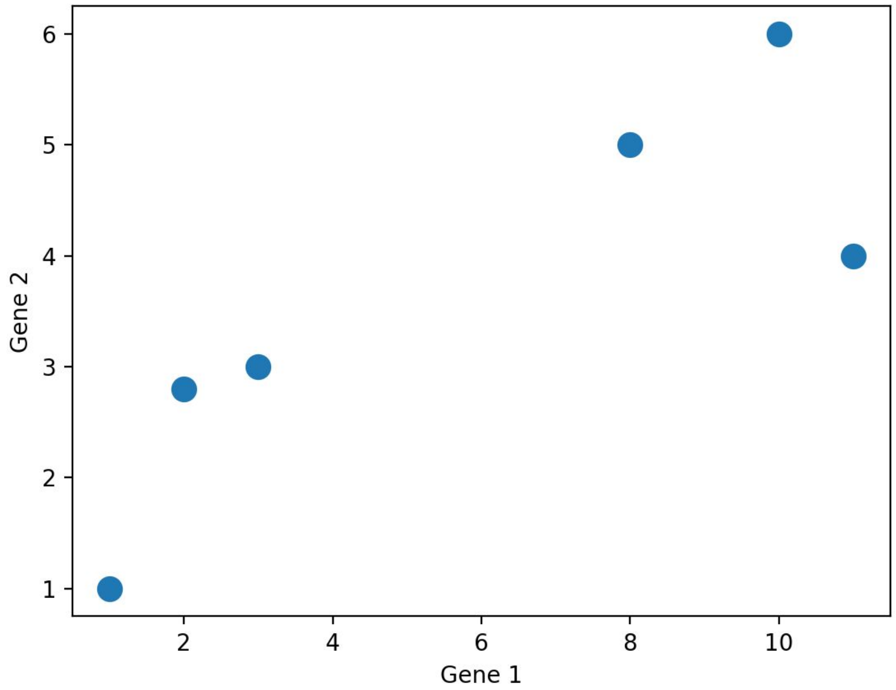
| | Gene 1 | Gene 2 | Gene 3 | Gene 4 |
|---|---|---|---|---|
| **0** | 10 | 6.0 | 12.0 | 5 |
| **1** | 11 | 4.0 | 9.0 | 7 |
| **2** | 8 | 5.0 | 10.0 | 6 |
| **3** | 3 | 3.0 | 2.5 | 2 |
| **4** | 2 | 2.8 | 1.3 | 4 |
| **5** | 1 | 1.0 | 2.0 | 7 |



1039867

Visualization can help us identify clusters in our dataset.

| | Gene 1 | Gene 2 | Gene 3 | Gene 4 |
|---|---|---|---|---|
| **0** | 10 | 6.0 | 12.0 | 5 |
| **1** | 11 | 4.0 | 9.0 | 7 |
| **2** | 8 | 5.0 | 10.0 | 6 |
| **3** | 3 | 3.0 | 2.5 | 2 |
| **4** | 2 | 2.8 | 1.3 | 4 |
| **5** | 1 | 1.0 | 2.0 | 7 |

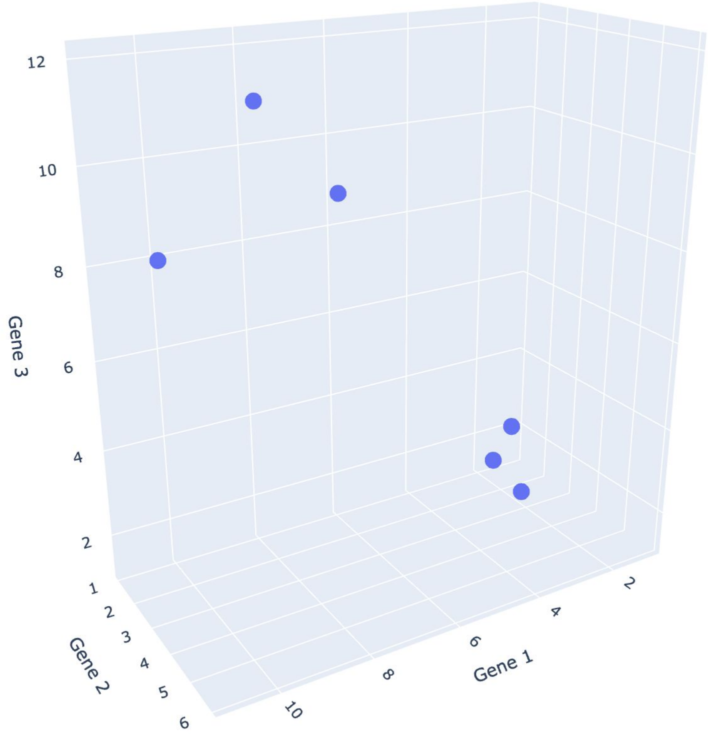


# Visualizing Gene Data

1039867

Visualization can help us identify clusters in our dataset.

