Unsupervised Learning, Recommenders, Reinforcement Learning

Clustering is an **unsupervised machine learning** technique used to group similar data points together **without using labeled data**.

🧠 Definition:

Clustering involves organizing a dataset into **clusters** (groups) such that:

- Data points in the **same cluster** are **more similar** to each other.
- Data points in **different clusters** are **more different**.

🔍 Goal:

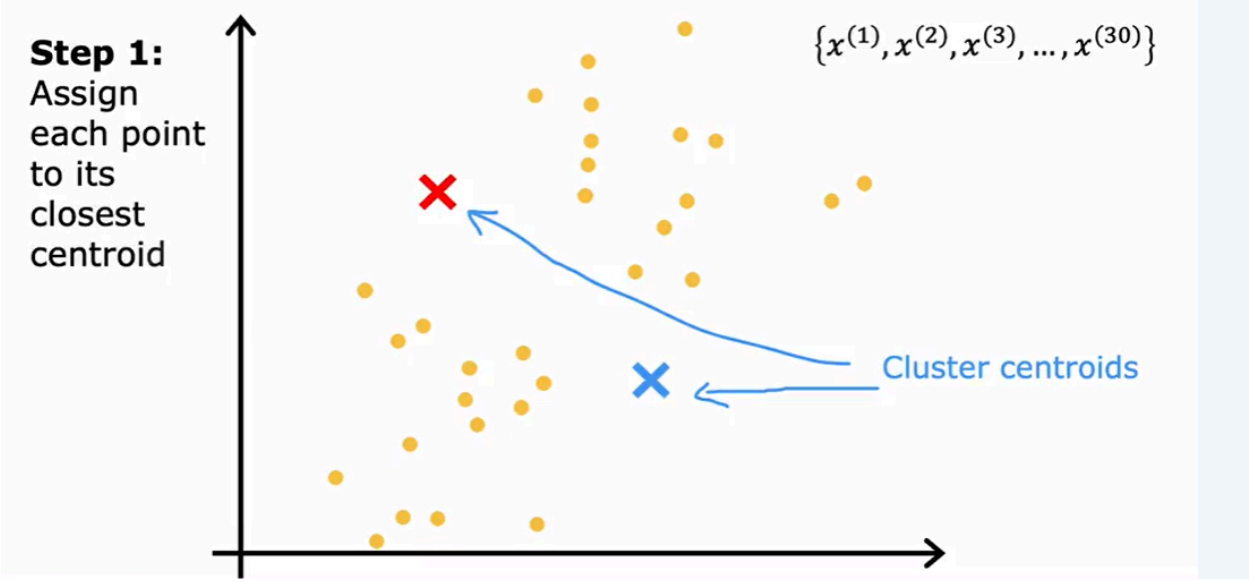
Find **natural groupings** or structure in the data — for example:

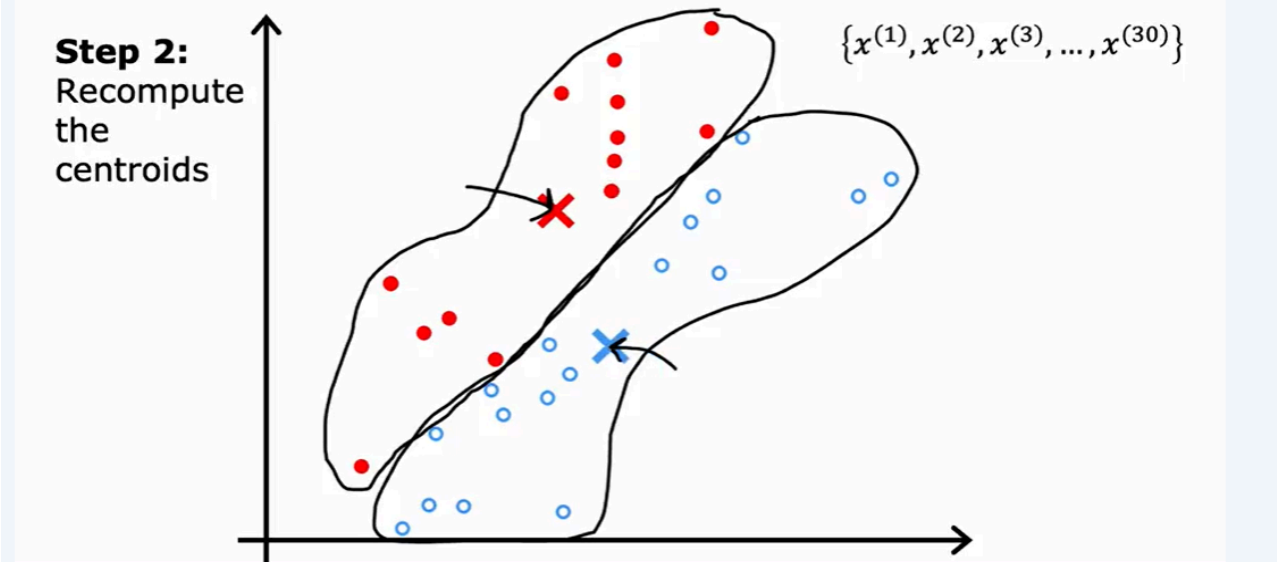
- Grouping customers by buying behavior.
- Organizing documents by topic.
- Segmenting images based on color or texture.

Popular Clustering Algorithm:

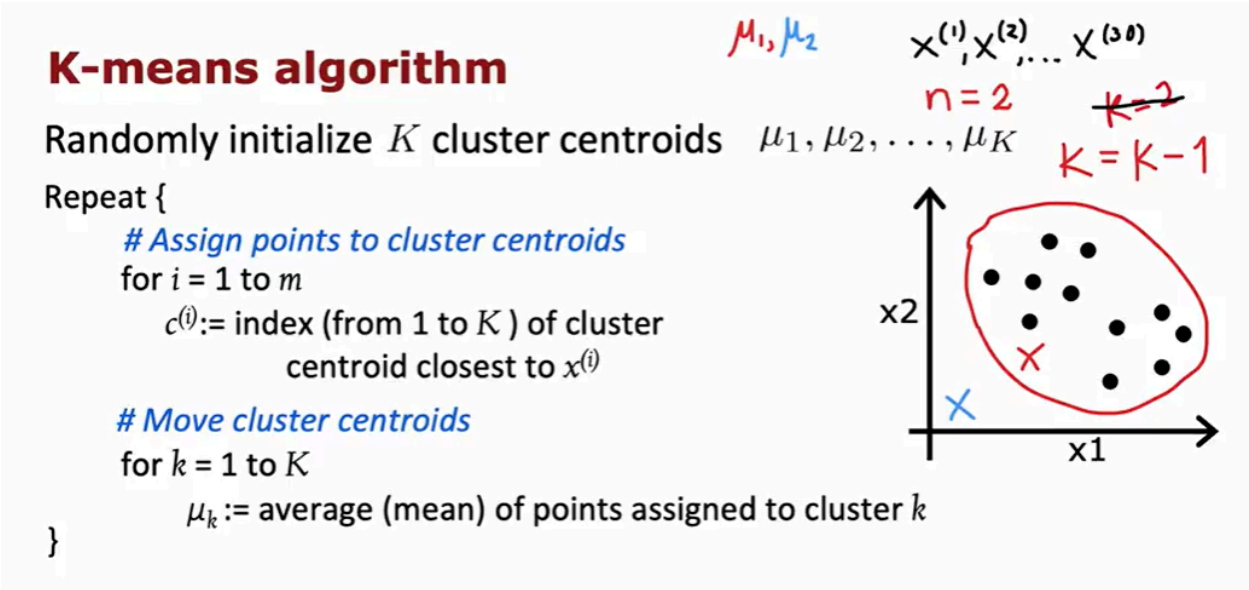
Algorithm	Description
K-Means	Partitions data into `K` clusters by minimizing the distance to cluster centroids.
Hierarchical Clustering	Builds nested clusters using a tree-like structure (dendrogram).
DBSCAN	Groups based on density, great for finding clusters of **arbitrary shapes** and removing outliers.
Gaussian Mixture Models (GMM)	Uses **probabilistic** approach assuming data comes from a mixture of Gaussians.

K -means Clustering :

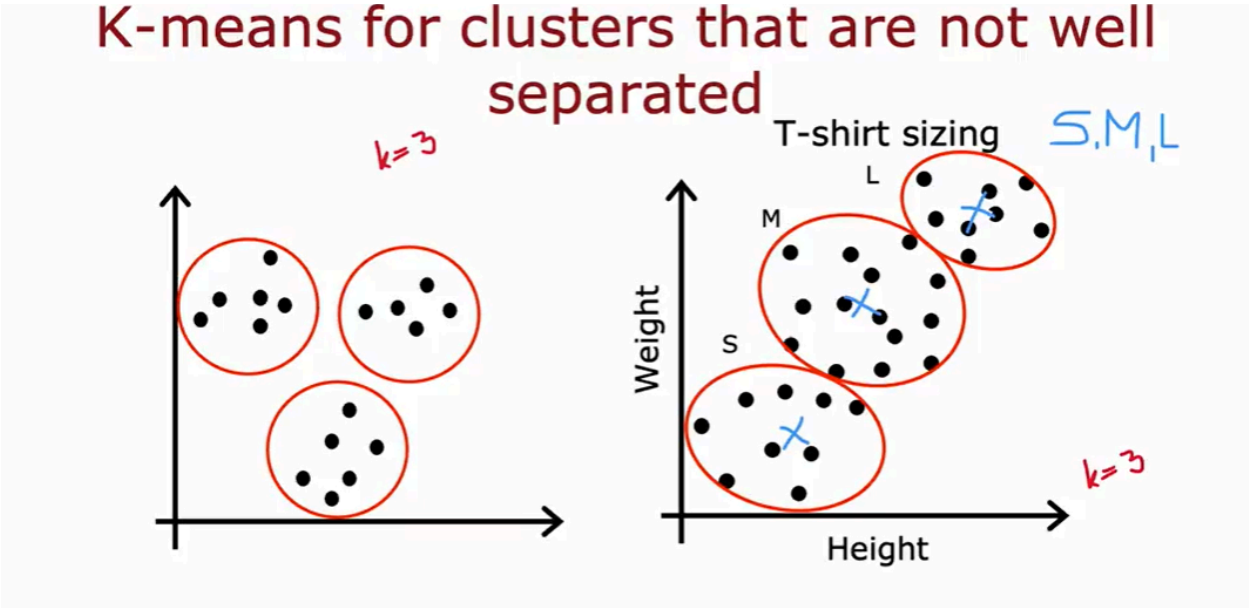




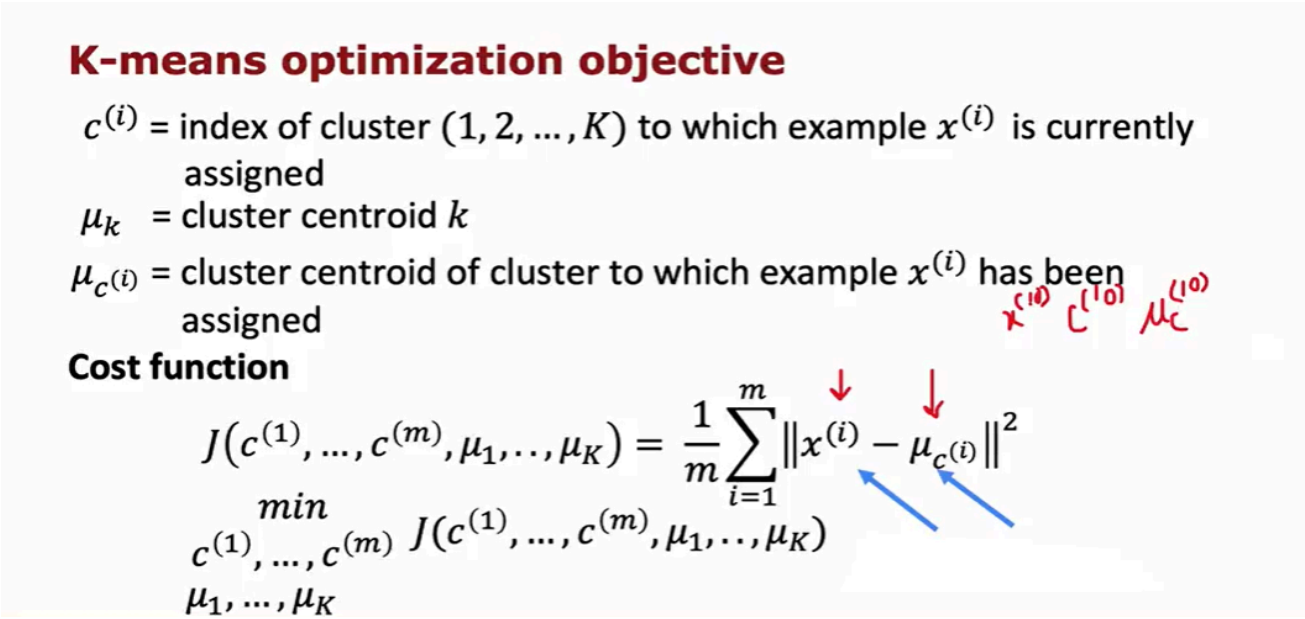
K means clustering algorithm:



Here , if any cluster is not associated with any point then just remove that cluster. To find the new cluster position just find average coordinate point of both x and y for the point that cluster associated with. That will be new point of that cluster.



Even if the data are not lied in well separated , the k-means clustering can do clustering on them.



The objective function here is used to check for a classification problem how many number of cluster is needed and in this case number of cluster vs objective function value is taken where number of cluster will be in x -axis and objective function value will be in y-axis and an elbow shape curved will be created and from that graph the number of cluster of that elbow point is appropriate.

💡 Cost Function in K-Means (a.k.a. Objective Function)

K-Means aims to minimize the **total intra-cluster variance**, also called the **Within-Cluster Sum of Squares (WCSS)**.

🎛️ Mathematical Form:

$$J = \sum_{i=1}^{K} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

Where:

- K: number of clusters
- C_i: set of points in the **i-th** cluster
- μ_i: centroid (mean) of cluster C_i
- x_j: data point in cluster C_i
- $\|x_j - \mu_i\|^2$: squared Euclidean distance from the point to the cluster center

🔍 What It Means:

- For each cluster, compute the squared distance between each point and its centroid.
- Sum up all those distances across all clusters.
- The **goal is to minimize this total cost**, so that data points are as close as possible to their respective centroids.

📉 Intuition:

- The better the clustering, the smaller the total distance between points and their cluster centers.
- During training, K-Means **reassigns points and recomputes centroids** to keep reducing this cost.

📈 Elbow Method & Cost:

You can use the cost function values (WCSS) at different values of `K` to visualize the **elbow method**, which helps decide the optimal number of clusters.

