

SELF-STUDY HANDBOOK

AI & Machine Learning Foundations

CommunityForce Inc.

Modules in this edition:

- **AI-01** AI, ML, Deep Learning, GenAI & Agents: The Big Picture
- **AI-02** How Machines Learn: $y = f(x)$
- **AI-03** Types of Machine Learning
- **AI-04** The ML Workflow: From Data to Deployment
- **AI-05** Neural Networks & Deep Learning Intuition
- **AI-06** Data, Analytics & Data Types
- **AI-07** Describing Data: Central Tendency & Spread
- **AI-08** Probability Essentials for ML
- **AI-09** Python Basics: Variables, Types & Input
- **AI-10** Python Collections: Lists, Tuples, Sets & Dictionaries
- **AI-11** Writing Functions

Screens are from recorded training sessions. Names, avatars and account details are replaced or removed; replaced values are marked DEMO DATA in the videos.

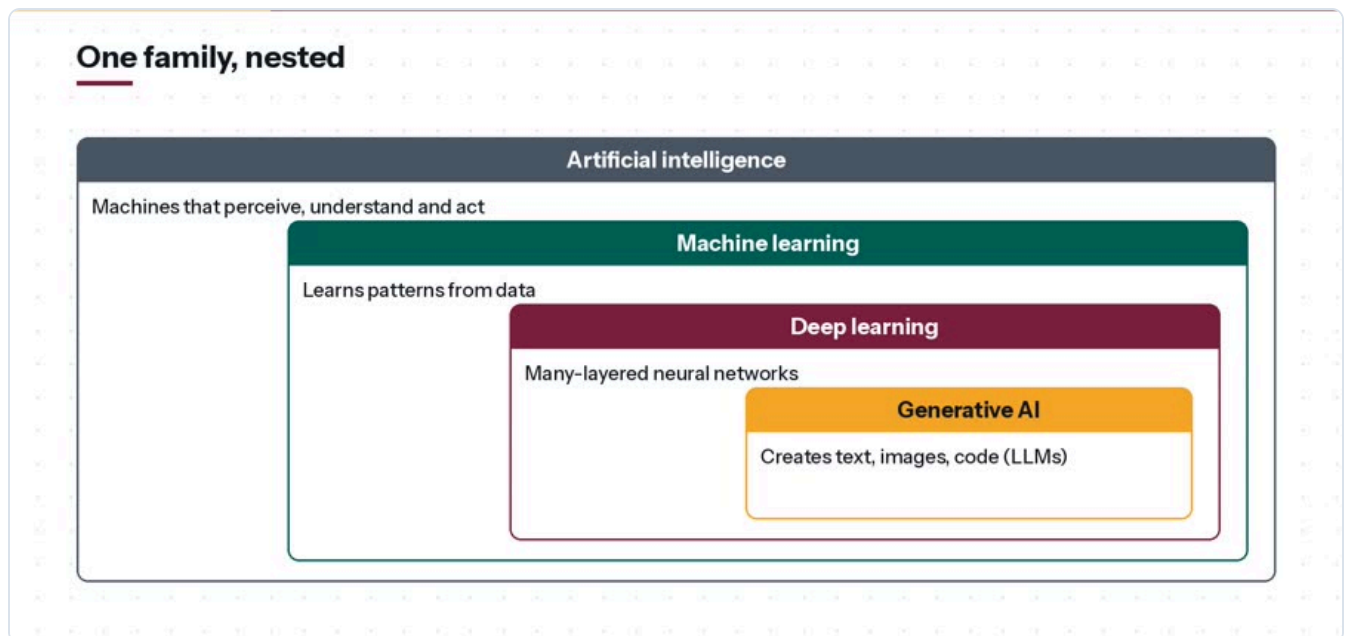
AI, ML, Deep Learning, GenAI & Agents: The Big Picture

Video: 01_AI-01_AI_ML_Deep_Learning_GenAI_and_Agents.mp4 · 4:36 · Stage: Foundations · Audience: Anyone starting the AI & ML course; no maths or coding needed

Learning objectives

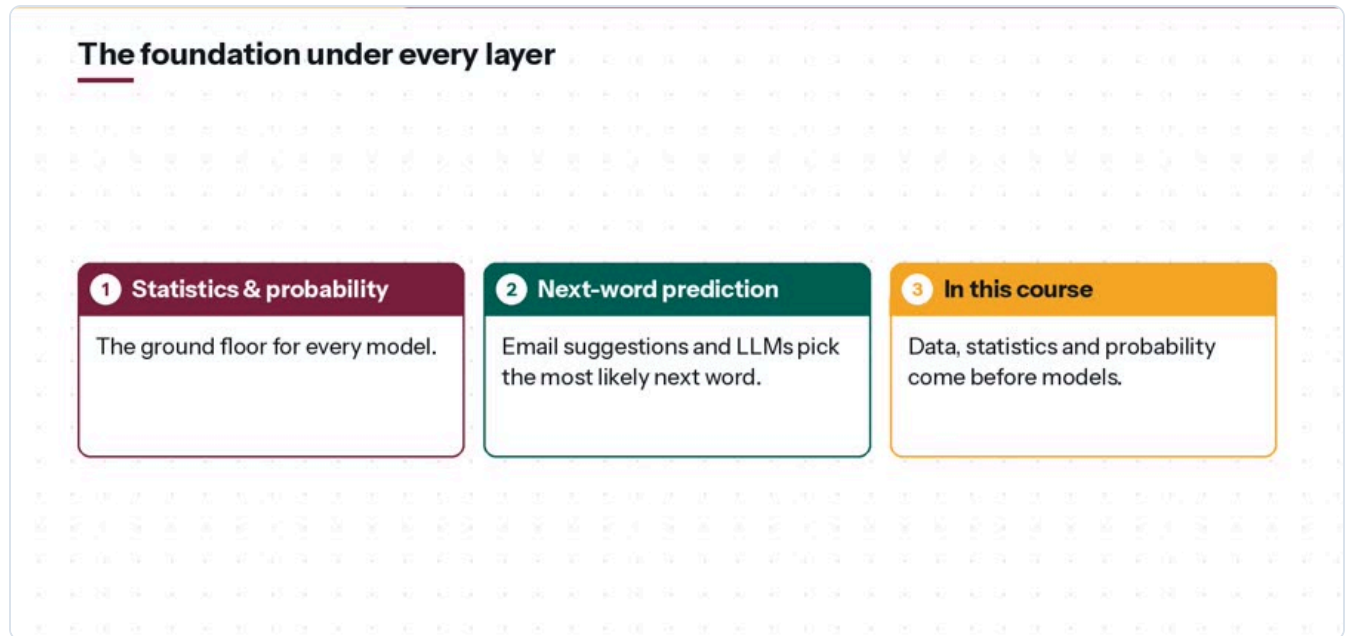
- Place AI, ML, deep learning and GenAI in one picture
- Name the building blocks of an AI agent
- Explain why AI took off now

One family, nested



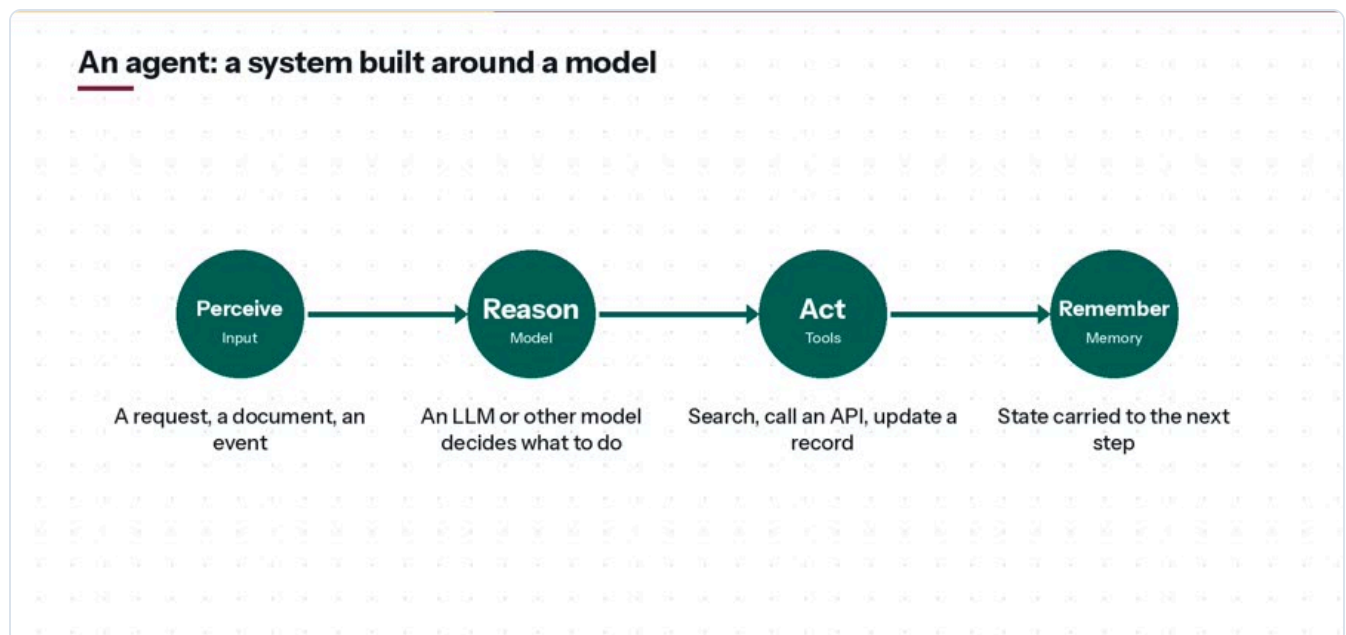
Think of it as a set of nested circles. Artificial intelligence is the broad goal: machines that perceive, understand, and act intelligently. Machine learning is the part of AI where systems learn patterns from data instead of following hand-written rules. Deep learning is machine learning with many-layered neural networks. And generative AI, including large language models, uses deep learning to create new text, images, or code.

The foundation under every layer



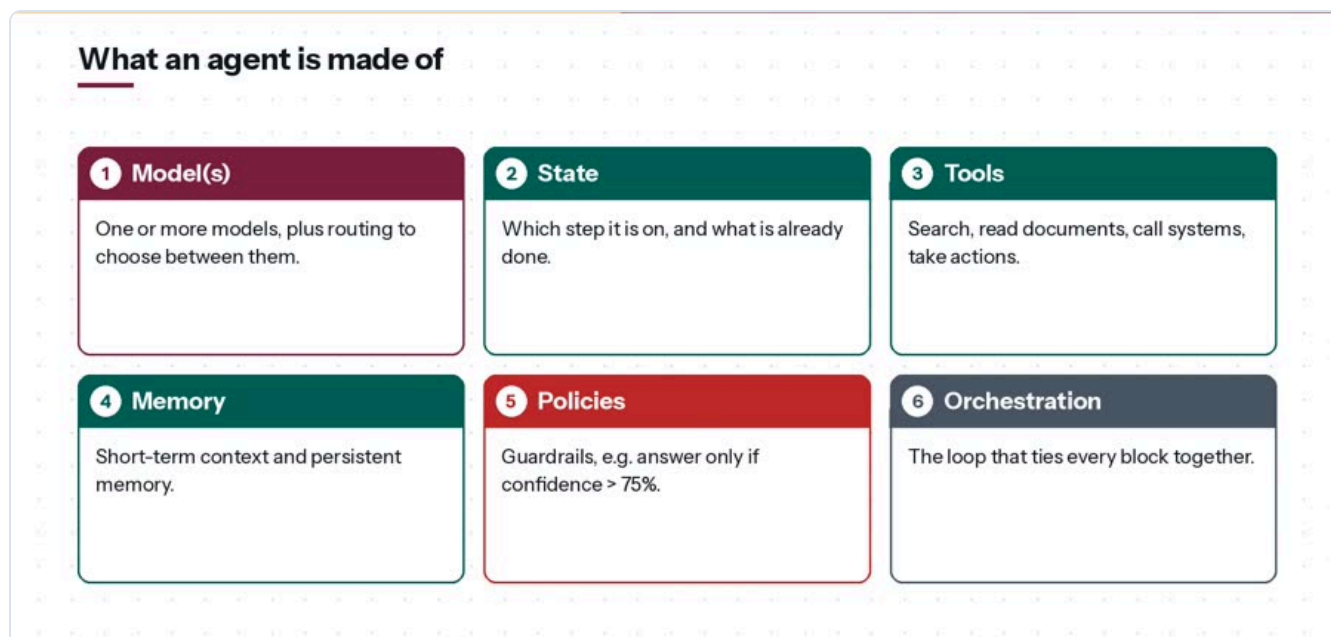
Underneath every layer sits the same foundation: statistics and probability. When your email app suggests the next words of a sentence, a model is predicting the most likely next word. Large language models do the same thing at a huge scale, and image models and agents rely on probabilities too. That's why this course covers data, statistics, and probability before building models.

So where do agents fit? An agent is not a single model



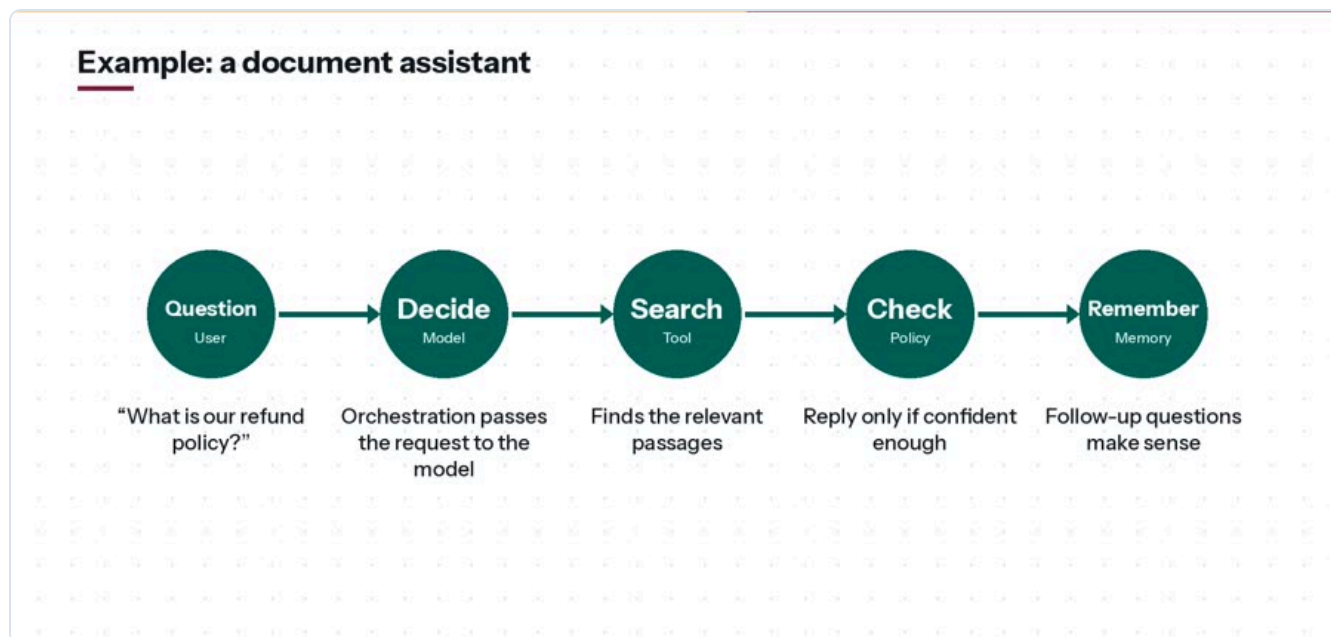
So where do agents fit? An agent is not a single model. It's a system built around one. It perceives a request or a situation, reasons about it with a model, then takes action using tools, and remembers what it has done so far.

What an agent is made of



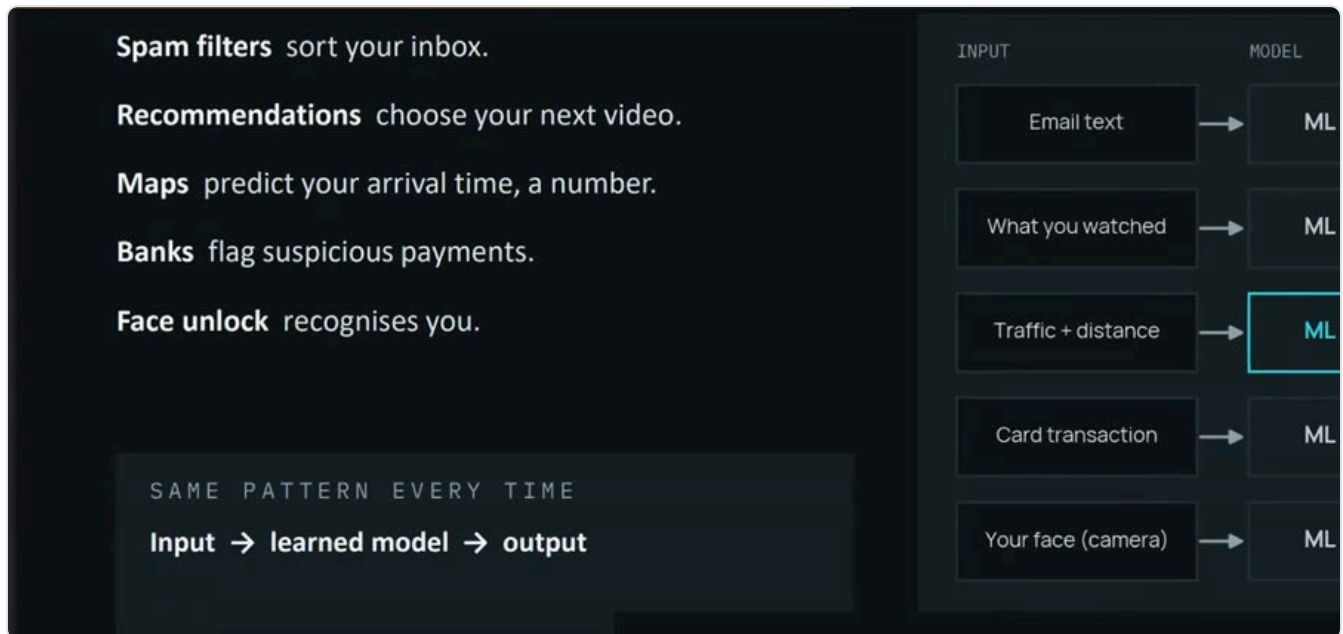
Here are the building blocks the trainer listed. A model, or several, with routing to pick the right one. State, so the agent knows which step it's on. Tools, to search, read, and act. Memory, short-term and persistent. Policies, or guardrails, such as only answering when confidence is above a threshold. And orchestration, the loop that ties all of these together.

Here's how those blocks work together in a simple document assistant



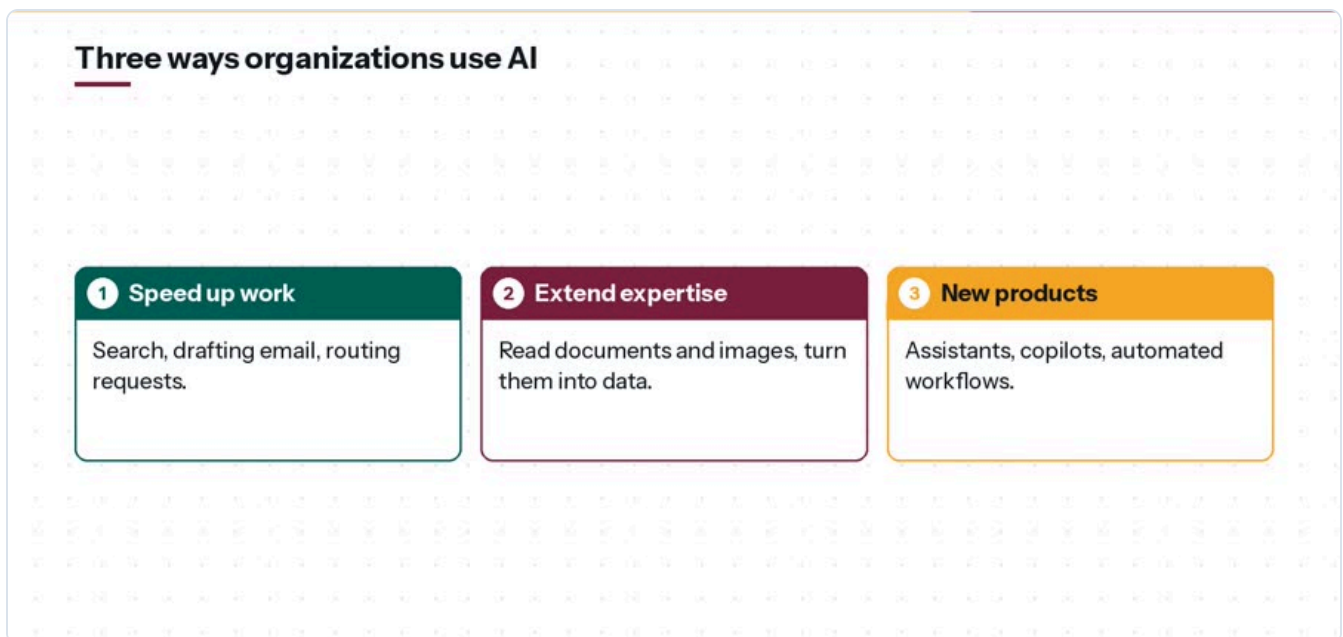
Here's how those blocks work together in a simple document assistant. A user asks a question. The orchestration loop passes it to a model, which decides to search. A search tool finds the relevant passages. The model drafts an answer, and a policy checks its confidence before replying. Memory keeps the conversation, so a follow-up question still makes sense.

You already use machine learning every day



You already use machine learning every day. Spam filters sort your inbox. Recommendations choose your next video. Maps predict your arrival time, which is a number. Banks flag suspicious card payments. Every case follows the same pattern: an input goes into a learned model, and an output comes back.

Three ways organizations use AI



In organizations, AI usually shows up in three ways. First, it speeds up everyday work: search, drafting emails, and routing requests. Second, it extends expertise, with systems that read documents and images and turn them into usable data. Third, it powers new products: assistants, copilots, and automated workflows.

Why AI took off now

Why AI took off now

The term "artificial intelligence" dates to the 1950s. What changed recently?

1 Data

Digital records, sensors, text and images at huge scale.

2 Compute

Cheaper processing, especially GPUs and cloud, made training practical.

3 Results

Breakthrough results and open research attracted investment.

Artificial intelligence isn't new. The term dates back to the nineteen fifties, and many of the core ideas are decades old. So why now? Three things came together. There is far more data than ever before. Computing power became much cheaper, especially for training large models. And a steady stream of success stories and research kept pushing the field forward.

Choose the right tool for the problem

Choose the right tool for the problem

Problem	Typical approach	Example
Predict a number or category from tables	Core machine learning	House price, loan approval, churn
Images, speech, video	Deep learning	Face unlock, speech to text
Open-ended language tasks	Generative AI + agents	Drafting email, answering from documents

One more point from the trainer: not every problem needs an agent. Predicting a number or a category from tables of data is often best solved with core machine learning. Images, speech, and video usually call for deep learning. And open-ended language tasks, like drafting or answering questions over documents, are where generative AI and agents shine.

Key takeaways

- Nested: AI → ML → deep learning → GenAI
- Agent = model, state, tools, memory, policies, loop
- Data, compute and results drove the boom

Practice

1. List three products you used today that rely on machine learning. For each, write the input and the output.
2. Pick a task at work and decide whether it needs core ML, deep learning, or generative AI with an agent. Justify your choice in one sentence.

Check your understanding

1. Which is the broadest term: AI, ML, deep learning or generative AI?
2. Is an agent the same as a model?
3. Name the three reasons AI took off recently.
4. Which approach usually suits predicting a number from tabular data?

Answers at the back of this handbook.

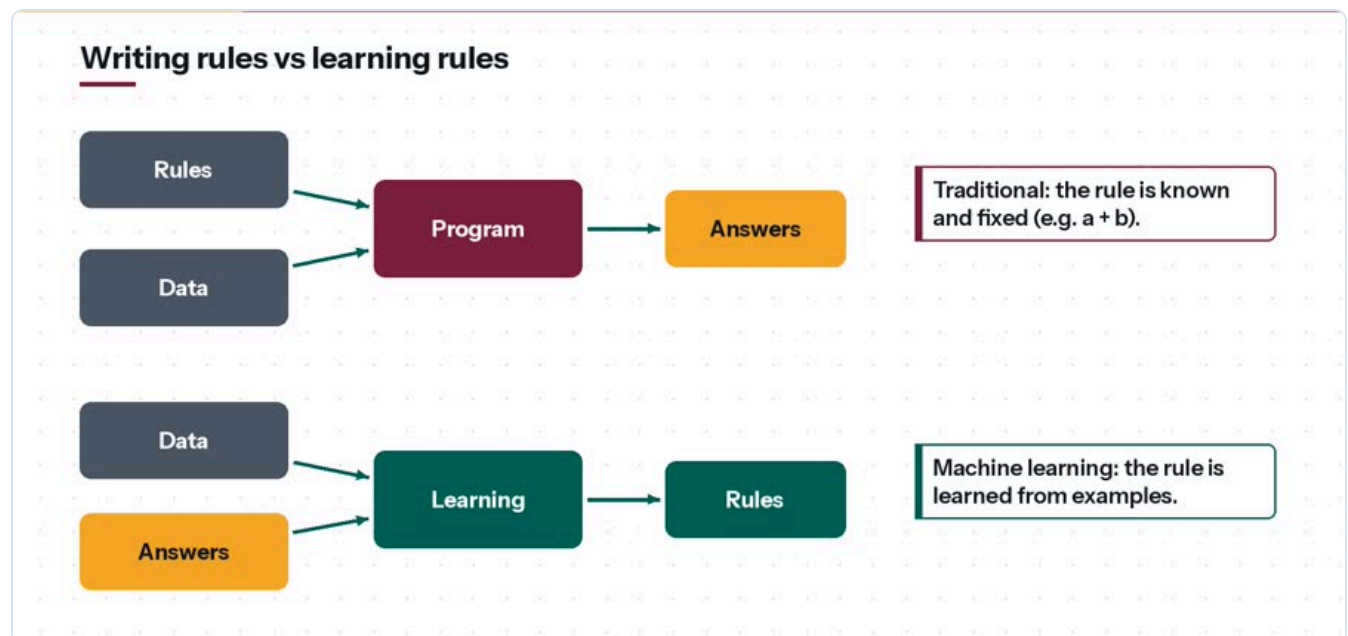
How Machines Learn: $y = f(x)$

Video: 02_AI-02_How_Machines_Learn.mp4 · 4:07 · Stage: Foundations · Audience: Beginners who completed AI-01

Learning objectives

- Describe learning as finding f in $y = f(x)$
- Contrast writing rules with learning rules
- Tell y from $\hat{y}$ and measure accuracy

Writing rules vs learning rules



In traditional programming, a developer writes the rules. You give the computer rules and data, and it produces answers. Adding two numbers is a perfect example, because the rule is universal and never changes. Machine learning flips this around. You give the computer data and the answers, and it works out the rules for itself.