| | Gene 1 | Gene 2 | Gene 3 | Gene 4 |
|---|---|---|---|---|
| **0** | 10 | 6.0 | 12.0 | 5 |
| **1** | 11 | 4.0 | 9.0 | 7 |
| **2** | 8 | 5.0 | 10.0 | 6 |
| **3** | 3 | 3.0 | 2.5 | 2 |
| **4** | 2 | 2.8 | 1.3 | 4 |
| **5** | 1 | 1.0 | 2.0 | 7 |



# Visualizing Gene Data

1039867

Since we are all 3D beings, we can't visualize beyond three dimensions! However, many datasets come with more than three features. What can we do?

| | Gene 1 | Gene 2 | Gene 3 | Gene 4 |
|---|---|---|---|---|
| **0** | 10 | 6.0 | 12.0 | 5 |
| **1** | 11 | 4.0 | 9.0 | 7 |
| **2** | 8 | 5.0 | 10.0 | 6 |
| **3** | 3 | 3.0 | 2.5 | 2 |
| **4** | 2 | 2.8 | 1.3 | 4 |
| **5** | 1 | 1.0 | 2.0 | 7 |

?

We reduce the dataset to lower dimensions → **Dimensionality reduction**

# Dimensionality

Lecture 24, Data 100 Spring 2024

# Intrinsic Dimension of Data

1039867

Suppose we have a dataset of:

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

In Linear Algebra:

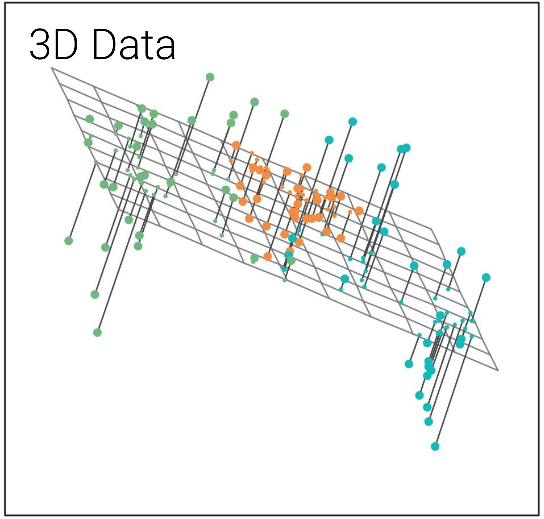
- **N points/row vectors** in a **d**-D space, OR
- **d** column vectors in an **N**-D space.

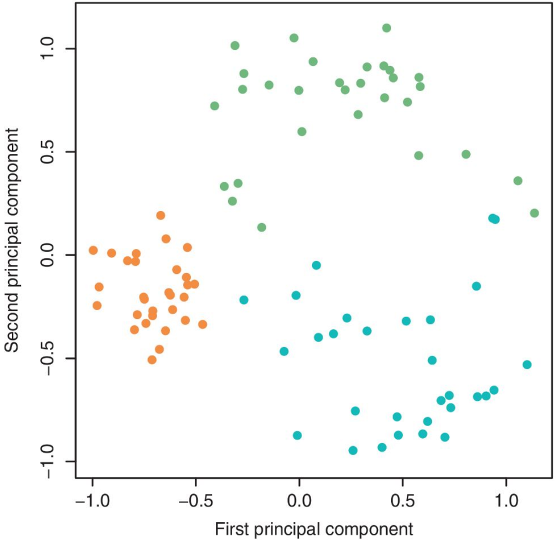
**Intrinsic Dimension** of a dataset is the **minimal set of dimensions** needed to approximately represent the data.

**Example:**

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

Intrinsic Dimension ≃ 2



# Dimensionality of the Column Space

1039867

Suppose we have a dataset of:

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

In Linear Algebra:

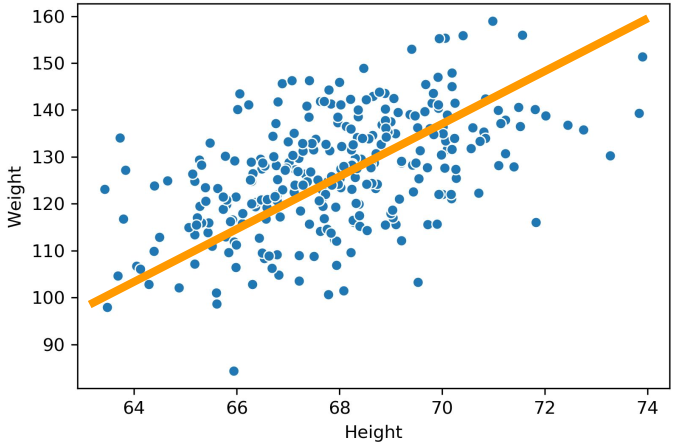
- **N points/row vectors** in a **d**-D space, OR
- **d** column vectors in an **N**-D space.

**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)

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




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

**Example:** 2 dimensions

# Dimensionality of the Column Space

1039867

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

# Dimensionality of the Column Space of Data?

9867

Consider the datasets shown.

What would you call the columns space of these datasets.

