

SELF-STUDY HANDBOOK

AI & Machine Learning Foundations

CommunityForce Inc.

Modules in this edition:

- **AI-12** Arguments, Lambdas, Filter & Map
- **AI-13** Classes & Objects: How scikit-learn Models Work
- **AI-14** Linear Regression Explained
- **AI-15** Hands-on: Predicting Ice-Cream Demand with scikit-learn
- **AI-16** Multiple Regression & Prediction Intervals
- **AI-17** From Regression to Classification: Logistic Regression
- **AI-18** Hands-on: Pass/Fail Classifier & Evaluation Metrics
- **AI-19** Clustering & K-Means Concepts
- **AI-20** Hands-on: Customer Segmentation with K-Means
- **AI-21** Decision Trees, Overfitting & Random Forests
- **AI-22** Feature Engineering & Data Cleaning

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

Arguments, Lambdas, Filter & Map

Video: 12_AI-12_Arguments_Lambdas_Filter_and_Map.mp4 · 4:24 · Stage: Python Foundations · Audience: Beginners who completed AI-11 Writing Functions

Learning objectives

- Use keyword and default arguments
- Accept any number of arguments with *args
- Write lambdas and use filter and map

By default, arguments are matched by position

2. The order is not important
3. The parameter names are important

```
def f1(name, msg):          # Define a function with two parameters
    print("Hello:", name, msg) # Display the values received by name and msg

f1(name="AI", msg="We learn about models") # Pass both values using parameter names
... Hello: AI We learn about models

# Changing the Order of Keyword Arguments
def f1(name, msg):          # Define a function with name and msg parameters
    print("Hello:", name, msg) # Display the received values

f1(name="AI", msg="We learn about models") # Pass name first and msg second
```

By default, arguments are matched by position. With keyword arguments you name the parameter in the call, like `name equals AI` and `msg equals We learn about models`. Because each value is labelled, the order no longer matters, and the call reads more clearly.

You can mix the two styles, with one important rule

Important Rule

Positional arguments must come first.

Keyword arguments must come after positional arguments.

```
def f1(name, msg):  
    print("Hello:", name, msg)  
  
f1("AI", msg="We learn about models")  
  
Hello: AI We learn about models
```

Define a function with two parameters
Display the received values
Pass name positionally and msg by keyword

```
# Incorrect Combination  
def f1(name, msg):  
    print("Hello:", name, msg)
```

Define a fu
Display the

TIP

f1("AI", msg="Hi") works. f1(name="AI", "Hi") is a SyntaxError.

You can mix the two styles, with one important rule. Positional arguments must come first, and keyword arguments come after them. Here AI is passed by position, and msg by keyword, so it works. Put a keyword before a positional value and Python raises a syntax error.

Tip f1("AI", msg="Hi") works. f1(name="AI", "Hi") is a SyntaxError.

A default value makes a parameter optional

```
# TypeError: the required course argument is missing

Course is: AI

# Function with a Default Value
def f1(course="Java"):
    print("Course is:", course)

# Define course with Java as its default value
# Display the value stored in course

f1()

# Do not supply a value, so Python uses Java

f1("AI")

# Supply AI, so Python uses AI instead of Java

... Course is: Java
Course is: AI

# Default Value Is Not Permanent Replacement
def f1(course="Java"):
    print("Course is:", course)

# Set Java as the default course
# Display the selected course
```

A default value makes a parameter optional. Here `course` equals `Java` in the definition. Calling `f` one with no argument uses `Java`. Supplying `AI` overrides the default for that call only. Default parameters are everywhere in machine learning libraries, which is why a model can be created with no settings at all.

Sometimes you don't know how many values will arrive

```
def f1(*n):  
    print(n)  
  
# Collect any number of arguments  
# Display the tuple  
  
f1()  
f1(10)  
f1(10, 20)  
f1(10, 20, 30)
```

TIP
def total(*values): return sum(values) → total(1, 2, 3) = 6

(
(10,
(10, 20)
(10, 20, 30)

Using pass with Variable-Length Arguments

Sometimes you don't know how many values will arrive. A star in front of a parameter, written star n, collects any number of positional arguments into a tuple. Call it with none, one, two, or three values, and n holds them all.

Tip def total(*values): return sum(values) → total(1, 2, 3) = 6

A lambda is a small, unnamed function written on one line: the lamb...

```
lambda Function with One Argument

lambda with One Argument

lambda function can accept one argument and return the result of an expression.

double = lambda number: number * 2      # Create a lambda that doubles a number

result = double(10)                      # Pass 10 as the input argument

print("Double value:", result)           # Display the returned value

• Double value: 20

Start coding or generate with AI.
```

A lambda is a small, unnamed function written on one line: the lambda keyword, the arguments, a colon, then a single expression. Double equals lambda number, colon, number times two. Calling double with ten returns twenty. The result of the expression is returned automatically.

Compare a normal function with a lambda

```
# Normal function

def multiply(a, b):                      # Define a normal multiplication function
    return a * b                        # Explicitly return the result

# Lambda function

multiply_lambda = lambda a, b: a * b    # The result is returned automatically

print(multiply(5, 4))                  # Call the normal function

print(multiply_lambda(5, 4))           # Call the lambda function

• 20
  20
```

Compare a normal function with a lambda. Multiply uses def, a name, and an explicit return. The lambda version does the same in one line, with no return keyword. Both print twenty. Use lambdas for short, throwaway logic, and def for anything longer or reused.

Functions can be passed to other functions

```
def is_even(number):  
    # Accept the number when it is even  
    if number % 2 == 0:  
        return True  
    else:  
        # Execute when the number is not even  
        # Reject the number when it is odd  
        return False  
  
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # Create the input list  
  
filtered_values = filter(is_even, numbers) # Apply is_even to every list element  
  
even_numbers = list(filtered_values) # Convert the filter object into a list  
  
print("Even numbers:", even_numbers) # Display the selected numbers  
  
Even numbers: [2, 4, 6, 8, 10]
```

Start coding or generate with AI.

Functions can be passed to other functions. Filter takes a test function and a list, and keeps only the items where the test returns True. Here is even is applied to the numbers one to ten. Filter returns a filter object, so we wrap it in list to see the even numbers.

Combine filter with a lambda and you don't even need a named test f...

```
numbers = [1, 2, 3, 4, 5, 6] # Create a list of numbers  
  
even_numbers = list(  
    filter(  
        lambda number: number % 2 == 0,  
        numbers  
    )  
) # Convert the result into a list  
# Select matching numbers  
# Check each number  
# Input list  
  
print(even_numbers) # Display the even numbers  
  
[2, 4, 6]
```

Python Arrays

Start coding or generate with AI.

Combine filter with a lambda and you don't even need a named test function. The lambda checks whether number modulo two equals zero, filter applies it to each item, and list collects the result: two, four, and six.

map(): transform every item

map(): transform every item

```
1 celsius = [0, 20, 25, 30]
2
3 fahrenheit = list(
4     map(lambda c: c * 9 / 5 + 32, celsius)
5 )
6
7 print(fahrenheit)
8 # [32.0, 68.0, 77.0, 86.0]
```

filter keeps some items; map transforms all of them.

Map is filter's partner. Instead of keeping some items, map applies a function to every item and returns the transformed values. Here a lambda converts each temperature from Celsius to Fahrenheit. Filter selects, map transforms, and together they replace many simple loops.

Teaser: NumPy arrays

Teaser: NumPy arrays

```
1 import numpy as np
2
3 celsius = np.array([0, 20, 25, 30])
4 print(celsius * 9 / 5 + 32)
5 # [32. 68. 77. 86.]
6
7 table = np.array([[1, 2, 3], [4, 5, 6]])
8 print(table.shape) # (2, 3)
```

2 rows × 3 columns, like a small table of features.

One last teaser. A Python list can mix types, but for numbers, machine learning uses NumPy arrays instead. An array holds values of one type, and does maths on every element at once, with no loop or lambda needed. It also handles two-dimensional tables of data with rows and columns. You'll meet NumPy again when we work with real datasets.

Your function toolkit

Your function toolkit		
Tool	Example	Use it to
Keyword args	<code>f1(name="AI", msg="Hi")</code>	Make calls clear, any order
Defaults	<code>def f1(course="Java")</code>	Make parameters optional
*args	<code>def total(*values)</code>	Accept any number of values
lambda	<code>lambda x: x * 2</code>	Short, throwaway functions
filter	<code>filter(test, items)</code>	Keep items that pass a test
map	<code>map(func, items)</code>	Transform every item

Here's your function toolkit at a glance. Keyword arguments make calls clear. Defaults make parameters optional. Star args accepts any number of values. A lambda is a one-line function. Filter keeps items that pass a test, and map transforms every item.

Key takeaways

- Keyword arguments come after positional ones
- Defaults make parameters optional
- *args collects extra values into a tuple
- lambda + filter + map process whole lists

Practice

1. Write `greet(name, greeting="Hello")` and call it three ways: positional only, with a keyword, and with the default.
2. Write `average(*scores)` that returns the mean of any number of scores.
3. Use filter with a lambda to keep only the scores above 70 from a list.
4. Use map with a lambda to convert a list of prices from dollars to cents.

Check your understanding

1. Which must come first in a call: positional or keyword arguments?
2. What does `def f1(course="Java")` let you do?
3. What type does `*args` collect values into?
4. What does `filter` return before you wrap it in `list()`?
5. What is the difference between `filter` and `map`?
6. Why use a NumPy array instead of a list for numbers?

Answers at the back of this handbook.