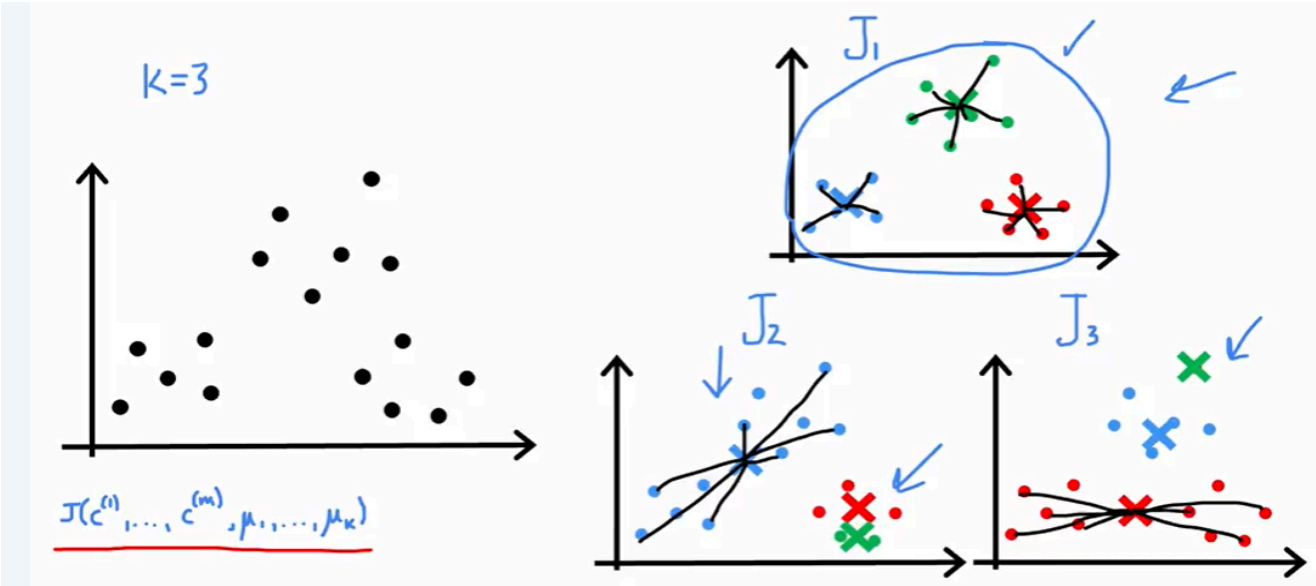
🔁 Optimization Strategy:

K-Means uses an **iterative greedy approach** (Lloyd's Algorithm):

1. Assign points to the nearest cluster.
2. Recompute the centroids.
3. Re-evaluate the cost.
4. Repeat until convergence (no major change in cost or assignments).

After every single iteration of k-means the WCSS function or distortion function value should go down otherwise there is a bug in the algorithm .

Initializing K-means:



K- means algorithm:

K-means algorithm

Step 0: Randomly initialize K cluster centroids $\mu_1, \mu_1, \ldots, \mu_k$

Repeat {

Step 1: Assign points to cluster centroids

Step 2: Move cluster centroids

}

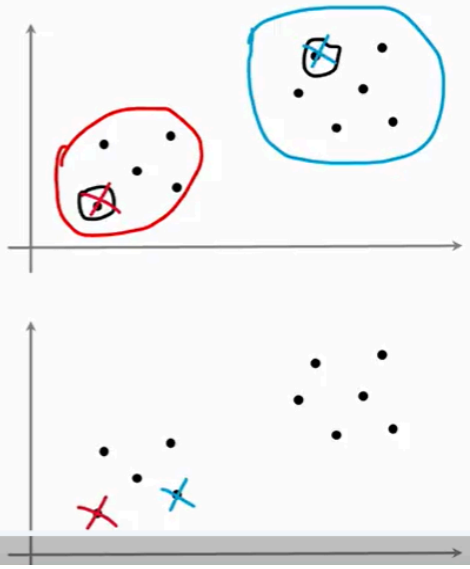
Step-0:

Random initialization

Choose $K < m$

Randomly pick K training examples.

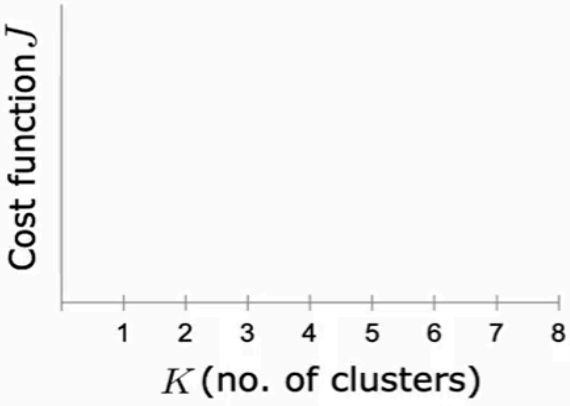
Set $\mu_1, \mu_1, \ldots, \mu_k$ equal to these K examples.

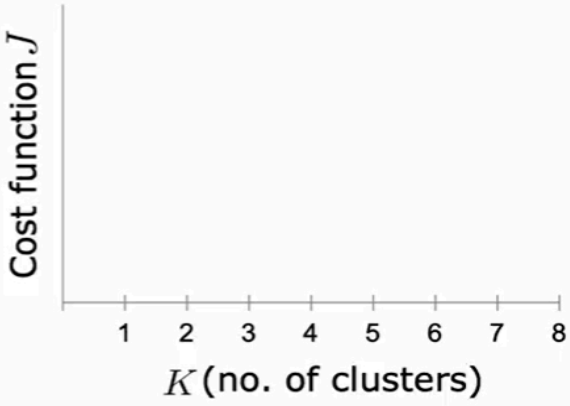


Choosing the right value of K:

Choosing the value of K

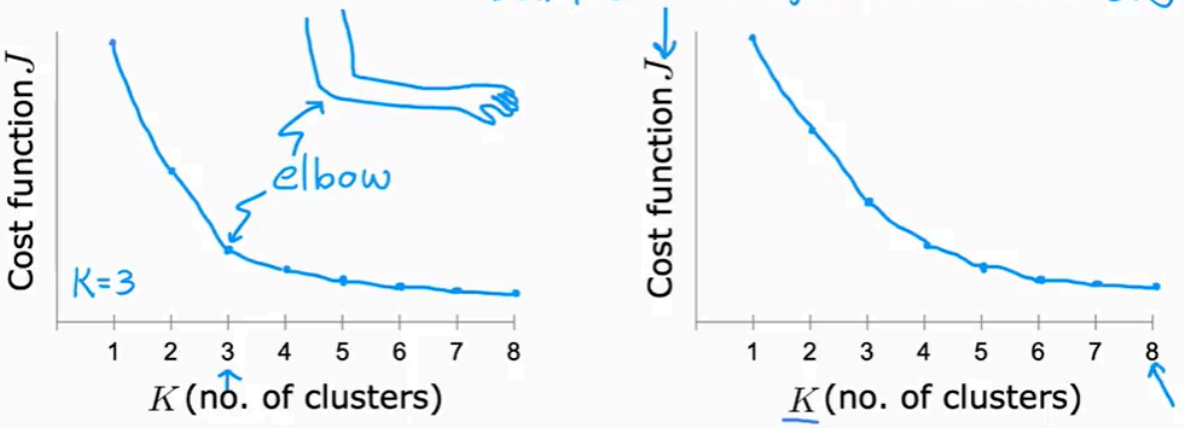
Elbow method





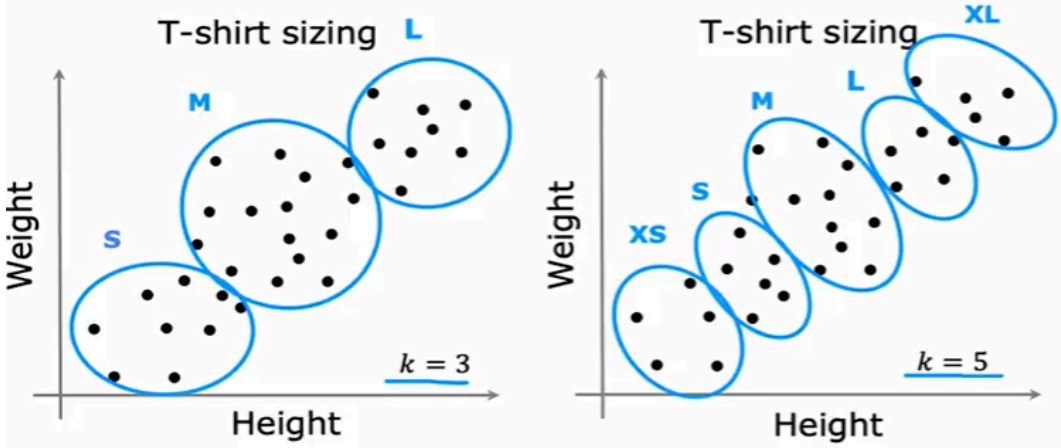
Choosing the value of K

Elbow method

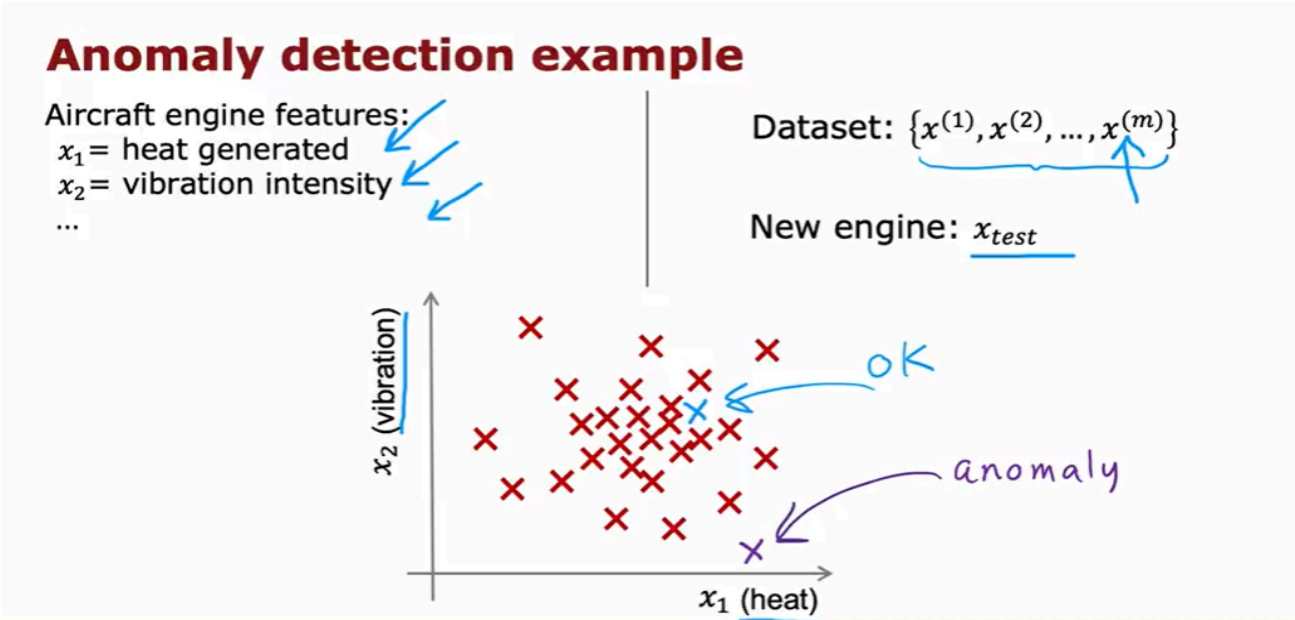


Choosing the value of K

Often, you want to get clusters for some later (downstream) purpose.
Evaluate K-means based on how well it performs on that later purpose.



Anomaly detection :



📊 The Data (Training Set)

- You have a **dataset** of normal engine behavior:

$$\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$$

  - Each `x` is a point with features like:
    - x_1: heat generated
    - x_2: vibration intensity
  - These are shown as **red crosses** clustered together on the plot.

🆕 New Data Point

- You get a **new engine** data point: x_{test}
- You want to check if it's behaving **normally** or it's an **anomaly**.

🧠 How to Detect Anomaly

You typically do this in one of two ways:

✅ 1. **Distance-based Approach**:

- If x_{test} is **too far** from the cluster of normal data → It's an anomaly.
- In the graph:
  - The blue point is **OK** (close to the cluster).
  - The purple point (bottom-left corner) is **far away**, so it's marked as **anomaly**.

✅ 2. **Probability-based (Gaussian)**:

- Fit a probability distribution (like Gaussian) on normal data.
- If the **probability** of x_{test} under that model is **very low**, it's likely an anomaly.

🎯 Final Goal:

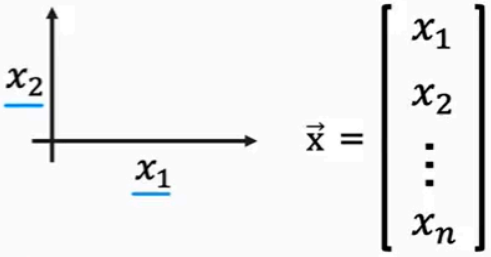
> Identify whether the new data point is **consistent** with what you've seen before or **suspiciously different**.

***For a single number gaussian or normal distribution works to detect anomaly , if the probability of the single feature output is very low then it is an anomaly. But in case of multi featured case there are algorithms for anomaly detection.**

Anomaly detection for multi featured dataset:

Density estimation

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(m)}\}$
Each example $\vec{x}^{(i)}$ has n features



$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \cdots * p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

$\sum$ "add" $\prod$

$p(x_1 = \text{high temp}) = 1/10$
$p(x_2 = \text{high vibra}) = 1/20$
$p(x_1, x_2) = p(x_1) * p(x_2)$
$= \frac{1}{10} \times \frac{1}{20} = \frac{1}{200}$

Anomaly detection example

how often log in?
how many web pages visited?
transactions?
posts? typing speed?

Fraud detection:
- $x^{(i)}$ = features of user i's activities
- Model $p(x)$ from data.
- Identify unusual users by checking which have $p(x) < \varepsilon$

perform additional checks to identify real fraud vs. false alarms

Manufacturing:
$x^{(i)}$ = features of product i

airplane engine
circuit board
smartphone

ratios

Monitoring computers in a data center:
$x^{(i)}$ = features of machine i
- x_1 = memory use,
- x_2 = number of disk accesses/sec,
- x_3 = CPU load,
- x_4 = CPU load/network traffic.

Gaussian (Normal) Distributions:

The **Gaussian distribution is** the same as the **Normal distribution** — they are two names for the **same concept** in statistics.

✅ Quick Explanation:

- "**Normal distribution**" is the more common name in general statistics.
- "**Gaussian distribution**" is named after **Carl Friedrich Gauss**, who described it mathematically.