Here's the trainer's slide

computer finds the rules.

The rules it finds are called the model.

TIP
"The ability to learn without being explicitly programmed" — usually credited to Arthur Samuel (1959).

DEFINITION

**Machine learning = teaching computers to learn
learn patterns from examples, not fixed rules.**

Here's the trainer's slide. Traditional programming: we write the rules, and the computer applies them. Machine learning: we give data plus answers, and the computer finds the rules. The rules it finds are called the model, a word you'll hear all through this course. The definition at the bottom puts it simply: computers learn patterns from examples, not fixed rules. A classic definition says the same: the ability to learn without being explicitly programmed.

Tip "The ability to learn without being explicitly programmed" — usually credited to Arthur Samuel (1959).

Why hand-written rules break

Why hand-written rules break

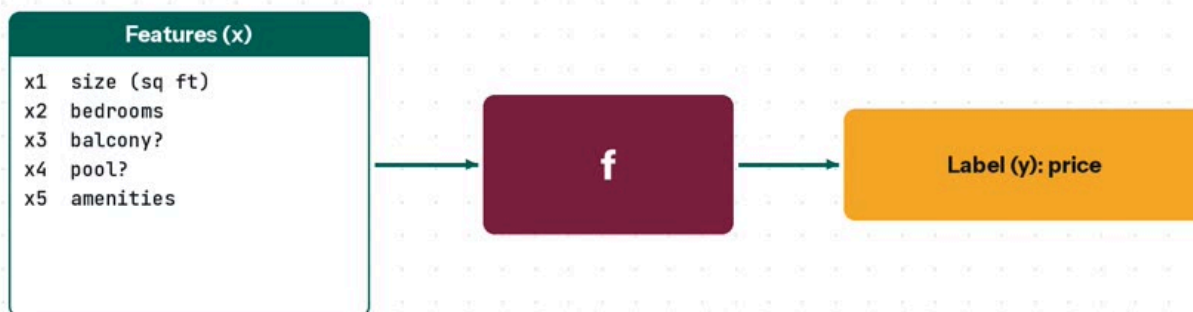
```
1 // Rule-based spam filter
2 if 'lottery' in email:
3     label = 'spam'
4 elif 'you won' in email:
5     label = 'spam'
6 elif 'l0ttery' in email: # spammers adapt
7     label = 'spam'
8 # ...and another rule, forever
```

ML instead: learn the pattern from thousands of labelled emails.

Why bother learning rules? Take spam. You could write a rule: if an email contains the word lottery, mark it as spam. But spammers simply change the word. Then you need another rule, and another, forever. Instead, we show a model thousands of emails already labelled spam or not spam, and let it learn the pattern.

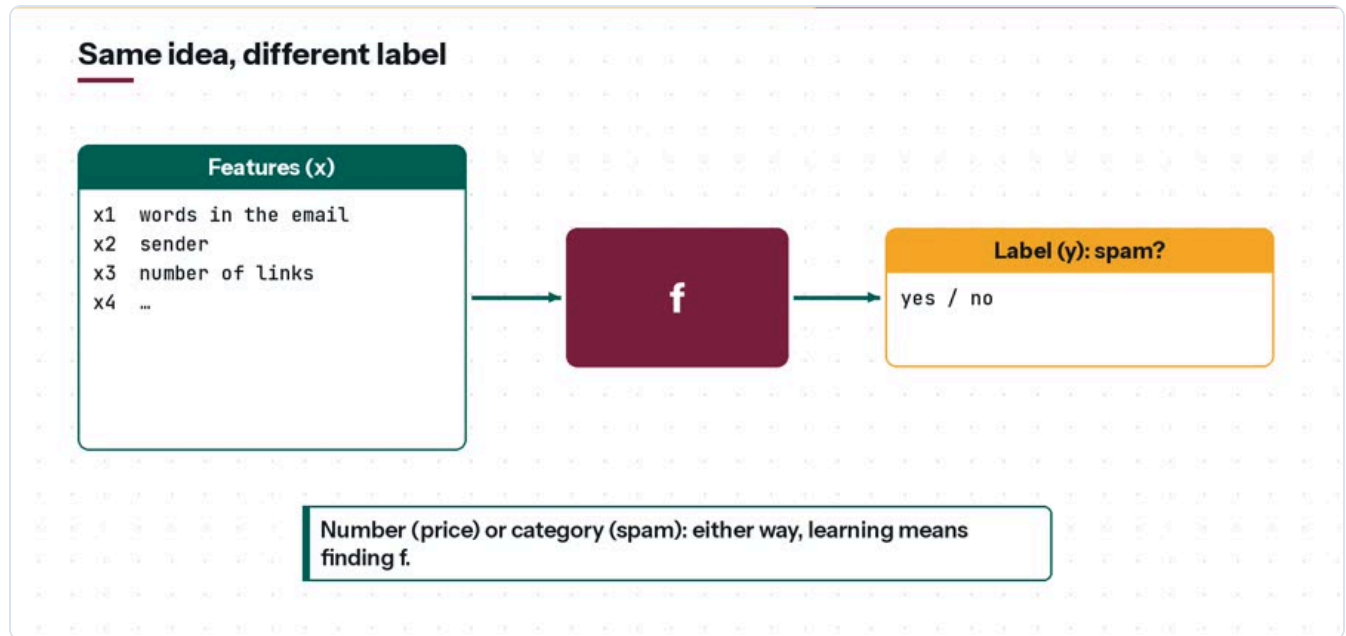
$y = f(x)$: features in, label out

$y = f(x)$: features in, label out



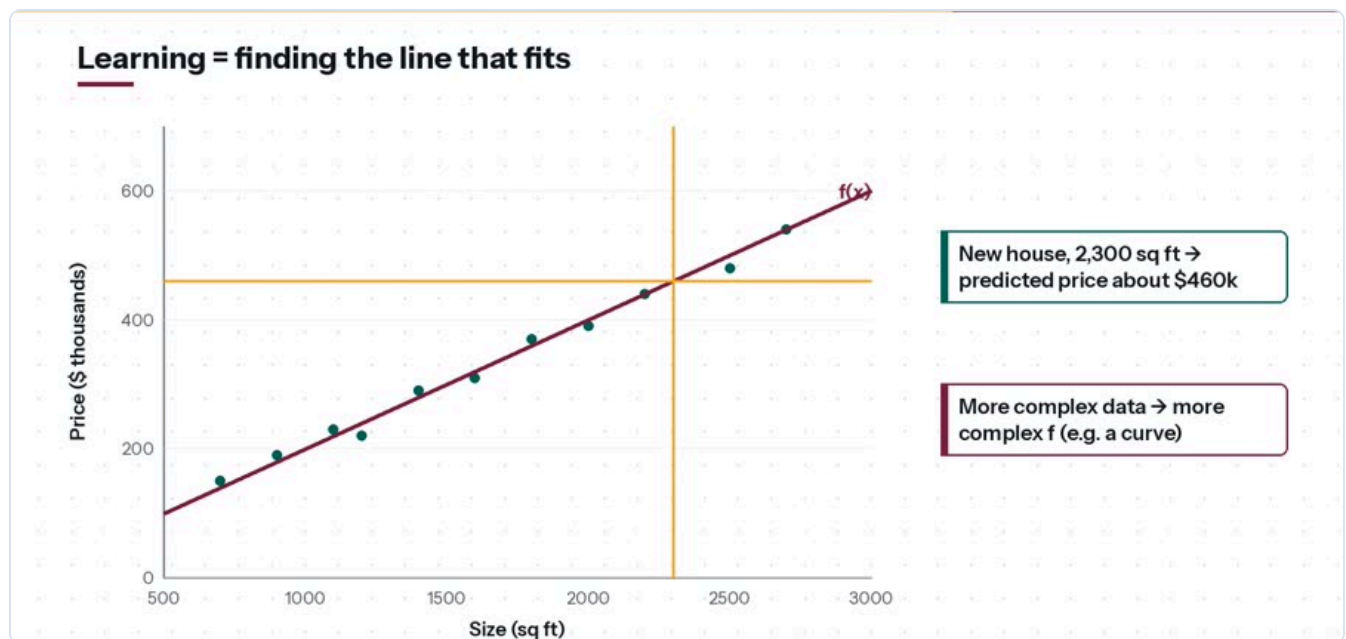
Here's the core idea. Suppose we want to predict a house price. The inputs, called features, are things like size, number of bedrooms, a balcony, a pool, and the amenities. We write them as x one, x two, x three, and so on. The output we want, the price, is called the label, or y . Learning means finding the function f that turns the features into the label.

Same idea, different label



The same idea works when the answer is a category instead of a number. For an email, the features might be the words it contains, who sent it, and how many links it has. The label is spam or not spam. A trained model takes the features of a new email and answers yes or no. Whether y is a number or a category, the job is the same: learn f from examples.

Learning = finding the line that fits



Let's make it visual with one feature. Each dot is a house we already know: its size and its sale price. Learning finds the line that best fits these points. That line is our function f . Now, for a new house of any size, we simply read its price off the line. That's a prediction. As data gets more complex, f can become a curve instead of a straight line.

The trainer's vocabulary slide sums it up

Dataset: the full table of examples

Example: one row (one house)

Features (X): inputs used to predict

Label (y): the answer we want to predict

Model: the learned pattern from X to y

Training / prediction: learn it, then use it on new data

The trainer's vocabulary slide sums it up. A dataset is the full table of examples. Each row is one example, here one house. Features are the inputs, x: size, bedrooms, and age. The label is the answer we want, y, here the price. The model is the learned pattern from x to y. Training means learning it, and prediction means using it on new data.

y vs $\hat{y}$: checking predictions

y vs $\hat{y}$: checking predictions

Email	True answer (y)	Prediction ($\hat{y}$)	Match?
#1	spam	spam	Yes
#2	not spam	not spam	Yes
#3	spam	not spam	No
#4	not spam	not spam	Yes
... 10 emails		7 match	70%

Accuracy = correct predictions ÷ total predictions. Later modules add better metrics.

How do we know if the model is any good? We hold back some examples where we already know the true answer, y, but hide it from the model. The model makes its prediction, called $\hat{y}$. Then we compare. If seven out of ten predictions match the true answers, the model is seventy percent accurate on that data.

Key takeaways

- Rules written vs rules learned
- Model = f , from features x to label y
- Compare $\hat{y}$ with y to measure accuracy

Practice

1. Write two tasks where hand-written rules work well, and two where they would break. Explain why.
2. For predicting a used car's price, list five features (x) and the label (y).

Check your understanding

1. In traditional programming, what do you give the computer?
2. In $y = f(x)$, what is x ?
3. What is $\hat{y}$?
4. 8 of 10 predictions are correct. What is the accuracy?

Answers at the back of this handbook.

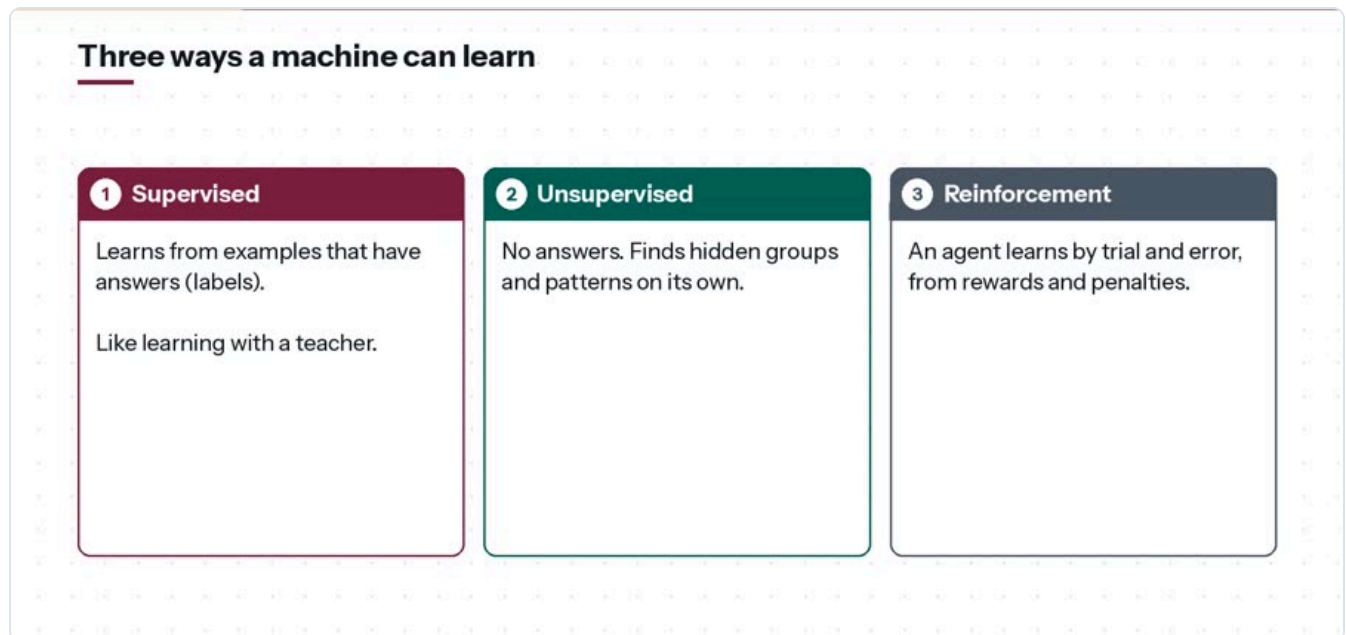
Types of Machine Learning

Video: 03_AI-03_Types_of_Machine_Learning.mp4 · 4:15 · Stage: Foundations · Audience: Beginners who completed AI-02

Learning objectives

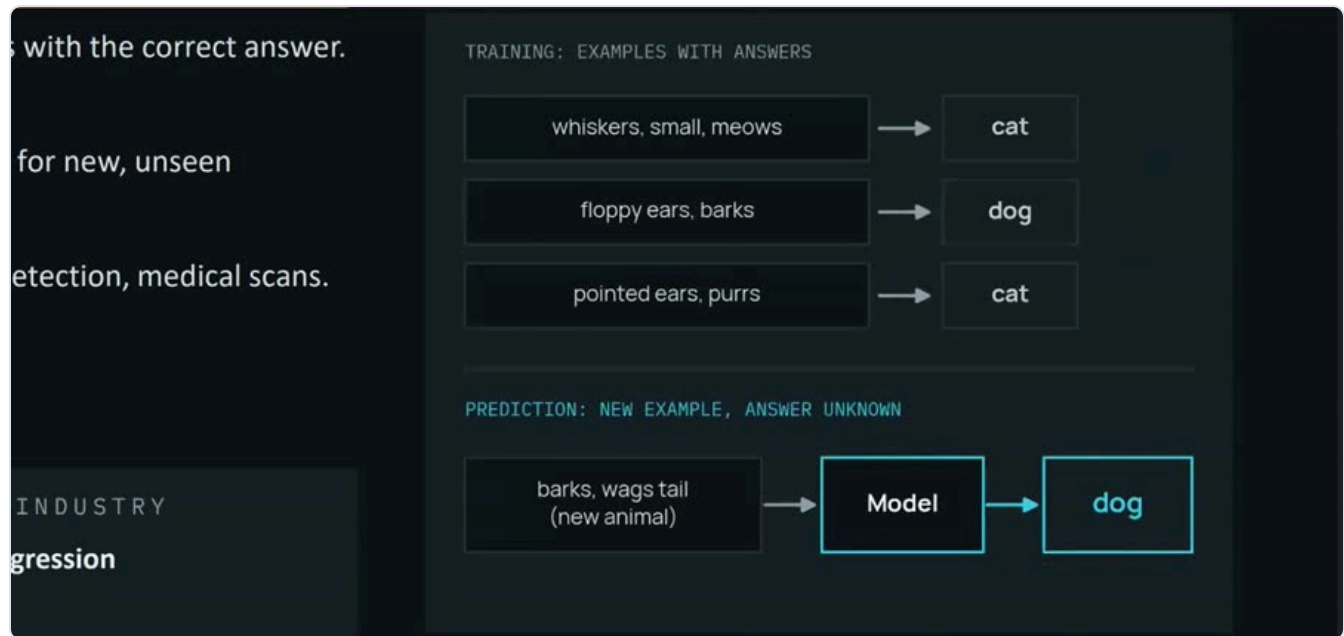
- Distinguish supervised, unsupervised and reinforcement learning
- Decide classification vs regression from a problem statement
- Give real-world examples of each

Three ways a machine can learn



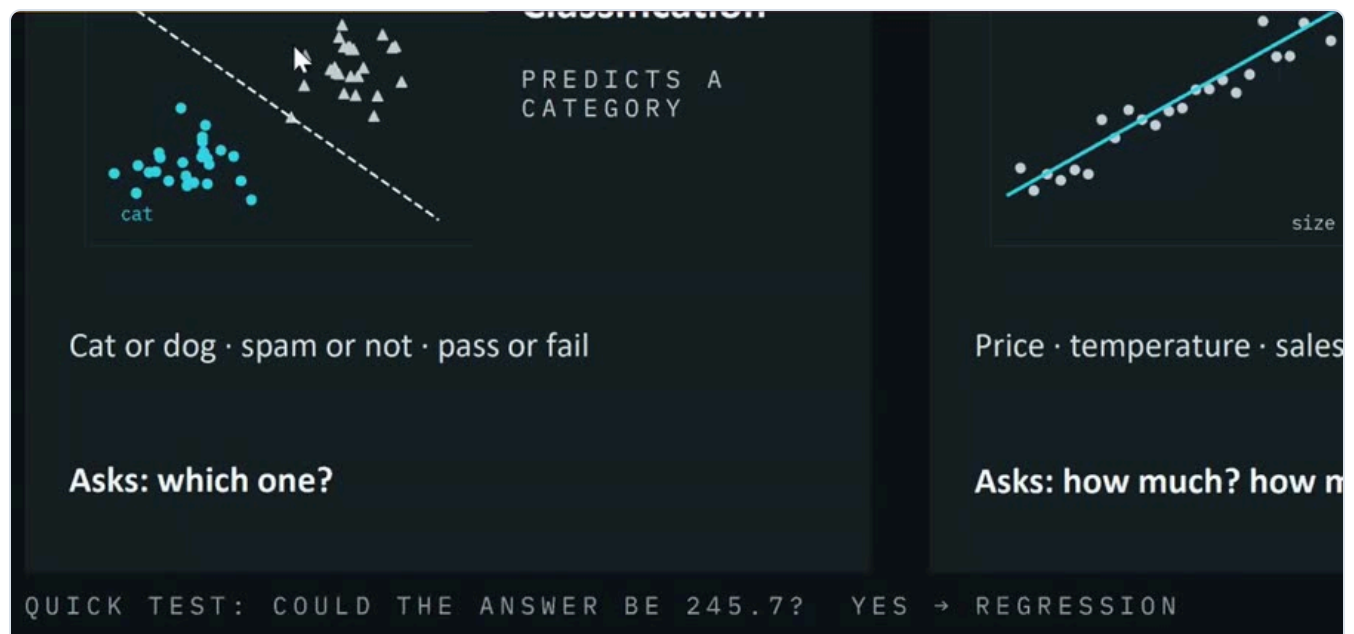
There are three ways a machine can learn. In supervised learning, every training example comes with the correct answer, like learning with a teacher. In unsupervised learning, there are no answers, and the model finds hidden groups or patterns on its own. And in reinforcement learning, an agent learns by trial and error, collecting rewards and penalties.

Supervised learning is the most common type in industry



Supervised learning is the most common type in industry. During training, every example comes with the correct answer: whiskers, small, meows, is a cat. Floppy ears and barks, is a dog. Then, for a new animal the model has never seen, it predicts the answer itself.

Supervised learning does two jobs



Supervised learning does two jobs. Classification predicts a category: cat or dog, spam or not, pass or fail. It asks, which one? Regression predicts a number: a price, a temperature, sales, or a delivery time. It asks, how much, or how many? A quick test: could the answer be two hundred forty-five point seven? If yes, it's regression.

Classification or regression?

Classification or regression?		
Question	Answer looks like	Type
Will it rain tomorrow?	Yes / no	Classification
How much rain will fall?	12.4 mm	Regression
Is this card payment fraud?	Fraud / not fraud	Classification
How many units will we sell?	1,240	Regression
Which of three products will they pick?	A / B / C	Classification (3 classes)

Try a quick quiz. The trainer's first question was about rain. Will it rain tomorrow? The answer is yes or no, so it's classification. How much rain will fall? That's a number, so regression. Is this card payment fraud? Classification. How many units will we sell next month? Regression. And which of three products will a customer pick? Still classification, just with three classes.

Unsupervised learning has no labels

No labels: nobody tells the computer the answer.

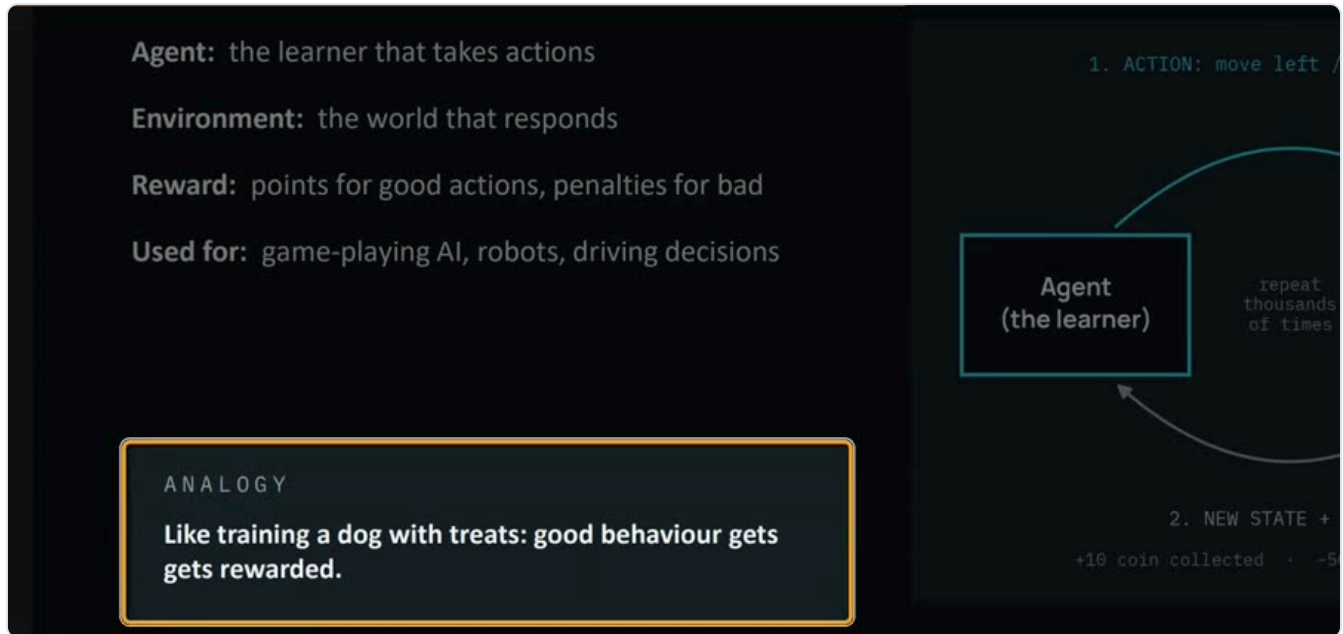
Clustering groups similar items together (e.g. K-Means).

Humans name the groups after they are found.

Used for: customer segments, grouping news, spotting unusual transactions.