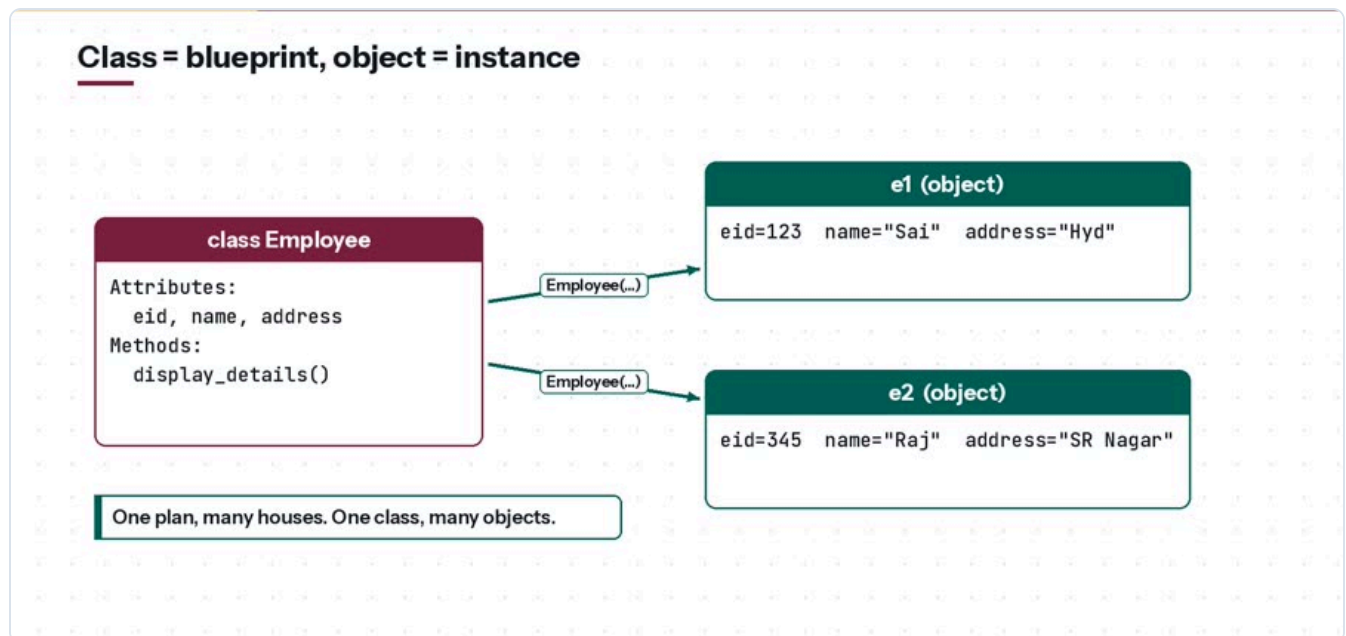
Classes & Objects: How scikit-learn Models Work

Video: 13_AI-13_Classes_and_Objects.mp4 · 3:59 · Stage: Python Foundations · Audience: Beginners who completed AI-12; preparing for the hands-on machine learning modules

Learning objectives

- Define a class with `__init__` and `self`
- Create objects and call their methods
- Read scikit-learn code as objects and methods

Class = blueprint, object = instance



A class is a blueprint, and objects are the things built from it. Think of a house plan: one plan, many houses, each with its own address and colour. An `Employee` class describes what every employee has, its attributes, and what it can do, its methods. Each object holds its own values.

The constructor is a special method of the class

```
Employee Salary : 100000  
Employee Address : Hyd  
Employee Age : IT
```

Introduction to Constructor

Constructor in Python

A constructor is a special method of a class.

In Python, the `__init__()` method is commonly called the constructor.

The constructor is used to initialize the variables of an object.

Important Properties

1. The constructor is a special method

The constructor is a special method of the class. In Python it's named `init`, with two underscores on each side. Its job is to initialize the variables of each new object.

Its key properties: it runs automatically when an object is created...

1. The constructor is a special method
2. Its name is `__init__`
3. It runs automatically when an object is created
4. It is normally used to create and initialize instance variables
5. Its first parameter normally uses the name `self`
6. It can accept additional parameters
7. It normally executes once for each newly created object

Syntax

```
class ClassName:
```

Its key properties: it runs automatically when an object is created, it usually creates and initializes instance variables, and its first parameter is normally called `self`. It can accept extra parameters, and it runs once for each new object.

Here's the trainer's Employee class

```
class Employee:
    """ This is the employee class for storing a
    - Employee : is the name of the class
    """
    (parameter) self: Self@Employee

    def __init__(self, eid, ename, salary, e_address, department):
        self.eid = eid
        self.ename = ename
        self.salary = salary
        self.e_address = e_address
        self.department = department

    def display_emp_details(self):
        print("Employee ID :", self.eid)
        print("Employee Name :", self.ename)
        print("Employee Salary :", self.salary)
        print("Employee Address :", self.e_address)
```

TIP
self = the object the method was called on.
Python passes it automatically.

Here's the trainer's Employee class. The class keyword and a CamelCase name, then a docstring. The constructor takes self plus employee details, and stores each one with self dot eid equals eid, and so on. A second method, display emp details, prints them. self is how a method reaches this object's own data.

Tip self = the object the method was called on. Python passes it automatically.

Constructor or method? The constructor is commonly named init, runs...

Start coding or generate with AI.	
Constructor vs Method	
Constructor	Method
Special method	Function defined inside a class
Commonly named <code>__init__()</code>	Can have any valid name
Runs automatically during object creation	Runs when it is called
Initializes object data	Performs an operation or business logic
Normally runs once for each newly created object	Can be called any number of times
First parameter is normally <code>self</code>	First parameter of an instance method is normally <code>self</code>
Start coding or generate with AI.	

Constructor or method? The constructor is commonly named init, runs automatically during object creation, and normally runs once per object. A regular method can have any valid name, runs only when you call it, and can be called any number of times. Both take self as the first parameter.

In this first version, the constructor assigned fixed values

```
.display_emp_details()           # Call the display method using the first object
int()                            # Print an empty line between the two records
.display_emp_details()           # Call the display method using the second object

Employee ID is: 1234
Employee name is: Sai
Employee address is: Hyd
Employee age is: 45

Employee ID is: 1234
Employee name is: Sai
Employee address is: Hyd
Employee age is: 45
```

What happens During Execution?

In this first version, the constructor assigned fixed values. Create two objects, e one and e two, call display on each, and both show the same employee. Fixed values defeat the point of having many objects.

The fix is to pass values in

```
print("Employee ID is:", self.eid)
print("Employee name is:", self.e_name)
print("Employee address is:", self.e_address)
print("Employee age is:", self.e_age)
```

Display this object's employee ID
Display this object's address
Display this object's age

TIP
Formal parameters: names in def __init__(self, eid, ...). Actual arguments: values in Employee(123, ...).

Creating Objects with Different Values

```
= Employee(
    123,
    "Sai",
    "Hyd",
    34
)
```

Create the first Employee object
Pass 123 to the eid parameter
Pass Sai to the e_name parameter
Pass Hyd to the e_address parameter
Pass 34 to the e_age parameter

The fix is to pass values in. The constructor now receives eid, name, address, and age, and the call supplies them: Employee with one two three, a name, a city, and thirty-four. These are the actual arguments, matched to the constructor's formal parameters.

Tip Formal parameters: names in def __init__(self, eid, ...). Actual arguments: values in Employee(123, ...).

Now each object keeps its own values

```
--- Second Employee ---  
Employee ID is: 345  
Employee name is: Raj  
Employee address is: SR Nagar  
Employee age is: 45
```

Understanding Assignment Inside the Constructor

Understanding `self.eid = eid` Markdown Cell

Parameter vs Instance Variable

Consider this statement:

```
self.eid = eid
```

TIP

Instance variables (`self.x`) belong to one object.
Change `e1` and `e2` is unaffected.

Now each object keeps its own values. The second employee shows ID three four five, Raj, and SR Nagar, while the first keeps its own details. One class, many independent objects.

Tip Instance variables (`self.x`) belong to one object. Change `e1` and `e2` is unaffected.

scikit-learn models are objects

scikit-learn models are objects

```
1 from sklearn.linear_model import LinearRegression
2
3 model = LinearRegression()      # object
4 model.fit(X_train, y_train)    # method
5 print(model.coef_)            # attribute
6 y_pred = model.predict(X_test) # method
```

predict() uses the learned attributes on new data.

Now look at scikit-learn with fresh eyes. LinearRegression is a class. Calling it with brackets runs the constructor and gives you a model object; default parameters fill in the settings. fit is a method that learns from your data and stores the results as attributes, like the coefficients. And predict is another method that uses them.

OOP vocabulary

OOP vocabulary

Term	Employee example	scikit-learn example
Class	class Employee	LinearRegression
Object	e1 = Employee(123, ...)	model = LinearRegression()
Constructor	def __init__(self, ...)	Runs when you call LinearRegression()
Attribute	self.eid / e1.eid	model.coef_
Method	e1.display_emp_details()	model.fit(), model.predict()

Here's the vocabulary side by side. The class is the blueprint. The object is one instance, created by calling the class. The constructor sets it up. Attributes hold its data, accessed with self inside the class or the object name outside. And methods are functions that belong to the class.

Key takeaways

- Class = blueprint; object = instance
- `__init__` runs automatically and stores values on self
- Call methods with dot notation
- scikit-learn: create, fit, predict

Practice

1. Write a class `Product` with `__init__(self, name, price)` and a method `total(self, quantity)` that returns `price * quantity`.
2. Create two `Product` objects with different values and print the total for each.
3. Add a method `apply_discount(self, percent)` that changes `self.price`. Show that only one object changes.
4. Label each line of: `model = LinearRegression(); model.fit(X, y); model.predict(X_new)` as class, object, constructor or method.