A. 1-dimensional,
B. 2-dimensional,
C. 3-dimensional
D. Something else

| Weight (lbs) | Weight (kg) |
|---|---|
| 113.0 | 51.3 |
| 136.5 | 61.9 |
| 153.0 | 69.4 |

Dataset 3

| Height (in) | Weight (kg) | Weight (lbs) | Age |
|---|---|---|---|
| 65.8 | 51.3 | 113.0 | 17 |
| 71.5 | 61.9 | 136.5 | 21 |
| 69.4 | 69.4 | 153.0 | 18 |

Dataset 4

slido

1039867

# What would you call these datasets?

ⓘ Start presenting to display the poll results on this slide.

# Dimensionality of Data?

1039867

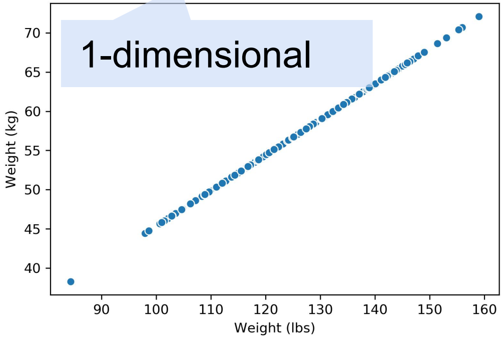
Consider the datasets shown.

What would you call these datasets?

**A.** 1-dimensional,
**B.** 2-dimensional,
**C.** 3-dimensional
**D.** Something else

| Weight (lbs) | Weight (kg) |
|---|---|
| 113.0 | 51.3 |
| 136.5 | 61.9 |
| 153.0 | 69.4 |

Dataset 3




| Height (in) | Weight (kg) | Weight (lbs) | Age |
|---|---|---|---|
| 65.8 | 51.3 | 113.0 | 17 |
| 71.5 | 61.9 | 136.5 | 21 |
| 69.4 | 69.4 | 153.0 | 18 |

Dataset 4

# Dimensionality of Data?

1039867

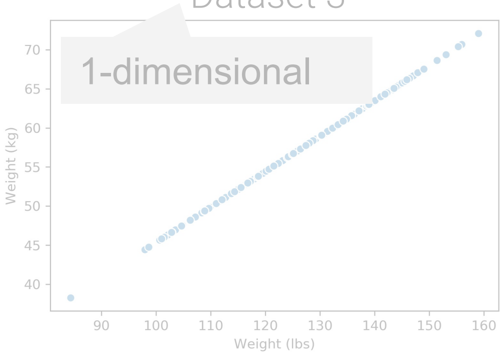
Consider the datasets shown.

What would you call these datasets?

**A.** 1-dimensional,
**B.** 2-dimensional,
**C.** 3-dimensional
**D.** Something else

| Weight (lbs) | Weight (kg) |
|---|---|
| 113.0 | 51.3 |
| 136.5 | 61.9 |
| 153.0 | 69.4 |

Dataset 3




| Height (in) | Weight (kg) | Weight (lbs) | Age |
|---|---|---|---|
| 65.8 | 51.3 | 113.0 | 17 |
| 71.5 | 61.9 | 136.5 | 21 |
| 69.4 | 69.4 | 153.0 | 18 |

Dataset 4

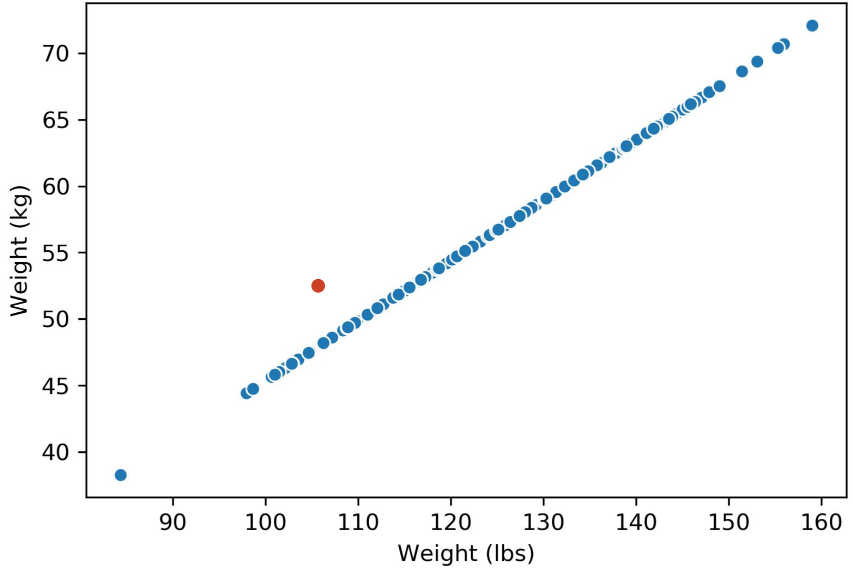
- **3-dimensional**, because two weight columns are redundant.
- Notice: Matrix of dataset has (column) **rank** 3!

# Dimensionality - what does it mean…?

1039867

Note that in the dataset below, I've added one **outlier** point to Dataset 3

- 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.