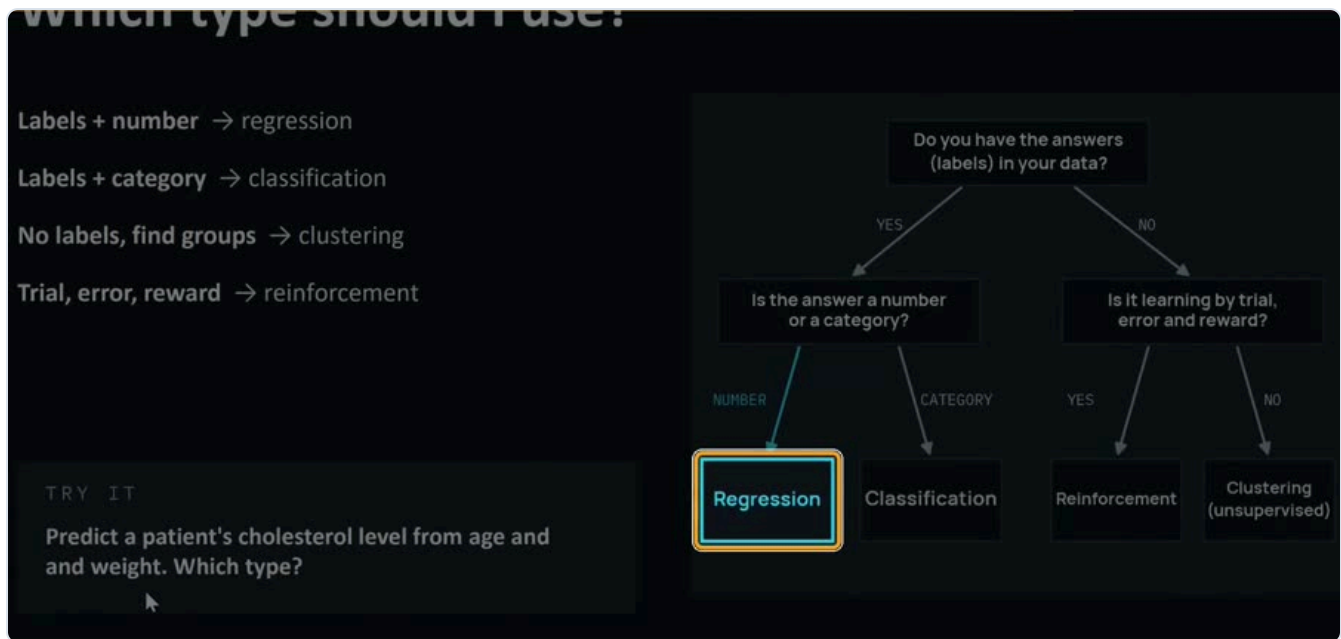
Unsupervised learning has no labels. Nobody tells the computer the answer. A clustering algorithm, such as K-means, groups similar items together. Here, raw customers plotted by income and spending fall into three groups. Humans then name the groups: big spenders, budget, and savers. It's used for customer segments, grouping news, and spotting unusual transactions.

Reinforcement learning works differently



Reinforcement learning works differently. An agent, the learner, takes an action in an environment, such as a game. The environment responds with a new state and a reward: plus ten for collecting a coin, minus fifty for falling in a hole. Repeated thousands of times, the agent learns which actions pay off. It's like training a dog with treats.

So which type should you use? Start at the top



So which type should you use? Start at the top. Do you have the answers, or labels, in your data? If yes, ask whether the answer is a number or a category: a number means regression, a category means classification. If there are no labels, is it learning by trial, error, and reward? Then it's reinforcement. Otherwise, you're clustering. Try this one: predict a patient's cholesterol level from age and weight. It's a number, so regression.

Same patient, two different questions

Same patient, two different questions

Question	Label	Type
Is this patient diabetic?	Yes / no	Classification
What is the patient's glucose level?	e.g. 140 mg/dL	Regression
Rule of thumb	Which one? / How much?	Category / number

The features can be identical. The label you want decides the type of problem.

The trainer's patient example shows how the question decides the type. Is this patient diabetic, yes or no? That's classification. What is the patient's glucose level? That's a number, so regression. Same patient, same features, but a different label changes the job. So always look at the label first, and ask: which one, or how much?

Key takeaways

- Supervised: labelled data → classification or regression
- Unsupervised: find groups without labels
- Reinforcement: learn from rewards
- Number → regression; category → classification

Practice

1. Write five prediction questions from your own work and label each as classification, regression, clustering or reinforcement.
2. Explain in two sentences why customer segmentation is usually unsupervised.

Check your understanding

1. Predicting tomorrow's temperature is which type?
2. What makes learning 'unsupervised'?
3. What does a reinforcement learning agent learn from?
4. Spam or not spam is which type?

Answers at the back of this handbook.

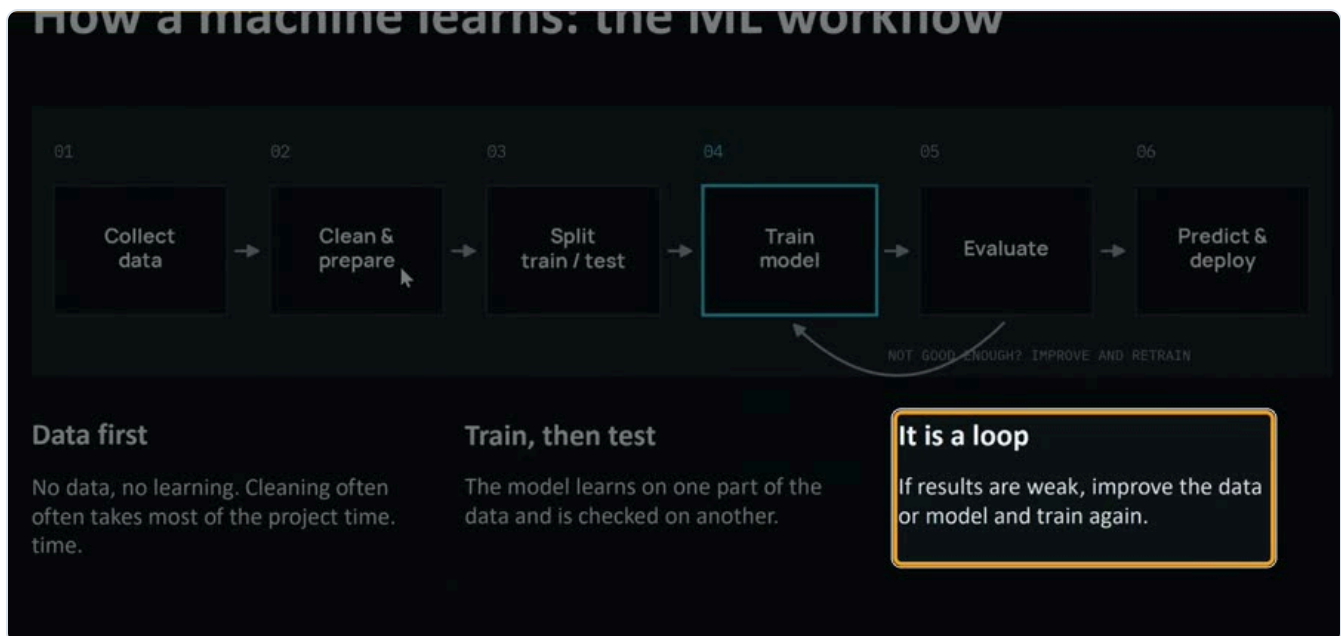
The ML Workflow: From Data to Deployment

Video: 04_AI-04_The_ML_Workflow.mp4 · 3:46 · Stage: Foundations · Audience: Beginners who completed AI-03

Learning objectives

- List the stages of the ML workflow
- Explain why test data must stay unseen
- Recognise overfitting and drift

Here's the workflow from the trainer's slides



Here's the workflow from the trainer's slides. First, collect the data. Then clean and prepare it. Split it into a training set and a test set. Train the model on the training data. Evaluate it on the test data. And finally, use it to predict, and deploy it. If the evaluation isn't good enough, you improve the data or the model and train again. It's a loop, not a straight line.

Preparing the data

Preparing the data

No data, no learning, and cleaning often takes most of the project time.

1 Missing values

Remove incomplete rows, or fill gaps with the mean, median or mode.

2 Text → numbers

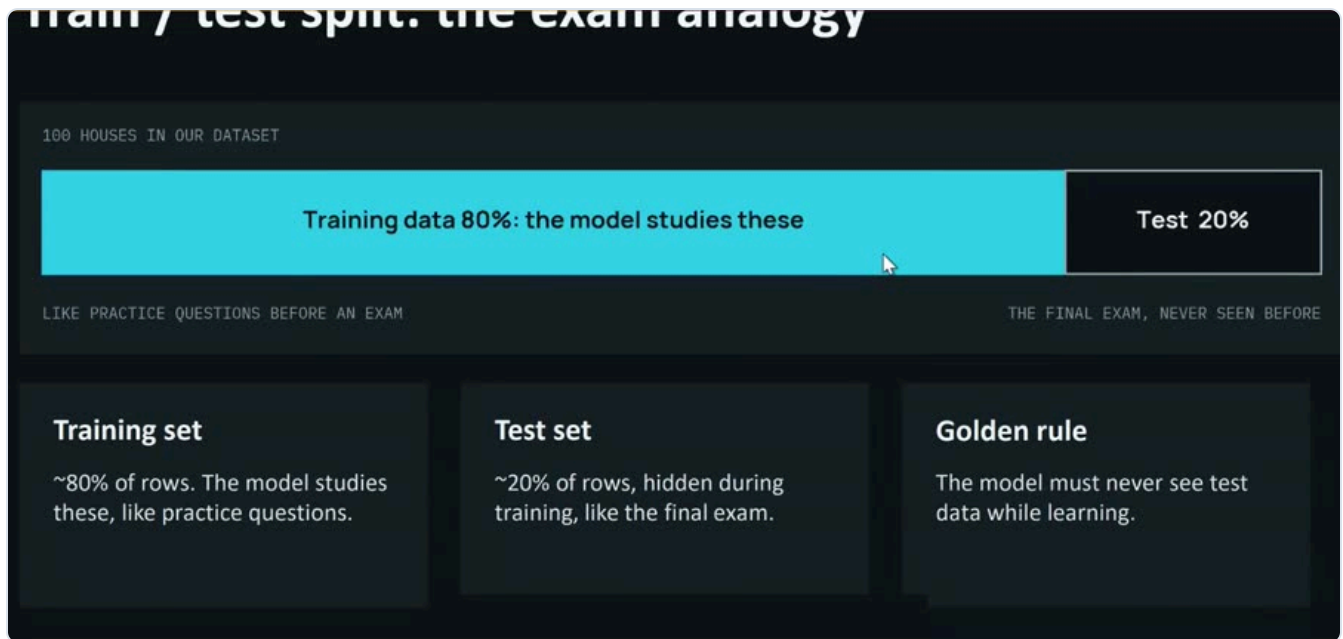
Models only understand numbers. One-hot encoding turns a category into 0/1 columns.

3 Representative

Does the data cover the cases the model will really see?

Data preparation usually takes most of a project's time. Three jobs come up again and again. First, handle missing values: remove the incomplete rows, or fill the gaps with the mean, median, or mode. Second, convert text categories into numbers, for example with one-hot encoding, because models only understand numbers. And third, check that the data actually represents the problem you're trying to solve.

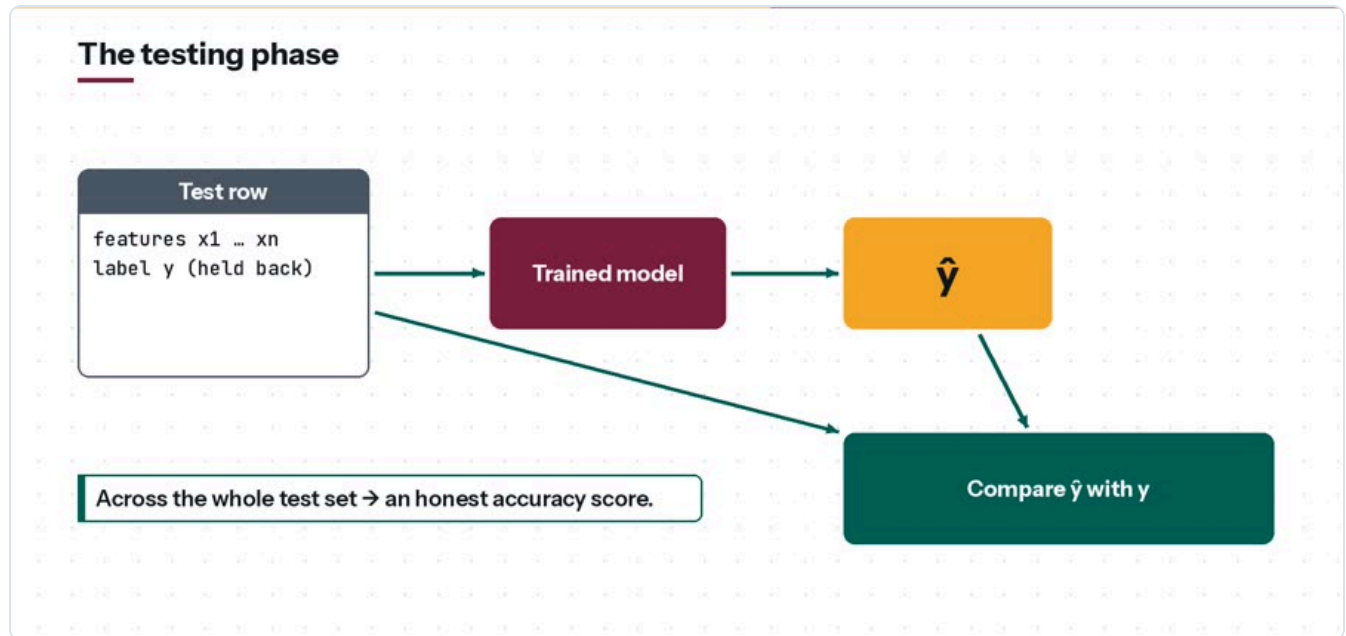
Next, the split



Next, the split. A common rule of thumb is eighty-twenty. About eighty percent of rows are training data: the model studies these, like practice questions before an exam. The other twenty percent is the test set, the final exam it has never seen. Some teams use sixty, twenty, twenty, keeping a third slice for validation while they tune. And the golden rule: the model must never see the test data while learning.

Tip Variant: 60% train · 20% validation · 20% test

The testing phase



So how does testing actually work? For each test row, we hold back the label and pass only the features to the trained model. The model returns its prediction, $\hat{y}$. Then we compare $\hat{y}$ with the true answer we kept aside. Doing this across the whole test set gives us an honest score, because the model never saw these answers.

Train and test on the same kind of data

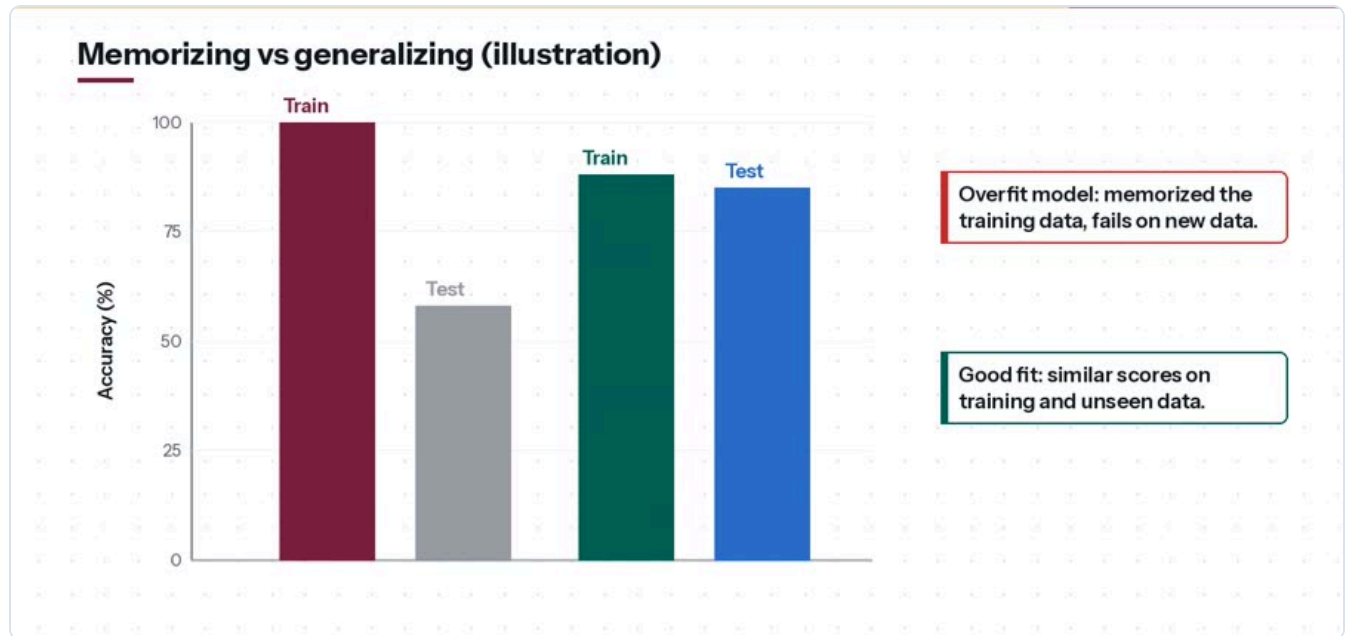
Train and test on the same kind of data

Trained on	Used on	What goes wrong
Fruit from region A	Fruit from region B	Size, color and texture differ
One country's customers	Another country's customers	Different transaction patterns

Patterns learned from one population may not carry over to another. Test on data like the data the model will really see.

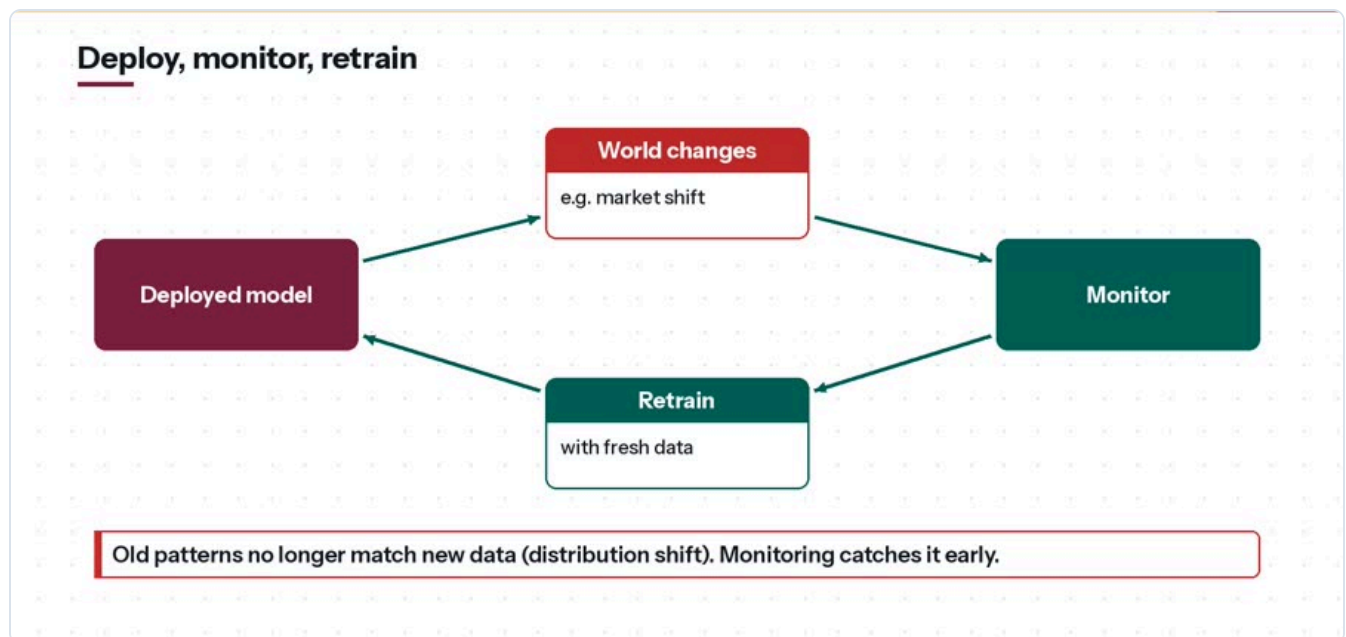
There's a second rule: training and test data must come from the same kind of source. The trainer gives two examples. Train a fruit classifier on fruit from one region, then test it on fruit from another, and the sizes, colors, and textures differ. Or train a loan model on one country's customers and apply it in another. In both cases the patterns don't carry over.

Memorizing vs generalizing (illustration)



Why the golden rule? If a model is tested on the same data it trained on, it can simply memorize the answers. Its training score looks perfect. But on new data, the score drops sharply, because it never learned to generalize. This is called overfitting. A well-fitted model scores a little lower in training, but holds up on unseen data.

Deploy, monitor, retrain



Deployment isn't the end. The world changes. The trainer's example: if a market shifted over the last two months, a model trained on five to ten years of older data will start giving wrong answers. This is called distribution shift. So we monitor the model, collect fresh data, and retrain. The loop keeps running for as long as the model is in use.

Key takeaways

- Collect → clean → split → train → evaluate → deploy
- Keep the test set unseen (about 80/20)
- Watch for overfitting
- Monitor and retrain when the world changes

Practice

1. You have 1,000 labelled loan applications. How many would you put in the training set and the test set with an 80/20 split?
2. Describe one way the world could change that would make a sales-forecast model go stale.

Check your understanding

1. What is the golden rule of testing?
2. A model scores 100% on training data but 55% on new data. What is this?
3. Why convert text categories to numbers?
4. Why retrain a deployed model?

Answers at the back of this handbook.

Neural Networks & Deep Learning Intuition

Video: 05_AI-05_Neural_Networks_and_Deep_Learning.mp4 · 4:57 · Stage: Foundations · Audience: Beginners who completed AI-04

Learning objectives

- Describe a neuron: weights, bias, activation
- Explain layers and probability (softmax) outputs
- Match CNN, RNN/LSTM and transformers to data types

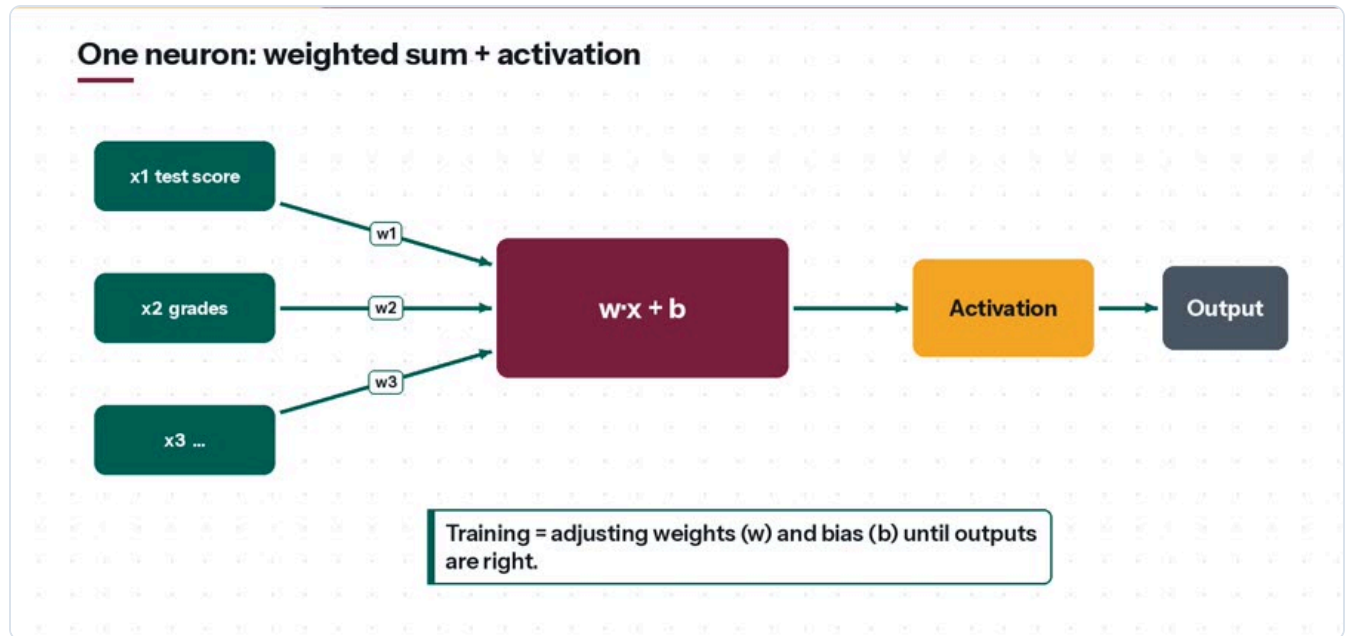
Machine learning vs deep learning

Machine learning vs deep learning

	Classic machine learning	Deep learning
Analogy	One person working on a task	A team of ten sharing what they learn
Structure	One model, often hand-picked features	Many layers that learn features themselves
Needs	Less data, modest compute	Lots of data and compute (GPUs)
Best at	Tables of numbers	Images, speech and text

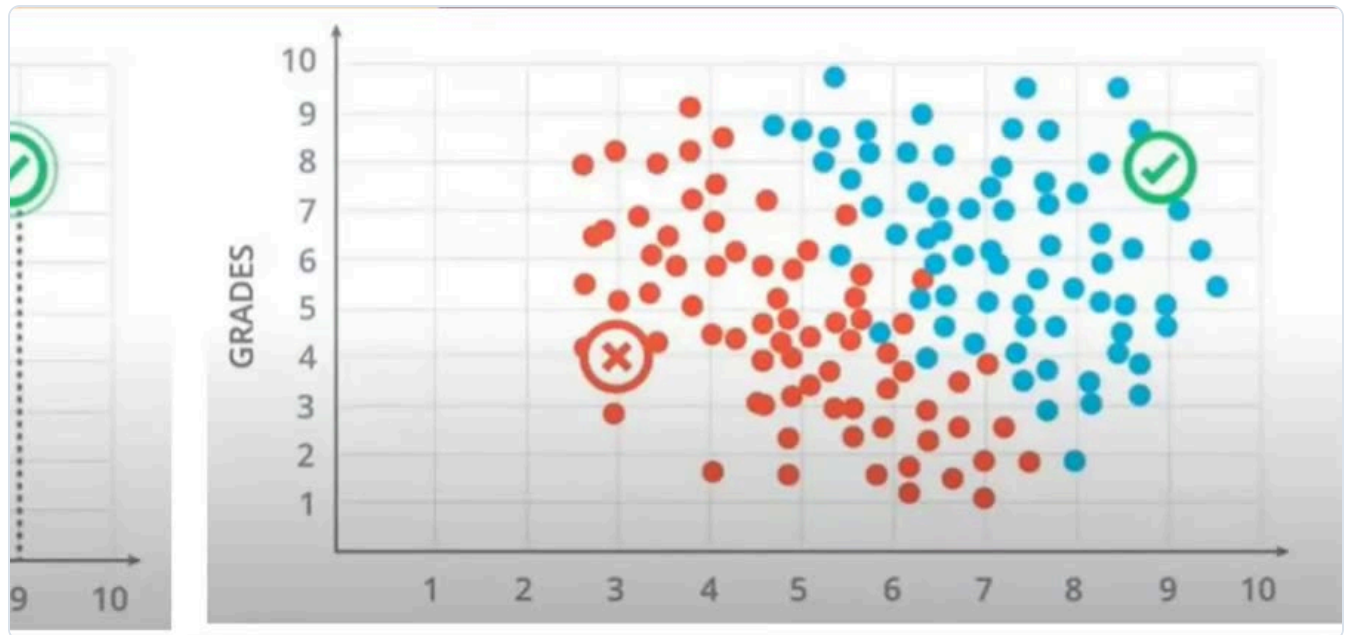
Deep learning is machine learning with many layers. The trainer's analogy: classic machine learning is like one person working on a task, sharing only their own knowledge. Deep learning is like a team of ten, each passing what they learn to the next. It needs much more data and computing power, but it shines on images, speech, and text.