Check your understanding

1. What is the constructor method called in Python?
2. What does `self` refer to inside a method?
3. How many times does the constructor normally run per object?
4. In `model = LinearRegression()`, what is `model`?
5. Is `model.fit()` a function or a method?

Answers at the back of this handbook.

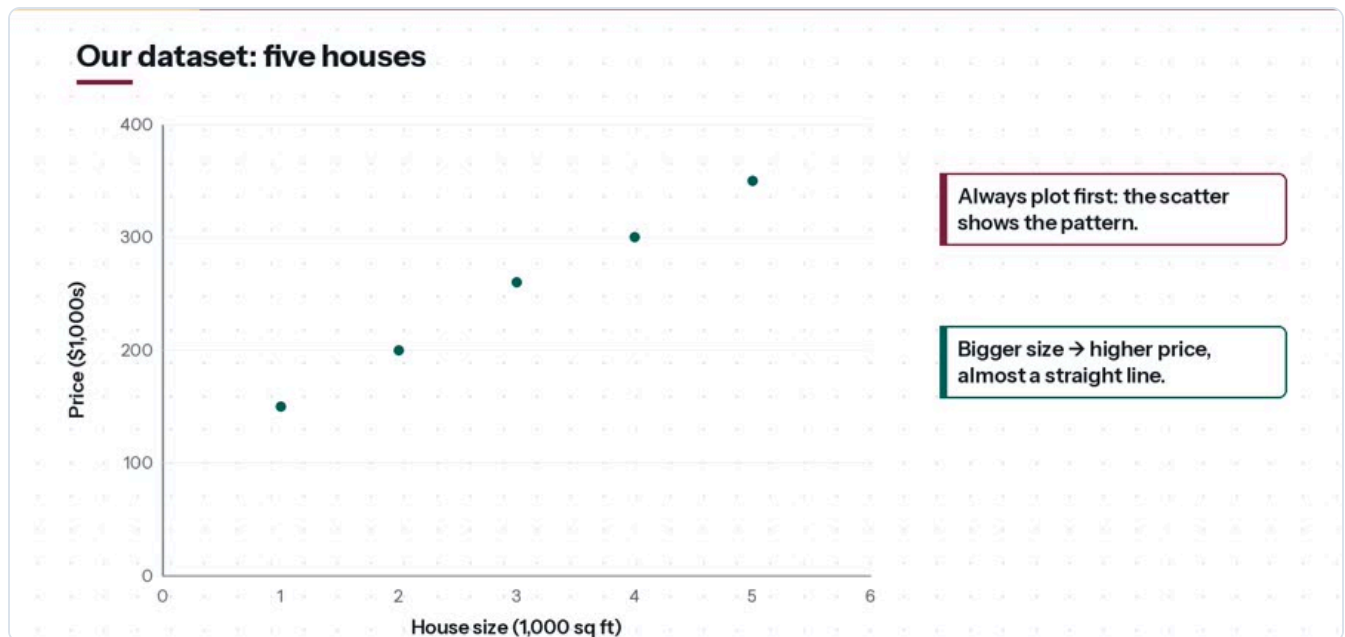
Linear Regression Explained

Video: 14_AI-14_Linear_Regression_Explained.mp4 · 4:39 · Stage: Modeling · Audience: Learners who completed the ML workflow and statistics modules (AI-04 to AI-08)

Learning objectives

- Read slope and intercept in $y = mx + c$
- Explain residuals, squared errors and MSE
- Describe least squares, gradient descent and R-squared

Our dataset: five houses



Here's the trainer's example: five houses, with their size in thousands of square feet and their price in thousands of dollars. Always plot your data first. The scatter shows a clear pattern. Bigger houses cost more, and the points sit almost on a straight line.

The model is a straight line



The model is a straight line: y equals $m \times x$ plus c . The slope, m , is how much y changes when x grows by one. Here it's fifty, so each extra thousand square feet adds about fifty thousand dollars. The intercept, c , is the value of y when x is zero. Here it's one hundred and two. Training simply means finding the best values of m and c .

The trainer's slide builds the intuition

Bigger slope → steeper line; negative slope → line goes down

Bigger intercept → whole line moves up

Slope 50: each extra 1,000 sq ft adds about \$50k

Intercept 102: base value of about \$102k

CAREFUL

Trust predictions only inside the data range (no wild extrapolation).

Change slope (m)

$m = 2$ (steep)

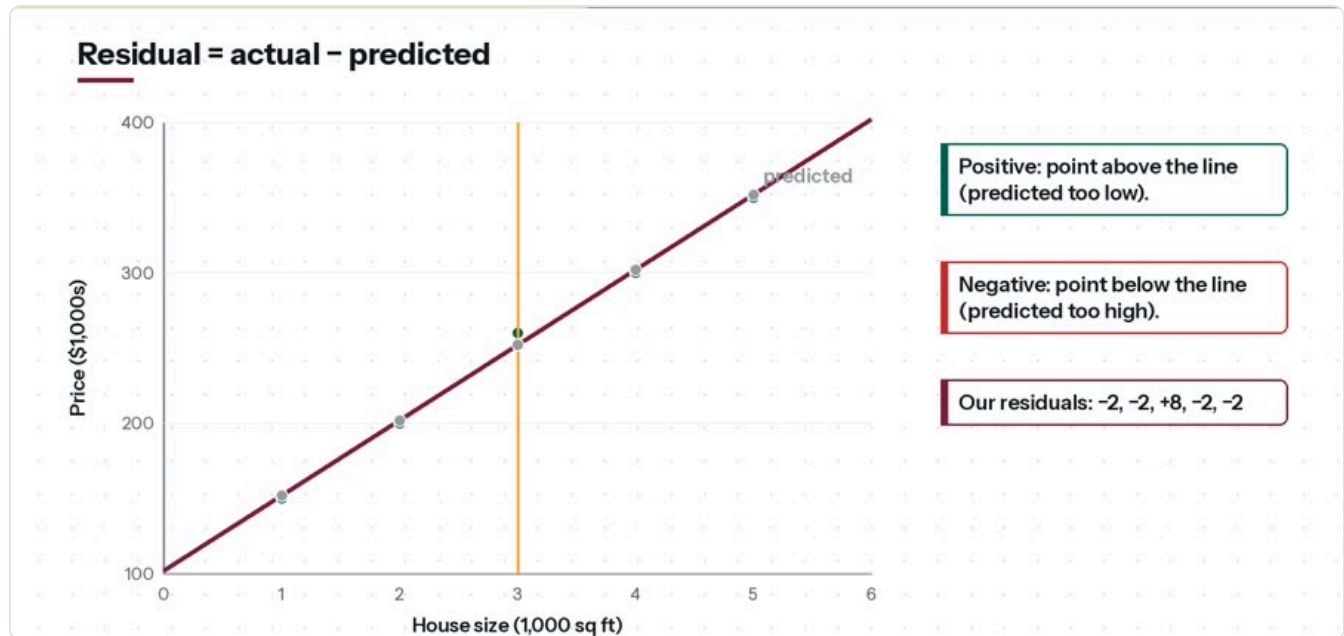
$m = 1$

$m = 0.3$ (flat)



The trainer's slide builds the intuition. A bigger slope gives a steeper line, and a negative slope makes it go down. A bigger intercept moves the whole line up. For our houses, each extra thousand square feet adds about fifty thousand dollars, on top of a base value of about one hundred and two thousand. And be careful: trust predictions only inside the range of your data, with no wild extrapolation.

Residual = actual - predicted



How wrong is the line? For every house, the error, or residual, is the actual price minus the predicted price. A positive residual means the point sits above the line, so the model predicted too low. A negative residual means it predicted too high. For our five houses, the residuals are minus two, minus two, plus eight, minus two, and minus two.

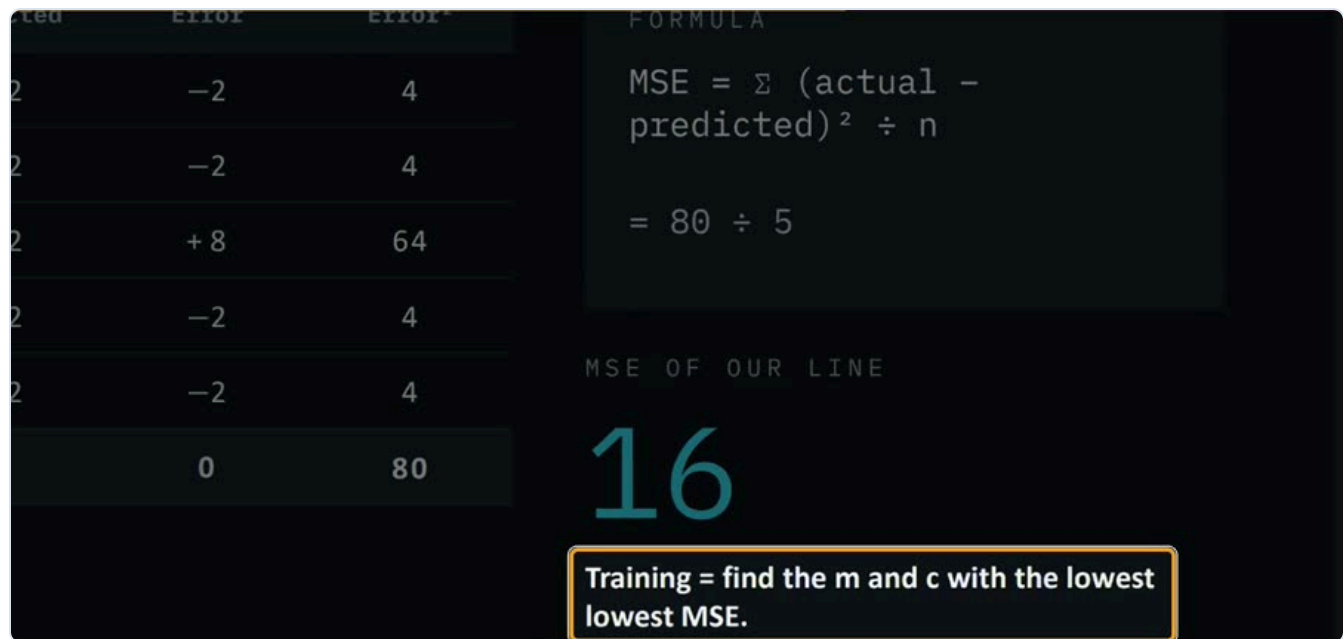
Why we square the errors

Why we square the errors

- 1 Errors cancel**
+3 and -3 add up to 0, so a bad line can look perfect.
- 2 Square them**
Every squared error is positive.
 $2 \rightarrow 4$
 $10 \rightarrow 100$
Big mistakes count more.
- 3 MSE**
MSE = average of the squared errors.
Lower is better.