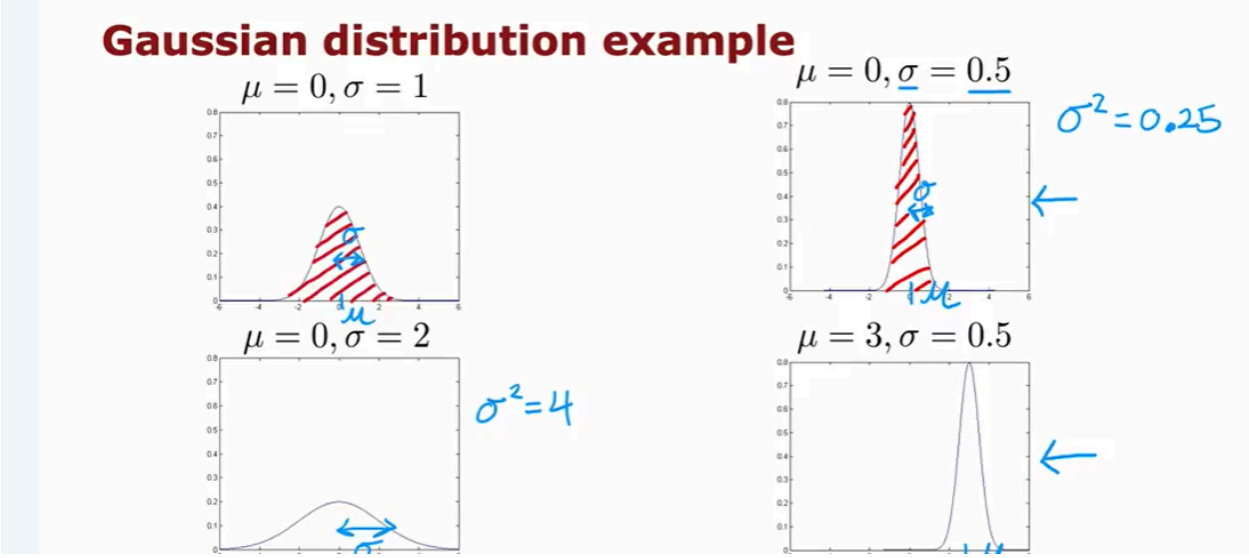
📌 Characteristics (as shown in your image):

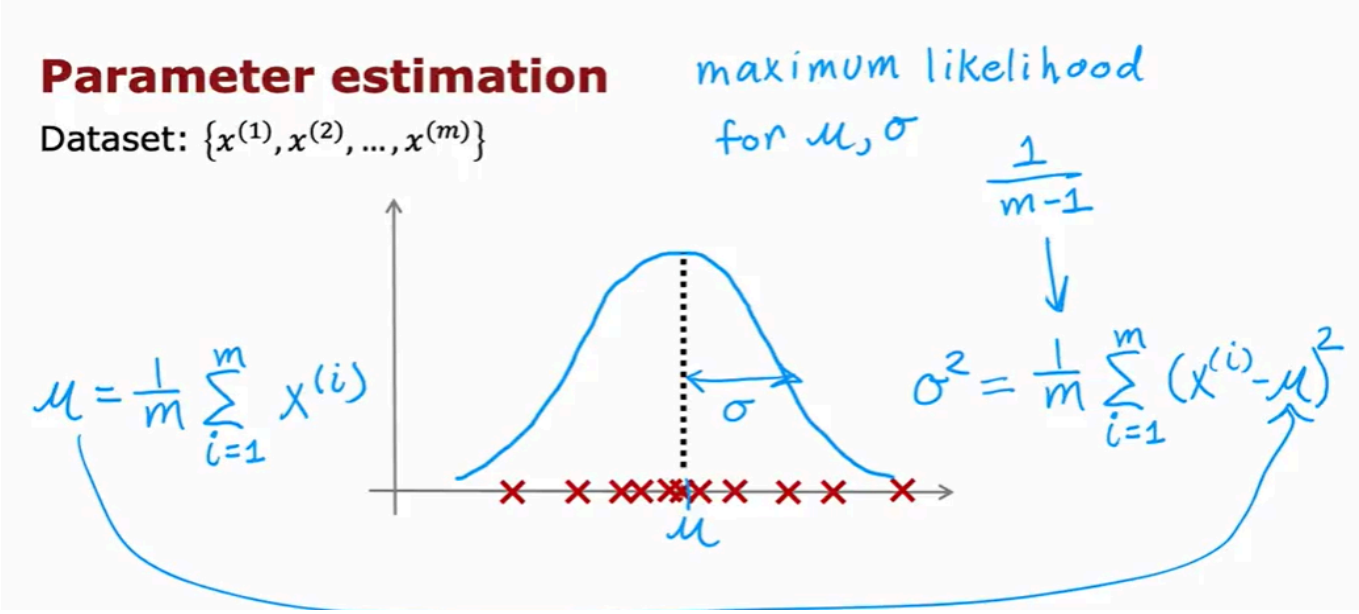
- Bell-shaped curve (like the **Liberty Bell** shown for analogy).
- Centered around **mean (μ)**.
- Spread is measured by **standard deviation (σ)**.
- The **variance** is σ^2.
- The peak of the curve is at **μ**, and the spread is determined by **σ**.

🤓 In math form:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Different choices of mu and sigma affects the gaussian distribution.





Algorithm for anomaly detection:

Anomaly detection algorithm

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

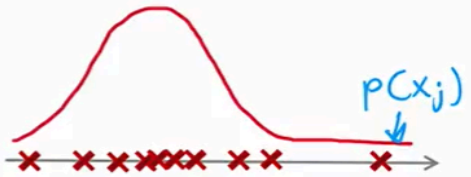
Vectorized formula

$$\vec{\mu} = \frac{1}{m}\sum_{i=1}^{m} \vec{x}^{(i)} \qquad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_n \end{bmatrix}$$

3. Given new example x, compute $p(x)$:

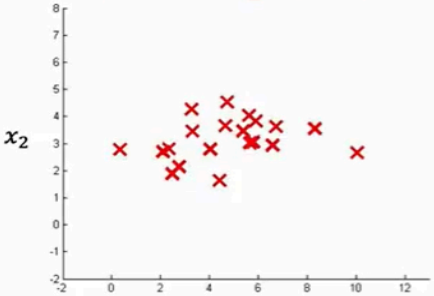
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} exp(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2})$$

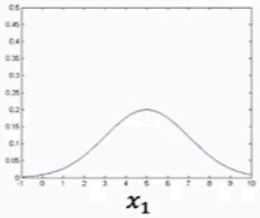
Anomaly if $p(x) < \varepsilon$



When multiplied p(x1) and p(x2) then the graph looks like a 3d shape.

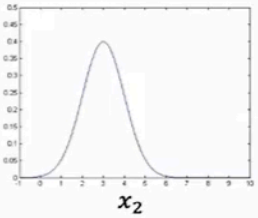
Anomaly detection example





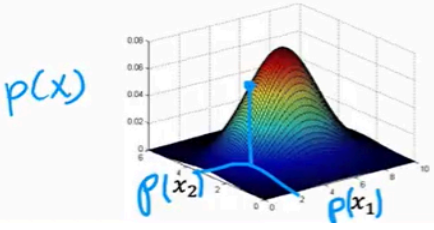
$\mu_1 = 5,\ \sigma_1 = 2$

$p(x_1; \mu_1, \sigma_1^2)$

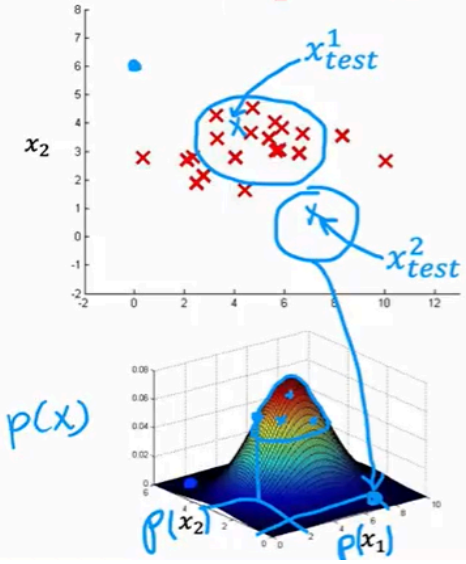


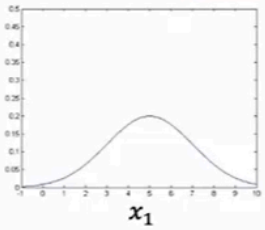
$\mu_2 = 3,\ \sigma_2 = 1$

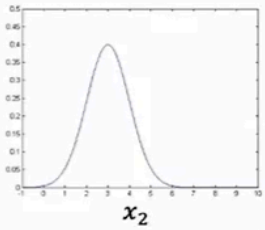
$p(x_2; \mu_2, \sigma_2^2)$



Anomaly detection example







$\mu_1 = 5,\ \sigma_1 = 2$

$p(x_1; \mu_1, \sigma_1^2)$

$\mu_2 = 3,\ \sigma_2 = 1$

$p(x_2; \mu_2, \sigma_2^2)$

$\varepsilon = 0.02$

$p\left(x_{test}^{(1)}\right) = 0.0426 \longrightarrow$ "ok"

$p\left(x_{test}^{(2)}\right) = 0.0021 \longrightarrow$ anomaly

course 2 week 3

skewed datasets

Algorithm evaluation

Fit model $p(x)$ on training set $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

On a cross validation/test example x, predict

$$y = \begin{cases} 1 & if\ p(x) < \varepsilon\ (anomaly) \\ 0 & if\ p(x) \geq \varepsilon\ (normal) \end{cases}$$

10

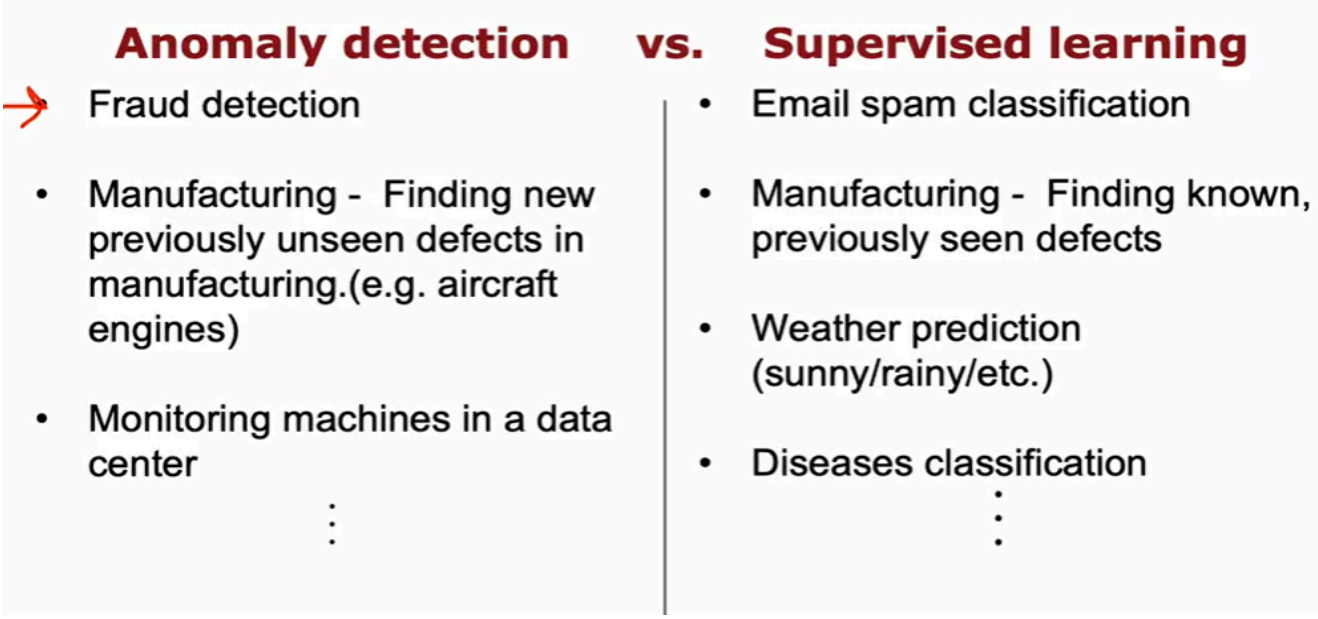
2000

Possible evaluation metrics:

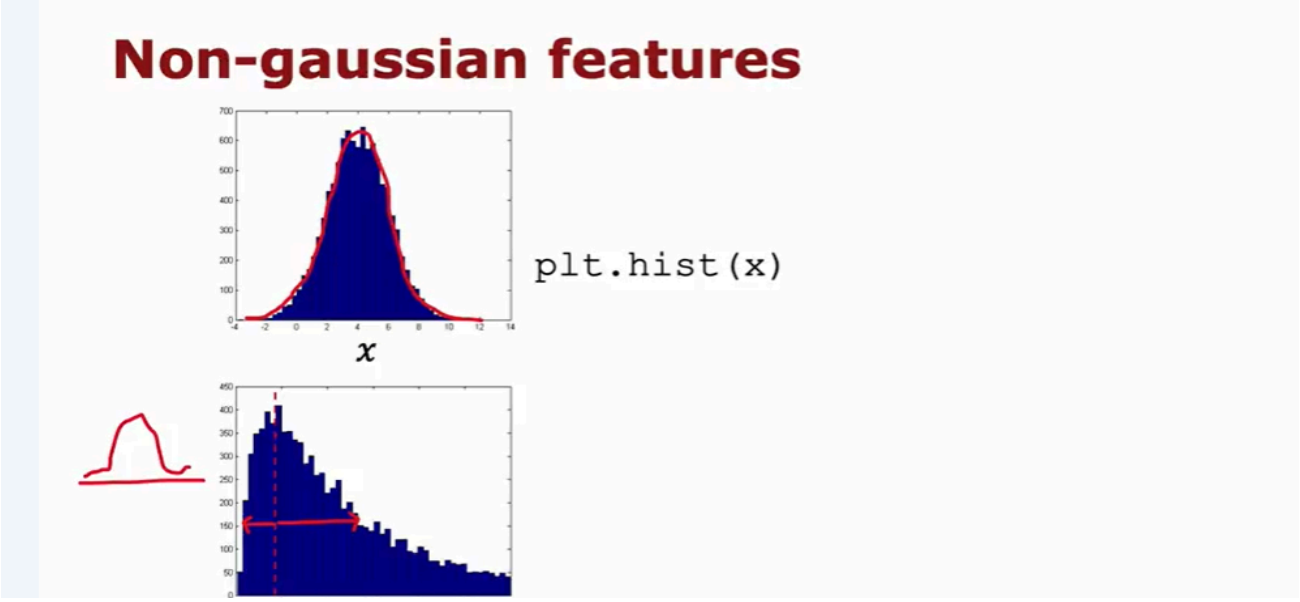
- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

Use cross validation set

Anomaly detection vs supervised learning :



Choosing what features to use:

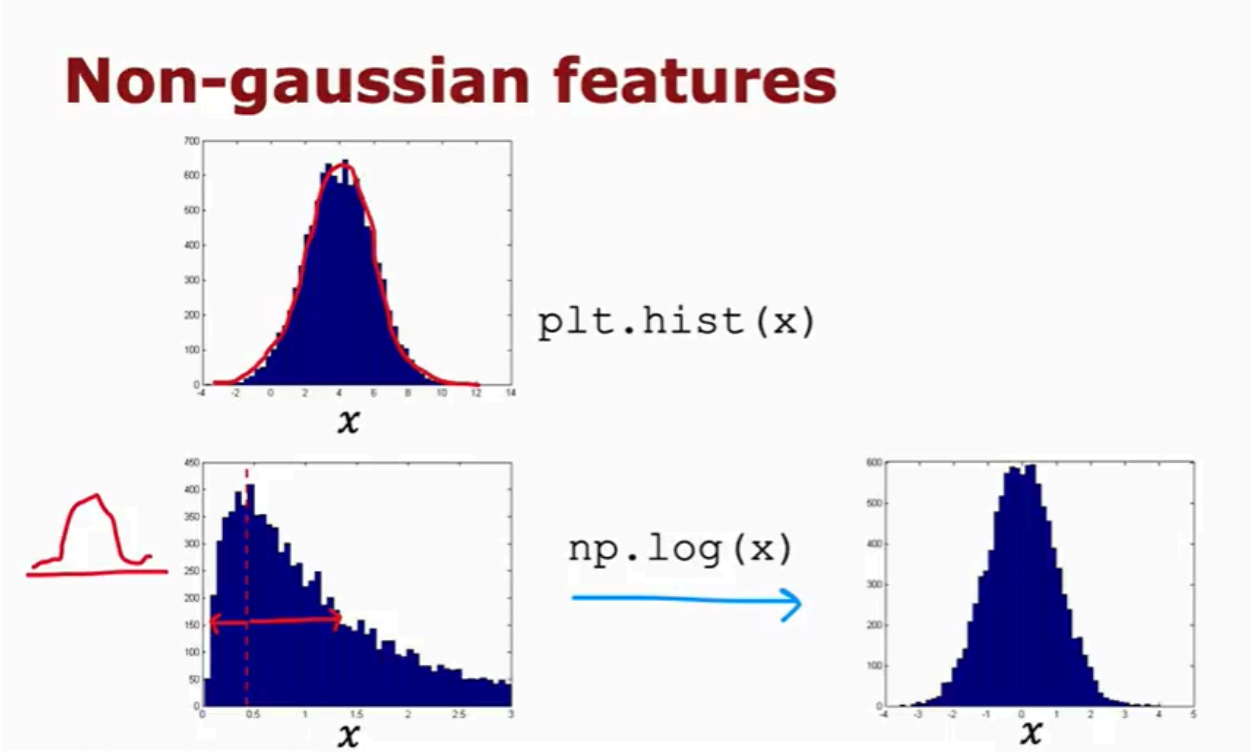


The first plot is looking like gaussian but actually it is not a gaussian features.