One neuron: weighted sum + activation



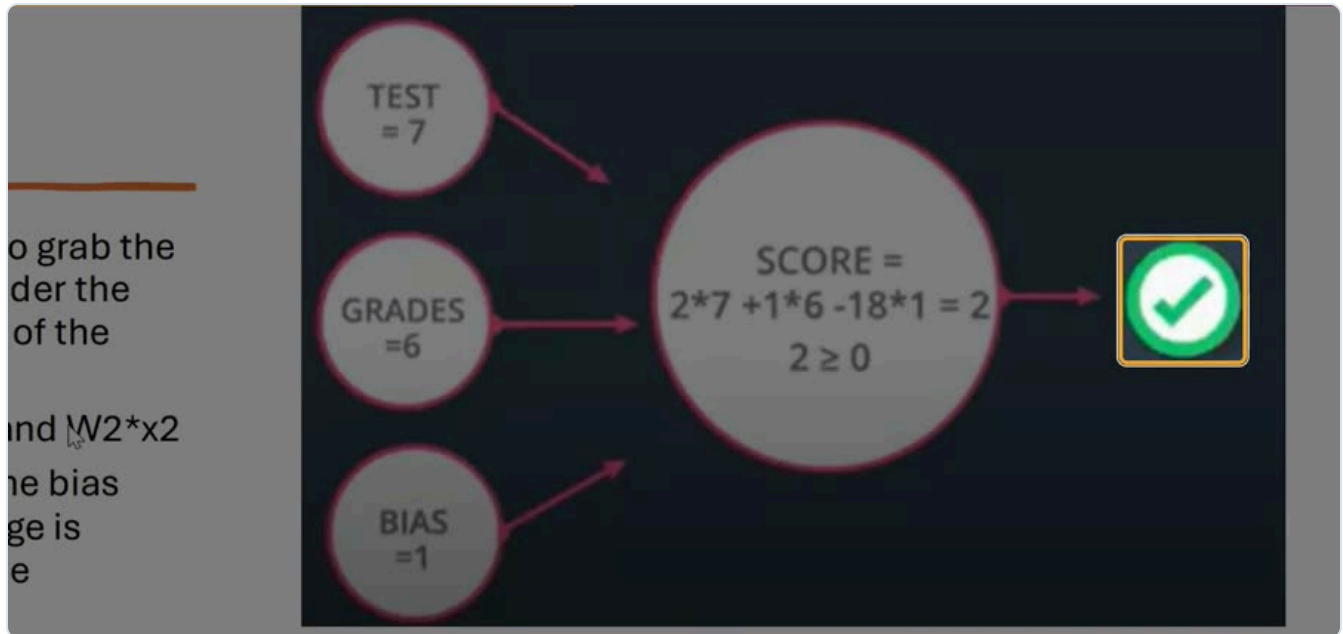
A single artificial neuron is simple. It takes inputs, multiplies each one by a weight that says how important it is, adds them up, and adds a bias. In short, $w \cdot x$ plus b . Then an activation function decides how strongly the neuron fires. Training means adjusting the weights and bias until the outputs are right.

The trainer made this concrete with an admissions department



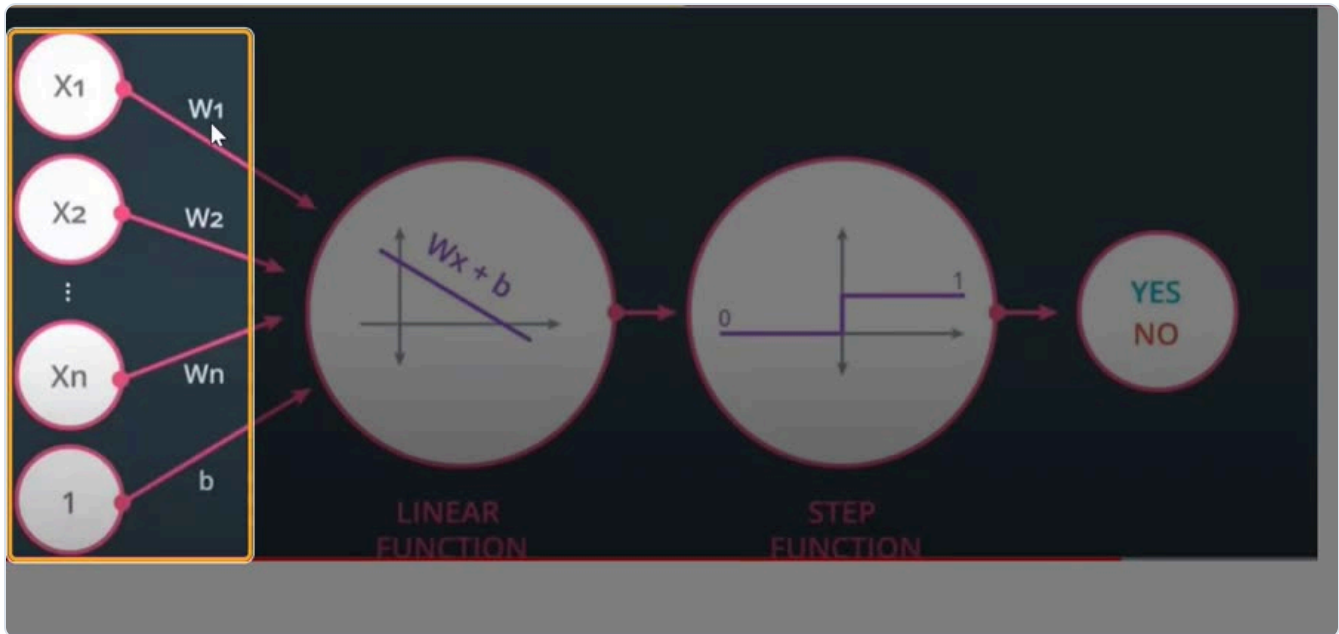
The trainer made this concrete with an admissions department. Student one scored nine on the test and eight for grades, and was accepted. Student two scored three and four, and was rejected. But what about student three, with seven and six? To decide, we plot the past students by test score and grades. Accepted students sit at the top right, and rejected students at the bottom left.

A single neuron learns a straight line that separates the two groups



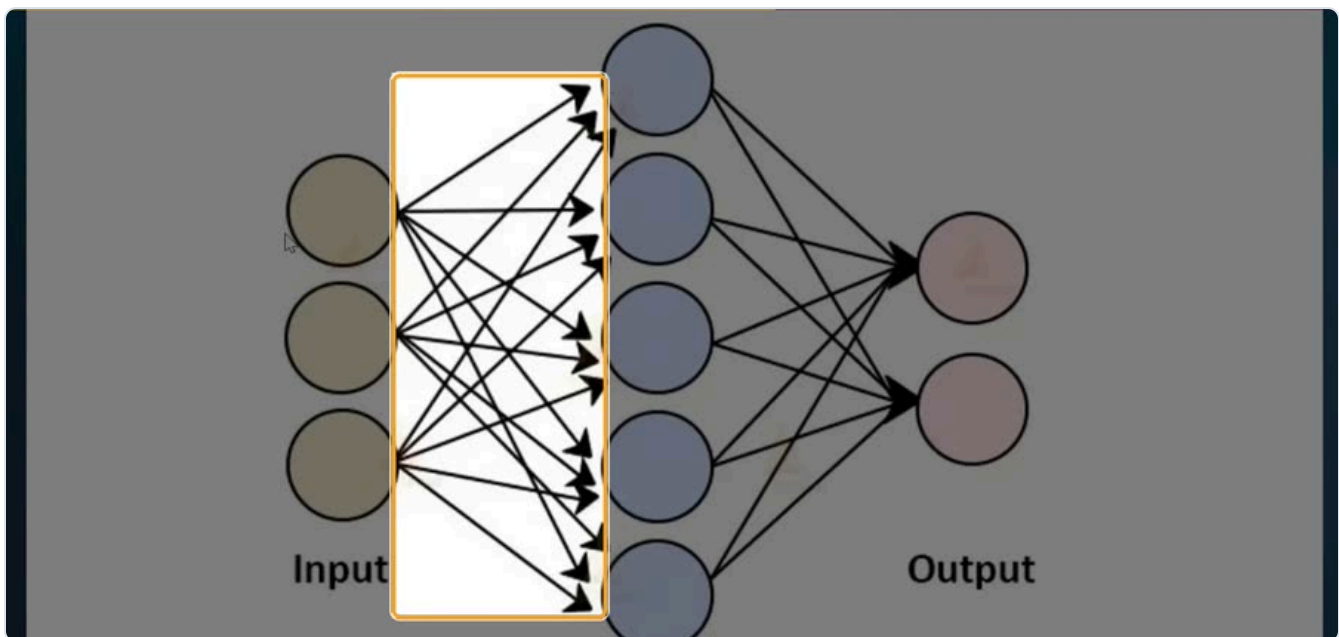
A single neuron learns a straight line that separates the two groups. On the trainer's slide, the line is two times the test, plus the grade, minus eighteen equals zero. That gives a score: two times the test, plus the grade, minus eighteen. If the score is positive, accept the student. If it's negative, reject. For student three, two times seven, plus six, minus eighteen, gives two. Two is at least zero, so the answer is accept.

This is the perceptron, the simplest neural network



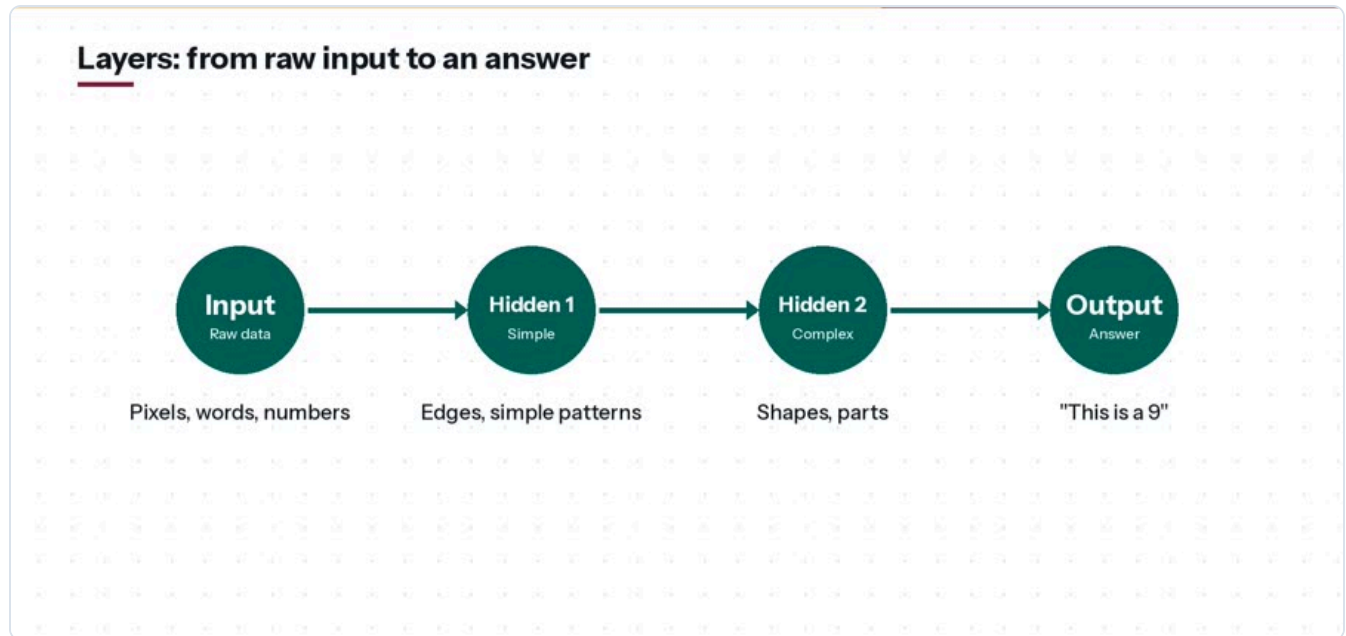
This is the perceptron, the simplest neural network. Each input is multiplied by its weight, and a bias is added: $w \times x$ plus b . That's the linear function. Then a step function turns the result into a yes or a no: one if the score is above zero, otherwise zero. With more features, like class rank, you simply add more weighted inputs.

One neuron draws one line, but real problems need more



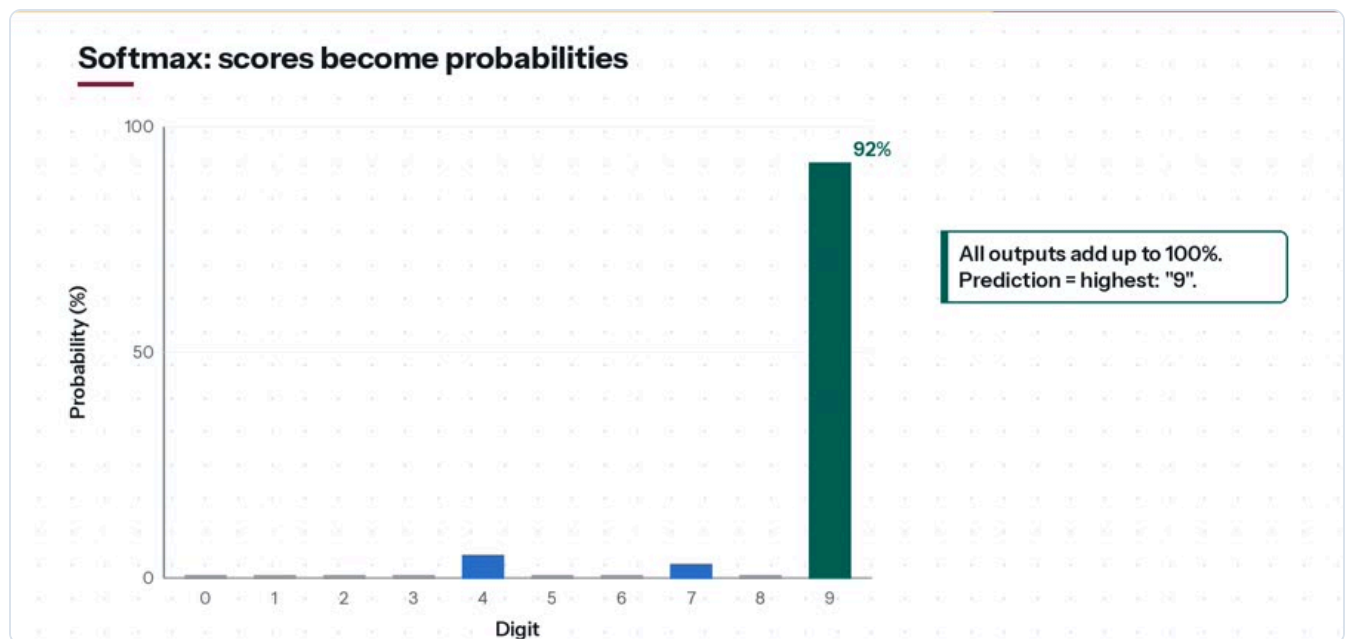
One neuron draws one line, but real problems need more. So we connect many neurons. On the trainer's diagram, the input layer on the left feeds a hidden layer in the middle, and every neuron passes its result forward to the output layer on the right. Each connection carries its own weight.

Stacking layers lets a network build up understanding step by step



Stacking layers lets a network build up understanding step by step. The input layer receives the raw data, such as pixels. The hidden layers each combine what the previous layer found, from edges, to shapes, to whole objects. The output layer gives the answer. When there are many hidden layers, we call it deep learning.

Softmax: scores become probabilities



For classification, the output layer usually ends with softmax, which turns raw scores into probabilities that add up to one hundred percent. Shown a handwritten digit, the network might say: ninety-two percent it's a nine, five percent a four, three percent a seven. The model picks the class with the highest probability.

Neural networks come in several architectures, each designed for di...

Neural networks

Neural networks come in various architectures, each designed to handle different types of data and tasks

- Artificial Neural Networks (ANNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)

Neural networks come in several architectures, each designed for different types of data and tasks. The trainer's slide lists four: artificial neural networks, convolutional neural networks, recurrent neural networks, and long short-term memory networks.

Network families and their data

Network families and their data

Family	How it works	Typical data
CNN	Scans small patches of an image	Images, video
RNN / LSTM	Reads data one step at a time, with memory	Time series, sentences
Transformer	Looks at the whole sequence with attention	Language (LLMs), also images

Here's how they differ. Convolutional networks, or CNNs, scan small patches of an image, so they suit images and video. Recurrent networks, RNNs, and their improved form, LSTMs, read data one step at a time with memory, so they suit time series and sentences. And transformers, which power today's large language models, look at a whole sequence at once using attention.

Key takeaways

- Neuron = $w \cdot x + b$, then activation
- Many hidden layers = deep learning
- Softmax → probabilities that sum to 100%
- CNN: images · RNN/LSTM: sequences · Transformer: language

Practice

1. Sketch a network that reads three exam scores and predicts pass or fail. Label the input, hidden and output layers.
2. For each task, pick a network family: detecting defects in product photos, forecasting daily sales, summarizing reports.

Check your understanding

1. What does a single neuron compute before its activation?
2. What makes a network 'deep'?
3. What does softmax produce?
4. Which family is typically used for images?

Answers at the back of this handbook.

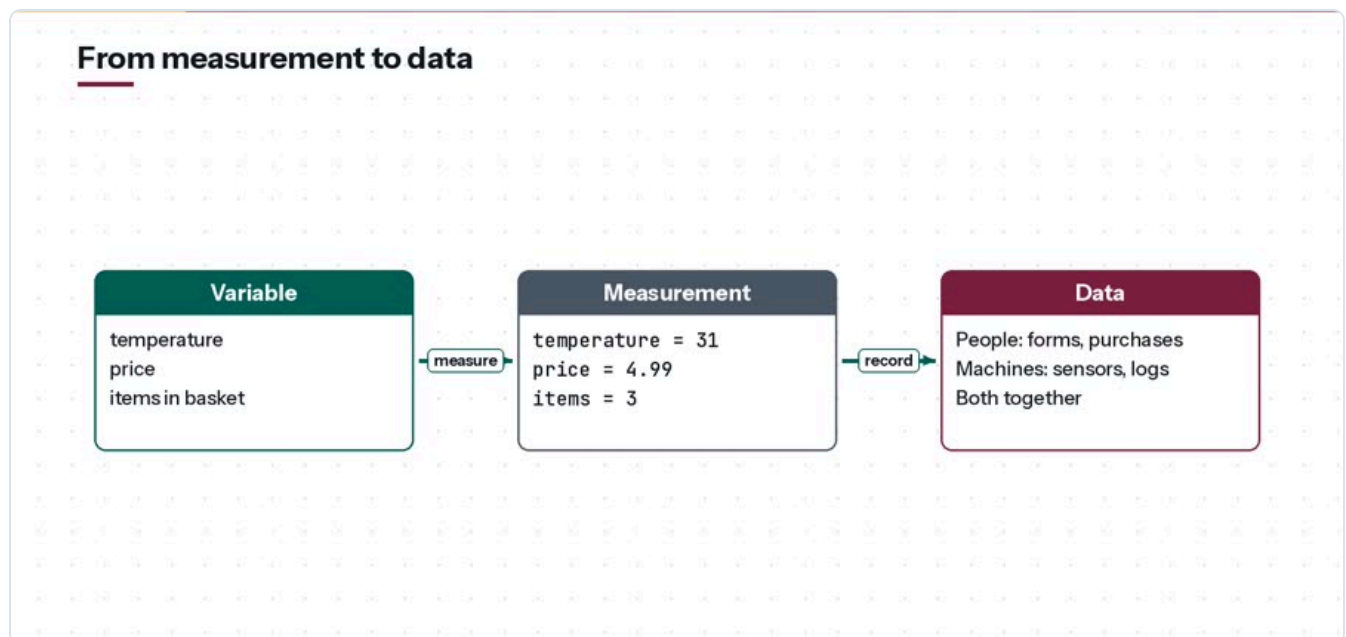
Data, Analytics & Data Types

Video: 06_AI-06_Data_Analytics_and_Data_Types.mp4 · 5:02 · Stage: Data Understanding · Audience: Beginners in AI and ML; no statistics or coding needed

Learning objectives

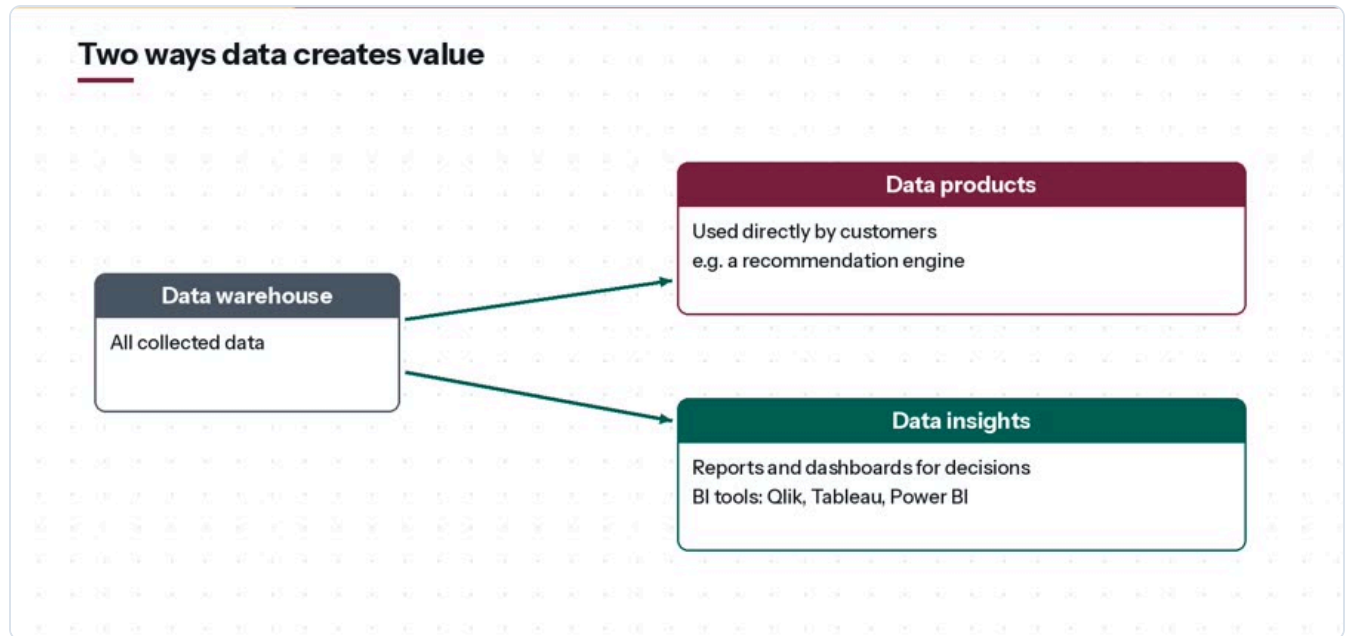
- Explain what data is and where it comes from
- Tell the four types of analytics apart
- Classify data as nominal, ordinal, interval, ratio, discrete or continuous

From measurement to data



Start with a variable: anything we can measure that can change, like temperature, price, or the number of items in a basket. When we record those measurements, we get data. Data is generated by people, like form entries and purchases, by machines, like sensors and logs, and increasingly by both together.