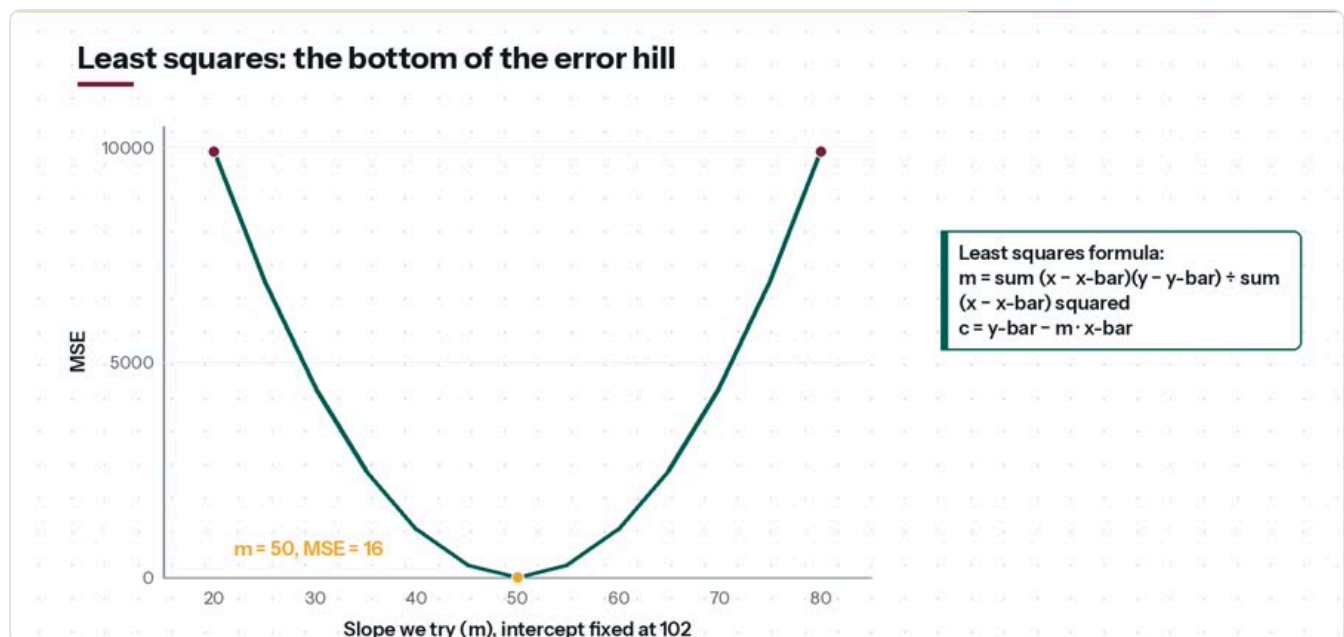
Why do we square the errors? Because positive and negative errors cancel out. A flat, obviously wrong line can still add up to zero. Squaring makes every error positive. It also punishes big mistakes much more than small ones: an error of two becomes four, but an error of ten becomes one hundred. The average of the squared errors is called the mean squared error, or MSE.

Here's the trainer's table for our line



Here's the trainer's table for our line. Each row shows the actual price, the predicted price, the error, and the squared error. House three is off by eight, so its squared error is sixty-four. The errors add up to zero, but the squared errors add up to eighty. Divide by five houses, and the MSE of our line is sixteen. Training means finding the m and c with the lowest MSE.

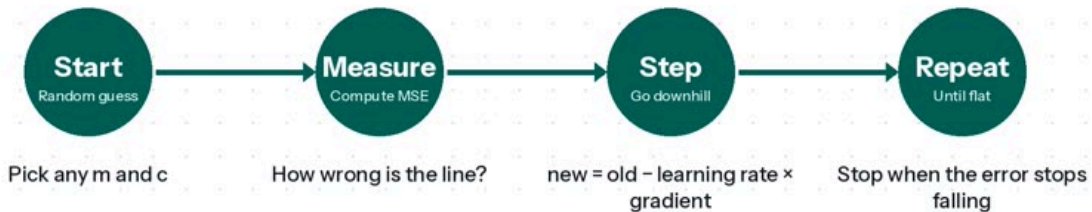
Least squares: the bottom of the error hill



Least squares picks the line with the smallest total of squared errors. The trainer's way to picture it is a hill. Try different slopes and plot the MSE for each one. A slope of twenty or eighty gives a huge error. The error falls as you approach fifty, the bottom of the hill, where the MSE is just sixteen. For a straight line there's a direct formula for this minimum.

For bigger models with many inputs, we use gradient descent instead

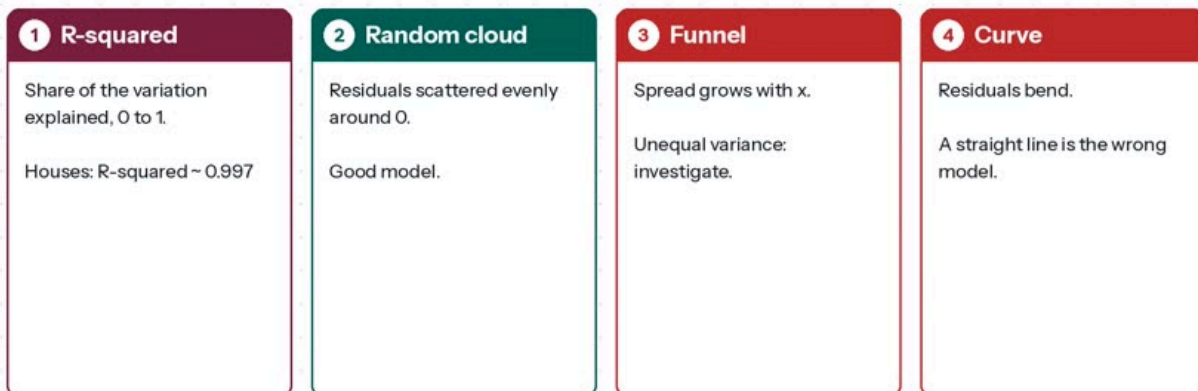
Gradient descent: walking down the error hill



For bigger models with many inputs, we use gradient descent instead. Picture walking down that hill in fog. Start from a random guess. Measure how wrong the line is. Take a small step downhill, sized by the learning rate. Then repeat until the error stops falling. This same idea trains much bigger models, including neural networks.

Is the line any good?

Is the line any good?



Once the line is fitted, check it two ways. First, R squared tells you the share of the variation in y that the line explains. It runs from zero to one, and our houses score about zero point nine nine seven. Second, plot the residuals. A random cloud around zero is good. A funnel shape means the errors grow with x . And a curve means a straight line is the wrong model.

Key takeaways

- Model: $y = mx + c$ (slope, intercept)
- Residual = actual – predicted; square them
- Least squares or gradient descent minimise MSE
- Check R-squared and the residual plot

Practice

1. For the five houses, compute the predicted price and the residual for each house using $\text{price} = 102 + 50 \times \text{size}$, then compute the MSE.
2. Try the line $\text{price} = 100 + 52 \times \text{size}$. Is its MSE higher or lower than 16? What does that tell you?

Check your understanding

1. In $y = mx + c$, what does m mean?
2. How is a residual calculated?
3. Why are errors squared?
4. What does a funnel-shaped residual plot signal?

Answers at the back of this handbook.

Hands-on: Predicting Ice-Cream Demand with scikit-learn

Video: 15_AI-15_Predicting_Ice_Cream_Demand.mp4 · 5:15 · Stage: Train & Evaluate · Audience: Learners who completed AI-14 and the Python modules; first end-to-end model

Learning objectives

- Load data into pandas and plot it
- Split, fit and predict with LinearRegression
- Evaluate with MAE, RMSE and R-squared

Here's the problem, straight from the notebook

Ice Cream Shop

Story: Ravi runs a small ice cream kiosk near the beach. Some days he runs out of ice cream by lunch time. On other days he has too much stock that melts in the evening.

Ravi wrote down the **temperature** and the **number of ice creams sold** for 20 days.

Job: build a model that looks at tomorrow's temperature and tells Ravi how many ice creams to keep ready.