🔍 About the First Plot

- It **looks** like a bell (symmetric and unimodal), but:
  - **Just looking bell-shaped ≠ truly Gaussian**.
  - It might **not** follow the **exact mathematical formula** of a Gaussian.
  - Even small deviations (e.g., **heavy tails**, **skew**, **kurtosis**) make it *non-Gaussian*.

So, it's **visually similar** to a Gaussian, but the underlying distribution might not be a perfect Gaussian.

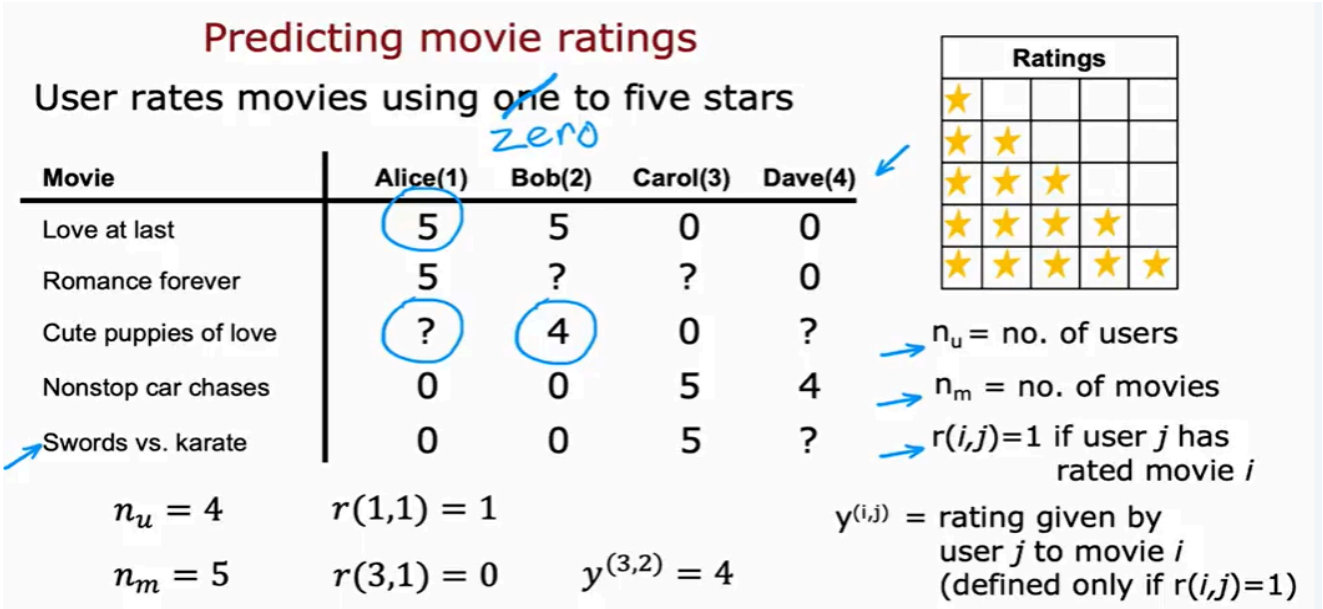


From non-gaussian features to turn into gaussian like features.

Common ways to make a feature non-gaussian:

Transformation	Use Case	Code Example
Log Transform	Skewed right (long tail on the right)	`np.log(x + 1)`
Square Root Transform	Moderate right-skew	`np.sqrt(x)`
Box-Cox Transform	Best for positive data, tries different powers	`scipy.stats.boxcox(x)`
Yeo-Johnson Transform	Handles both positive and negative values	`PowerTransformer(method='yeo-johnson')`
Z-score Standardization	Not Gaussianizing, but helps shape & scale	`(x - mean) / std`

Recommender System :



PCA[Principal Component Analysis] :

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction. It transforms a set of possibly correlated variables (features) into a smaller number of uncorrelated variables called principal components. These components capture the most significant variations in the data, with the first component accounting for the most variance, the second for the second most, and so on.

PCA is widely used for:

1. **Reducing the number of features**: This is particularly useful when you have high-dimensional data (many features) and want to reduce computational complexity without losing much information.

2. **Data visualization**: By reducing dimensions to 2 or 3 components, you can plot data points and observe patterns.

3. **Noise reduction**: By keeping only the most significant components, PCA helps remove noise and irrelevant variations.

The steps involved in PCA:

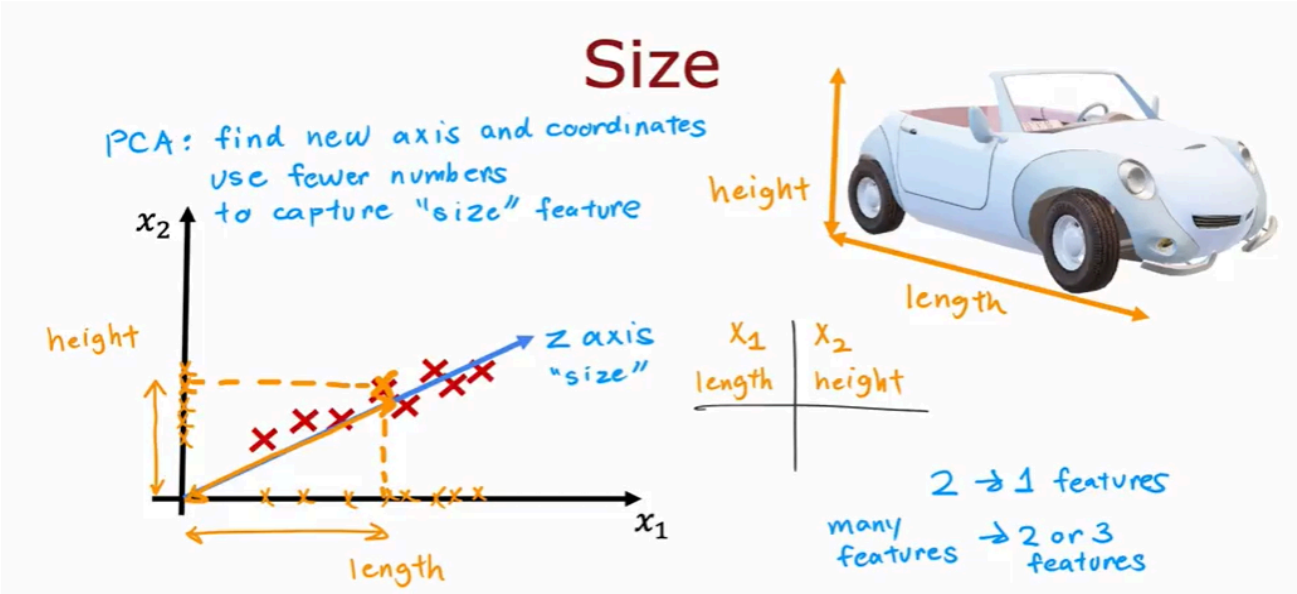
1. **Standardize** the data (subtract the mean and divide by the standard deviation) to ensure that all features are on the same scale.

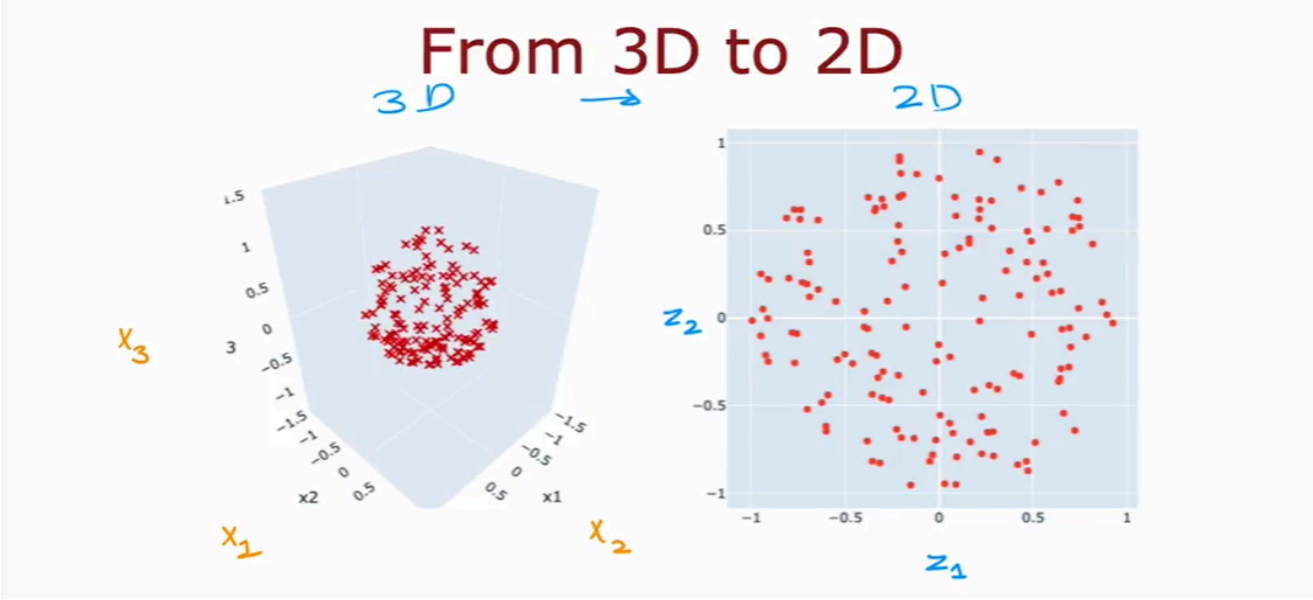
2. **Calculate the covariance matrix** to understand how the features are correlated with each other.

3. **Compute the eigenvalues and eigenvectors** of the covariance matrix.

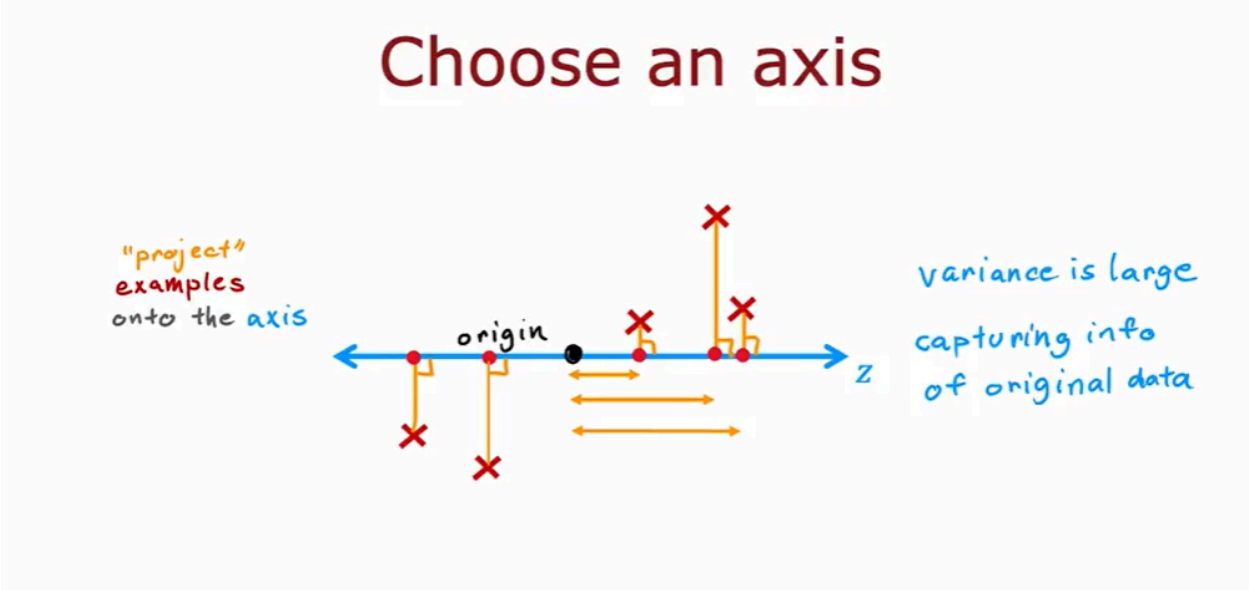
4. **Select the principal components** (the eigenvectors corresponding to the largest eigenvalues).

5. **Transform the data** into the new coordinate system formed by the principal components.



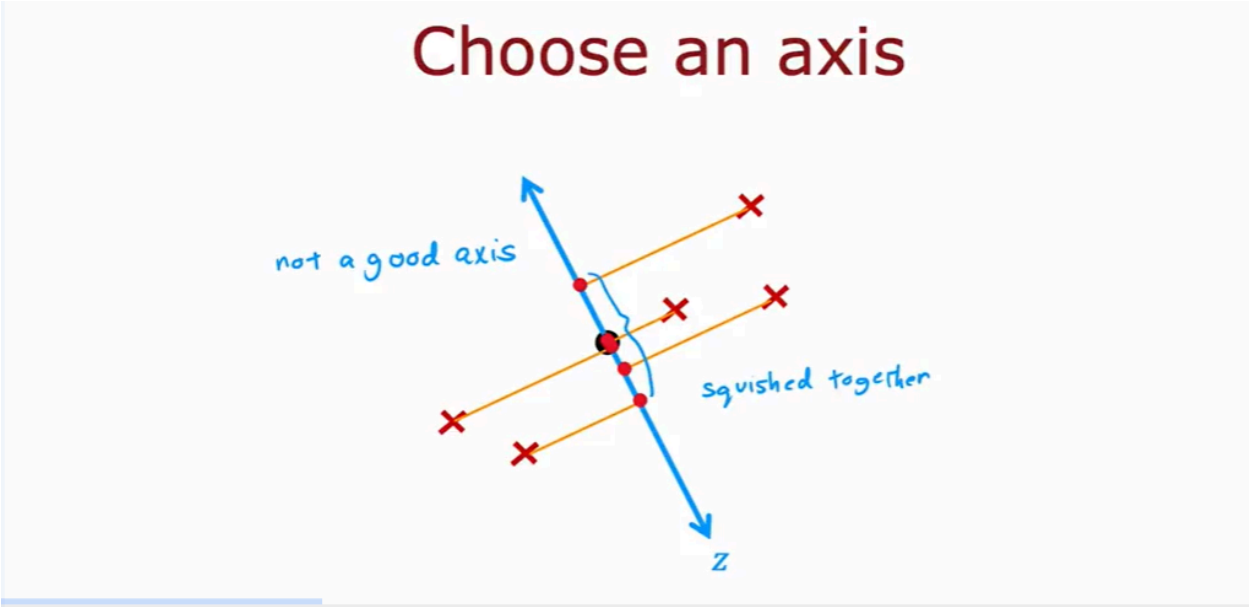


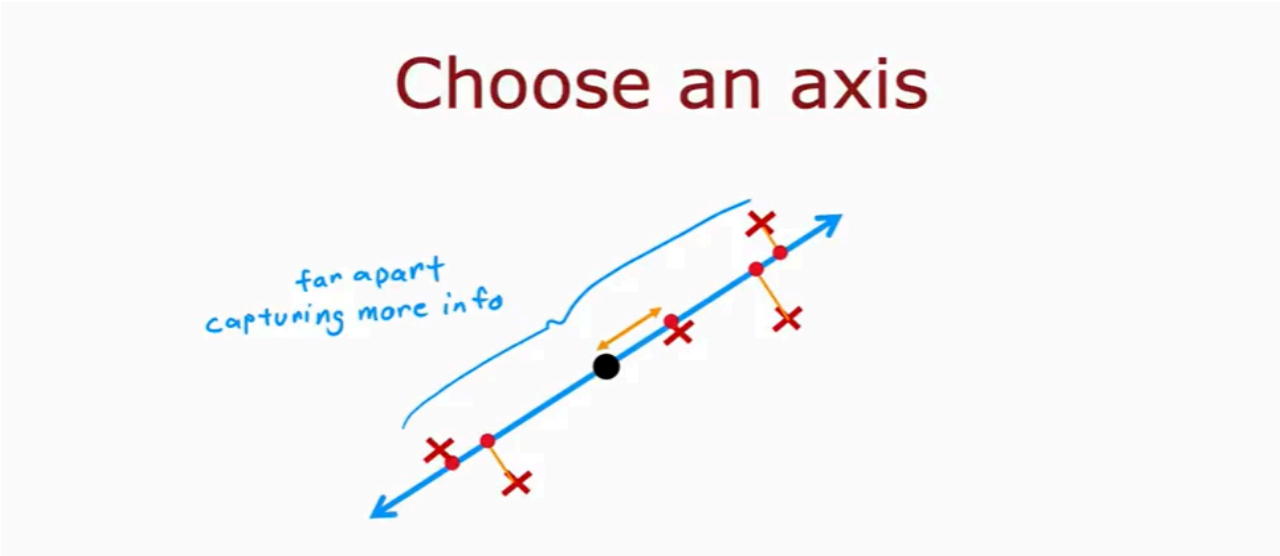
Instead of having 3 features x1,x2,x3 reduces it into z1 and z2 axis and it is convenient way to visualize.