Two ways data creates value



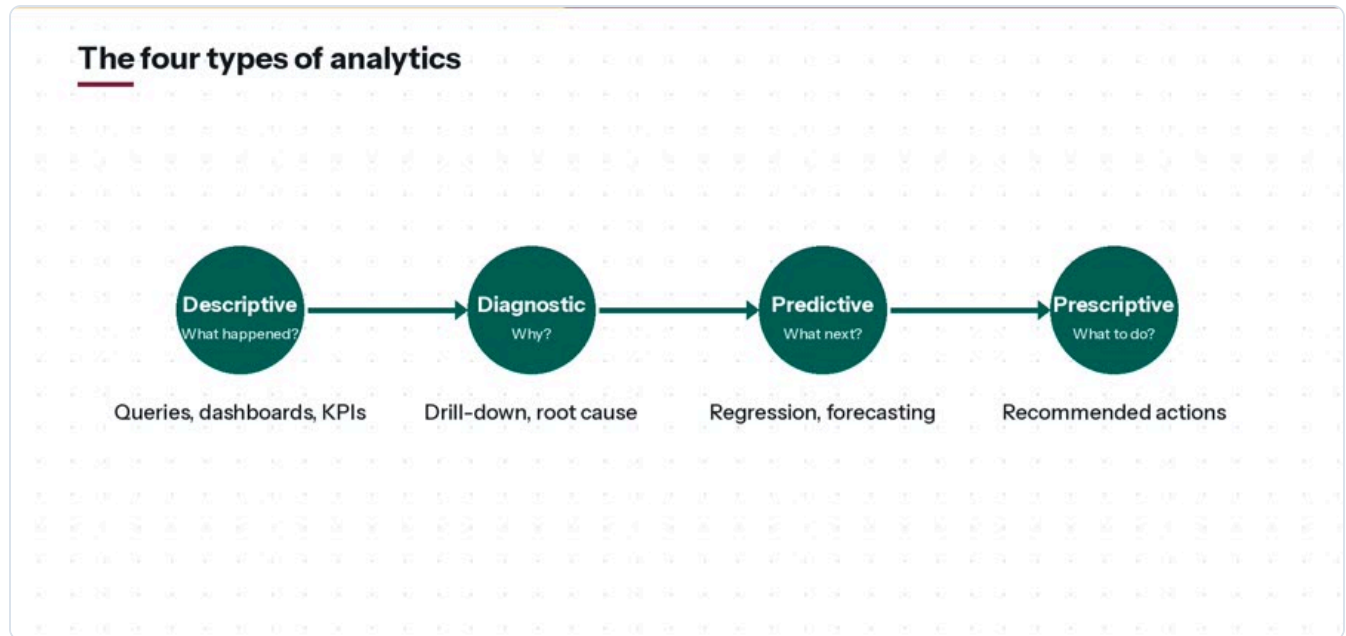
Organizations collect that data in a data warehouse. From there it flows two ways. Data products are things customers use directly, like a recommendation engine that suggests what to buy next. Data insights are reports and dashboards that help people decide, which is the world of BI tools such as Qlik, Tableau, and Power BI.

Two words are often mixed up

	Data Analytics
Historical (past) data.	Broader concept that includes data analysis.
What happened? Why did it happen?	Uses data, statistics, technology, and models to support decision-making.
Collecting, and interpreting data.	Focuses on: What happened? Why did it happen? What will happen? What should we do next?
Gain past events.	Goal: Generate insights and improve future outcomes.
Sales fell by 15% last month. Data analysis finds the reason.	Example: Predict next month's sales and recommend actions to increase revenue.

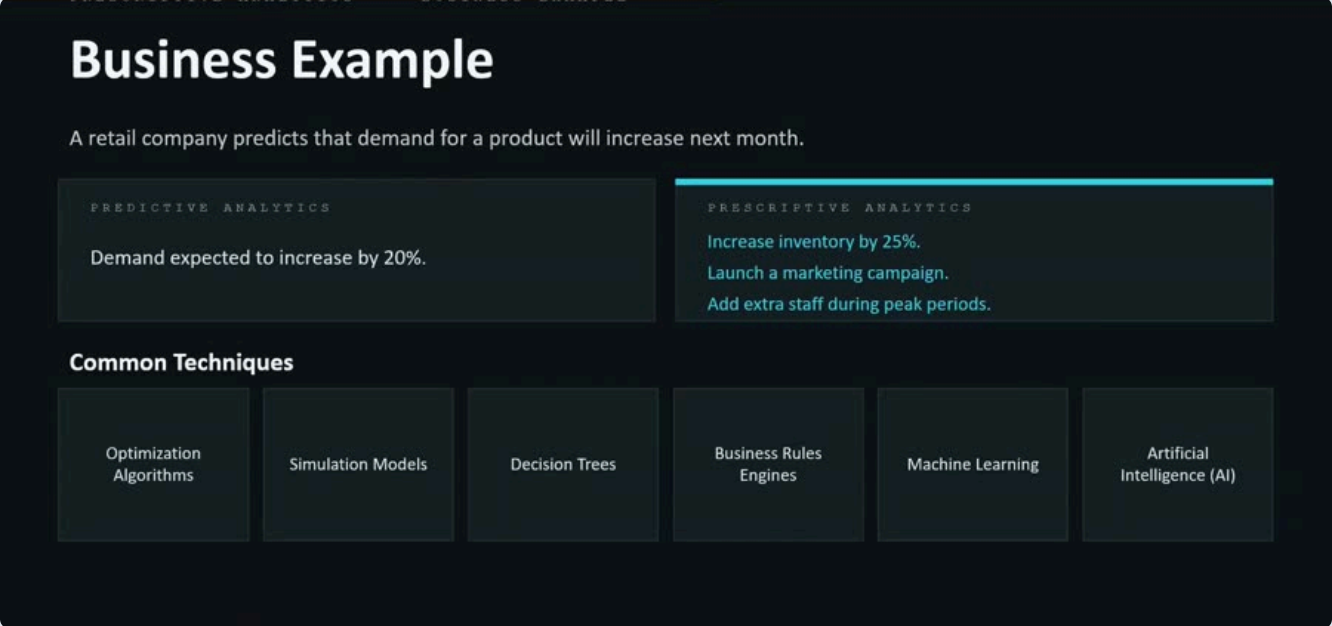
Two words are often mixed up. On the trainer's slide, data analysis focuses on historical data. It asks what happened, and why, and its goal is to understand past events. For example, sales fell fifteen percent last month, and analysis finds the reason. Data analytics is the broader idea. It includes analysis, but also uses statistics, technology, and models to ask what will happen, and what we should do next. That forward-looking side is where machine learning lives.

Analytics comes in four types, each answering a harder question



Analytics comes in four types, each answering a harder question. Descriptive analytics tells you what happened, using queries and dashboards. Diagnostic analytics asks why, by drilling down to the root cause. Predictive analytics estimates what is likely to happen, with methods like regression and time-series forecasting. And prescriptive analytics recommends what to do next.

The trainer walks one retail story through the types



Business Example

A retail company predicts that demand for a product will increase next month.

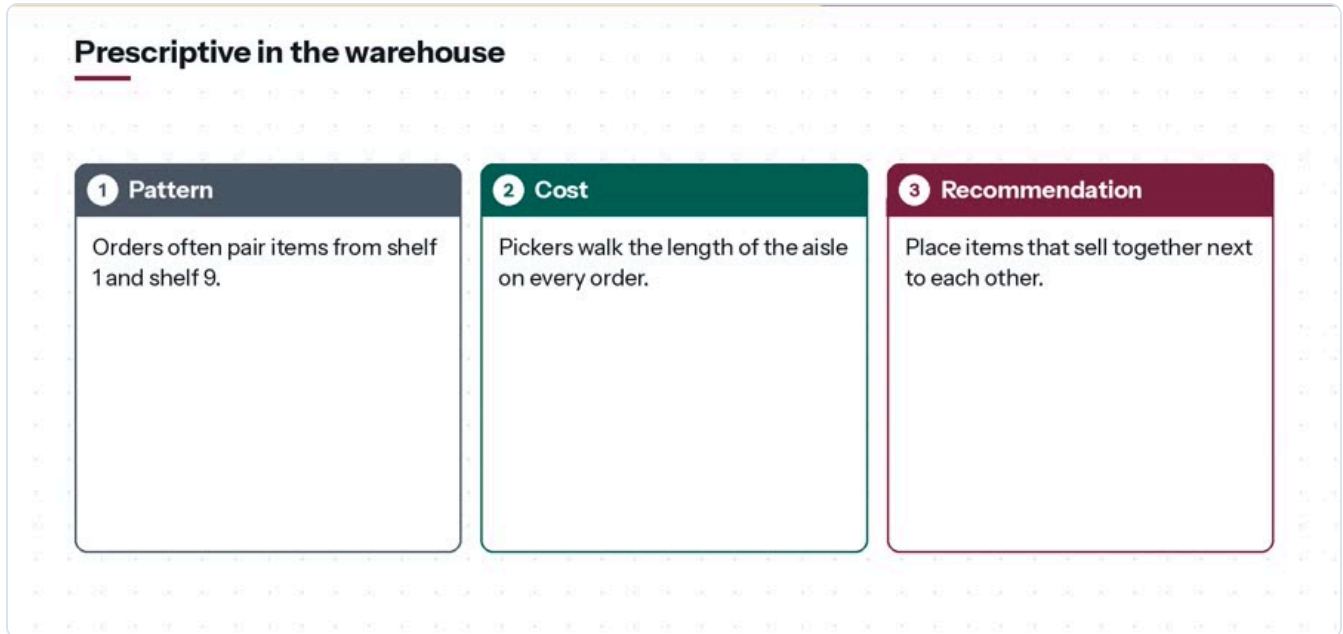
PREDICTIVE ANALYTICS	PRESCRIPTIVE ANALYTICS
Demand expected to increase by 20%.	- Increase inventory by 25%. - Launch a marketing campaign. - Add extra staff during peak periods.

Common Techniques

Optimization Algorithms	Simulation Models	Decision Trees	Business Rules Engines	Machine Learning	Artificial Intelligence (AI)
-------------------------	-------------------	----------------	------------------------	------------------	------------------------------

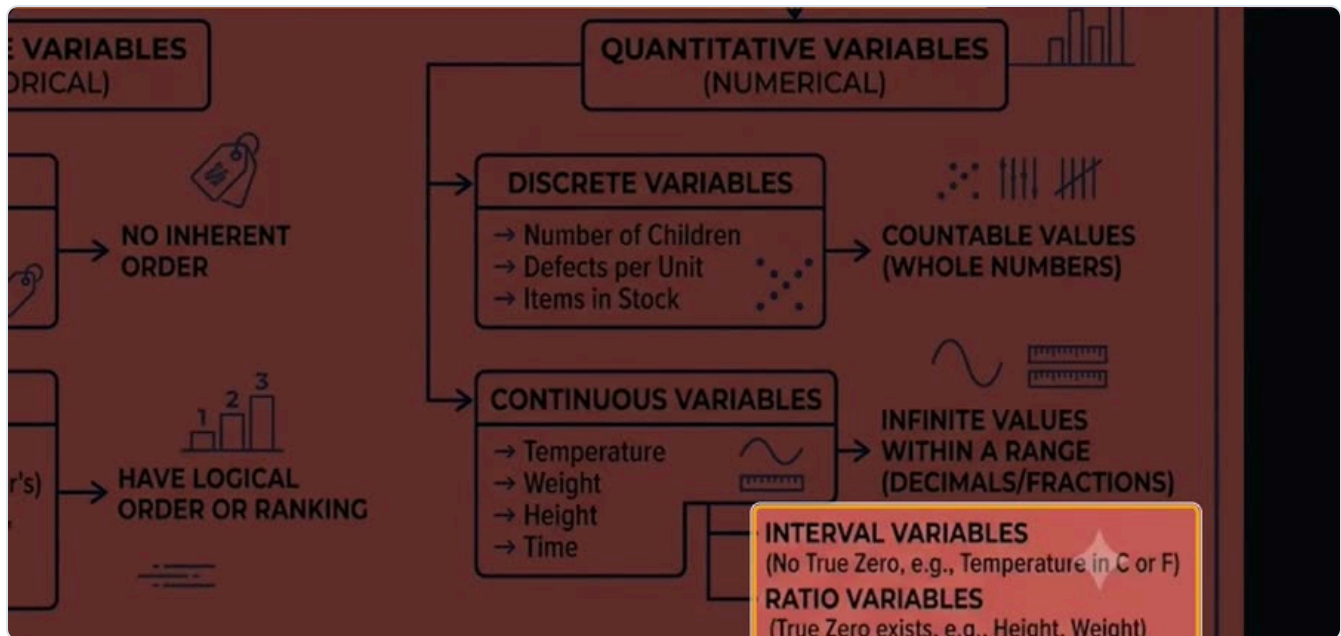
The trainer walks one retail story through the types. Descriptive: sales decreased by twenty percent in June. Diagnostic: why? The analysis shows fewer customer visits, product stock shortages, and reduced marketing. Then the story moves forward. Predictive: demand for a product is expected to rise twenty percent next month. Prescriptive: increase inventory by twenty-five percent, launch a marketing campaign, and add staff for peak periods. Machine learning is one of the techniques behind these recommendations.

Prescriptive in the warehouse



The trainer added a warehouse example of prescriptive analytics. If orders keep combining items from the first shelf and the ninth shelf, pickers waste time walking between them. Descriptive analytics shows the pattern, and prescriptive analytics recommends placing items that sell together side by side.

Now, the types of data



Now, the types of data. The trainer's schematic starts with variables, which are either qualitative, meaning categorical, or quantitative, meaning numerical. Categorical variables are nominal, like colours or brands, with no inherent order, or ordinal, like education level or survey ratings, which do have an order. Numerical variables are discrete, counted in whole numbers, like the number of children. Or they're continuous, measured on a scale, like height, weight, or time. Continuous data splits again into interval and ratio.

Four levels of measurement

Four levels of measurement

Level	What it means	Examples	You can...
Nominal	Labels, no order	Colour, product brand	Count
Ordinal	Ordered, unequal gaps	Education level, 1-5 rating	Rank
Interval	Equal gaps, no true zero	Temperature in °C	Add, subtract
Ratio	Equal gaps and a true zero	Weight, income, sales	Multiply, divide

Together these give four levels of measurement. Nominal data are labels with no order. Ordinal data have an order, but the gaps aren't equal. Interval data have equal gaps but no true zero, like temperature in Celsius: twenty degrees isn't twice as hot as ten. Ratio data have a true zero, like weight, income, or sales, so you can say one value is twice another.

Why data types matter in ML

Why data types matter in ML

1 Valid statistics

Average a ratio field, never a nominal label. Use the mode for categories.

2 Encoding

Ordinal → ordered numbers (1, 2, 3).
Nominal → one-hot columns.

3 Choosing the task

Numeric target → regression.
Categorical target → classification.

Why does this matter for machine learning? The data type decides what you can do with it. You can average ratio data, but not nominal labels. Ordinal categories can be encoded as ordered numbers, while nominal ones need one-hot encoding. And the type of your target decides the task: a number means regression, a category means classification.

Key takeaways

- Data = recorded measurements of variables
- Analysis looks back; analytics looks ahead
- Descriptive → diagnostic → predictive → prescriptive
- Categorical or numerical; four levels of measurement

Practice

1. List five fields from a dataset you know and label each as nominal, ordinal, interval or ratio, and as discrete or continuous.
2. Take a business problem from your work and describe what descriptive, diagnostic, predictive and prescriptive analytics would each add.

Check your understanding

1. Which analytics type answers 'why did it happen?'
2. A 1–5 satisfaction rating is measured at which level?
3. Is temperature in °C interval or ratio data?
4. A recommendation engine is a data product or a data insight?

Answers at the back of this handbook.

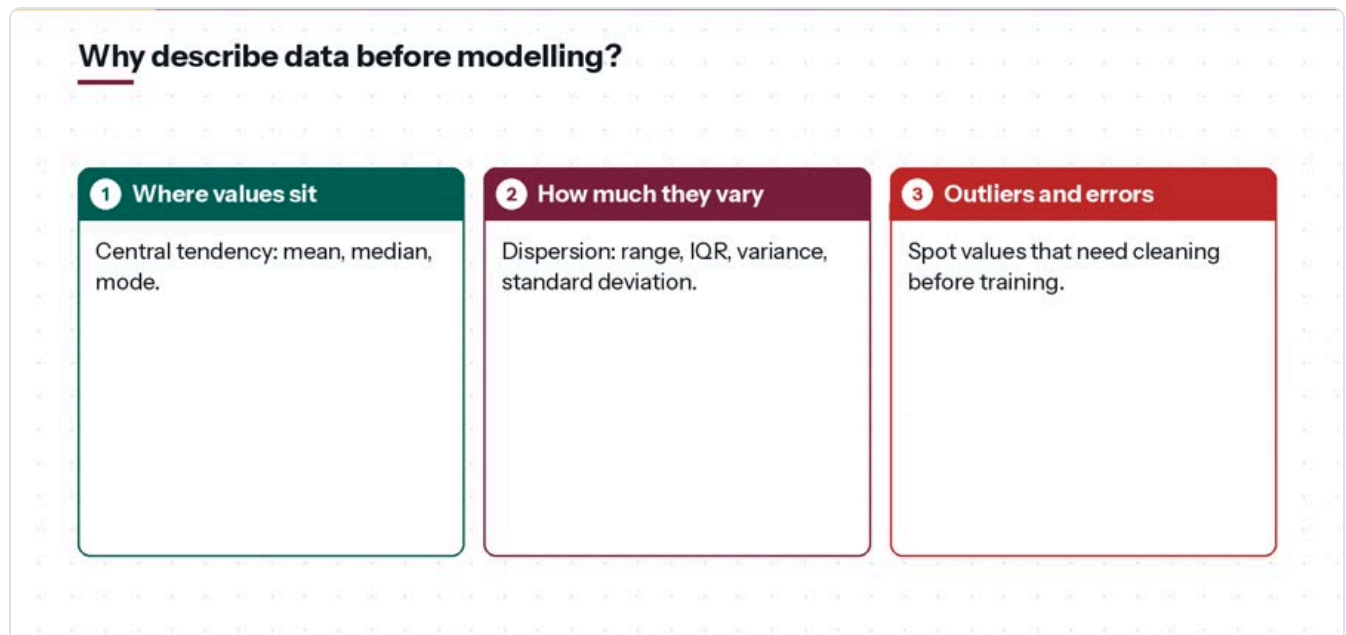
Describing Data: Central Tendency & Spread

Video: 07_AI-07_Central_Tendency_and_Spread.mp4 · 5:08 · Stage: Exploratory Data Analysis · Audience: Beginners in the AI & ML track who completed AI-06

Learning objectives

- Choose between mean, median and mode
- Compute range, IQR, variance and standard deviation
- Recognise skewness and kurtosis

Why describe data before modelling?



Why describe data before modelling?

- 1 Where values sit**
Central tendency: mean, median, mode.
- 2 How much they vary**
Dispersion: range, IQR, variance, standard deviation.
- 3 Outliers and errors**
Spot values that need cleaning before training.

Why describe data first? Summary statistics show you where most values sit, how much they vary, and whether there are outliers or errors to clean. Models learn from whatever you give them, so problems you miss here end up inside your predictions.

Mean, median and mode

Mean, median and mode

Example data: 3, 5, 5, 7, 10

Mean

Sum ÷ count
 $(3+5+5+7+10) \div 5 = 6$

Median

Middle value when sorted
3, 5, [5], 7, 10 → 5

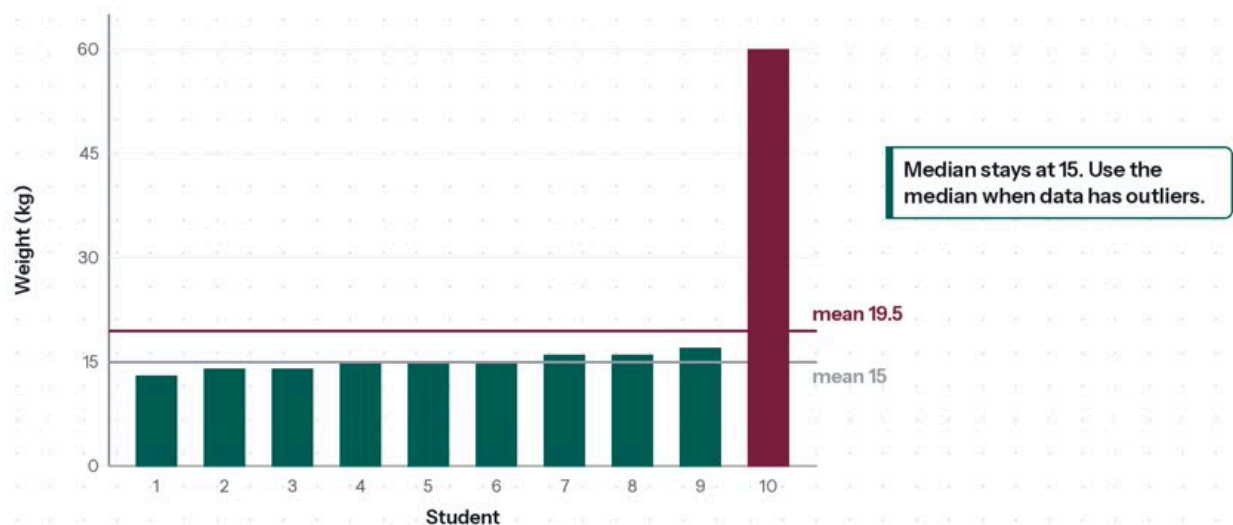
Mode

Most frequent value
5 appears twice → 5

Central tendency summarizes a whole group with one typical number. The mean is the sum of the values divided by how many there are. The median is the middle value once you sort them. And the mode is the value that appears most often. For three, five, five, seven, and ten, the mean is six, the median is five, and the mode is five.

Outliers pull the mean

Outliers pull the mean



The mean has one big weakness: extreme values pull it. Imagine the weights of nine young pupils, all around fifteen kilograms. Their mean is fifteen. Now one older student sits in by mistake, weighing sixty kilograms. The mean jumps to nineteen and a half, which describes nobody in the room. The median barely moves, staying at fifteen.

Population mean vs sample mean

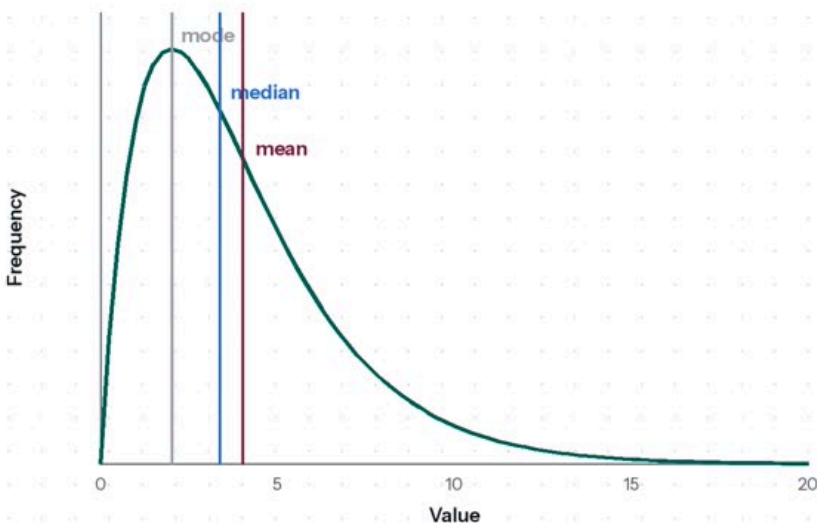
Population mean vs sample mean

	Population	Sample
Data	Every member of the group	A subset of the group
Symbol	μ	$\bar{x}$
Formula	$\mu = \text{sum of } x \div N$	$\bar{x} = \text{sum of } x \div n$
In ML	Rarely available	What we usually have

A quick note on notation. When you have every member of a group, you compute the population mean, written as the Greek letter mu, and divide by capital N. When you only have a sample, you compute the sample mean, written x bar, and divide by lowercase n. In machine learning we almost always work with samples.

A right-skewed distribution

A right-skewed distribution



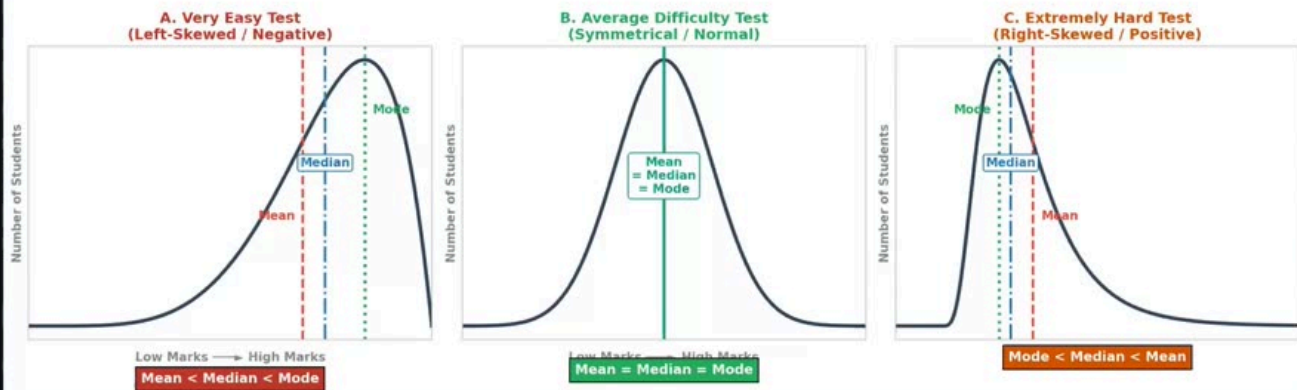
Skewed data? Report the median.

Real data is often skewed. Here's a right-skewed distribution, like scores on a very hard exam, or household incomes: most values are low, with a long tail to the right. The mode sits at the peak. The median is a little to the right. And the mean is pulled furthest along the tail. When data is skewed, report the median.

The trainer's slide shows all three shapes with exam scores

Understanding Mean, Median, Mode

Grade 10 Statistics: Understanding Exam Score Distributions



The trainer's slide shows all three shapes with exam scores. On a very easy test, most students score high, and a few low marks form a tail on the left. So the mean is less than the median, which is less than the mode. On a test of average difficulty, the curve is symmetric, and mean, median, and mode are equal. On an extremely hard test, the tail runs to the right, and the mean is the largest. So before choosing a measure, look at the histogram first.

The centre isn't enough; we also need to know how spread out the da...

Quartiles: Example

Ordered array: 106, 109, 114, 116, 121, 122, 125, 129

TIP
IQR = 123.5 - 111.5 = 12

Q₁ (First Quartile)
 $i = 25/100 \times 8 = 2$
 $Q_1 = (109 + 114) / 2 = 111.5$

Q₂ (Second Quartile / Median)
 $i = 50/100 \times 8 = 4$
 $Q_2 = (116 + 121) / 2 = 118.5$

Q₃ (Third Quartile)
 $i = 75/100 \times 8 = 6$
 $Q_3 = (122 + 125) / 2 = 123.5$

The centre isn't enough; we also need to know how spread out the data is. The range is simply the maximum minus the minimum. The interquartile range is more robust. In the trainer's example, eight sorted values run from one hundred six to one hundred twenty-nine, so the range is twenty-three. The first quartile, the twenty-fifth percentile, is one hundred eleven point five. The median, Q two, is one hundred eighteen point five. And Q three, the seventy-fifth percentile, is one hundred twenty-three point five. So the IQR, Q three minus Q one, is twelve: the spread of the middle half of the data.