# Matrix Decomposition (Factorization)

Lecture 24, Data 100 Spring 2024

1039867

# Dimensionality Reduction as Matrix Factorization

1039867

**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

1039867

**Original Dataset**

**Reduced Dimension Dataset**

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

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

1039867

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





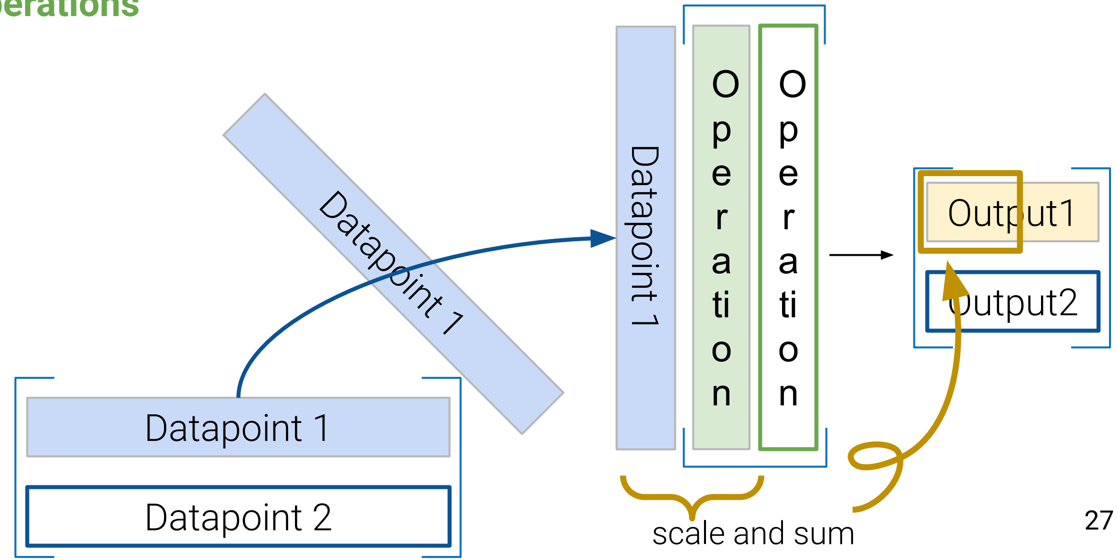
$$\begin{bmatrix}2 & 2 & 2\\ 5 & 8 & 0\end{bmatrix} \times \begin{bmatrix}2 & 1\\ 1 & 1\\ 4 & 1\end{bmatrix} = \begin{bmatrix}14 & 6\\ 18 & 13\end{bmatrix}$$

data

operations

View 1: Perform multiple linear operations on data.

- We use this view when building linear models.



# Multiplication View 2/2: Right Matrix Transforms Features

[LinAlg Review]

1039867

| | | |
|---|---|---|
| 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.

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

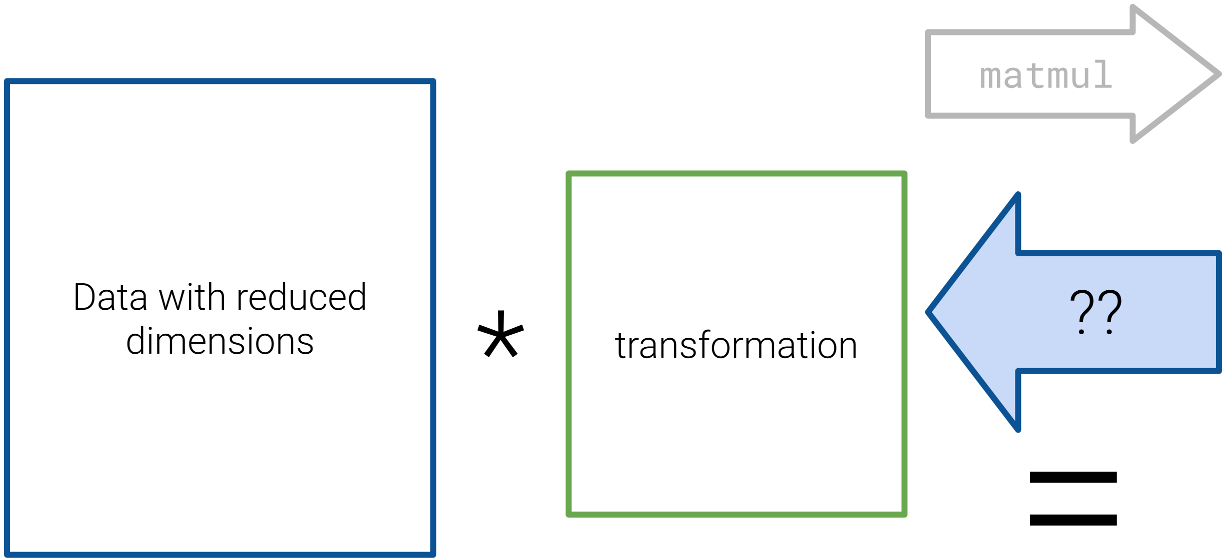
$$= \boxed{2} \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \boxed{1} \begin{bmatrix} 2 \\ 8 \end{bmatrix} + \boxed{4} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 14 \\ 18 \end{bmatrix}$$

View 2: Multiplication is a column transformation.