The projection from the real data point to the z axis should be in such a way that the variance among the data point projected onto the z axis should be maximum . This means the z axis should capture as much information as it can from the real data point .

This type of choice is squished and the variance is much less here. This type of distribution is discouraged to apply as principle component.

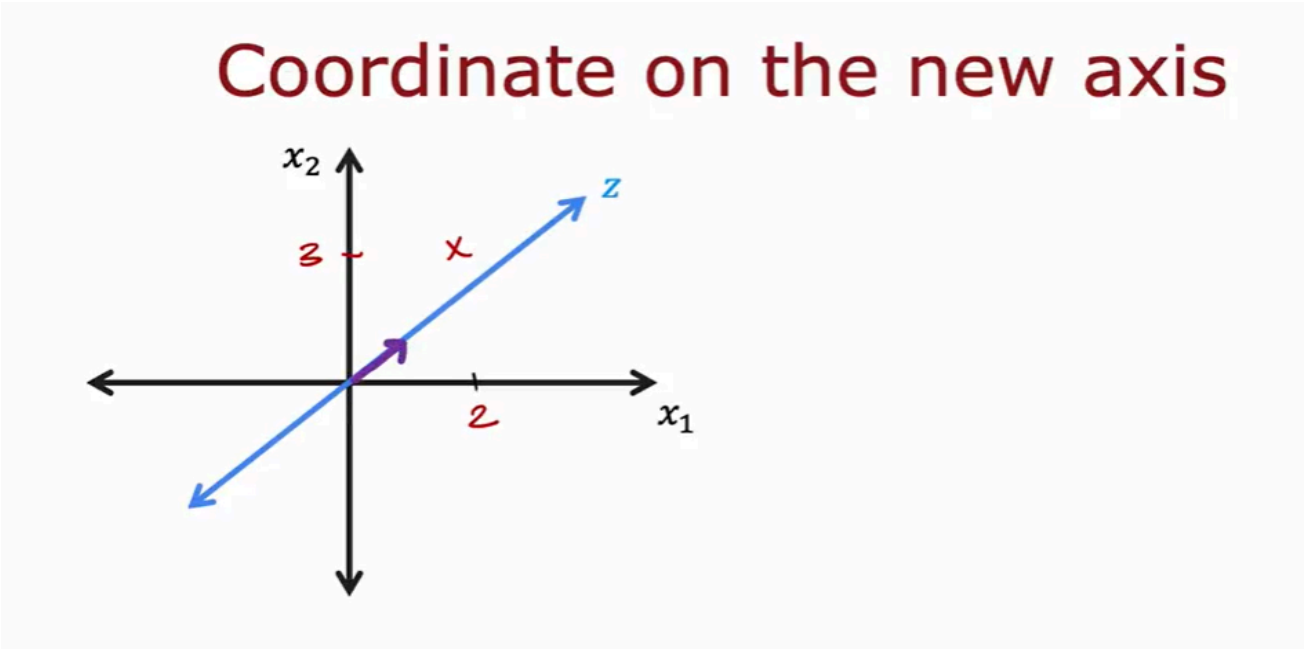


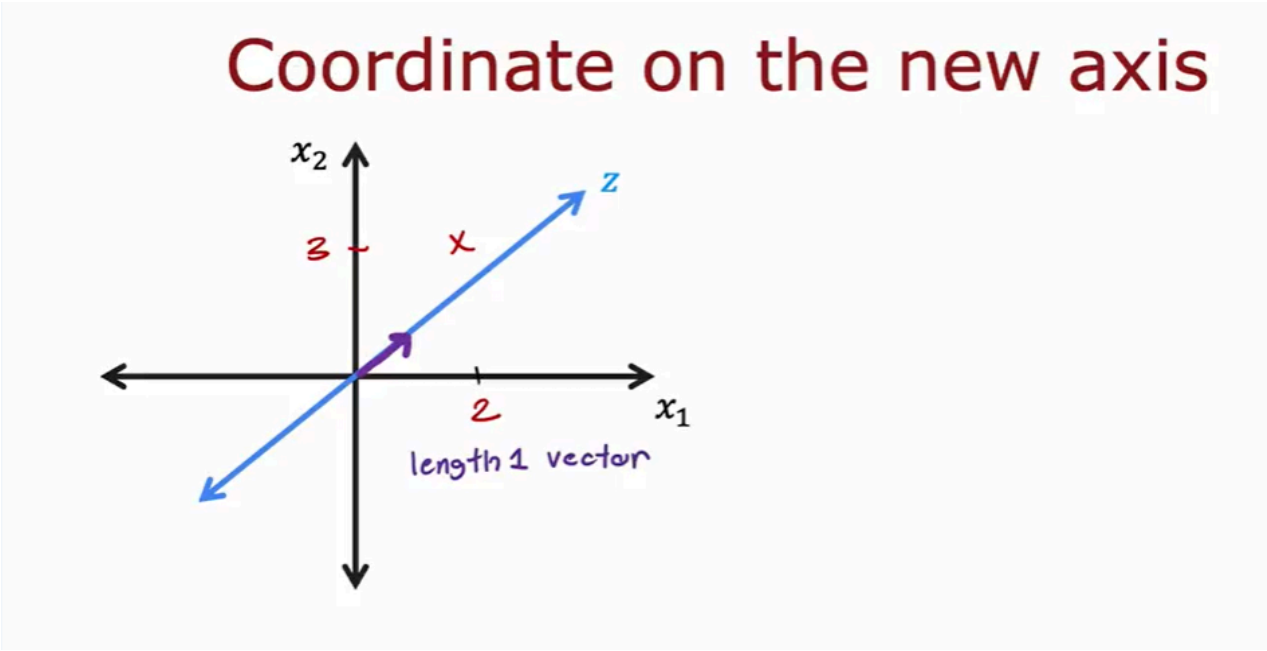


This is capturing a large amount of information and it is called principle component.

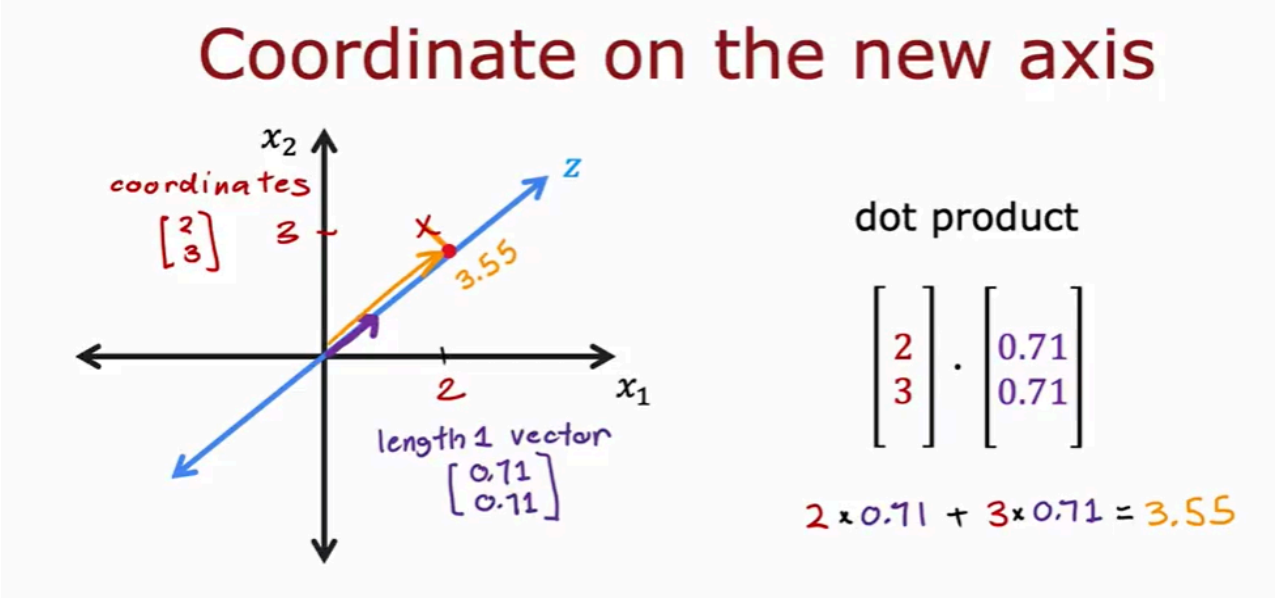
Example:

Let’s say for 2 data point 2 and 3 i want to draw or find the PCA it can be done by doing so



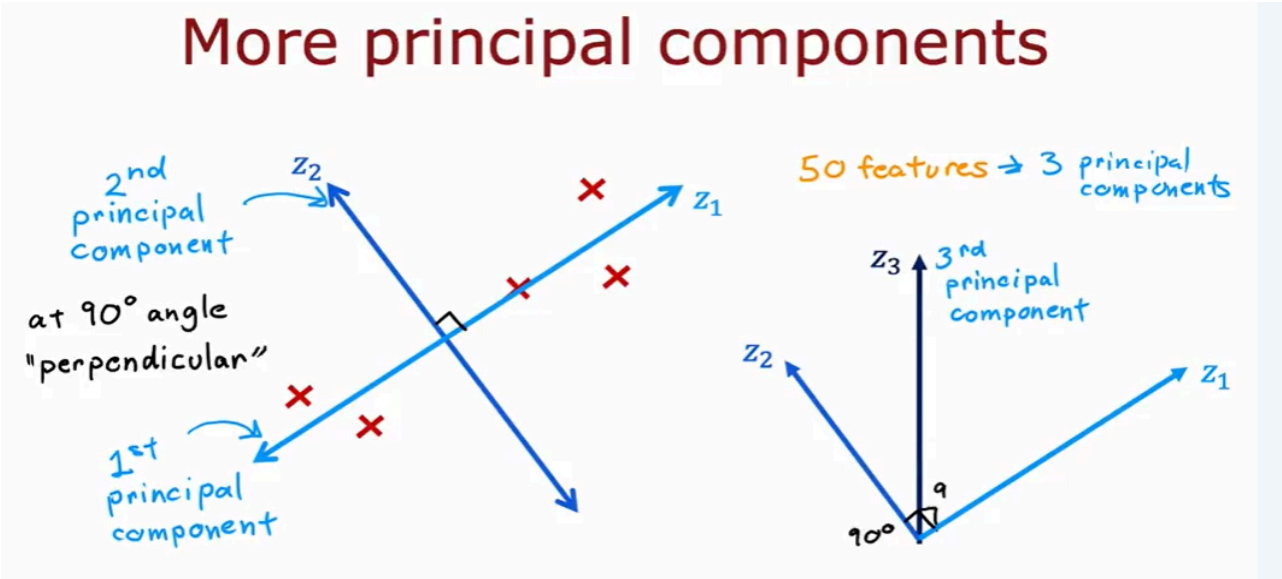


The PCA axis is z axis. Length 1 vector .



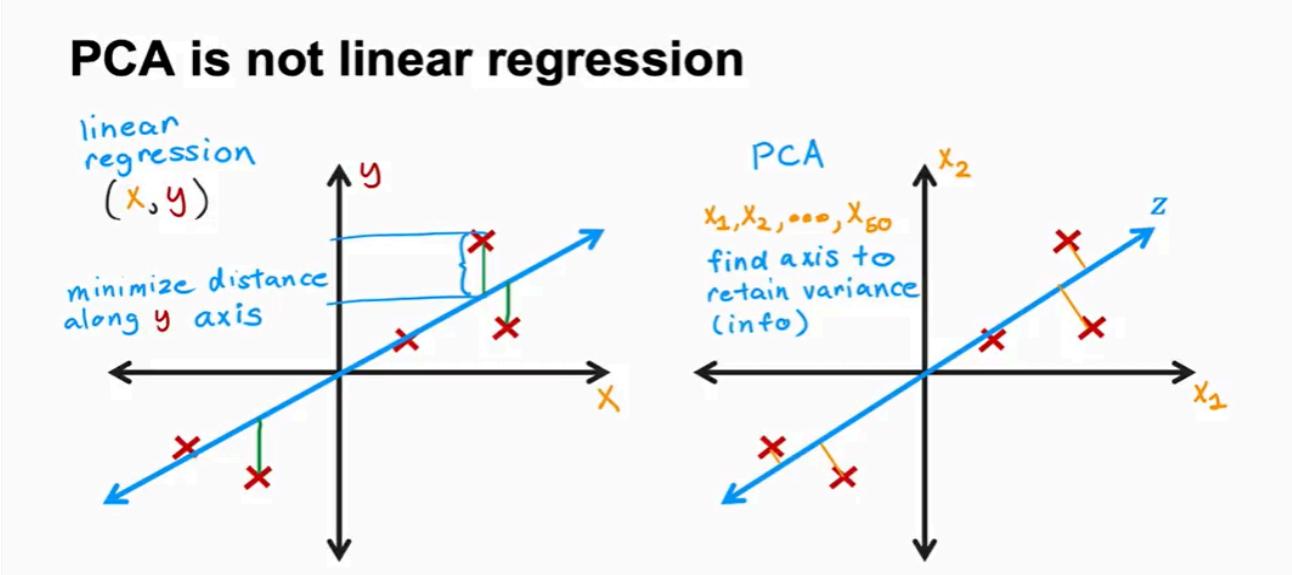
3.55 is the distance from the origin through the pca axis or z axis.