Input (X) = temperature in °C

Output (y) = ice creams sold in one day

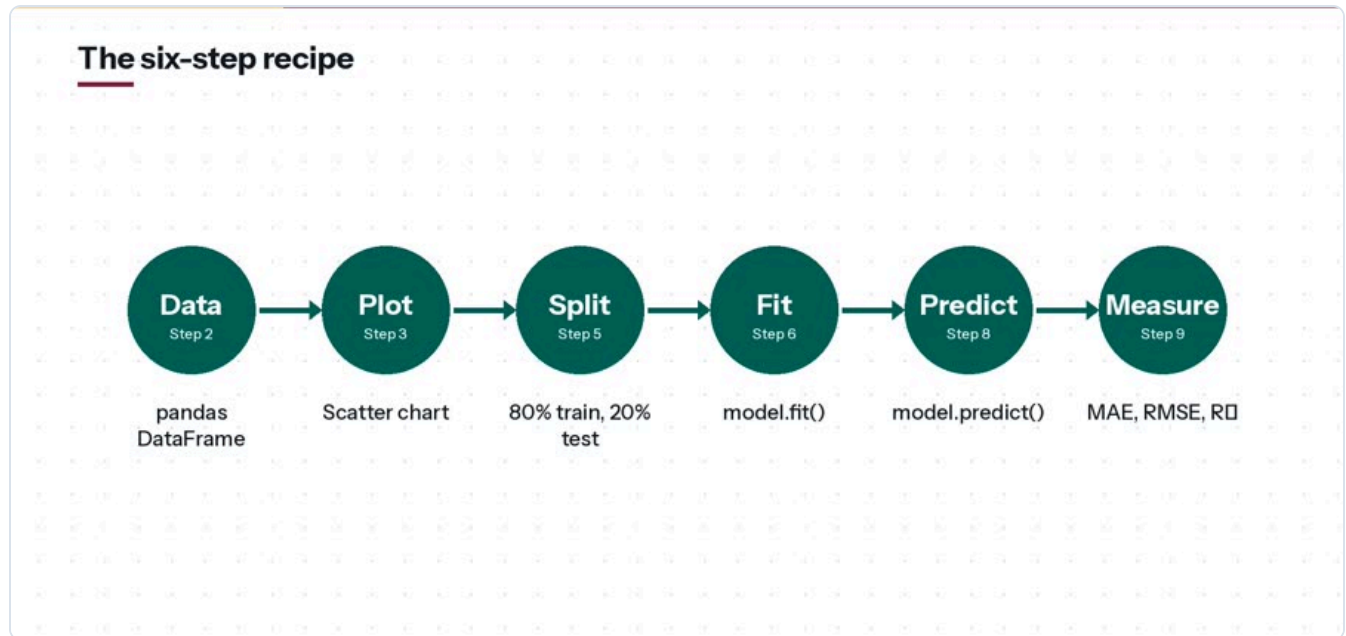
We'll use **simple linear regression**: one input, one output, one straight line.

In Python, a comment starts with `#`. Everything after `#` on a line is only for humans. Python ignores it. Every line of code below has a comment that explains it.

Run each cell with **Shift + Enter**, from top to bottom.

Here's the problem, straight from the notebook. Ravi runs a small ice-cream kiosk near the beach. Some days he runs out by lunchtime; other days he throws away melted stock. He wrote down the temperature and the number of ice creams sold for 20 days. Our job: build a model that reads tomorrow's temperature and tells Ravi how many ice creams to prepare. One input, one output, so this is simple linear regression.

The notebook follows six steps that you'll reuse in almost every ma...



The notebook follows six steps that you'll reuse in almost every machine learning project. Create the data, plot it, split it, fit the model, predict the hidden days, and measure the errors. Let's walk through each one.

Every notebook starts with imports

```
library is ready-made code that someone else wrote, so we don't have to write it ourselves.
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

print("All libraries are ready")
```

```
# pandas works with tables (rows and columns), like Excel
# matplotlib draws charts; we call it plt for short
# the linear regression model (draws the best straight line)
# splits data into a training part and a testing part
# measures the average size of our mistakes
# measures the average of the squared mistakes
# measures how well the line fits (1.0 = perfect)

# if you see this message, the imports worked
```

```
All libraries are ready
```

Step 2: Create the data

vi's notebook for 20 days. We type the data directly, so no file upload is needed.

Every notebook starts with imports. Pandas handles tables. Matplotlib draws charts. From scikit-learn we take the LinearRegression model, the train_test_split helper, and three metrics that measure the model's mistakes. When the cell prints All libraries are ready, the imports worked.

Next, the data

Ravi's notebook for 20 days. We type the data directly, so no file upload is needed.

TIP

Small data can be typed in. Real projects load a file: `pd.read_csv('sales.csv')`.

```
temperature = [22, 24, 25, 27, 28, 29, 30, 31, 32, 33,
               34, 35, 36, 37, 38, 39, 40, 41, 42, 44] # temperature (°C) for each of the 20 days

ice_creams_sold = [93, 116, 121, 117, 125, 129, 145, 145, 159, 141,
                  178, 169, 182, 181, 185, 198, 207, 204, 211, 230] # ice creams sold on the same 20 days

data = pd.DataFrame({
    "temperature": temperature,
    "ice_creams_sold": ice_creams_sold
})

print(data.shape)
data.head()
```

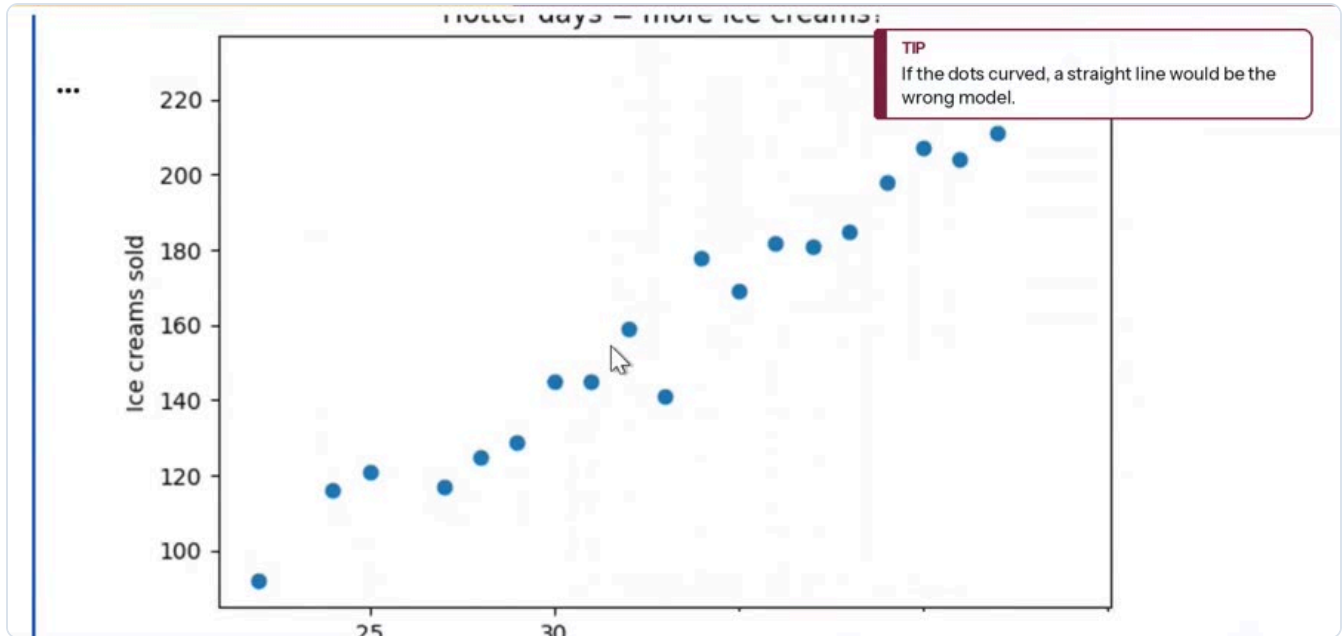
... (20, 2)

temperature	ice_creams_sold
-------------	-----------------

Next, the data. Ravi's 20 temperatures go in one list, and the 20 sales figures in another, in the same order. Then `pd dot DataFrame` combines the two lists into a table with two columns. Printing its shape confirms 20 rows and 2 columns.

Tip Small data can be typed in. Real projects load a file: `pd.read_csv('sales.csv')`.

Always look at your data before modelling



Always look at your data before modelling. The scatter plot shows one dot per day. As temperature rises, sales rise too, and the dots roughly follow a straight line. That pattern tells us a straight-line model is a sensible choice.

Tip If the dots curved, a straight line would be the wrong model.

Now choose the input and the output

4: Choose X (input) and y (output)

```
X = data[["temperature"]] # X = input. Double brackets keep it as a TABLE (sklearn needs a
y = data["ice_creams_sold"] # y = output. Single brackets give one column of values

print("X shape:", X.shape) # (20, 1) -> 20 rows, 1 column
print("y shape:", y.shape) # (20,) -> 20 values

X.shape: (20, 1)
y.shape: (20,)
```

5: Split into training data and test data

The model learns from the **training** days. We hide the **test** days and use them later, like a surprise exam.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, # split X and y into 4 parts
```

Now choose the input and the output. Capital X holds temperature in double brackets, so it stays a table with 20 rows and 1 column, which is the shape scikit-learn expects. Lowercase y is the sales column, a simple list of 20 values.

train_test_split divides the 20 days

Step 5: Split into training data and test data

The model learns from the **training** days. We hide the **test** days and use them later, like a surprise exam.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, # split X and y into 4 parts
                                                    test_size=0.2, # the data we want to split
                                                    random_state=42 # 20% of rows go to the test part (4 days)
                                                    ) # fixes the random shuffle, so everyone gets the same split
                                                    # close the function call

print("Training days:", len(X_train)) # should print 16
print("Test days:", len(X_test)) # should print 4

Training days: 16
Test days: 4
```

train_test_split divides the 20 days. A test size of 0.2 hides 20%, which is 4 days, for a surprise exam later. Random state fixes the shuffle, so everyone who runs the notebook gets the same split. The model trains on the other 16.