[3blue1brown's Essence of Linear Algebra]

# 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 |

**Goal**

**Transformation**

# Matrix Decomposition (Matrix Factorization)

1039867

**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

| Age | Height (in) |
|---|---|
| 182 | 28 |
| 399 | 30 |
| 725 | 33 |

*

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 1/12 |

2x3

**Goal** **Transformation**

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

# Matrix Decomposition: Infinite Ways

1039867

**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...

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

3x2

| | |
|---|---|
| 182 | 28 |
| 399 | 30 |
| 725 | 33 |

X

2x3

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 1/12 |

3x3

| | | |
|---|---|---|
| 182 | 28 | 2.33 |
| 399 | 30 | 2.5 |
| 725 | 33 | 2.75 |

X

3x3

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

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

slido

1039867

# What are possible matrix factorizations? Select all that apply.

ⓘ Click **Present with Slido** or install our Chrome extension to activate this poll while presenting.

# Matrix Decomposition: Limited by Rank

1039867

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 (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.

✅ (3x2) x (2x3)

✅ (3x3) x (3x3)

**C.** (3x1) x (1x3)

✅ (3x4) x (4x3)

✅ Something else

# Matrix Decomposition: Limited by Rank

1039867

3x4

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

X

4x3

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 99 | 31 | 17 |

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

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

Impossible, because rank of original > 1!

ax/bx = a/b = 182/399
ay/by = a/b = 28/30
**Contradiction!**

3x1

| |
|---|
| a |
| b |
| c |

X

1x3

| | | |
|---|---|---|
| x | y | z |

What are possible matrix factorizations? Select all that apply.

✅(3x2) x (2x3) ❌(3x1) x (1x3) ✅ Something else
✅(3x3) x (3x3) ✅(3x4) x (4x3)

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

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

# Matrix Decomposition: Limited by Rank

1039867

$$3\text{x}4 \quad \begin{bmatrix} 182 & 28 & 2.33 & 0 \\ 399 & 30 & 2.5 & 0 \\ 725 & 33 & 2.75 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 99 & 31 & 17 \end{bmatrix} \quad 4\text{x}3$$

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

Impossible, because rank of original > 1!

ax/bx = a/b = 182/399
ay/by = a/b = 28/30
**Contradiction!**

$$3\text{x}1 \quad \begin{bmatrix} a \\ b \\ c \end{bmatrix} \times \begin{bmatrix} x & y & z \end{bmatrix} \quad 1\text{x}3$$

What are possible matrix factorizations? Select all that apply.

✅(3x2) x (2x3) ❌(3x1) x (1x3) ✅ Something else
✅(3x3) x (3x3) ✅(3x4) x (4x3)

| 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?

# Automatic factorization

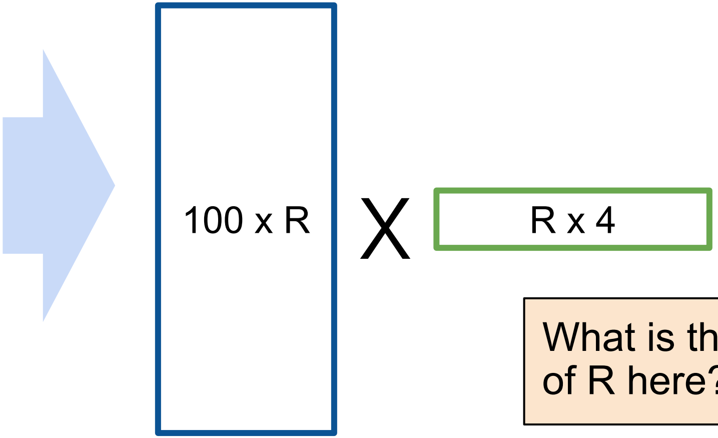
1039867

**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

1039867

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

Lecture 24, Data 100 Spring 2024



1039867

# Principal Component Analysis (PCA)

1039867

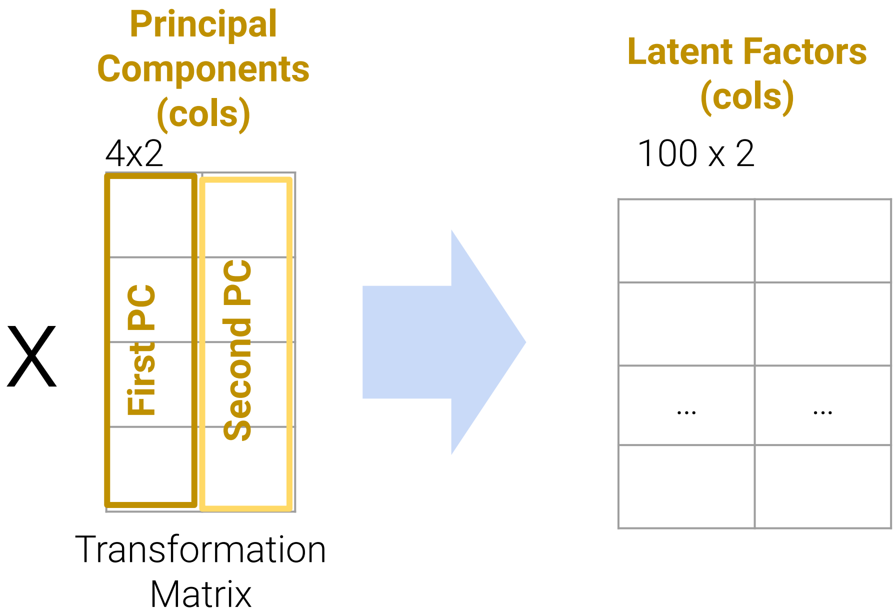
**Goal**: Transform observations from high-dimensional data down to **low dimensions** (often 2) through linear transformations.