More PCA :

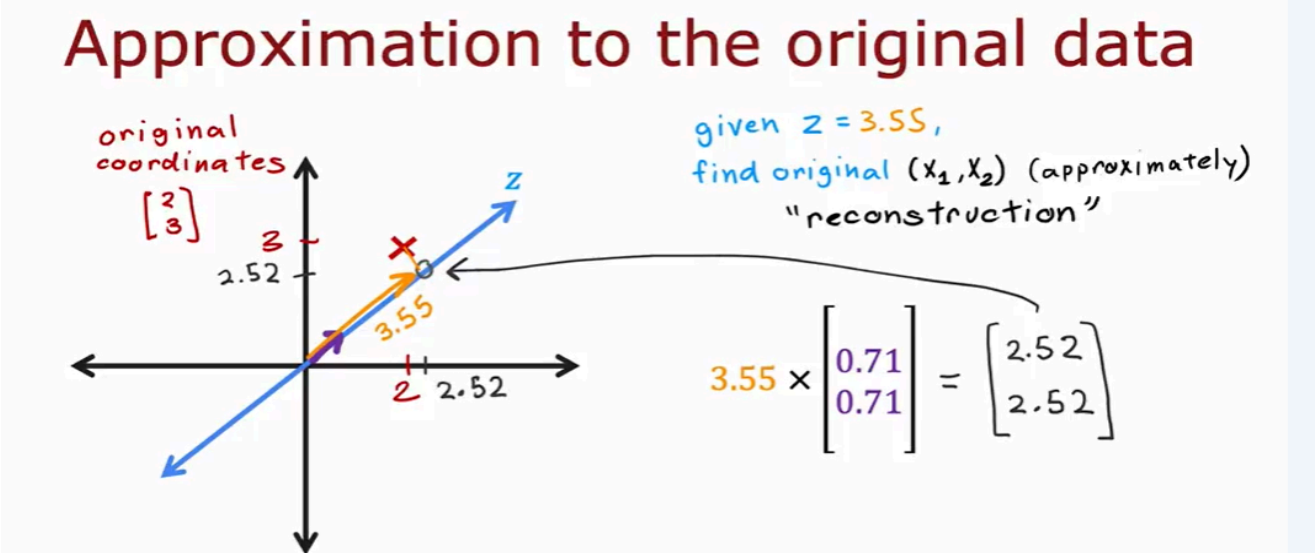


There can be more than one PCA for data point.

PCA vs Linear regression:

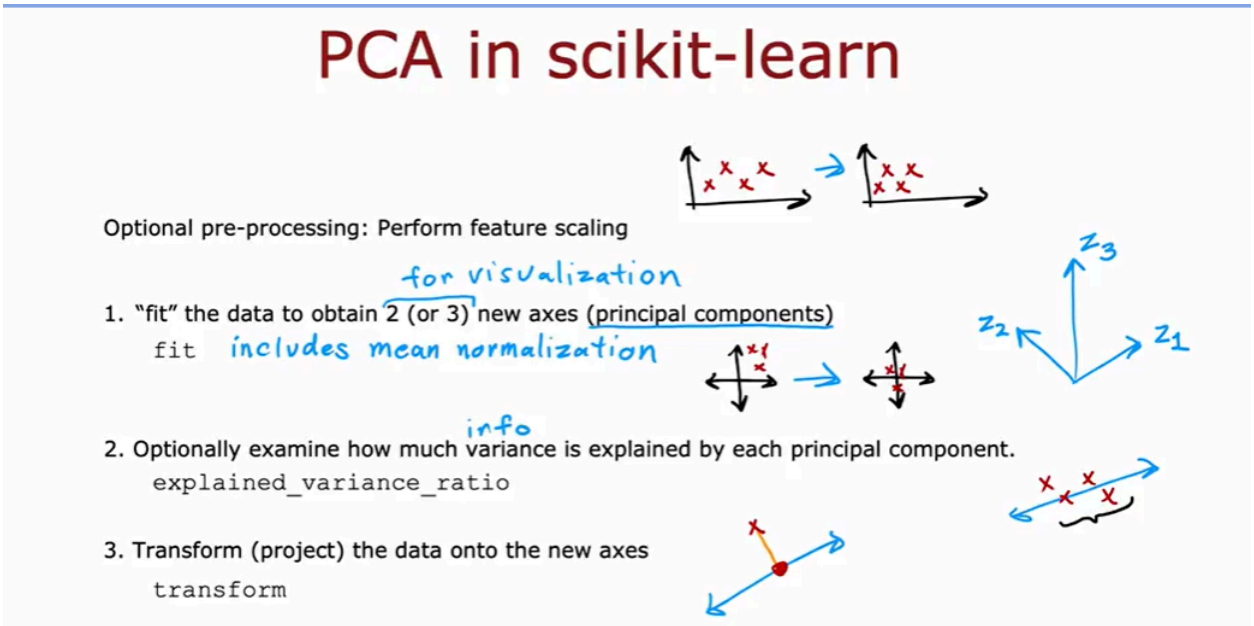


Approximate to the original datapoint:

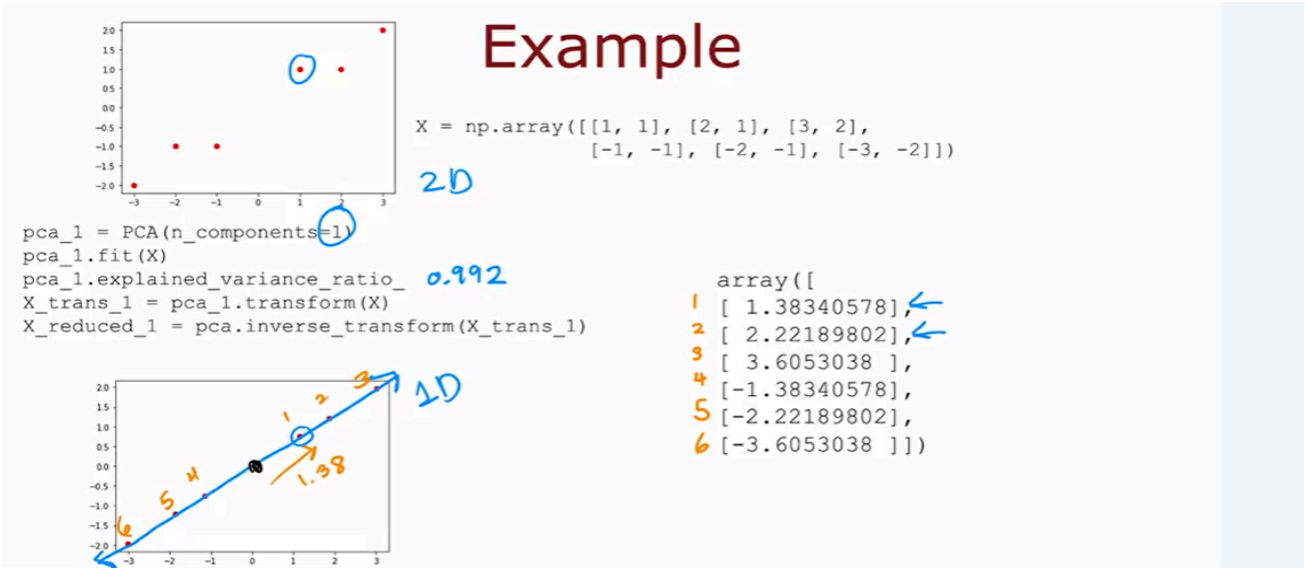


Reconstruction step of PCA.

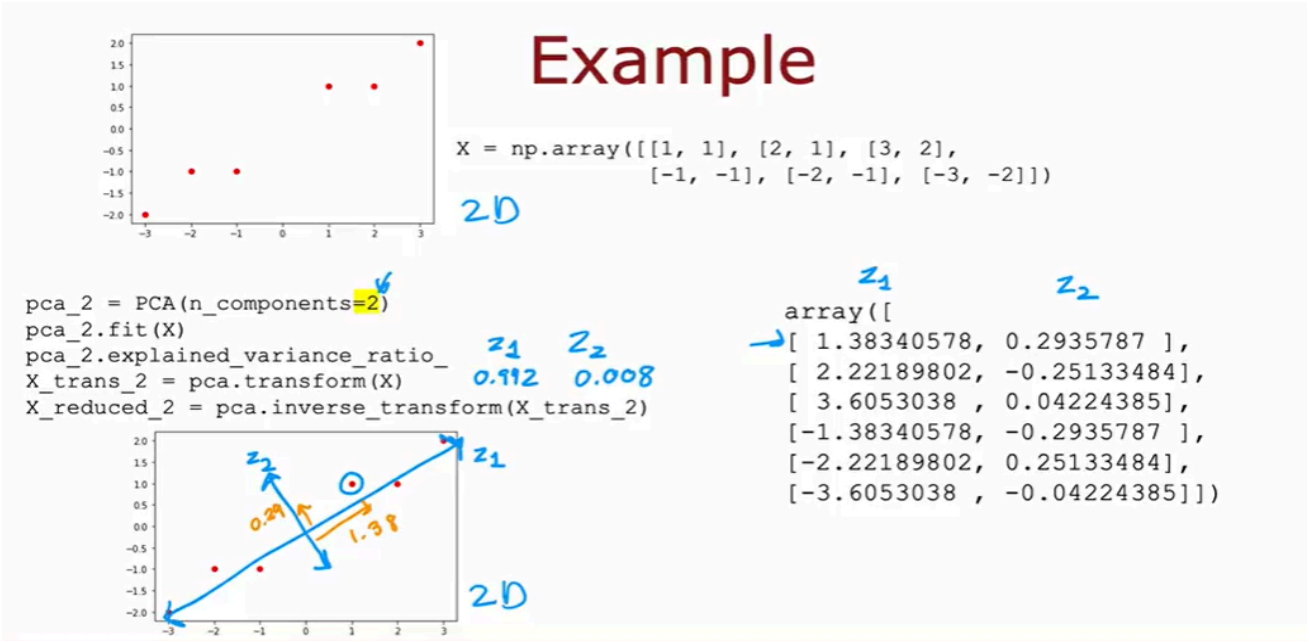
PCA step in scikit learn:

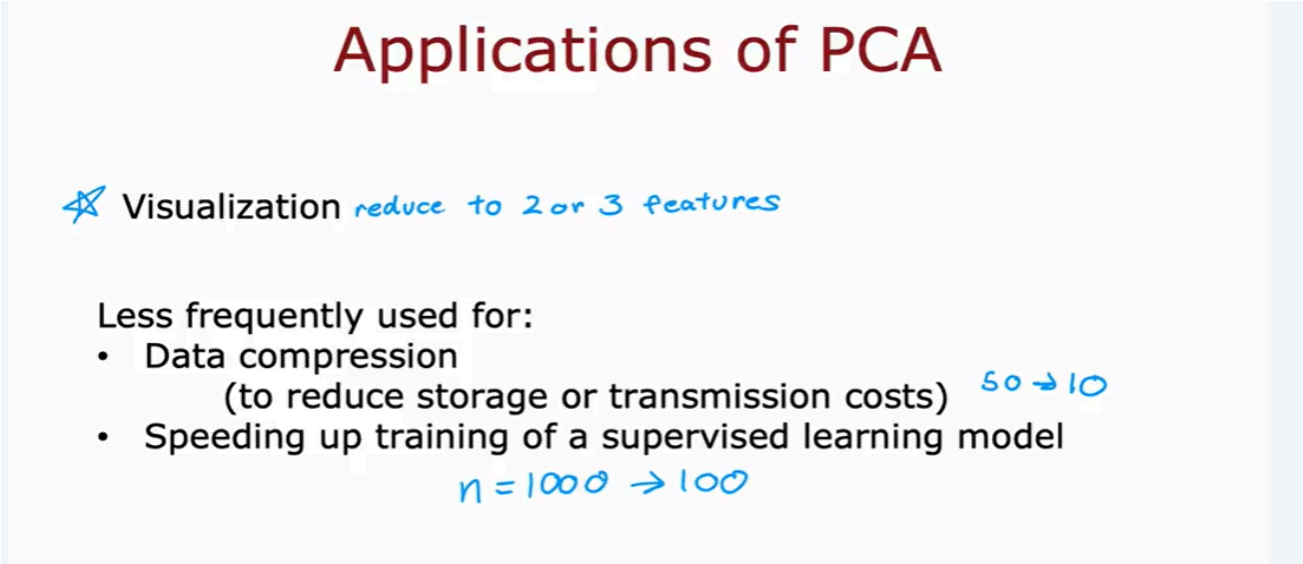


Example:


Example
X = np.array([[1, 1], [2, 1], [3, 2],
[-1, -1], [-2, -1], [-3, -2]])
2D
pca_1 = PCA(n_components=1)
pca_1.fit(X)
pca_1.explained_variance_ratio_ 0.992
X_trans_1 = pca_1.transform(X)
X_reduced_1 = pca.inverse_transform(X_trans_1)
array([
[1.38340578],
[2.22189802],
[3.6053038],
[-1.38340578],
[-2.22189802],
[-3.6053038]])
1D
1.38

For 2 PCA


Example
X = np.array([[1, 1], [2, 1], [3, 2],
[-1, -1], [-2, -1], [-3, -2]])
2D
pca_2 = PCA(n_components=2)
pca_2.fit(X)
pca_2.explained_variance_ratio_ z1 z2
0.992 0.008
X_trans_2 = pca.transform(X)
X_reduced_2 = pca.inverse_transform(X_trans_2)
z1 z2
array([
[1.38340578, 0.2935787],
[2.22189802, -0.25133484],
[3.6053038 , 0.04224385],
[-1.38340578, -0.2935787],
[-2.22189802, 0.25133484],
[-3.6053038 , -0.04224385]])
z1
z2
1.38
2D



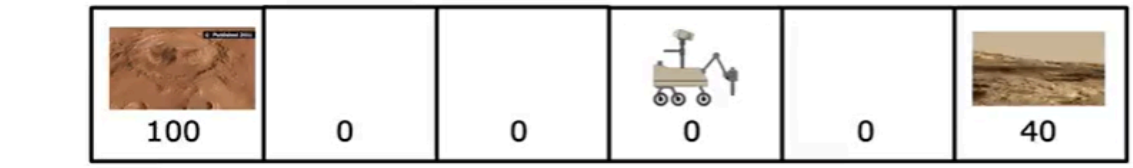
Reinforcement Learning:

Reinforcement Learning (RL) is a type of machine learning where an **agent** learns how to behave in an environment by **interacting with it** and receiving **rewards or penalties**. The goal of the agent is to **maximize the total cumulative reward** over time.

Core concept:

1. **Agent**: The learner or decision-maker.
2. **Environment**: The external system the agent interacts with.
3. **State (S)**: The current situation of the environment.
4. **Action (A)**: A choice the agent makes.
5. **Reward (R)**: Feedback from the environment after an action.
6. **Policy (π)**: Strategy the agent uses to decide actions based on states.
7. **Value Function (V)**: Expected long-term reward of a state.
8. **Q-Function (Q)**: Expected long-term reward of a state-action pair.
9. **Episode**: One run of interaction from start to finish.

Return



Return $= 0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100 = 0.729 \times 100 = 72.9$

Return $= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$ (until terminal state)

Discount Factor $\gamma = 0.9 \quad 0.99 \quad 0.999$

$\gamma = 0.5$

Return $= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 = 1$

Return in reinforcement learning is sum of the weighted reward

Example of Return

100	50 ←	25 ←	12.5 ←	6.25 ←	40
100	0	0	0	0	40
1	2	3	4	5	6

← return
← reward

$\gamma = 0.5$

The return depends on the actions you take.

100	2.5 →	5 →	10 →	20 →	40
100	0	0	0	0	40
1	2	3	4	5	6

$0 + (0.5)0 + (0.5)^2 40 = 10$

100	50 ←	25 ←	12.5 ←	20 →	40
100	0	0	0	0	40
1	2	3	4	5	6

$0 + (0.5)40 = 20$

Making decision : policies in reinforcement learning:

Policy (π) defines the agent's **way of behaving at a given time** — that is, **what action to take given a state**.