Tip Range = max - min

Tip IQR = 123.5 - 111.5 = 12

Variance measures spread using every value

Calculate Deviations

VALUE	MEAN	DEVIATION ($X - \mu$)
48	50	-2
49	50	-1
50	50	0
51	50	+1
52	50	+2

If we add deviations: $-2 - 1 + 0 + 1 + 2 = 0 \rightarrow$ Always zero!

So, we cannot simply average deviations.

Variance measures spread using every value. Class A scored forty-eight to fifty-two, and Class B scored ten to ninety. Both have a mean of fifty, but Class B is far more spread out. Step one: subtract the mean from each value. For Class A, the deviations run from minus two to plus two. But if you add them up, you always get zero, so we can't simply average them.

Variance and standard deviation

Variance and standard deviation

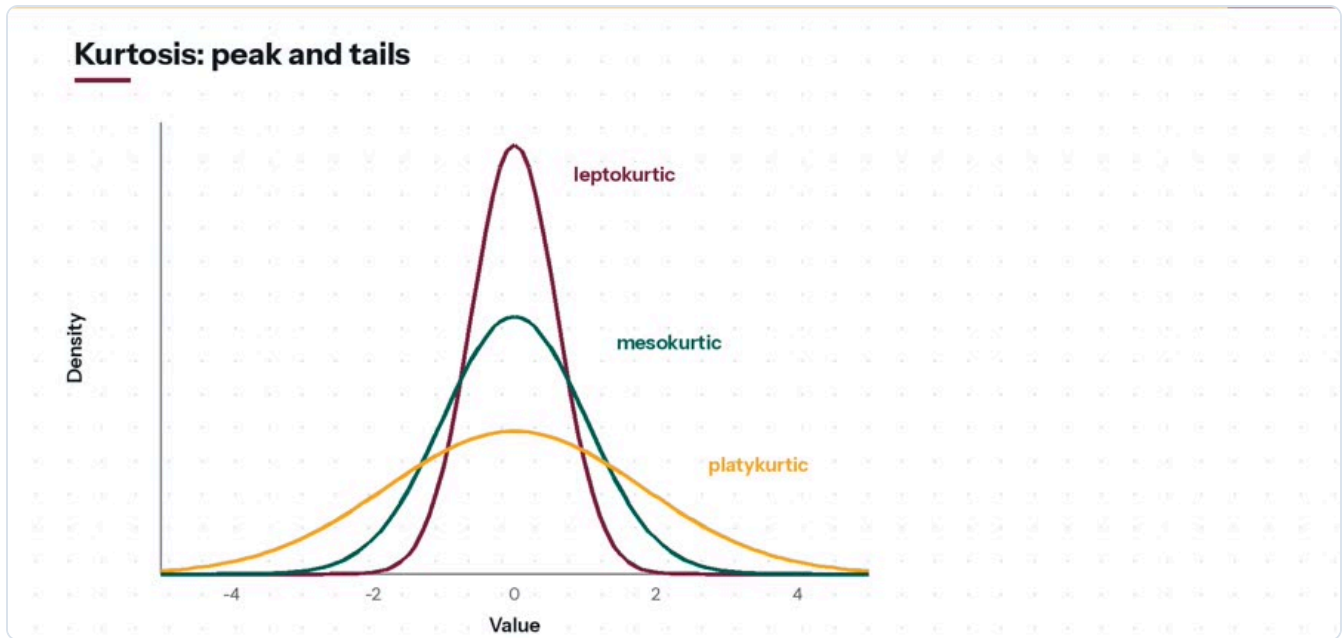
```
1 Class A    = 48, 49, 50, 51, 52
2 mean      = 250 / 5 = 50
3
4 deviation = -2, -1, 0, +1, +2
5 squared   = 4, 1, 0, 1, 4
6
7 variance  = 10 / 5 = 2
8 std dev   = sqrt(2) ≈ 1.41
9
```

```
10 Class B → variance 800, std dev ≈ 28.3
```

Same mean, much bigger spread.

Step two: square the deviations so negatives don't cancel out, and average them. That's the variance. For Class A, the squares add up to ten, so the variance is ten divided by five, which is two. Then take the square root to get back to the original units. That's the standard deviation, about one point four marks. Class B's standard deviation is about twenty-eight, which confirms how much more spread out it is.

Kurtosis: peak and tails



Finally, kurtosis describes how peaked a distribution is and how heavy its tails are. A leptokurtic distribution has a tall, narrow peak and heavier tails, so outliers are more likely. A mesokurtic distribution has the moderate shape of a normal bell curve. And a platykurtic distribution is flat and wide.

Key takeaways

- Mean, median, mode describe the centre
- Skewness shows which way the tail runs
- Range, IQR, variance, standard deviation
- Kurtosis describes peak and tails

Practice

1. For the data 6, 8, 8, 9, 11, 30 compute the mean, median and mode, then explain which best represents the data and why.
2. Compute the range and IQR for 2, 5, 7, 8, 10, 12, 15, 21.
3. Sketch a left-skewed, symmetric and right-skewed distribution and mark the mean and median on each.

Check your understanding

1. Which measure of centre is most affected by outliers?
2. In a right-skewed distribution, which is larger: mean or median?
3. What does the IQR measure?
4. Why do we square deviations when computing variance?

Answers at the back of this handbook.

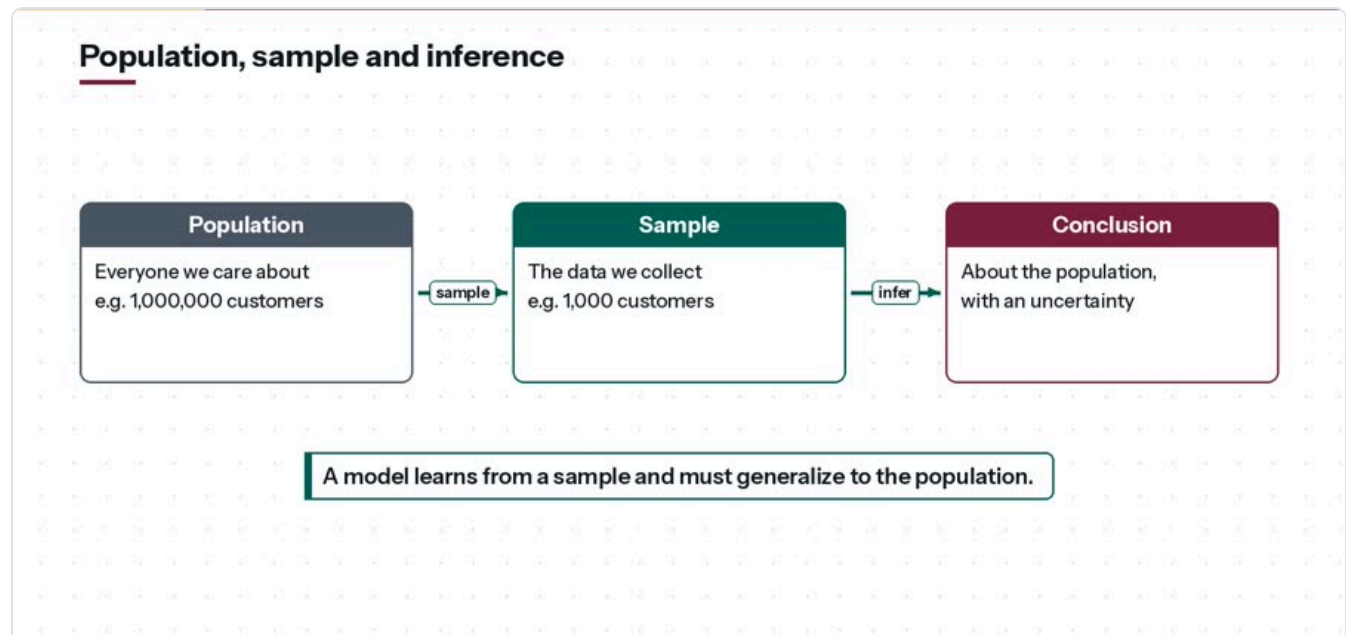
Probability Essentials for ML

Video: 08_AI-08_Probability_Essentials_for_ML.mp4 · 5:22 · Stage: Exploratory Data Analysis · Audience: Beginners in the AI & ML track who completed AI-07

Learning objectives

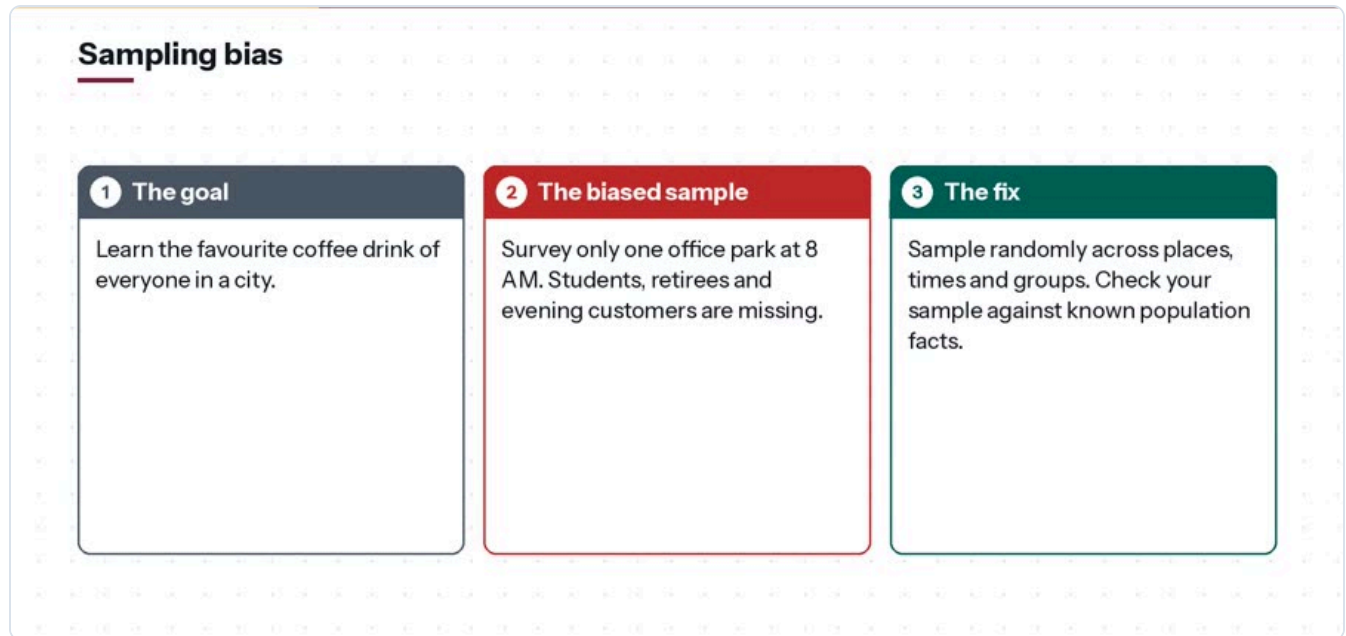
- Tell a population from a sample, and spot sampling bias
- Use classical, frequency and subjective probability
- Recognise binomial and Poisson problems

Population, sample and inference



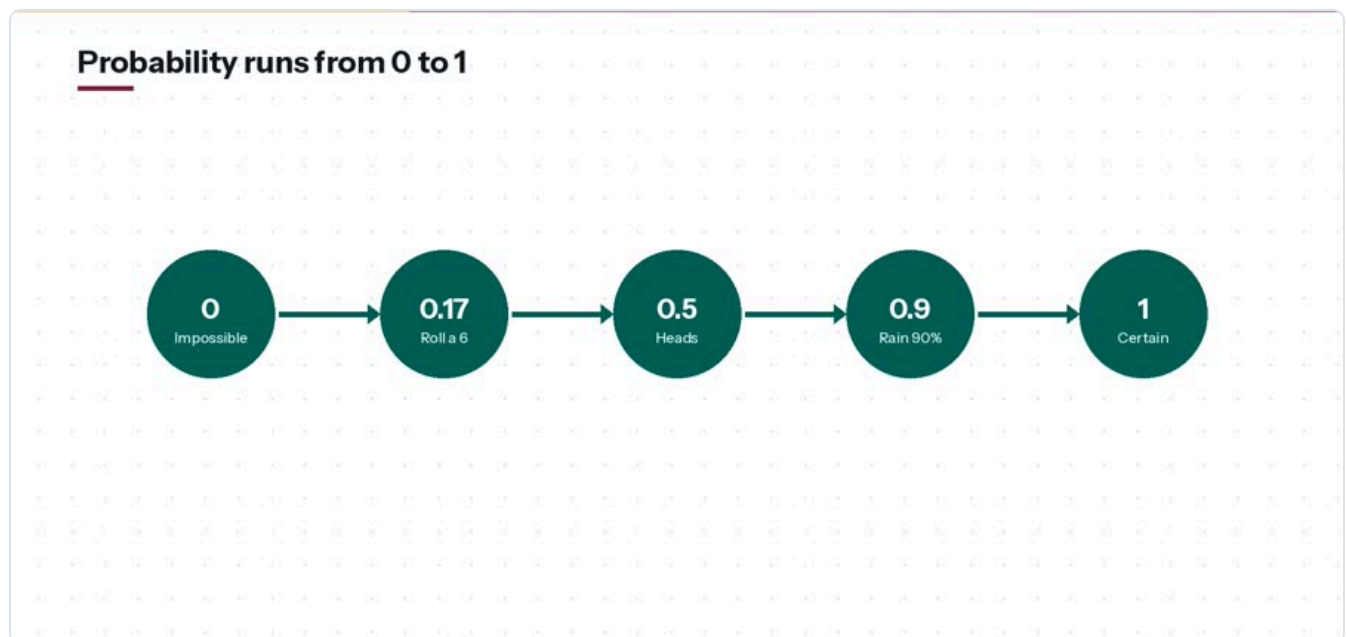
We almost never have data on everyone. The full group we care about is the population, say a million customers. We study a sample, perhaps a thousand of them, and use statistics to infer what's true for the whole population. Probability then tells us how uncertain those conclusions are. Training a model works the same way: it learns from a sample and must work on new data from the population.

Sampling bias



A sample is only useful if it represents the population. Suppose you want the favourite coffee drink of everyone in a city, but you only survey people outside one office park at eight in the morning. You'll hear mostly from commuters who want espresso, and miss students, retirees, and evening customers. That's sampling bias, and a model trained on biased data inherits the same blind spots.

A probability is a number between zero and one



A probability is a number between zero and one. Zero means impossible. One means certain. Rolling a six on a fair die has a probability of about point one seven. A fair coin landing heads is point five. And a forecast of ninety percent rain is point nine: likely, but still not certain.

Three ways to assign probability

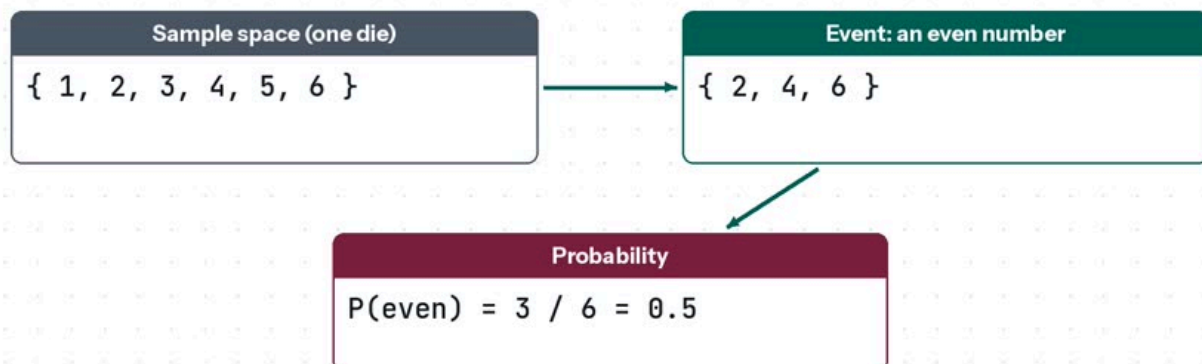
Three ways to assign probability

Approach	How	Example
Classical	Equally likely outcomes: favourable ÷ total	Rolling a 6: $1 \div 6$, about 0.17
Relative frequency	Observed history: times it happened ÷ trials	Rain on 30 of 100 similar days: 0.3
Subjective	Informed judgement; people may differ	Expert: 70% chance project ships on time

There are three ways to assign a probability. Classical probability applies when every outcome is equally likely: one favourable outcome out of six gives one sixth, and everyone gets the same answer. Relative frequency uses history: if it rained on thirty of the last hundred similar days, the probability is point three. And subjective probability is an informed judgement, like an expert estimating the chance a project finishes on time; different people may disagree.

Sample space and events

Sample space and events



An experiment has a sample space: every possible outcome. For one die, that's one to six. An event is any set of those outcomes. The event, an even number, contains two, four, and six. So its probability is three favourable outcomes out of six, which is one half.

Independent vs dependent events

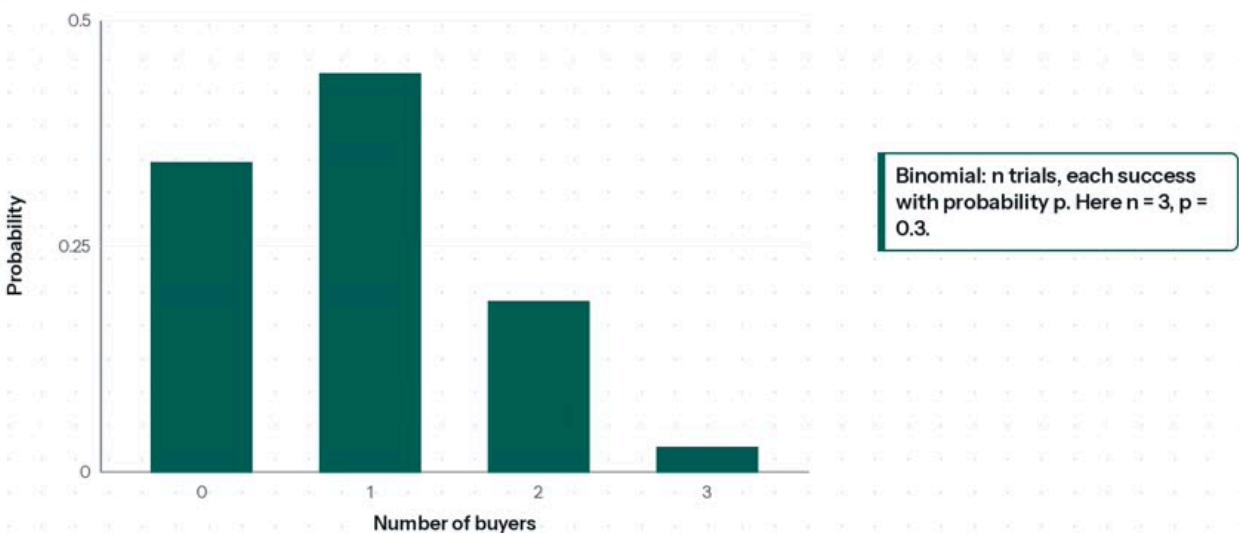
Independent vs dependent events

	Independent	Dependent
Meaning	One event doesn't change the other's chance	One event changes the other's chance
Example	Two coin tosses	Two cards drawn without replacement
Calculation	$P(A \text{ and } B) = P(A) \times P(B)$ $P(\text{two heads}) = 0.5 \times 0.5 = 0.25$	$P(2 \text{ aces}) = 4/52 \times 3/51$, about 0.0045

Two events are independent if one happening doesn't change the chance of the other, like two separate coin tosses. Then you can multiply: the chance of two heads is one half times one half, which is one quarter. Events are dependent when one affects the other, like drawing two cards without putting the first back. After an ace is drawn, there are fewer aces left.

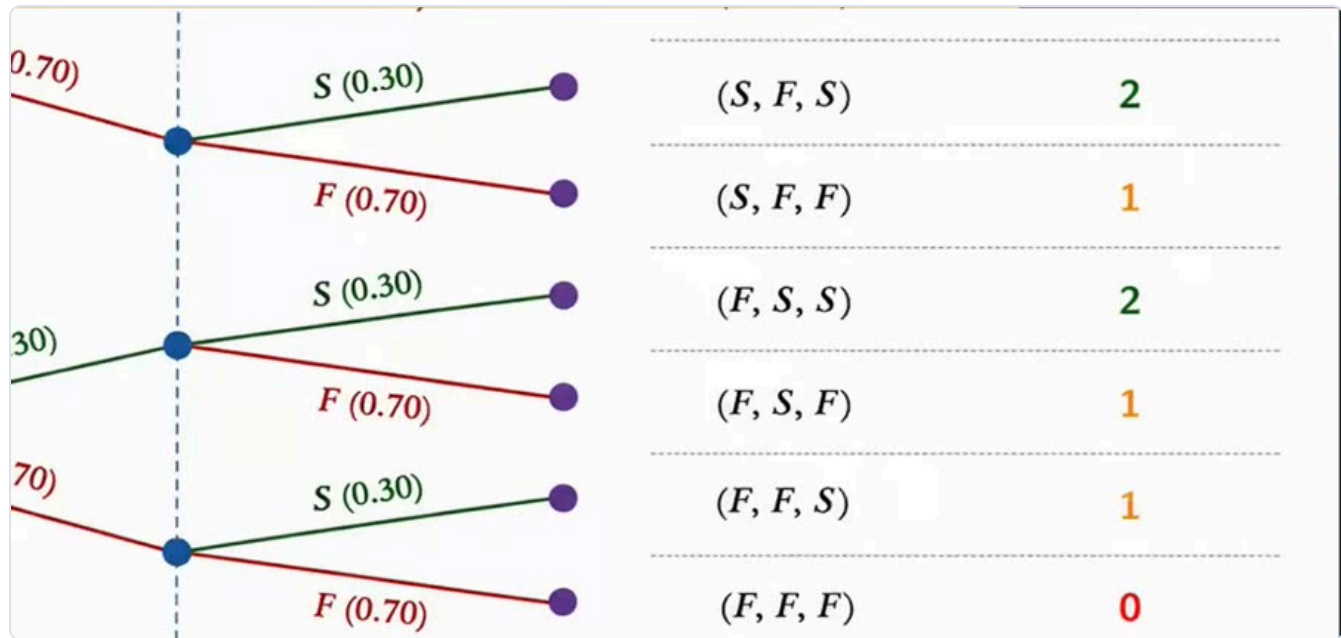
Binomial: how many of the next 3 customers buy? ($p = 0.3$)

Binomial: how many of the next 3 customers buy? ($p = 0.3$)



Some problems follow known patterns called distributions. The binomial distribution counts successes in a fixed number of yes or no trials. A shop owner knows thirty percent of visitors buy something. For the next three customers, the chance that none buy is about thirty-four percent, exactly one is forty-four percent, two is nineteen percent, and all three is under three percent.

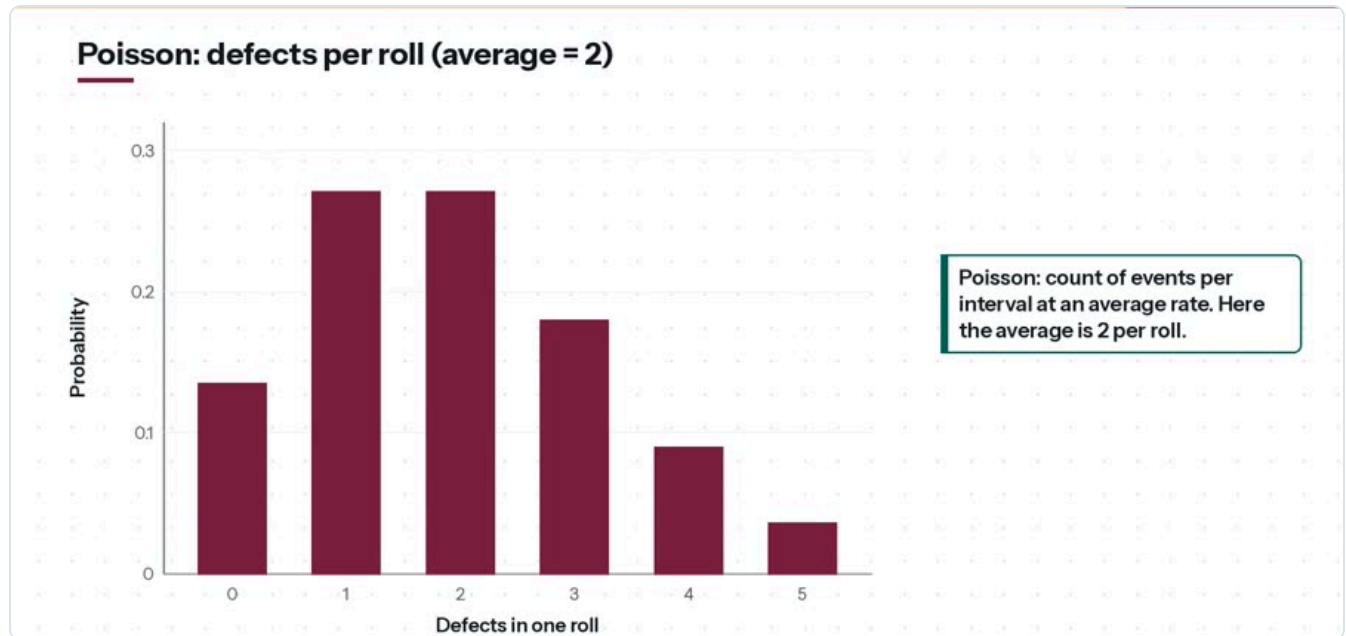
Here's the trainer's tree diagram for the same shop



Here's the trainer's tree diagram for the same shop. Each customer either buys, marked S, with probability point three, or doesn't, marked F, with probability point seven. Three customers give eight possible outcomes. Only the top path has all three buying. Exactly two buyers happens on three paths, such as S, S, F. Each of those paths has probability point three times point three times point seven, about point zero six three. So three paths give about nineteen percent. And the bottom path, where nobody buys, is point seven cubed, about thirty-four percent.