Training takes just two lines

Step 6: Create the model and train it

Training = the computer finds the best straight line through the training dots.

```
model = LinearRegression() # create an empty linear regression model
model.fit(X_train, y_train) # TRAIN the model: find the best line for the training data

print("Training finished") # a message so we know this step is done

.. Training finished
```

Step 7: Read the line: slope and intercept

The line is: **ice creams = intercept + slope × temperature**

Training takes just two lines. Create an empty `LinearRegression` model, then call `fit` with the training inputs and outputs. `fit` finds the line with the smallest squared errors through the sixteen training points, exactly the least squares idea from AI-14. When it prints `Training finished`, the model has learned.

Tip `fit()` = least squares: the line with the lowest MSE on the training data.

The trained model is just a line

```
intercept = model.intercept_

print("Slope (m):", round(slope, 2))          # round to 2 decimals so it is easy to read
print("Intercept (c):", round(intercept, 2))  # round to 2 decimals
print(f"Equation: ice creams = {intercept:.2f} + {slope:.2f} x temperature")  # print the full line equation

slope (m): 6.13
intercept (c): -43.21
Equation: ice creams = -43.21 + 6.13 x temperature
```

Does it mean?

Slope ≈ 6.13 → every 1 °C hotter, Ravi sells about **6 more** ice creams.

Intercept ≈ -43.21 → the line's value at 0 °C. Ravi never sells at 0 °C (and can't sell minus ice creams), so we don't use this number. It just positions the line.

8: Predict the test days and compare

The trained model is just a line. The slope is six point one three: every degree hotter, Ravi sells about six more ice creams. The intercept is about minus 43. It's the line's value at zero degrees, which never happens here, so it only positions the line; don't read it on its own.

Now the surprise exam

 `results` `# close the table`
`# show the comparison`

	temperature	actual_sold	predicted_sold	error
0	22	92	91.7	0.3
1	41	204	208.1	-4.1
15	39	198	195.9	2.1
1	24	116	103.9	12.1

Next steps:

[Generate code with results](#)

[New interactive sheet](#)

Now the surprise exam. `model.predict` runs on the four hidden days, and the table puts the guesses next to the truth. At 22 degrees, the model guessed 91.7 against 92 actually sold. Its biggest miss is at 24 degrees: about 104 predicted, against 116 sold.

Measuring the model on the 4 test days

Measuring the model on the 4 test days

1 MAE ~ 4.7

Average size of a mistake, in ice creams.

Smaller is better.

2 RMSE ~ 6.5

Like MAE, but big misses count more.

Smaller is better.

3 R-squared ~ 0.98

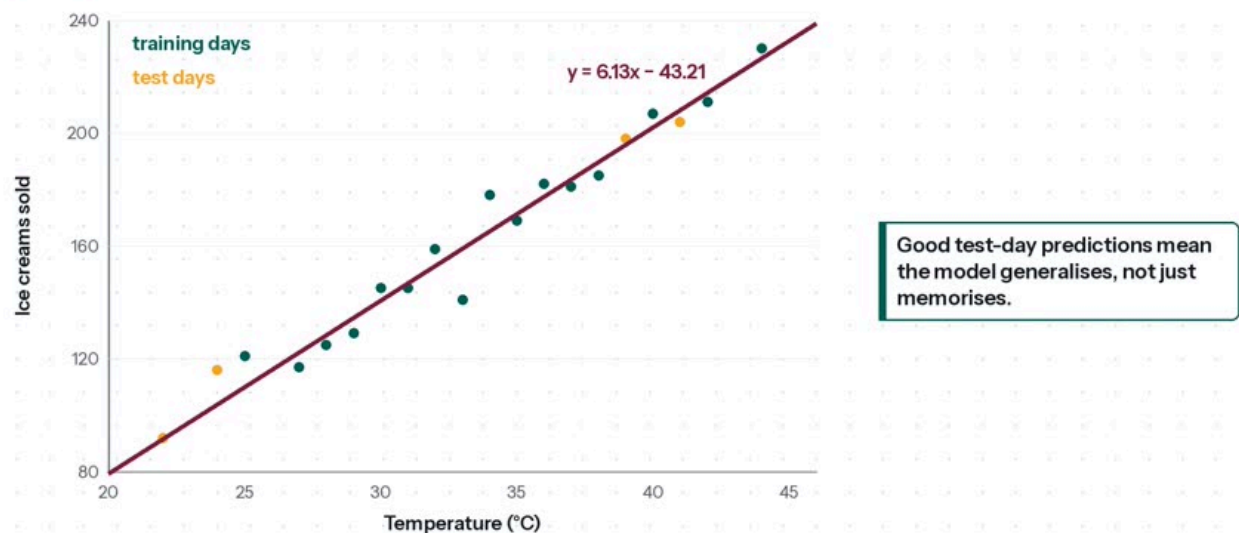
Share of the variation the line explains.

Closer to 1 is better.

How good is the model overall? We use three metrics on the test days. Mean absolute error is about four point seven: on average the model is off by fewer than five ice creams. Root mean squared error is about 6.5, and it punishes big misses more. And R squared is 0.98: the line explains about 98% of the variation in sales.

Training days, test days and the learned line

Training days, test days and the learned line



Here's the whole picture. The teal dots are the sixteen training days, the amber dots are the four test days, and the burgundy line is what the model learned. The line passes close to every point, including the days it never saw.

Finally, the forecast

Step 11: Use the model for tomorrow

The weather forecast says tomorrow will be **43 °C**. How many ice creams should Ravi prepare?

```
tomorrow = pd.DataFrame({"temperature": [43]})      # a new table with one row: tomorrow's temperature
prediction = model.predict(tomorrow)[0]             # predict, then take the first (and only) answer

print(f"Tomorrow at 43 °C, Ravi should prepare about {round(prediction)} ice creams") # final answer, rounded

... Tomorrow at 43 °C, Ravi should prepare about 220 ice creams
```

Bonus: check the line by hand

Finally, the forecast. The weather says tomorrow will be 43 degrees. Put that temperature in a one-row table with the same column name, and call predict. The model answers about 220 ice creams. You can check it by hand: multiply 6.13 by 43, then subtract 43.21, and you get about 220.

Tip $6.13 \times 43 - 43.21 = 220.38$, about 220

Key takeaways

- Loaded and plotted the data
- Split 80 / 20 into training and test days
- Fitted LinearRegression; read slope and intercept
- Measured MAE, RMSE and R-squared; made a forecast

Practice

1. Recreate the notebook in Colab or Jupyter with the twenty data points from the video and confirm slope ~ 6.13 and intercept ~ -43.21 .
2. Change test_size to 0.3 and random_state to 7. How do MAE and R-squared change, and why?
3. Predict sales for 30 °C and 46 °C. Which prediction would you trust less, and why?

Check your understanding

1. Why is X written with double brackets?
2. What does `test_size=0.2` do with 20 rows?
3. What does a slope of 6.13 mean here?
4. Which metric punishes big misses more, MAE or RMSE?

Answers at the back of this handbook.

Multiple Regression & Prediction Intervals

Video: 16_AI-16_Multiple_Regression_and_Intervals.mp4 · 4:21 · Stage: Modeling · Audience: Learners who completed AI-14 and AI-15

Learning objectives

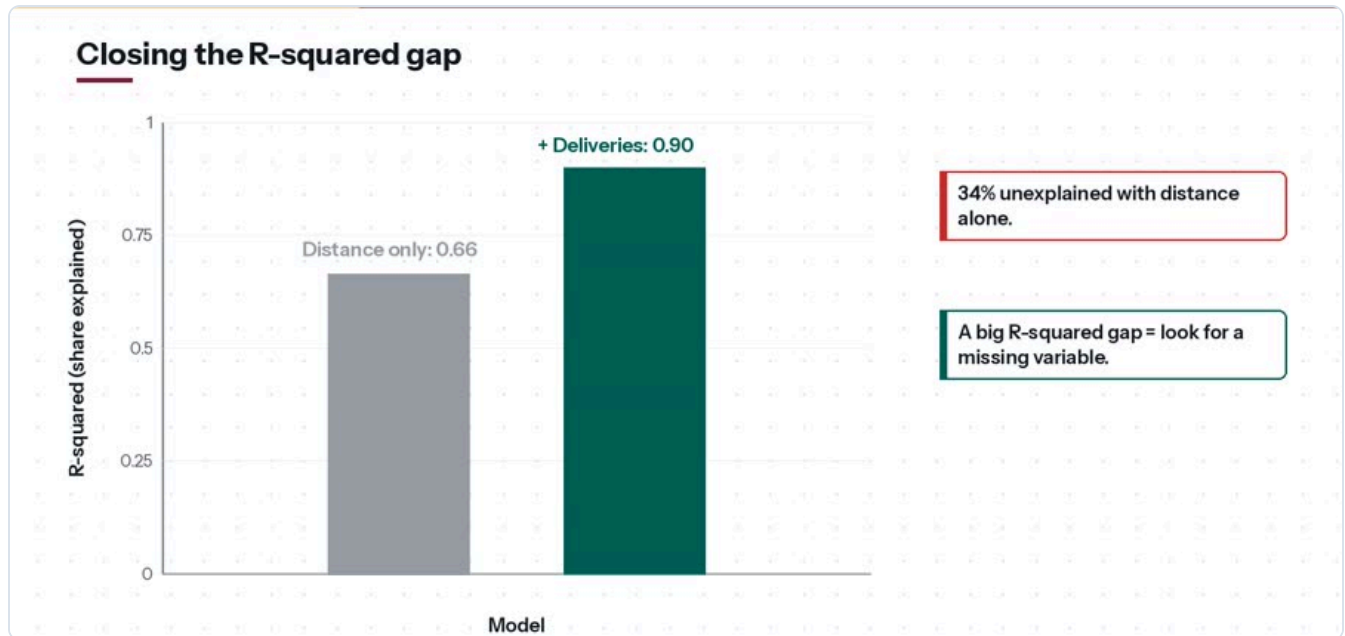
- Build a regression with several inputs
- Interpret coefficients "holding others constant"
- Tell confidence and prediction intervals apart

Simple vs multiple regression

Simple vs multiple regression		
	Simple	Multiple
Inputs	One (x)	Two or more (x1, x2, ...)
Equation	$y = b_0 + b_1x$	$y = b_0 + b_1x_1 + b_2x_2 + \dots$
How many?	Exactly one	No fixed limit: add what helps
Example	Travel time from distance	Travel time from distance and stops

Simple linear regression uses one input to predict an output. Multiple linear regression uses two or more. There's no fixed limit. You add the variables that genuinely help explain the outcome. A delivery company, for example, might predict travel time from both distance and the number of stops.