**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 |



# Why perform PCA?

1039867

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

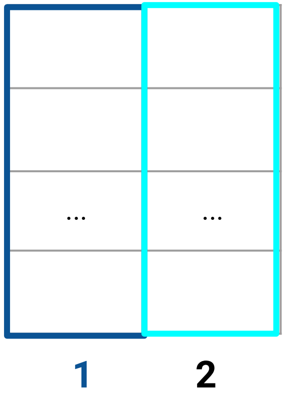
**Related Goal**: Low-dimension representation should capture the **variability** of the original data.

**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




Why **two** dimensions?

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

# Two Equivalent Framings of PCA

1039867

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

1039867

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 |
|---|---|---|---|---|

**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

1039867

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.

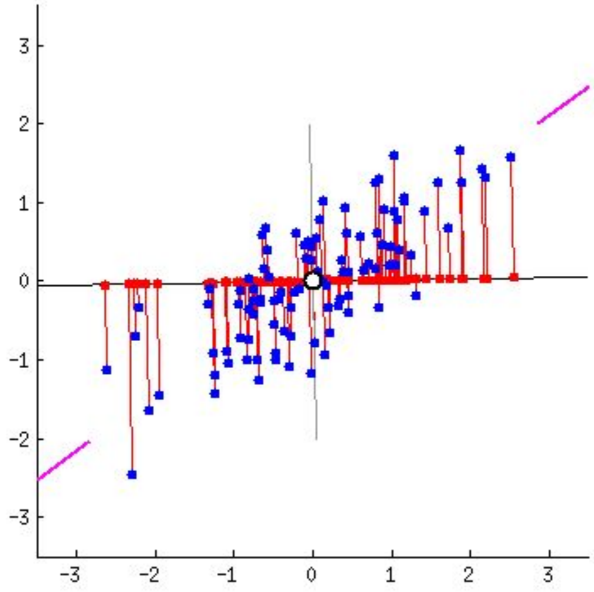
# Principal Component Analysis: If you're curious

(out of scope)

1039867

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.

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

[StackExchange]

1039867

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.

In practice, we don't carry out this procedure.

Instead, we use singular value decomposition (SVD) to find all principal components efficiently.

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

# Stretch Break!

1039867

# Deriving PCA as Error Minimization

Lecture 24, Data 100 Spring 2024

1039867

These are **new slides** for Spring 2024.

You are not expected to be able to be able to redo this derivation, however understanding the derivation may help with future assignments.