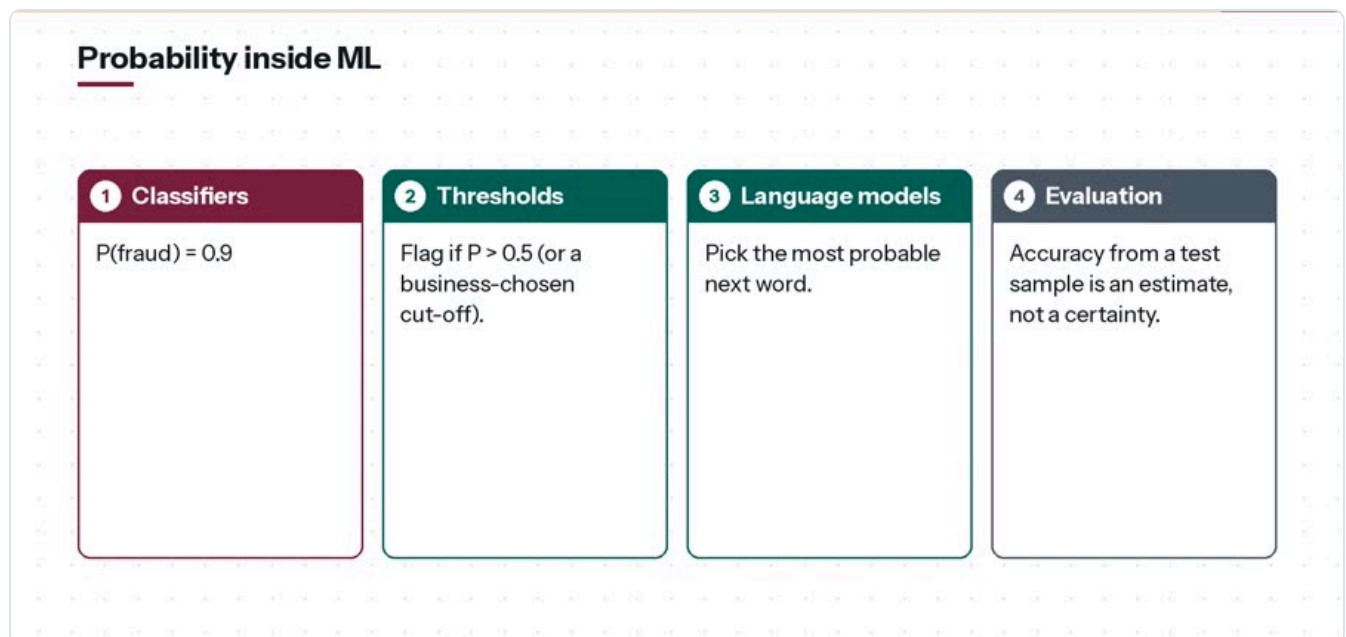
Tip One such path: $0.3 \times 0.3 \times 0.7 = 0.063$. Three paths: $3 \times 0.063 = 0.189$, about 19%.

Poisson: defects per roll (average = 2)



The Poisson distribution counts how many times an event happens in a fixed interval, when events occur independently at an average rate. Think of defects in a roll of fabric, or orders per minute. If a factory averages two defects per roll, the chance of zero defects is about fourteen percent, and one or two defects are each about twenty-seven percent.

Probability inside ML



Here's where probability shows up in machine learning. A classifier outputs a probability, such as point nine that a transaction is fraud. A threshold turns that into a decision. Language models predict the most probable next word. And every model's accuracy is itself an estimate from a sample, with uncertainty attached.

Key takeaways

- Samples represent populations; avoid bias
- Probability: classical, frequency, subjective
- Independent events: multiply
- Binomial and Poisson model counts

Practice

1. Describe a survey you might run at work and list two ways its sample could be biased, plus how to fix each.
2. A fair die is rolled. Write the sample space and compute $P(\text{number greater than 4})$.
3. Name one business count you could model with a Poisson distribution and one yes/no process you could model with a binomial distribution.

Check your understanding

1. What is a probability of 0 called?
2. Rain fell on 30 of 100 similar days. Which approach gives $P(\text{rain}) = 0.3$?
3. Two independent events each have probability 0.5. What is the chance both happen?
4. Which distribution counts events per interval at an average rate?

Answers at the back of this handbook.

Python Basics: Variables, Types & Input

Video: 09_AI-09_Python_Basics_Variables_Types_and_Input.mp4 · 5:00 · Stage: Environment & Tooling · Audience: Beginners in the AI & ML track; no programming experience needed

Learning objectives

- Open and run a notebook in Google Colab or Jupyter
- Write if / elif / else with correct indentation
- Use core types and convert input() text to numbers

Why Python for AI?

Why Python for AI?

1 Readable	2 Libraries	3 Community
Simple syntax; less code for the same idea.	NumPy, pandas (data) scikit-learn (ML) PyTorch, TensorFlow (deep learning)	Tutorials, answers and open-source examples for almost anything.

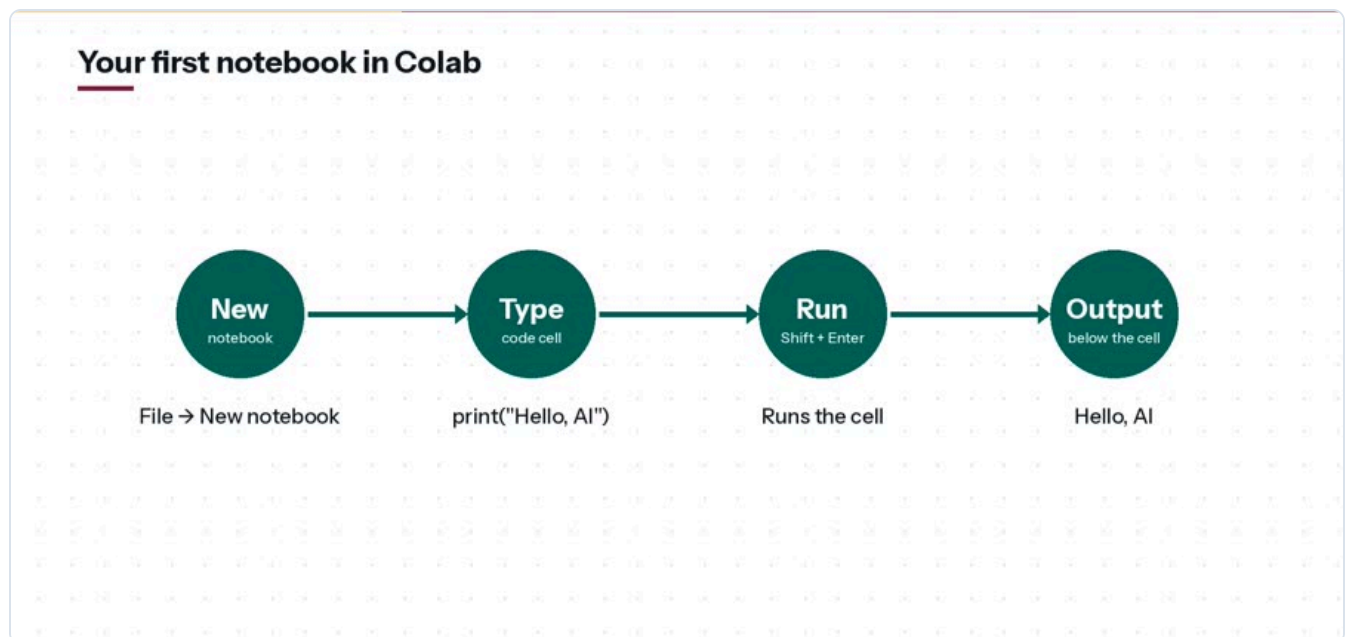
Why Python for AI? Its syntax reads almost like English, so you focus on the problem, not the punctuation. It has mature libraries for every step: NumPy and pandas for data, scikit-learn for classic machine learning, and PyTorch and TensorFlow for deep learning. And a huge community means answers and examples for almost anything.

Where to run Python

Where to run Python		
Option	Setup	Best for
Google Colab	Browser only: colab.research.google.com ; sign in with a Google account	Learning; no install; free GPU options
Jupyter Notebook	Install Anaconda (includes Python + Jupyter), then launch Jupyter	Working offline on your own machine
VS Code	Install Python + the Python extension	Larger projects and scripts

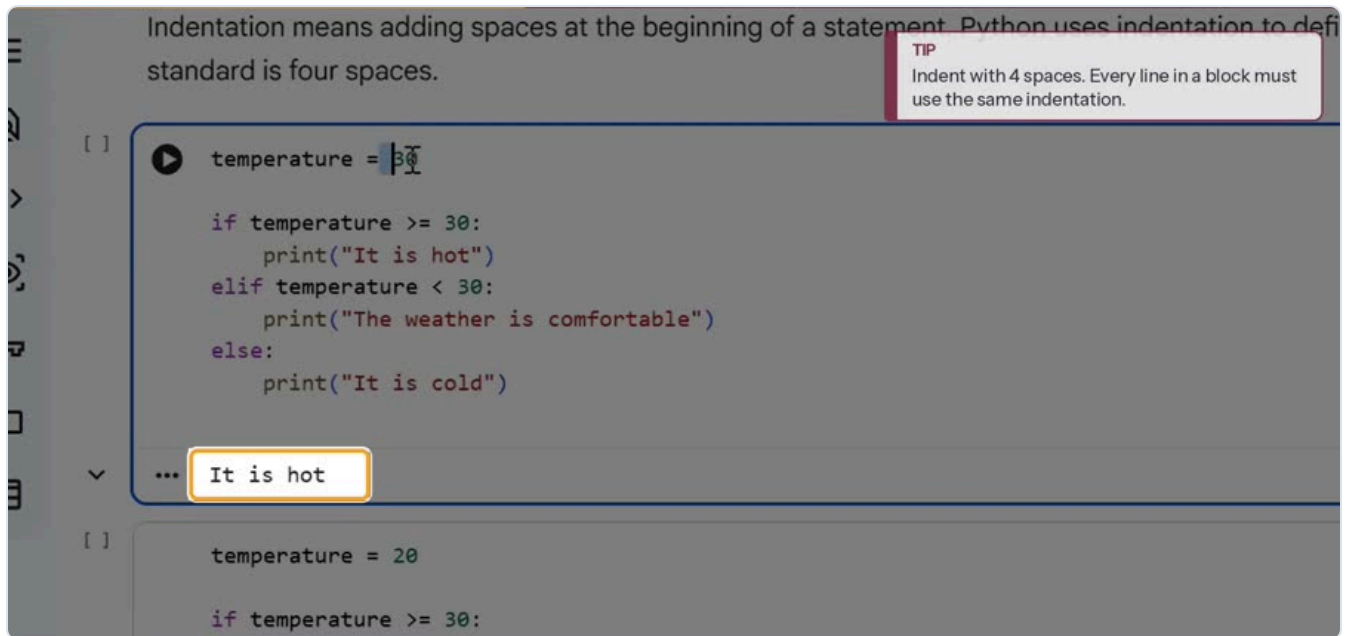
You'll need somewhere to run Python. The easiest is Google Colab: it runs in your browser, it's free, and there's nothing to install. Jupyter Notebook, usually installed through Anaconda, runs the same kind of notebook on your own computer. And VS Code with the Python extension suits larger projects. This course uses Colab, as the trainer did.

In Colab, it takes four steps



In Colab, it takes four steps. Open [colab dot research dot google dot com](https://colab.research.google.com) and create a new notebook. Type your code into a cell. Press Shift and Enter to run it. The output appears right below the cell, and you can add text cells to take notes between your code.

Here's the trainer's first notebook



Indentation means adding spaces at the beginning of a statement. Python uses indentation to define blocks of code. The standard is four spaces.

TIP
Indent with 4 spaces. Every line in a block must use the same indentation.

```
[ ] temperature = 30

    if temperature >= 30:
        print("It is hot")
    elif temperature < 30:
        print("The weather is comfortable")
    else:
        print("It is cold")

... It is hot

[ ] temperature = 20

    if temperature >= 30:
```

Here's the trainer's first notebook. The variable temperature is set to thirty. Then an if statement checks it: if temperature is at least thirty, print it is hot; else if it's below thirty, print the weather is comfortable; otherwise, print it is cold. Notice the indentation. Python uses spaces, four by convention, to mark which lines belong to each branch. The output says, it is hot.

Tip Indent with 4 spaces. Every line in a block must use the same indentation.

Remove the indentation from the print line and Python refuses to run



```
[2] ✓ 0s else:
    print("The weather is comfortable")

The weather is comfortable

[3] ! 0s temperature = 30

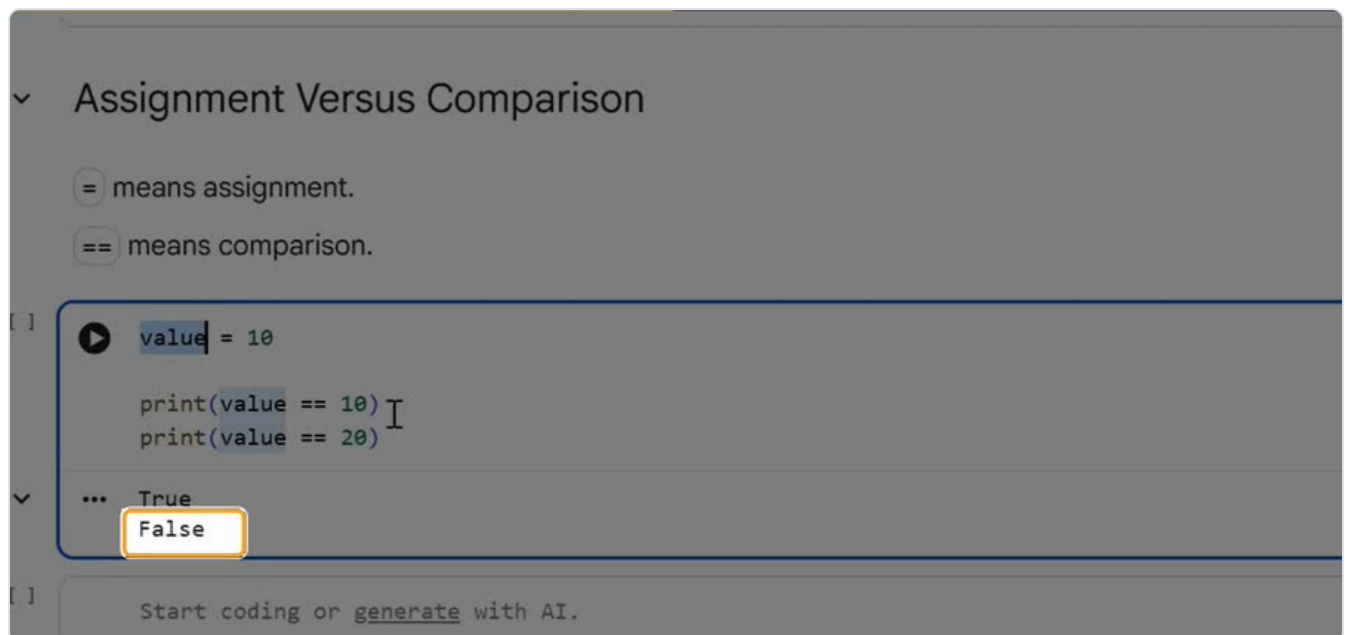
if temperature >= 30:
    print("It is hot")

... File "/tmp/ipykernel_1309/315900698.py", line 4
    print("It is hot")
    ^
IndentationError: expected an indented block after 'if' statement on line 3

Next steps: Explain error
```

Remove the indentation from the print line and Python refuses to run. The error says, expected an indented block after the if statement. Indentation isn't just style in Python: it's part of the syntax.

Don't mix up one equals sign and two



```
Assignment Versus Comparison

= means assignment.
== means comparison.

[ ] ▶ value = 10

    print(value == 10)
    print(value == 20)

... True
    False

[ ] Start coding or generate with AI.
```

Don't mix up one equals sign and two. A single equals sign assigns a value: value equals ten stores ten. A double equals sign compares: value double equals ten asks a question, and the answer is True. Value double equals twenty is False.

Naming rules and dynamic typing

Naming rules and dynamic typing

1 Allowed
student_name
average_score2
_total

2 Not allowed
2score (starts with a number)
my score (space)
if, for, True (keywords)

3 Dynamic typing
x = 10 # int
x = "ten" # now str
No type declarations.

Names you create are called identifiers. They can contain letters, numbers, and underscores, but can't start with a number, can't contain spaces or symbols, and can't be a reserved keyword like if, else, for, while, or True. Python is case sensitive, so Age and age are different variables. And Python is dynamically typed: you never declare a type, the value decides it.

Multiple assignment and print options

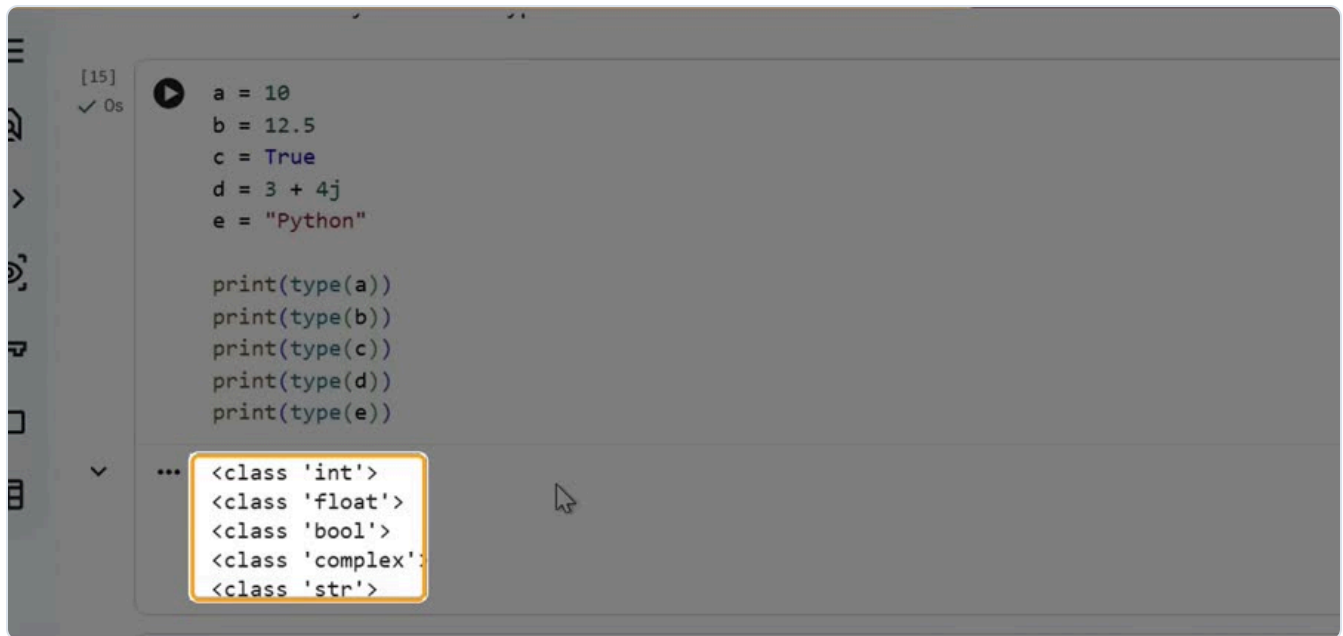
Multiple assignment and print options

```
1 name, age, score = "Asha", 34, 91.5
2 a = b = c = 10
3
4 x, y = 10, 20
5 x, y = y, x          # swap → 20 10
6
7 print(1, 2, 3, sep="-")      # 1-2-3
8 print("Loading", end="...") # Loading...
```

sep = between items; end = after the line (default is a new line).

Here are a few handy shortcuts. You can assign several variables in one line, or give them all the same value. You can swap two values without a temporary variable. And print takes options: sep changes what goes between items, and end changes what comes after the line.

Python has five core types you'll use constantly

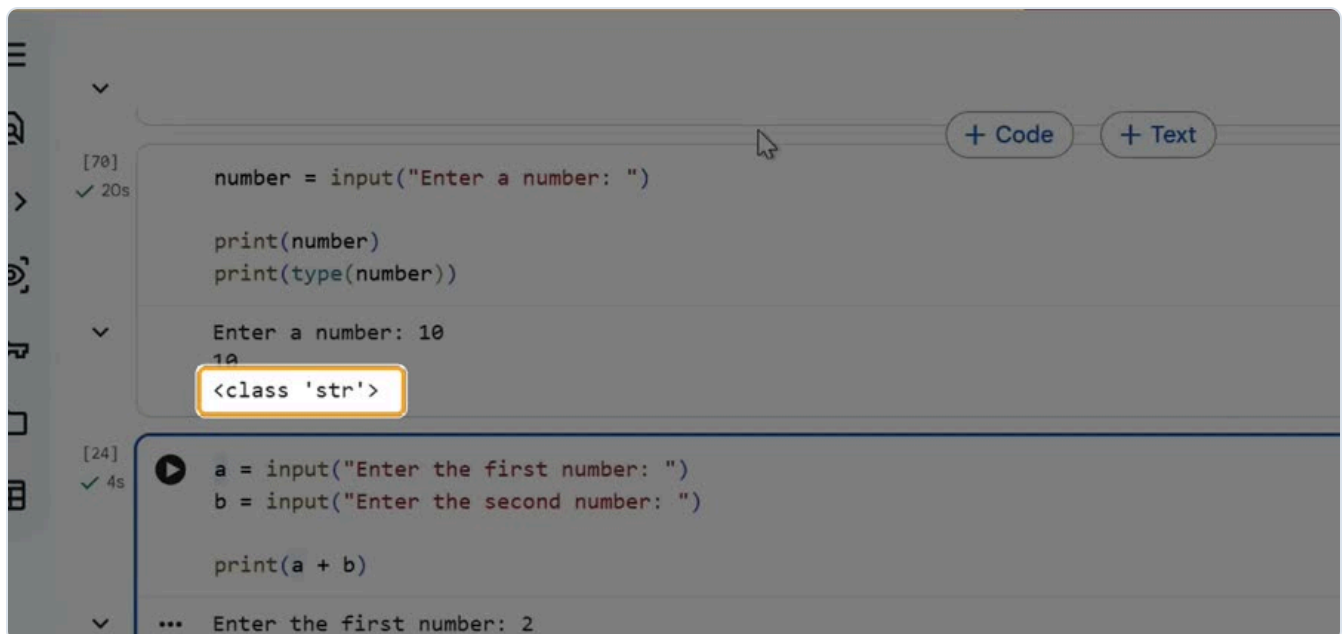


The screenshot shows a Python IDE with a code editor and a console. The code in the editor defines five variables: `a = 10`, `b = 12.5`, `c = True`, `d = 3 + 4j`, and `e = "Python"`. Below these, it uses `print(type(x))` to display the type of each variable. The console output shows the corresponding Python classes: `<class 'int'>`, `<class 'float'>`, `<class 'bool'>`, `<class 'complex'>`, and `<class 'str'>`. A yellow box highlights the first five lines of the console output.

```
[15] ✓ 0s  
a = 10  
b = 12.5  
c = True  
d = 3 + 4j  
e = "Python"  
  
print(type(a))  
print(type(b))  
print(type(c))  
print(type(d))  
print(type(e))  
  
... <class 'int'>  
    <class 'float'>  
    <class 'bool'>  
    <class 'complex'>  
    <class 'str'>
```

Python has five core types you'll use constantly. Ten is an int, a whole number. Twelve point five is a float. True is a bool. Three plus four j is a complex number. And Python in quotes is a str, a string of text. The type function tells you which is which.

Here's a classic beginner trap



The screenshot shows a Python IDE with two code snippets. The first snippet uses `input()` to get a number from the user, prints it, and shows its type. The second snippet asks for two numbers and adds them. The console shows the user entering '10' and the output being `<class 'str'>`, which is highlighted with a yellow box. This demonstrates that `input()` always returns a string, even for numeric input.

```
[70] ✓ 20s  
number = input("Enter a number: ")  
print(number)  
print(type(number))  
  
Enter a number: 10  
10  
<class 'str'>  
  
[24] ✓ 4s  
a = input("Enter the first number: ")  
b = input("Enter the second number: ")  
print(a + b)  
  
... Enter the first number: 2
```

Here's a classic beginner trap. The input function always returns a string, even when the user types a number. The trainer enters ten, and type shows class str.

So adding two inputs joins the text instead of doing maths

```
[70] ✓ 20s
print(number)
print(type(number))

Enter a number: 10
10
<class 'str'>

[71] ✓ 16s
a = input("Enter the first number: ")
b = input("Enter the second number: ")

print(a + b)

... Enter the first number: 5
Enter the second number: 6
56

[25] ✓ 3s
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
```

TIP
int("5") → 5 float("2.5") → 2.5 str(42) → "42"

So adding two inputs joins the text instead of doing maths. Five and six give fifty-six. The fix is to convert: wrap input in int, or float for decimals, and the same code now adds the numbers properly.