Formally:

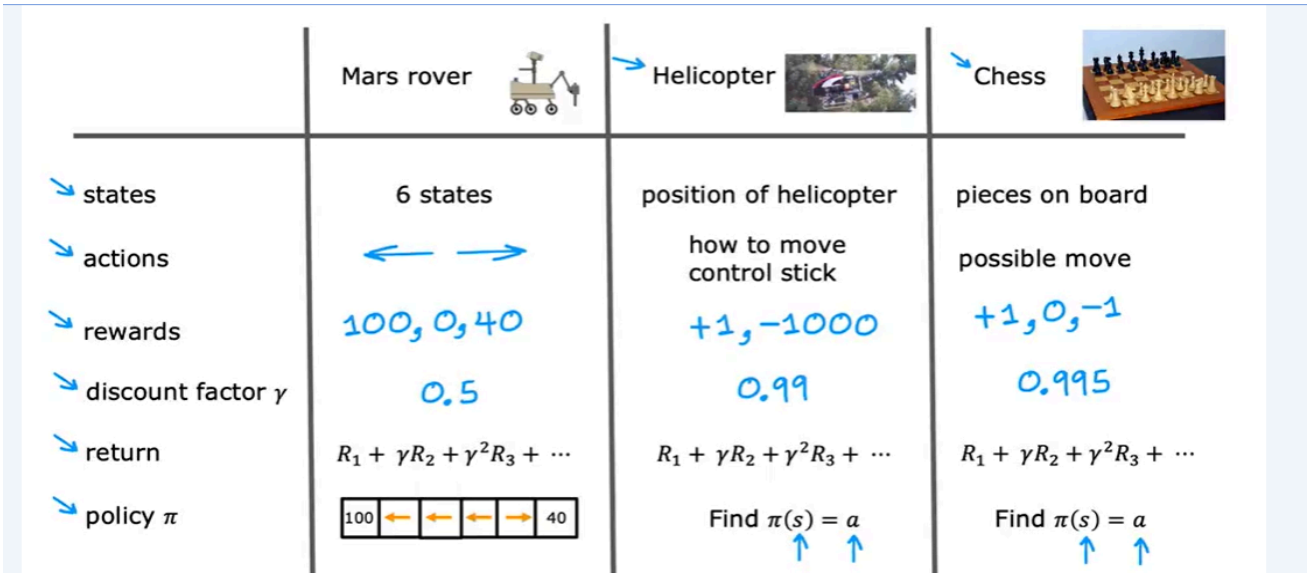
$$\pi(a|s) = P(A_t = a \mid S_t = s)$$

- It maps **states (s)** to a **distribution over actions (a)**.
- A policy **can be deterministic or stochastic**:
  - **Deterministic Policy**: Always picks a fixed action for a state.

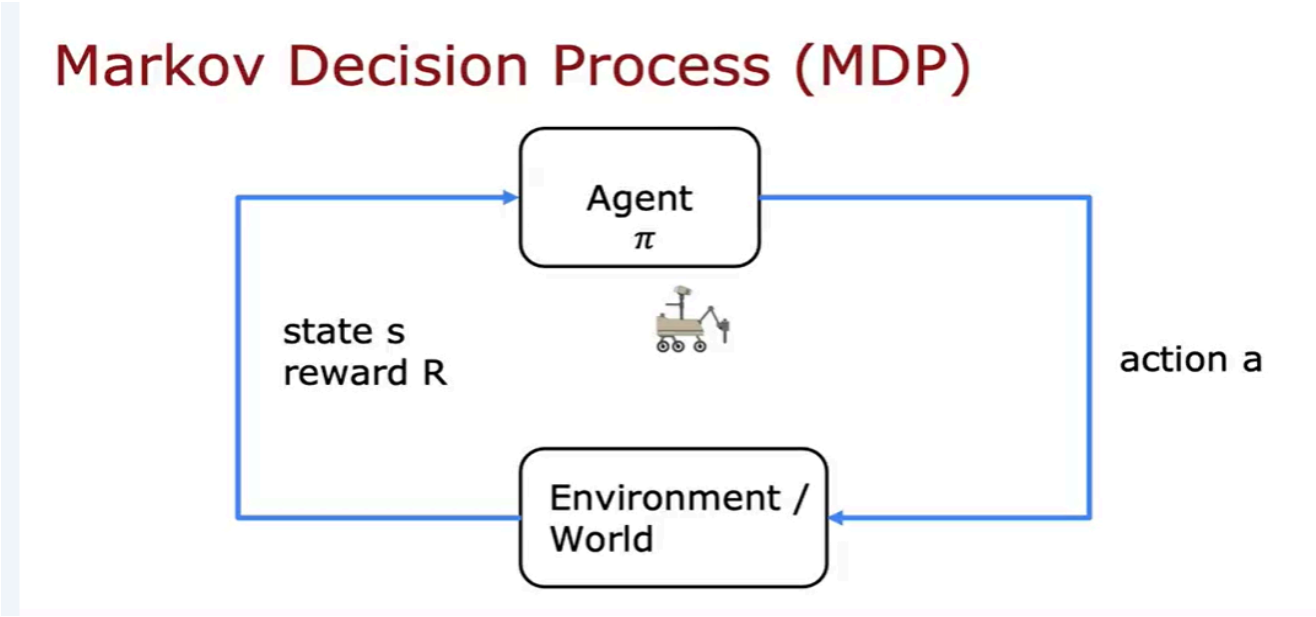
$$a = \pi(s)$$

  - **Stochastic Policy**: Picks actions based on a probability distribution.

$$\pi(a|s) = \text{Probability of taking action } a \text{ when in state } s$$



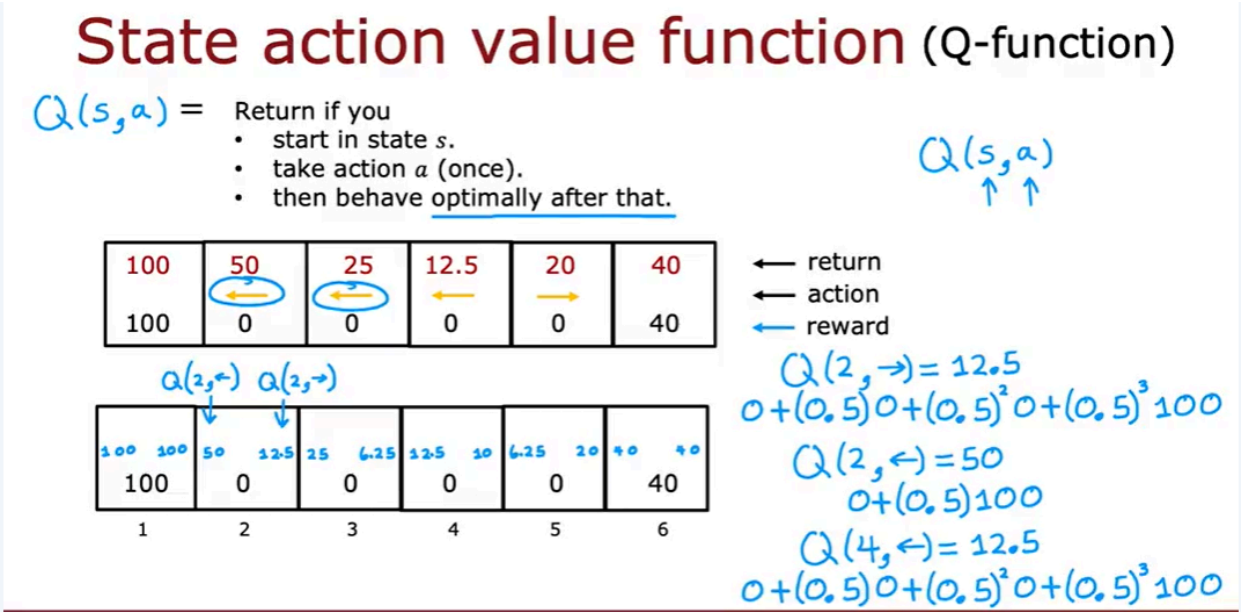
This formalism of reinforcement learning algorithm has a name , it is called MDP [Markob Decis on Process]

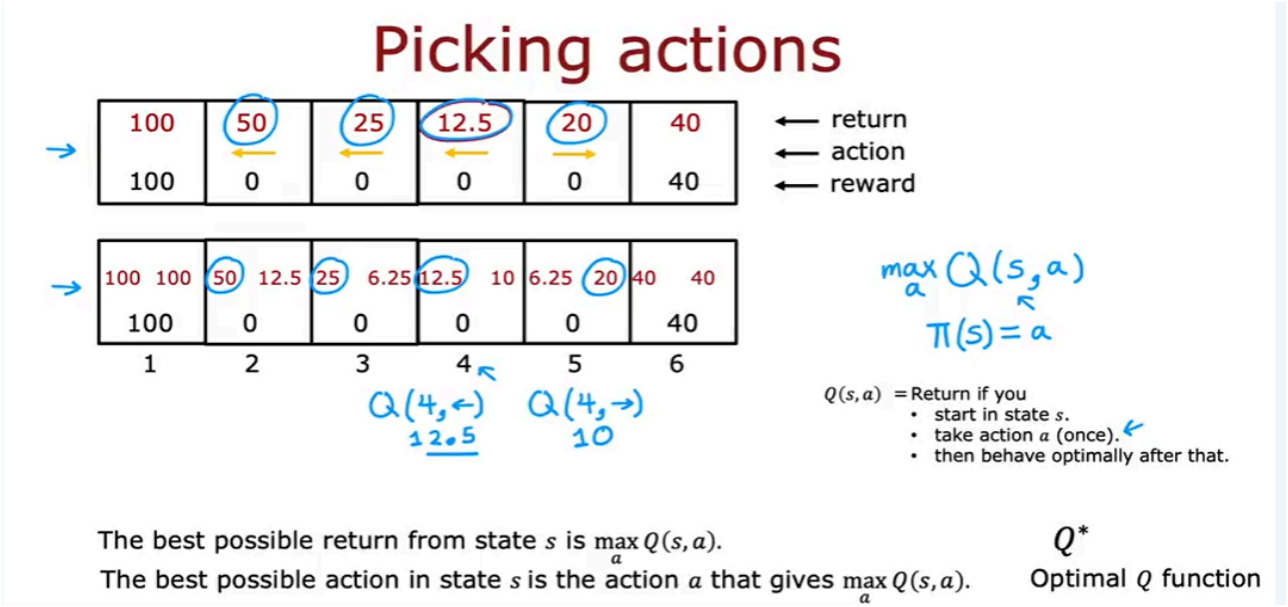


State action value:

The **State-Action Value function**, often denoted as **Q(s, a)**, tells us:

> **"What is the expected cumulative reward** if the agent starts in **state s**, takes **action a**, and **follows a policy π** afterward?"





🌟 What is $Q^*(s, a)$?

$Q^*(s, a)$ is the **maximum expected return** (cumulative discounted reward) **achievable** by **any policy**, starting from state s, taking action a, and **then following the optimal policy**.

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

It represents the **best possible value** for each (state, action) pair.

Bellman Equation:

🎯 What is the Bellman Equation **used for**?

The **Bellman Equation** helps us:

1. **Break down the value of a policy or action into immediate reward + future value.**
2. **Formulate recursive relationships** to solve for value functions or Q-values.
3. **Enable dynamic programming methods** like **value iteration** and **policy iteration**.
4. **Serve as the foundation for learning algorithms** like **Q-learning**, **SARSA**, and **Deep Q-Networks (DQN)**.

Explanation of Bellman Equation

$Q(s,a)$ = Return if you
- start in state s.
- take action a (once).
- then behave optimally after that.

$s \rightarrow s'$

→ The best possible return from state s' is $\max_{a'} Q(s',a')$

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

Reward you get right away

Return from behaving optimally starting from state s'.

$R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$

$= R_1 + \gamma [R_2 + \gamma R_3 + \gamma^2 R_4 + \cdots]$

Explanation of Bellman Equation

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$

100 100	50 12.5	25 6.25	12.5 10	6.25 20	40 40
100	0	0	0	0	40
1	2	3	4	5	6

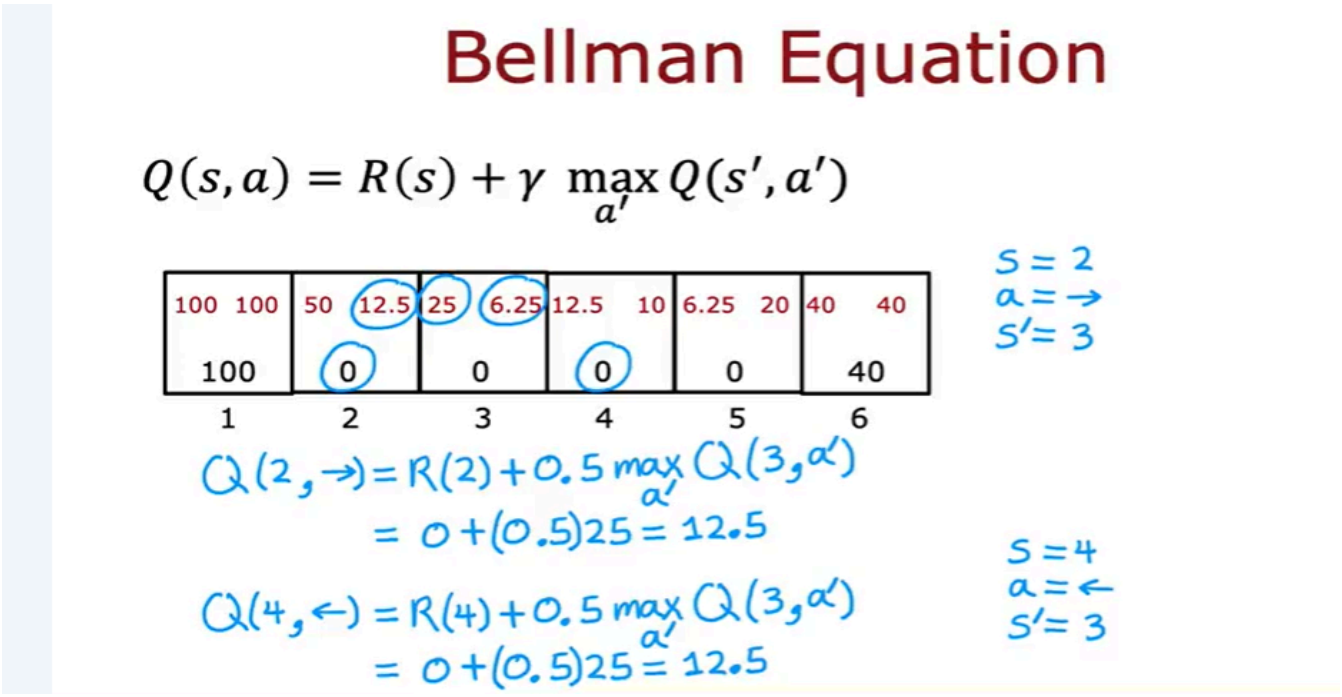
$Q(4, \leftarrow)$

$= 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100$

$= R(4) + (0.5)[0 + (0.5)0 + (0.5)^2 100]$

$= R(4) + (0.5) \max_{a'} Q(3, a')$

It is nothing but breaking down the reward what u get right away and what u will get right into the future.



*** It is like dynamic programming .

Random(Stochastic) Environment:

A **stochastic (random) environment** is one where the **next state and reward are not fully predictable**, even if the agent takes the same action in the same state.

> **In other words**: Taking the same action in the same situation **can lead to different results** — due to randomness in the environment.