# Derive PCA using Loss Minimization

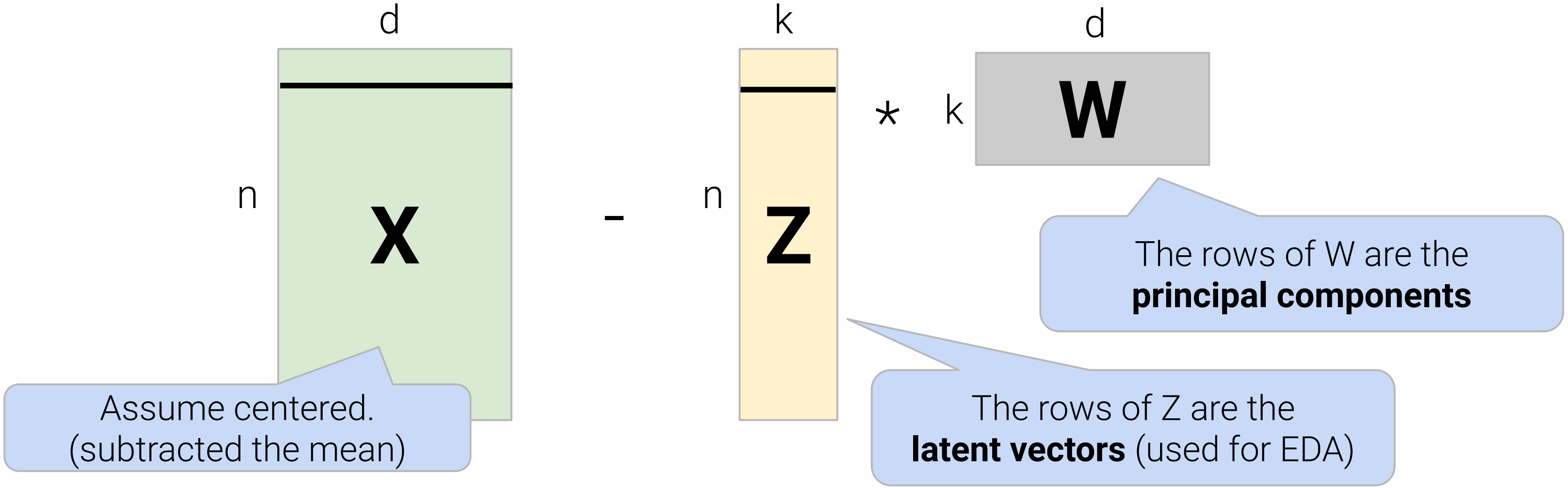
1039867

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

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

Row Vector

Row Vector



# Derive PCA using Loss Minimization

1039867

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

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

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

Row Vector

Column Vector

# Derive PCA using Loss Minimization

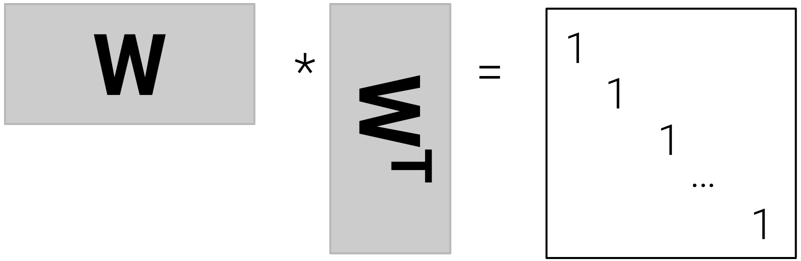
1039867

**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.

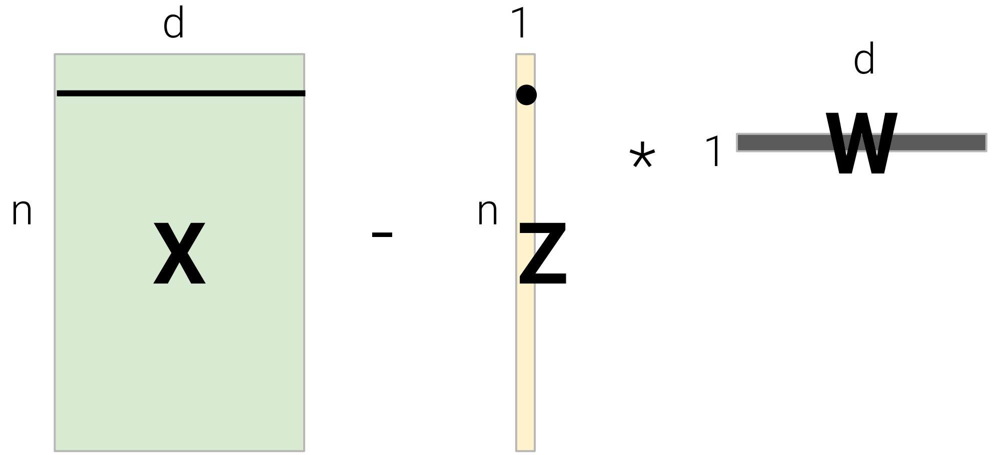


# Simplified Derivation: consider (k=1)

1039867

Let consider the situation when k=1:

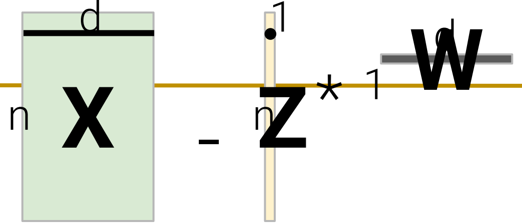
$$L(z,w) = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - z_i w\right)\left(X_i - z_i w\right)^T$$






Derivation based on Kevin Murphy's derivation in the excellent PML Textbook.

## Simplified Derivation: Differentiating wrt z




1039867

Let consider the situation when k=1:

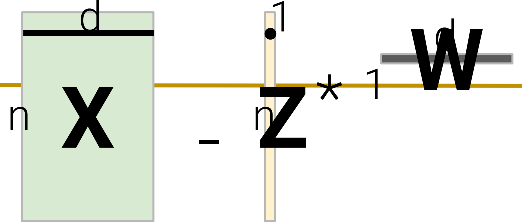
$$L(z,w) = \frac{1}{n}\sum_{i=1}^{n}(X_i - z_i w)(X_i - z_i w)^T$$

Expanding the loss:

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

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

# Simplified Derivation: Substituting soln for z




1039867

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

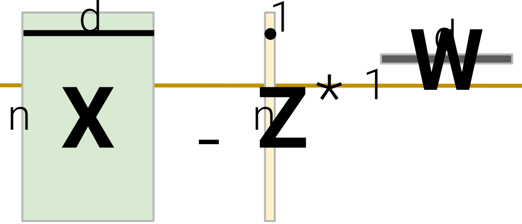
$$L(z, w) = \frac{1}{n}\sum_{i=1}^{n}\left(-2 z_i X_i w^T + z_i^2\right)$$