Closing the R-squared gap



Why add variables at all? Look at R squared. With distance alone, the delivery model explains about 66% of the variation in travel time. The other 34% is held by something the model can't see. Add the number of deliveries, and R squared jumps to about 90%. A big gap in R squared is a clue that an important variable is missing.

Here's the trainer's delivery model

In simple linear regression, we use one independent variable to predict the dependent variable. But travel time does not depend only on distance. It may also depend on the number of deliveries. So now we introduce a second independent variable.

MULTIPLE REGRESSION MODEL

Dependent variable (Y)	Travel Time
X ₁	Distance Travelled
X ₂	Number of Deliveries

REGRESSION EQUATION

$$\hat{Y} = -0.8687 + 0.0611 X_1 + 0.9234 X_2$$

$$\text{Travel Time} = -0.8687 + 0.0611 (\text{Distance}) + 0.9234 (\text{Deliveries})$$

TIP

$$-0.8687 + 0.0611 \times 100 + 0.9234 \times 3 = 8.01$$

Here's the trainer's delivery model. The output, y, is travel time. The first input is the distance travelled, and the second is the number of deliveries. The fitted equation is $\hat{y}$ equals minus 0.8687, plus 0.0611 times distance, plus 0.9234 times deliveries. So a route of 100 distance units with 3 deliveries comes out at about 8 units of travel time.

Tip $-0.8687 + 0.0611 \times 100 + 0.9234 \times 3 = 8.01$

Here's the trainer's slide with the fitted coefficients

FOR DISTANCE	FOR DELIVERIES
0.0611	0.9234
If distance increases by 1 unit, predicted travel time increases by 0.0611 units, keeping the number of deliveries constant.	If the number of deliveries increases by 1, predicted travel time increases by 0.9234 units, keeping distance constant.
That phrase "keeping the other variable constant" is very important in multiple regression.	
What Happened to R²?	
With distance only R² = 0.664 Distance explained 66.4% of the variation in travel time.	After adding deliveries R² ≈ 0.90 The two variables together explain approximately 90% of the variation.

Here's the trainer's slide with the fitted coefficients. For distance, 0.0611: if distance increases by one unit, predicted travel time increases by about 0.06 units, keeping the number of deliveries constant. For deliveries, 0.9234: one more delivery adds about 0.9 units of travel time, keeping distance constant. That phrase, keeping the other variable constant, is the heart of multiple regression. And the bottom row shows the R squared jump we just saw.

Multiple regression in scikit-learn

Multiple regression in scikit-learn

```
1 from sklearn.linear_model import LinearRegression
2
3 X = data[["distance", "deliveries"]] # two inputs
4 y = data["travel_time"]
5
6 model = LinearRegression()
7 model.fit(X, y)
8
9 print(model.coef_)      # one per input
10 print(model.intercept_) # the constant
```

Coefficients follow the column order in X.

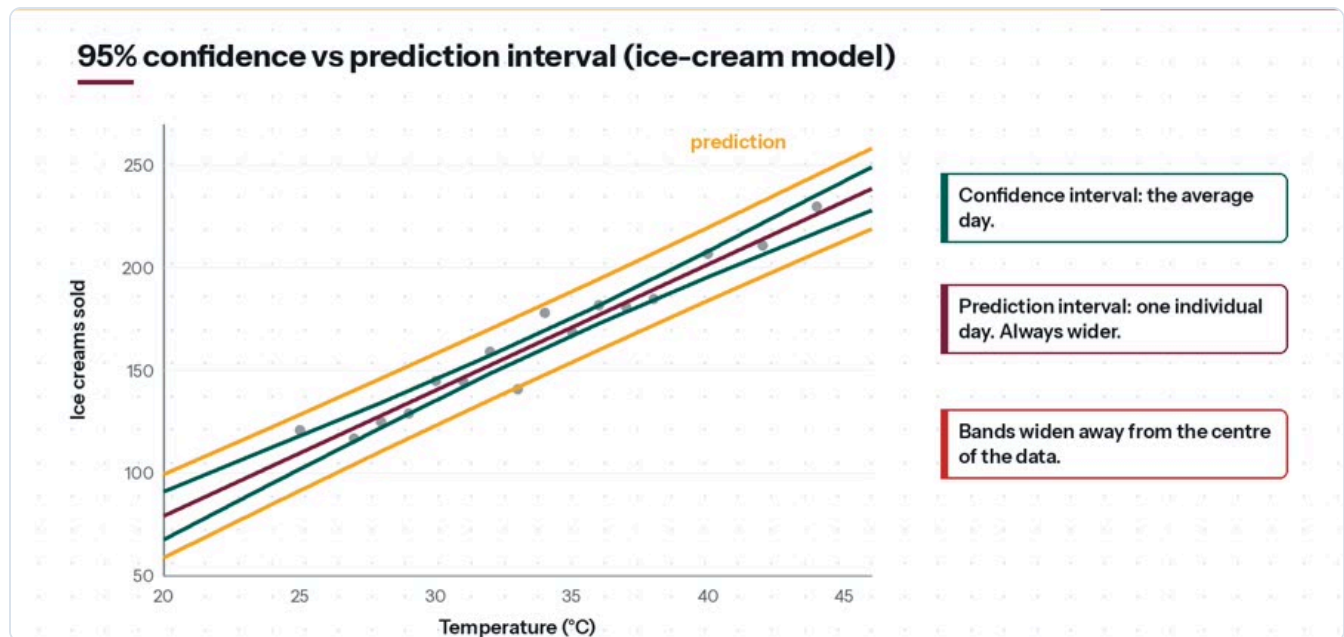
In scikit-learn, the only change from AI-15 is that X now holds several columns. Fit the model exactly as before. Then the coefficients come back as a list, one per input, in the same order as your columns, and the intercept gives the constant.

A single predicted number, called a point estimate, is almost never...

PREDICTION INTERVALS	
CONFIDENCE INTERVAL	PREDICTION INTERVAL
Average / Mean	One Individual
Narrower	Wider
Less uncertainty	More uncertainty
Confidence interval is mainly used when we want the average value of Y.	
Prediction interval is used when we want the possible value for one individual observation.	

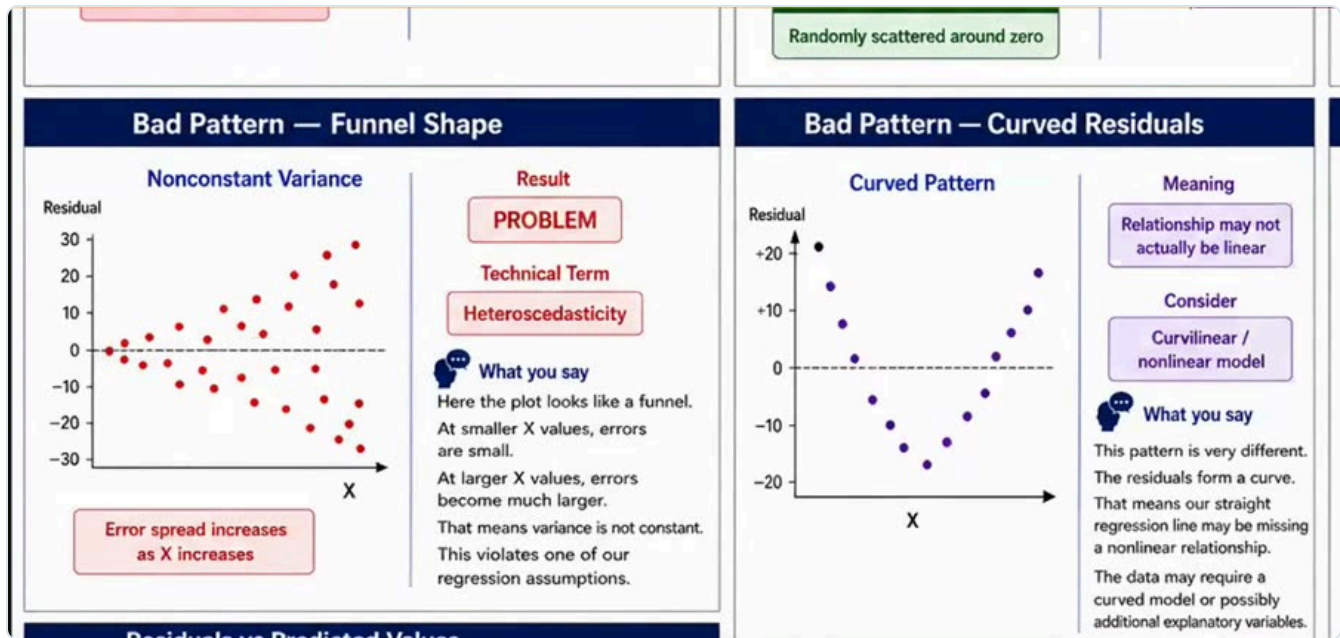
A single predicted number, called a point estimate, is almost never exactly right. So we give an interval instead. A confidence interval describes the average value of y for a given x . It's narrower, with less uncertainty. A prediction interval describes where one individual new observation might fall. It's wider, because individuals vary around the average.