Example:

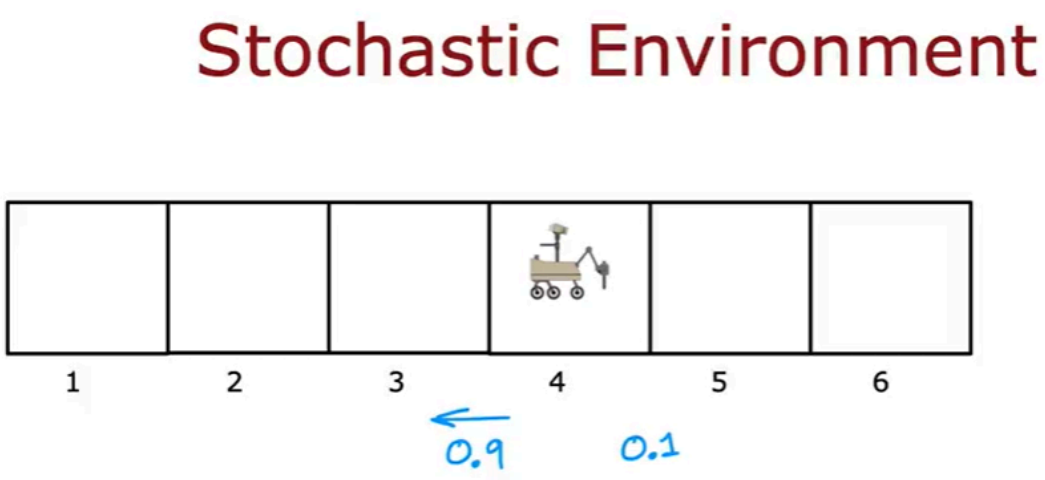
Let's say an agent is in a grid world (like a maze), and it decides to move "**right**" from a certain state.

In a **deterministic environment**, it always goes right.

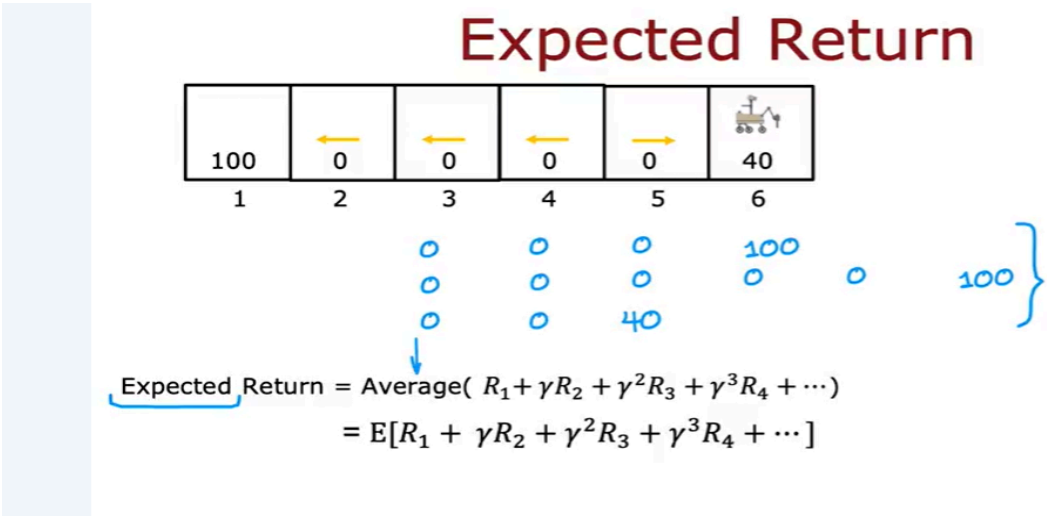
In a **stochastic environment**, it might:

- go right 80% of the time,
- go up 10% of the time,
- go down 10% of the time.

This randomness is what makes the environment stochastic.

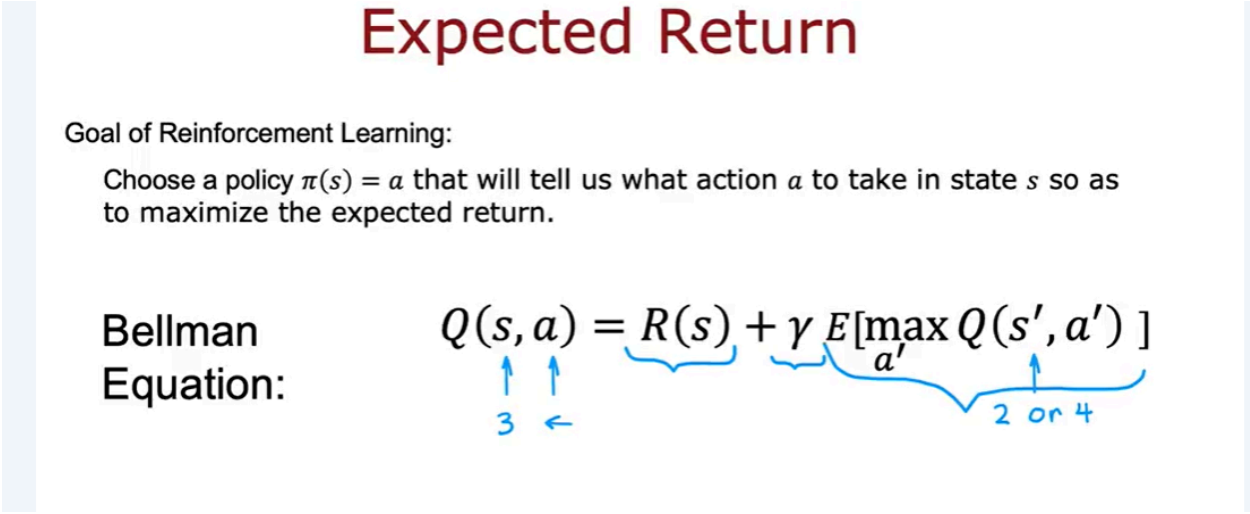


Here the mars rover is in 4 and here 0.9 percent chance that it will end up in index 3 and 0.1 percent chance that it will end up in index 5 so this kind of situation ios called stochastic environment .



Here , from a state randomly various sequence can be achieved by the agent and for each sequence the reward is averaged and this is called the expected value . The goal of this RL algorithm is to find the policy (pi) to maximize the sum of discounted rewards.

The previous bellman equation will be changed slightly here as expected issues will be merged.



Here from state 3 , agent can move to 2 or 4 randomly that means a` can be left or right and for this the average will be taken.
So the total return taking action a from state s is ,once in a behaving optimally is equal to the reward u getting right away + what u expect to get on average of the future returns.

Here the value of optimal policy is given and Q value is also given , with increase of mis_step the reward value gets reduced and the above is for mis_step 0.1 ,but when mis_step is 0.4 the value of optimal policy gets reduced more as mis_step 0.4 means 40% time the agent misses the step that is probability of going in wrong direction

Continuous State Space:

In a **continuous state space**, the **states are not countable** — they can take **any real value within a range**.

> Instead of a few discrete states (like s1, s2, s3), you might have **infinite possible states** like:

s=(x=3.52,y=−1.75)

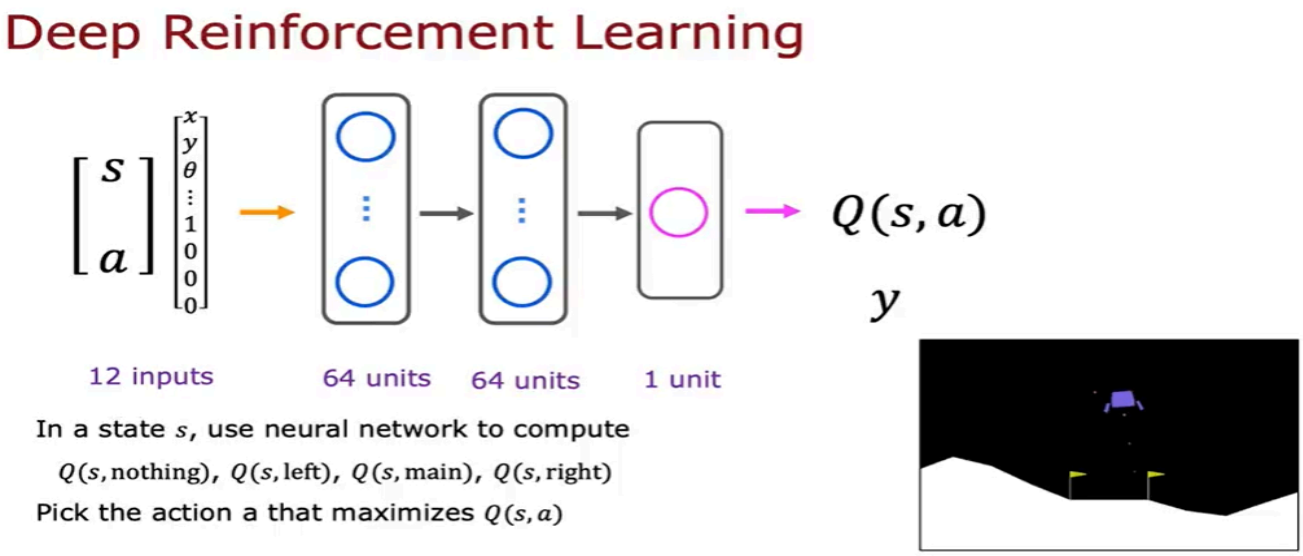
📌 Examples of Continuous States

Scenario	State Variables
Self-driving car	Position, velocity, steering angle, acceleration
Robot arm	Joint angles, torque, velocity
Stock market	Price, volume, moving average

🧠 Why is it Challenging?

1. **You can't store a value for every state**, because there are infinitely many.
2. You can't use **classic Q-tables** like in small discrete environments.
3. You need function approximators (e.g., neural networks) to represent the value or policy.

Deep RL :

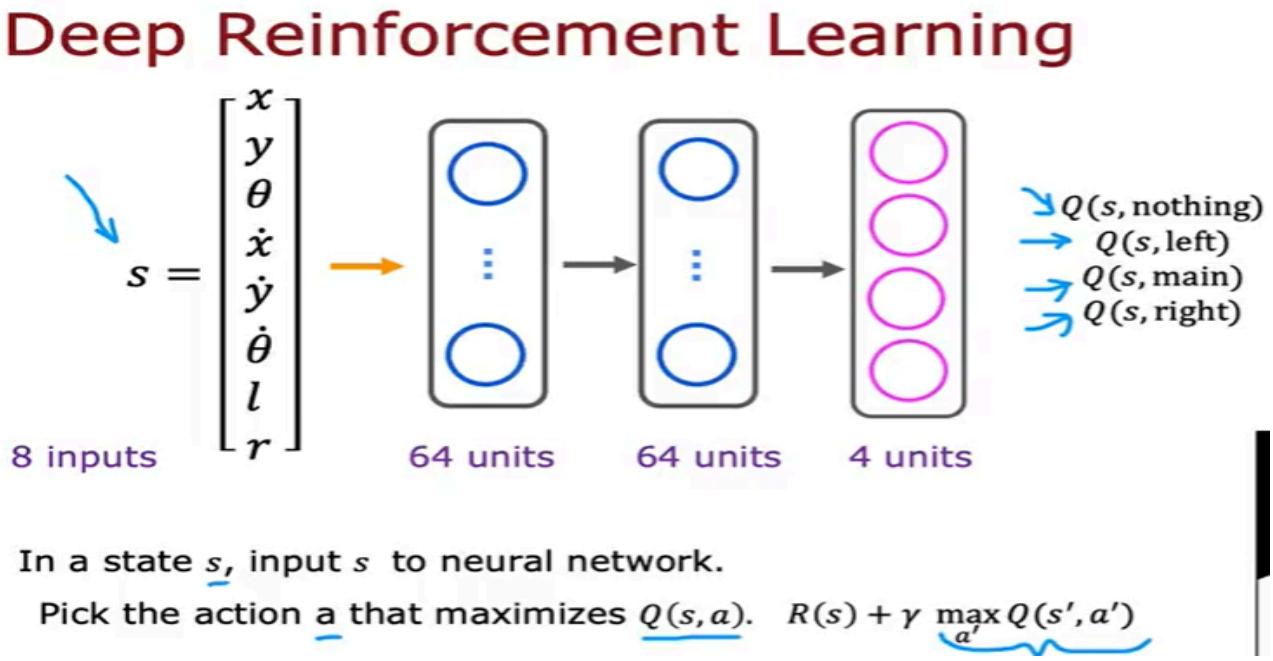


Deep Reinforcement Learning (Deep RL) combines **Reinforcement Learning (RL)** and **Deep Learning** to solve complex decision-making problems where traditional RL methods (like Q-tables) fall short.

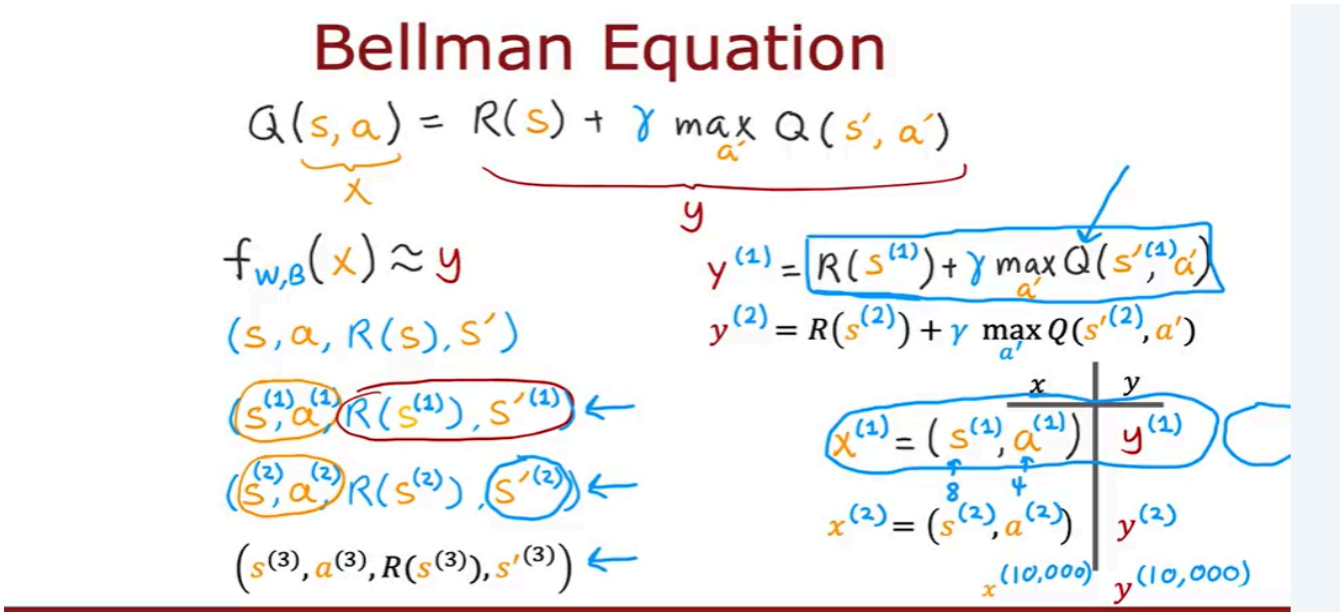
📘 What Is Deep RL?

> Deep RL = Reinforcement Learning (learn by interacting with environment)
> + Deep Learning (learn from high-dimensional data using neural networks)

It allows agents to learn optimal behavior **from pixels, raw sensors, or complex inputs** — instead of hand-coded features or discrete states.



Training Deep RL:



Here , This diagram illustrates how the **Bellman Equation** is used in **Deep Q-Learning** to update Q-values using neural networks.

🔁 Core Bellman Equation

$$Q(s, a) = R(s) + \gamma \max_{a'} Q(s', a')$$

- $Q(s, a)$: The value of taking action a in state s
- $R(s)$: Immediate reward
- γ: Discount factor
- s': Next state
- a': All possible next actions