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

**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




1039867

Let consider the situation when k=1:

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

Taking the derivative with respect to $z_i$:

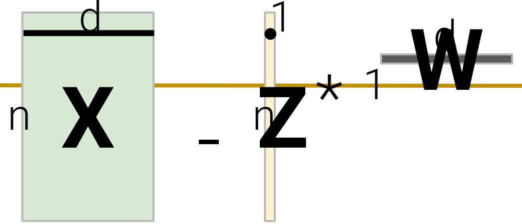
$$\frac{\partial}{\partial z_i} L(z, w) = \frac{1}{n} \left(-2 X_i w^T + 2 z_i\right)$$

Setting the derivative equal to 0 and solving for $z_i$:

$$z_i = X_i w^T$$

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

# Simplified Derivation: Substituting soln for z




1039867

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

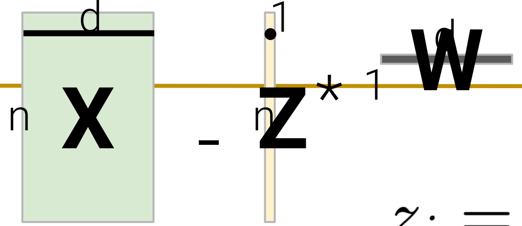
$$L(z, w) = \frac{1}{n} \sum_{i=1}^{n} \left(-2 z_i X_i w^T + z_i^2\right)$$

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

**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**

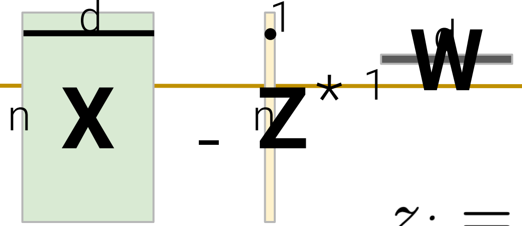
Use **Lagrange multiplier $\lambda$** to introduce the constraint **$ww^T$=1** to our optimization problem:

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

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




1039867

Use Lagrange multiplier $\lambda$ to introduce the constraint (**ww$^T$=1**)

$$z_i = X_i w^T$$

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

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 $\lambda$**

# 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 $w_1$:

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

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

# Extending the Derivation to the Second PC (Bonus)

1039867

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 $w_1$:

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

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

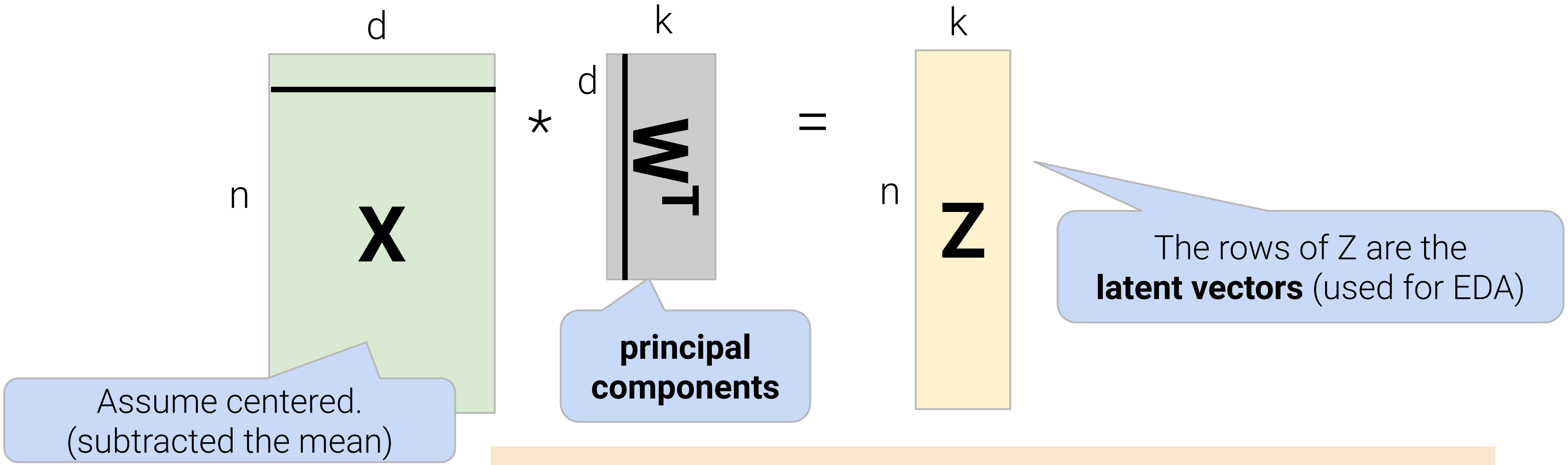
# Take Away from the Optimization Framing



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:




How do we compute the eigenvectors of the Cov. matrix?

# Lecture Stopped Here

1.

1039867

1039867

LECTURE 25

# PCA I

Content credit: Acknowledgments