Tip int("5") → 5 float("2.5") → 2.5 str(42) → "42"

Strings are immutable: once created, they can't be changed in place

```
[86] ! 0s
text = "Python"

text[0] = "J"

-----
TypeError                                Traceback (most recent call last)
/tmp/ipykernel_3111/211138803.py in <cell line: 0>()
      1 text = "Python"
      2
----> 3 text[0] = "J"

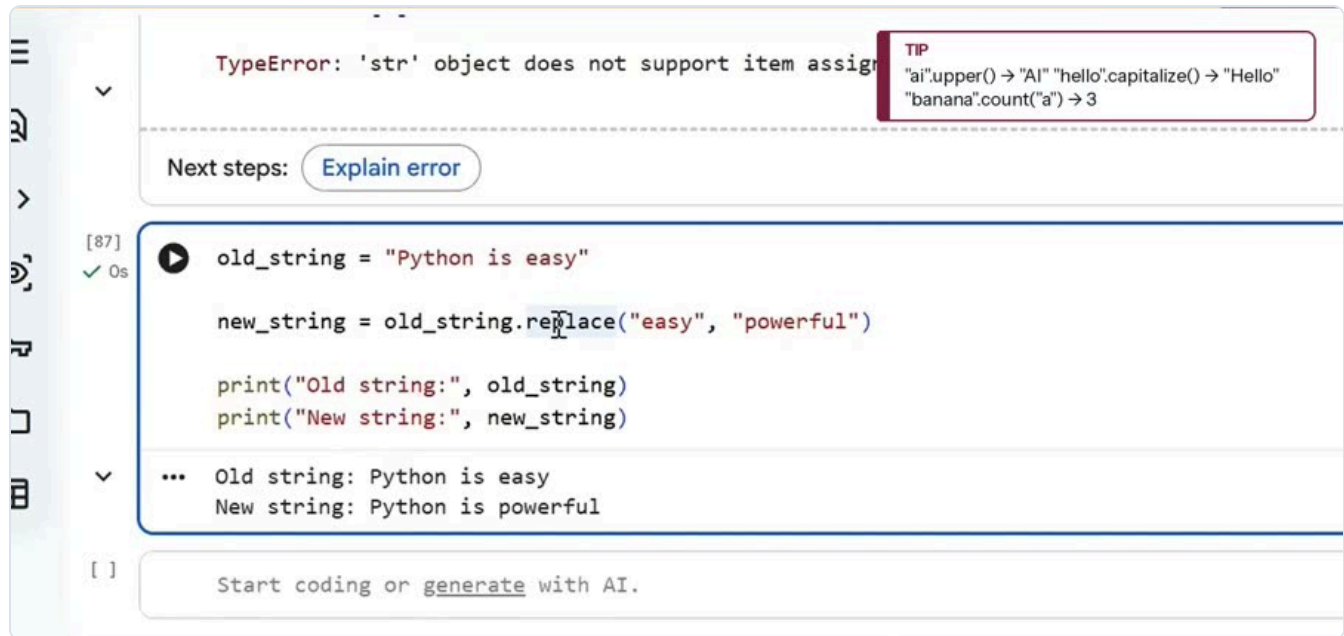
TypeError: 'str' object does not support item assignment

Next steps: Explain error

[46] ✓ 0s
p1d_string = "Python is easy"
```

Strings are immutable: once created, they can't be changed in place. Trying to replace the first letter of Python with J raises a TypeError: str object does not support item assignment.

Instead, string methods return a new string



The screenshot shows a Google Colab notebook interface. At the top, a red error message states: `TypeError: 'str' object does not support item assignment`. To the right of this message is a **TIP** box containing the following examples: `"ai".upper() → "AI"`, `"hello".capitalize() → "Hello"`, and `"banana".count("a") → 3`. Below the error message, a button labeled "Next steps: Explain error" is visible. The main code cell, labeled [87] and marked as successful with a green checkmark and "0s", contains the following Python code:

```
old_string = "Python is easy"
new_string = old_string.replace("easy", "powerful")
print("Old string:", old_string)
print("New string:", new_string)
```

Below the code cell, the output is displayed:

```
... Old string: Python is easy
New string: Python is powerful
```

At the bottom of the interface, there is a button that says "Start coding or generate with AI."

Instead, string methods return a new string. Replace swaps easy for powerful, and the original string is untouched. Other useful methods include upper, capitalize, and count.

Tip `"ai".upper() → "AI"` `"hello".capitalize() → "Hello"` `"banana".count("a") → 3`

Key takeaways

- Run Python free in Google Colab
- Indentation defines blocks; `=` assigns, `==` compares
- `int`, `float`, `bool`, `complex`, `str`
- `input()` returns a string: convert it

Practice

1. Open a new Colab notebook and write an `if` / `elif` / `else` block that prints a message for three temperature ranges. Then break the indentation on purpose and read the error.
2. Ask the user for two numbers with `input()`, add them without conversion, then fix it with `int()` and `float()`.
3. Create one variable of each core type and print `type()` for each.

Check your understanding

1. What does Python use to mark a block of code?
2. What is the difference between = and ==?
3. What type does input() always return?
4. What happens if you try `text[0] = 'J'` on a string?

Answers at the back of this handbook.

Python Collections: Lists, Tuples, Sets & Dictionaries

Video: 10_AI-10_Python_Collections.mp4 · 4:42 · Stage: Python Foundations · Audience: Beginners who completed AI-09 Python Basics

Learning objectives

- Create and change lists and tuples
- Choose list, tuple, set or dictionary
- Use dictionaries for key and value records

Python's built-in collections

Python's built-in collections

1 List	2 Tuple	3 Set	4 Dictionary
<code>[10, 20, 30]</code>	<code>(10, 20, 30)</code>	<code>{10, 20, 30}</code>	<code>{"id": 101}</code>
Ordered, changeable, allows duplicates.	Ordered, cannot be changed.	Unique values, no order, no indexing.	Values stored under keys.

Python has four built-in collections, plus strings, which behave like a sequence of characters. A list is ordered and changeable. A tuple is ordered but can't be changed. A set keeps only unique values, in no particular order. And a dictionary stores values under keys, like a labelled record.

Let's start with lists in the trainer's Colab notebook

List Properties

1. A list is ordered
2. A list supports indexing
3. A list allows duplicate values
4. A list can contain different data types
5. A list is mutable
6. A list is created using square brackets `[]`

[]

```
numbers = [10, 20, 30, 40, 50]
```

Let's start with lists in the trainer's Colab notebook. A list stores multiple values in one variable. It's ordered, it supports indexing, it allows duplicates, it can mix data types, it's mutable, and you create it with square brackets.

Here are four lists: whole numbers, decimals, names, and a mixed li...

```
integer_list = [10, 20, 30]

float_list = [10.5, 20.5, 30.5]

name_list = ["Asha", "Ravi", "Sara"]

mixed_list = [101, "Asha", 85.5, True, None]

print(integer_list)
print(float_list)
print(name_list)
print(mixed_list)]
```

```
... [10, 20, 30]
     [10.5, 20.5, 30.5]
     ['Asha', 'Ravi', 'Sara']
     [101, 'Asha', 85.5, True, None]
```

Here are four lists: whole numbers, decimals, names, and a mixed list holding a number, a string, a decimal, True, and None. Python happily keeps different types side by side, and printing each list shows its values in the same order you wrote them.

Every element has a position, called its index, starting from zero

Negative Indexing

✓ Negative Indexing

Negative indexing accesses elements from the right side.

Element	10	20	30	40	50
Negative index	-5	-4	-3	-2	-1

Index **-1** represents the last element.

```
[ ] numbers = [10, 20, 30, 40, 50]
```

Every element has a position, called its index, starting from zero. Python also counts backwards: minus one is the last element, minus two the second to last, and so on. Negative indexing is a quick way to grab the end of a list without knowing its length.

Tip numbers[0] is the first element. numbers[-1] is the last.

Slicing takes several elements at once, using start, colon, stop, c...

```
### Code Cell
numbers = [10, 20, 30, 40, 50, 60]

print("Index 1 to 3:", numbers[1:4])
print("Beginning to index 2:", numbers[:3])
print("Index 3 to end:", numbers[3:])
print("Every second value:", numbers[::2])
print("Reversed list:", numbers[::-1])
```

```
... Index 1 to 3: [20, 30, 40]
Beginning to index 2: [10, 20, 30]
Index 3 to end: [40, 50, 60]
Every second value: [10, 30, 50]
Reversed list: [60, 50, 40, 30, 20, 10]
```

TIP
list[start:stop:step]. Stop is excluded: [1:4] gives indexes 1, 2 and 3.

Slicing takes several elements at once, using start, colon, stop, colon, step. The stop index is excluded, so numbers one to four returns twenty, thirty, forty. Leave out start or stop to go from the beginning or to the end, use a step of two for every second value, and a step of minus one reverses the list.

Tip list[start:stop:step]. Stop is excluded: [1:4] gives indexes 1, 2 and 3.

Lists are mutable

`append()` adds its argument as one element.

`extend()` adds the individual elements from another collection.

[38]
✓ Os

```
list1 = [10, 20]
list1.append([30, 40])

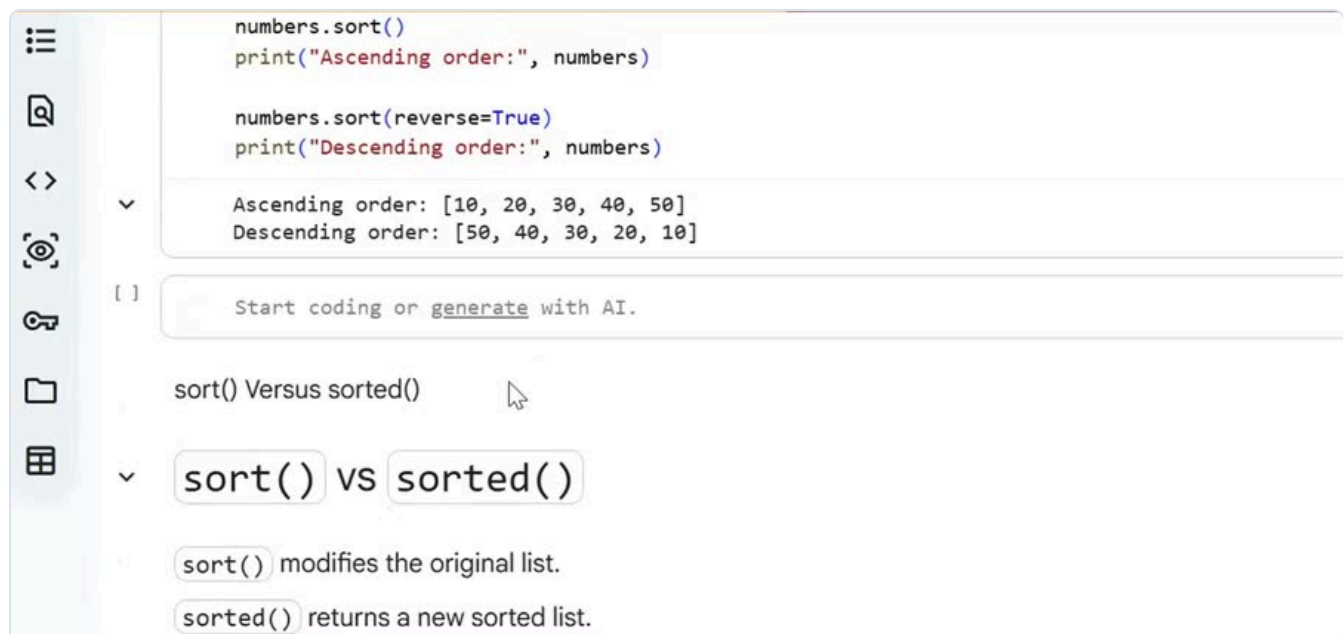
list2 = [10, 20]
list2.extend([30, 40])

print("Using append:", list1)
print("Using extend:", list2)
```

... Using append: [10, 20, [30, 40]]
Using extend: [10, 20, 30, 40]

Lists are mutable. Assigning marks zero equals seventy-five replaces the first value in place. To add items, `append` adds its argument as one element, while `extend` adds each element from another collection. Notice the difference: `append` puts a whole list inside the list, `extend` merges the values.

To remove items, `pop` takes out an element by index and hands it bac...



The screenshot shows a code editor with the following code:

```
numbers.sort()
print("Ascending order:", numbers)

numbers.sort(reverse=True)
print("Descending order:", numbers)
```

The output of the code is:

```
Ascending order: [10, 20, 30, 40, 50]
Descending order: [50, 40, 30, 20, 10]
```

The sidebar on the left shows a search result for `sort() vs sorted()` with the following text:

- `sort()` modifies the original list.
- `sorted()` returns a new sorted list.

To remove items, `pop` takes out an element by index and hands it back to you. `Pop` with one removes twenty. For ordering, `sort` changes the original list, and `sort` with `reverse` equals `True` sorts in descending order. The `sorted` function instead returns a new list and leaves the original alone.

A tuple looks like a list in round brackets, but it's immutable: af...

```
[43] ✓ 0s empty_tuple = ()

wrong_single_tuple = (10)

correct_single_tuple = (10,)

print("Empty tuple type:", type(empty_tuple))

print("(10) type:", type(wrong_single_tuple))

print("(10,) type:", type(correct_single_tuple))

... Empty tuple type: <class 'tuple'>
(10) type: <class 'int'>
(10,) type: <class 'tuple'>
```

TIP
(10) is an int. (10,) is a tuple.

A tuple looks like a list in round brackets, but it's immutable: after creating it you can't change, add, or remove elements. That makes tuples ideal for fixed data such as G P S coordinates. One gotcha: ten in brackets is just an integer. A single-element tuple needs a trailing comma.

Tip (10) is an int. (10,) is a tuple.

In the next session, the trainer moves on to sets

```

Set Indexing

A set is unordered.

Therefore, a set does not support positional indexing.

[98]
✓ 0s
items = {10, 20, 30}

print(items)

# The following statement produces an error:
# print(items[0])

... {10, 20, 30}
```

In the next session, the trainer moves on to sets. A set is an unordered collection of unique elements. Duplicates are dropped automatically, and because there's no order, a set doesn't support indexing. Asking for items zero would raise an error.

Watch out for empty curly braces

```

Empty curly braces create a dictionary, not a set.

Use set() to create an empty set.

[76]
✓ 0s
a = {}
b = set()

print("Value of a:", a)
print("Type of a:", type(a))

print("Value of b:", b)
print("Type of b:", type(b))

... Value of a: {}
Type of a: <class 'dict'>
Value of b: set()
Type of b: <class 'set'>
```

Watch out for empty curly braces. They create an empty dictionary, not a set. To create an empty set, call set with no arguments. The output confirms it: a is a dictionary, and b is a set.

A dictionary stores key and value pairs, like an employee record wi...

```
record = {  
    "name": "sha",  
    "age": 24,  
    "salary": 8500.5  
}  
  
print("Before modification:", record)  
  
record["salary"] = 9000.0  
  
print("After modification:", record)
```

... Before modification: {'name': 'sha', 'age': 24, 'salary': 8500.5}
After modification: {'name': 'sha', 'age': 24, 'salary': 9000.0}

A dictionary stores key and value pairs, like an employee record with an ID, a name, and a city. Keys must be unique, and you look values up by key, not by position. Dictionaries are mutable too: assigning record salary equals nine thousand updates that one value.

Tip Keys must be unique and unchangeable (strings, numbers, tuples). Values can be anything.

Which collection?

Which collection?

Collection	Ordered	Changeable	Duplicates	Use it for
List []	Yes	Yes	Yes	A sequence you'll change, e.g. readings
Tuple ()	Yes	No	Yes	Fixed values, e.g. coordinates
Set { }	No	Yes	No	Removing duplicates, fast membership tests
Dict {k: v}	Insertion order	Yes	Keys: no	Labelled records, e.g. one row of data

So which collection should you choose? Use a list for an ordered sequence you'll change, like a column of readings. Use a tuple for fixed values, like coordinates. Use a set to remove duplicates or test membership quickly. And use a dictionary for labelled records, like one row of a dataset.

Key takeaways

- Lists: ordered, changeable, index and slice
- Tuples: ordered but fixed
- Sets: unique values, no order
- Dictionaries: values under unique keys

Practice

1. Create a list of five temperatures. Print the first, the last (with a negative index) and the middle three (with a slice).
2. Append a new reading, then sort the list in descending order. Try `sorted()` and compare the original list.
3. Store one student's record as a dictionary with name, age and score. Update the score and print the record.
4. Turn a list with repeated values into a set and count the unique values.

Check your understanding

1. What does `numbers[-1]` return?
2. What does `numbers[1:4]` return for `[10, 20, 30, 40, 50]`?
3. How do you create a single-element tuple?
4. What do empty curly braces `{}` create?
5. What is the difference between `append` and `extend`?

Answers at the back of this handbook.

Writing Functions

Video: 11_AI-11_Writing_Functions.mp4 · 4:19 · Stage: Python Foundations · Audience: Beginners who completed AI-10 Python Collections

Learning objectives

- Define functions with def, parameters and a body
- Return values, and know print from return
- Explain local and global scope

Why bother with functions? Suppose ten lines of code perform a calc...

Why Do We Need Functions?

Suppose we write ten lines of code to perform a calculation.

If we need the same calculation again, we should not repeat those ten lines.

We can place the calculation inside a function and call the function whenever required.

DRY Principle

DRY means:

Do Not Repeat Yourself

Why bother with functions? Suppose ten lines of code perform a calculation. If you need that calculation again, you shouldn't copy those ten lines. Put them inside a function and call it whenever you need it. That's the DRY principle: do not repeat yourself.

Parts of a function

Parts of a function

```
1 def calculate_total(price, quantity):
2     """Return the cost of an order."""
3     total = price * quantity
4     return total
5
6 bill = calculate_total(25, 4)
7 print(bill)  # 100
```

return sends the result back to the caller.

A function has a few parts. The `def` keyword starts it, followed by the function's name, and parameters in round brackets, then a colon. An optional docstring describes what it does. The indented body does the work, and an optional `return` statement sends a result back.

Here's the simplest example in the notebook

Creating and Calling a Function

```
def f1():                # Define a function called f1
    print("Hello")        # Display Hello when the function executes

f1()                      # Call the function and execute its body
```

... Hello

Calling a Function Multiple Times

```
def f1():                # Define the function only once
    print("Hello")        # Display Hello whenever the function is called
```

Here's the simplest example in the notebook. `def f one, colon, then an indented print of Hello.` Defining a function doesn't run it. The code runs only when you call it by name with brackets, and then Hello appears.

Parameters make a function reusable

```
# Square of a Number
def square(number):
    result = number * number
    print("Square of", number, "is:", result)

square(10)
square(5)
square(4)
```

TIP
Parameter: the name in the definition.
Argument: the value you pass in the call.

Define the function
Multiply the number by itself
Display the original number and result

Pass 10 to the function
Pass 5 to the function
Pass 4 to the function

Square of 10 is: 100
Square of 5 is: 25
Square of 4 is: 16

Parameters make a function reusable. Square takes one parameter called number, multiplies it by itself, and prints the result. The values you pass when calling, ten, five and four, are the arguments. One definition, three different answers.

Tip Parameter: the name in the definition. Argument: the value you pass in the call.

A function can take several parameters

```
check_even_or_odd(9) # Call the function using an odd number

4 is even
9 is odd
```

TIP
add("Py", "thon") → Sum is: Python

```
# Function with Multiple Parameters
def add(a, b):
    result = a + b
    print("Sum is:", result)

# Define a function with two parameters
# Add the values received through a and b
# Display the result

add(20, 30) # Assign 20 to a and 30 to b

add(10.1, 20) # Add a floating-point value and an integer

add(20.3, 20.1) # Add two floating-point values

# Join two strings using the plus operator
```

A function can take several parameters. Add receives two values and prints their sum. Arguments are matched by position, so twenty goes to the first parameter and thirty to the second. Because Python works out types at runtime, the same function adds decimals, and even joins two strings with the plus operator.

Tip add("Py", "thon") → Sum is: Python

Now a common trap

1. Stored in a variable
2. Reused later
3. Used in another calculation
4. Passed to another function

```
def add(a, b):  
    result = a + b  
    print("Sum is:", result)  
  
value = add(10, 20)  
  
print("Stored value:", value)
```

... Sum is: 30
Stored value: None

Function That Returns

Now a common trap. This version of `add` prints the sum inside the function, and doesn't return anything. When we store the call in `value` and print it, we get `None`. Printing shows a result on screen, but it doesn't hand anything back to the caller.

With `return`, the function sends the sum back to the calling place

```
# Storing and Reusing a Returned Value  
def add(a, b):  
    return a + b  
  
result = add(10, 20)  
print("Sum is:", result)  
  
final_result = result * 5  
print("Final result:", final_result)
```

... Sum is: 30
Final result: 150

Start coding or [generate](#) with AI.

With `return`, the function sends the sum back to the calling place. Now `result` holds thirty, and you can reuse it: multiplying by five gives a final result of one hundred and fifty. Returning values is how functions chain together in a data pipeline.

print() vs return

print() vs return

	print()	return
Does	Displays a value on screen	Sends a value to the caller
Caller gets	None	The returned value
Stops function?	No	Yes, return ends it
Use for	Checking output	Reusable results, pipelines

Here's the difference side by side. Print displays a value, mainly for output, and the caller receives None. Return sends the value to the caller, ends the function, and lets you store or reuse the result. In data work, prefer return, and print only to check your results.

Chaining functions with return

Chaining functions with return

```
1 def average(total, count):
2     return total / count
3
4 def grade(score):
5     if score < 50:
6         return "Fail"
7     return "Pass"
8
9 result = grade(average(270, 3))
10 print(result) # Pass
```

average(270, 3) returns 90.0, which becomes grade's argument.

Here's how return lets functions work together. Average returns the mean of a total and a count. Grade takes a score and returns Pass or Fail. Because both functions return values, you can feed one straight into the other: two hundred and seventy over three is ninety, which grades as Pass. Small functions that return values are easy to test and reuse.

Next, scope: where a variable lives

```
Value of b: 20

# Local Variable Access Error
def f1():                                # Define the function
    a = 10                                # Create a local variable inside the function
    print(a)                             # Accessing a here is valid

f1()                                      # Call the function

#print(a)                                # This produces NameError because a is local to f1

10

#Global Variables
```

Next, scope: where a variable lives. A variable created inside a function is local. Here `a` equals ten exists only while `f1` runs. Printing `a` inside works, but printing it outside the function would raise a `NameError`, so the trainer commented that line out.

A variable created outside any function is global, and every functi...

The `global` keyword

The `global` keyword allows a function to modify a global variable.

```
a = 10 # Create a global variable with the value 10

def f1(): # Define the first function
    global a # Tell Python to use the global variable a
    a = 99 # Change the global value from 10 to 99
    print("Inside f1:", a) # Display the modified global value

def f2(): # Define the second function
```

TIP
Prefer passing values in as arguments and returning results over changing globals.

A variable created outside any function is global, and every function can read it. To change a global inside a function, declare it with the `global` keyword first. Here, the `global` statement, followed by the new value, changes the global from ten to ninety-nine. Use it sparingly: returning values is usually clearer.

Tip Prefer passing values in as arguments and returning results over changing globals.

Key takeaways

- Functions keep code DRY
- `def`, name, parameters, indented body
- `return` hands back a result; `print` only displays
- Variables inside a function are local

Practice

1. Write a function `celsius_to_fahrenheit(c)` that returns $c * 9 / 5 + 32$. Call it for 0, 25 and 100.
2. Rewrite it to print instead of return, store the call in a variable and print the variable.
Explain the None you see.
3. Write a function that takes a list of numbers and returns their average. Use it inside another calculation.
4. Create a global counter and a function that increases it with the global keyword. Then rewrite it to return the new value instead.
5. Write two small functions that return values, then call one inside the other (like `grade(average(...))`).

Check your understanding

1. Which keyword starts a function definition?
2. What is the difference between a parameter and an argument?
3. What does a function return if it has no return statement?
4. What happens if you print a local variable outside its function?
5. How do you change a global variable inside a function?

Answers at the back of this handbook.

Answer key

AI-01 • AI, ML, Deep Learning, GenAI & Agents: The Big Picture

1. Artificial intelligence (AI)
2. No. An agent is a system built around one or more models, with state, tools, memory, policies and orchestration.
3. More data, cheaper compute, and a stream of successful results
4. Core machine learning

AI-02 • How Machines Learn: $y = f(x)$

1. Rules and data; it produces answers
2. The features (inputs)
3. The model's prediction
4. 80%

AI-03 • Types of Machine Learning

1. Regression (supervised)
2. There are no labels (answers) in the data
3. Rewards and penalties from its environment
4. Classification (supervised)

AI-04 • The ML Workflow: From Data to Deployment

1. The model must never see the test data while learning
2. Overfitting
3. Models only understand numbers
4. Because the data changes over time (drift), so old patterns stop working

AI-05 • Neural Networks & Deep Learning Intuition

1. A weighted sum of its inputs plus a bias ($w \cdot x + b$)
2. Many hidden layers
3. Probabilities across classes that add up to 100%
4. Convolutional neural networks (CNNs)

AI-06 • Data, Analytics & Data Types

1. Diagnostic
2. Ordinal

3. Interval (no true zero)
4. A data product

AI-07 • Describing Data: Central Tendency & Spread

1. The mean
2. The mean
3. The spread of the middle 50% of the data (Q3 - Q1)
4. So negative and positive deviations don't cancel out

AI-08 • Probability Essentials for ML

1. Impossible
2. Relative frequency
3. 0.25
4. Poisson

AI-09 • Python Basics: Variables, Types & Input

1. Indentation (4 spaces by convention)
2. = assigns a value; == compares two values
3. str (a string)
4. TypeError: strings are immutable

AI-10 • Python Collections: Lists, Tuples, Sets & Dictionaries

1. The last element
2. [20, 30, 40]
3. With a trailing comma, e.g. (10,)
4. An empty dictionary
5. append adds one element; extend adds each element of another collection

AI-11 • Writing Functions

1. def
2. A parameter is the name in the definition; an argument is the value passed in the call
3. None
4. A NameError
5. Declare it with the global keyword first