95% confidence vs prediction interval (ice-cream model)



Here's what that looks like for the ice-cream model from AI-15. The gray dots are the training days, and the regression line runs through the middle. The narrow inner band is the confidence interval for the average day. The wide outer band is the prediction interval for a single day. Both bands are narrowest near the middle of the data, and widen toward the edges, where we have less information.

Before trusting any of these numbers, check the residuals, exactly ...



Before trusting any of these numbers, check the residuals, exactly as in AI-14. The trainer's summary slide puts it simply. A random cloud or horizontal band is good. A funnel means a variance problem. A curve means a linear model may be wrong.

Key takeaways

- Several inputs: $y = b_0 + b_1x_1 + b_2x_2$
- Read each coefficient holding others constant
- A big R-squared gap = a missing variable
- Confidence = average; prediction = one new case

Practice

1. Using the delivery model (0.0611 per distance unit, 0.9234 per delivery), how much longer would a route with 3 extra deliveries take, holding distance constant?
2. Write the scikit-learn code to fit a model predicting house price from size, bedrooms and age, then print the coefficients.
3. For a forecast you make at work, decide whether a confidence interval or a prediction interval is the right range to report, and explain why.

Check your understanding

1. What does "holding the others constant" mean for a coefficient?
2. R-squared rose from 0.66 to 0.90 after adding deliveries. What does that suggest?
3. Which interval is wider, confidence or prediction?
4. What does a curved residual plot suggest?

Answers at the back of this handbook.

From Regression to Classification: Logistic Regression

Video: 17_AI-17_Logistic_Regression.mp4 · 4:04 · Stage: Modeling · Audience: Learners who completed AI-14 to AI-16

Learning objectives

- Recognise binary classification problems
- Explain why a straight line fails for yes/no
- Read the sigmoid, threshold and decision boundary

The trainer's slide sums up the difference

Whenever our target or dependent variable has two possible outcomes, logistic regression is one important model we can use.

Linear Regression vs Logistic Regression	
LINEAR REGRESSION	LOGISTIC REGRESSION
Predict numerical value	Predict probability for a class
Salary	Buy / Not Buy
House price	Fraud / Not Fraud
Temperature	Churn / Not Churn
Output can vary continuously	Probability stays between 0 and 1

The trainer's slide sums up the difference. Linear regression predicts a numerical value, like a salary, a house price, or a temperature, and its output can vary continuously. Logistic regression predicts the probability of a class, like buy or not buy, fraud or not fraud, churn or not churn. Its output always stays between zero and one.

Here's the example we'll use, from the trainer's notebook

Simple Classification: Will the Student Pass?

The story: teaches an evening class. One week before the exam, she wants to know which student give them extra coaching.

For her last batch she wrote down two things for each student: **how many hours they studied**, and

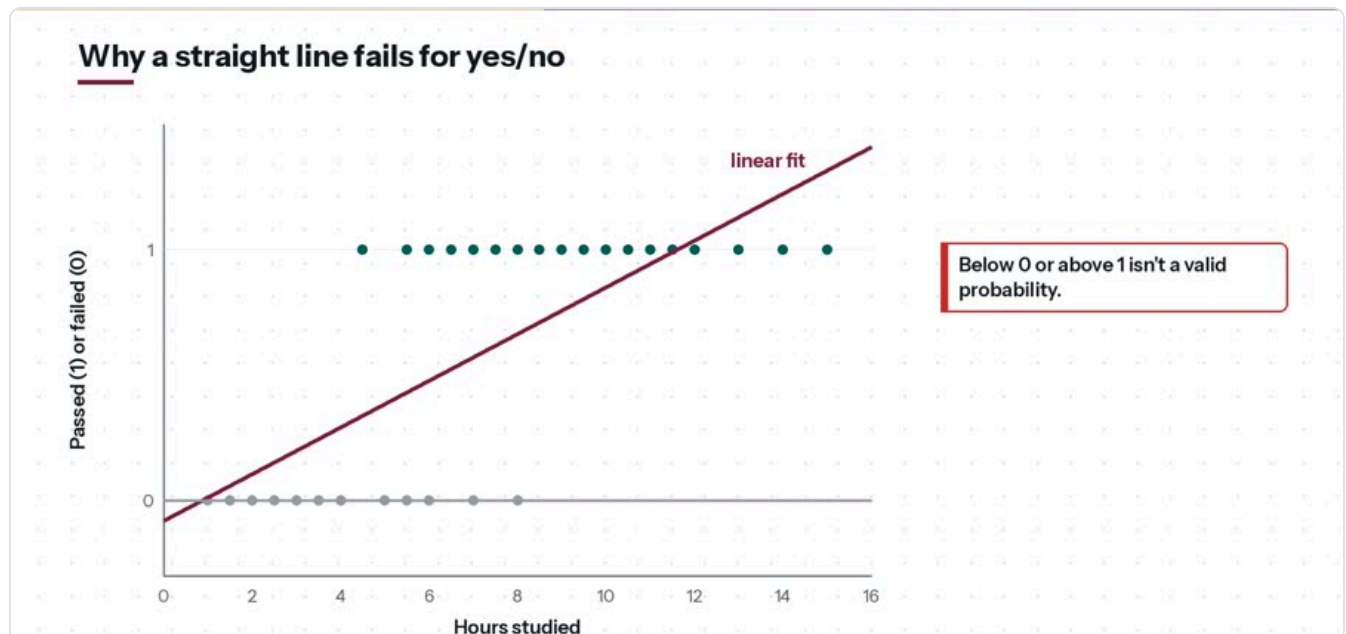
Our job: build a model that looks at the study hours of a new student and answers a yes or no que

- Input (X) = hours studied
- Output (y) = 1 for pass, 0 for fail

This is **classification**. The answer is a category (pass or fail), not a number. We use a model called "regression" in it, but it is a classification model.

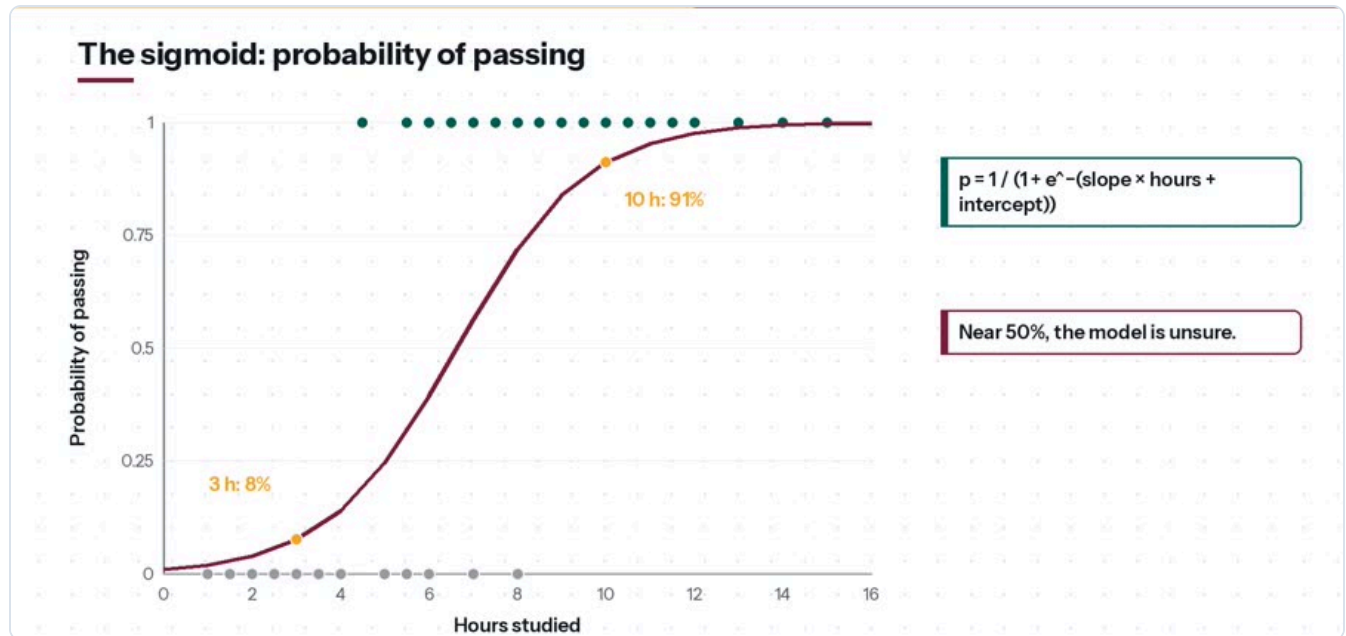
Here's the example we'll use, from the trainer's notebook. A teacher recorded how many hours each student studied, and whether they passed or failed. The input X is hours studied. The output y is one for pass and zero for fail. The answer is a category, not a number, so this is classification.

Why a straight line fails for yes/no



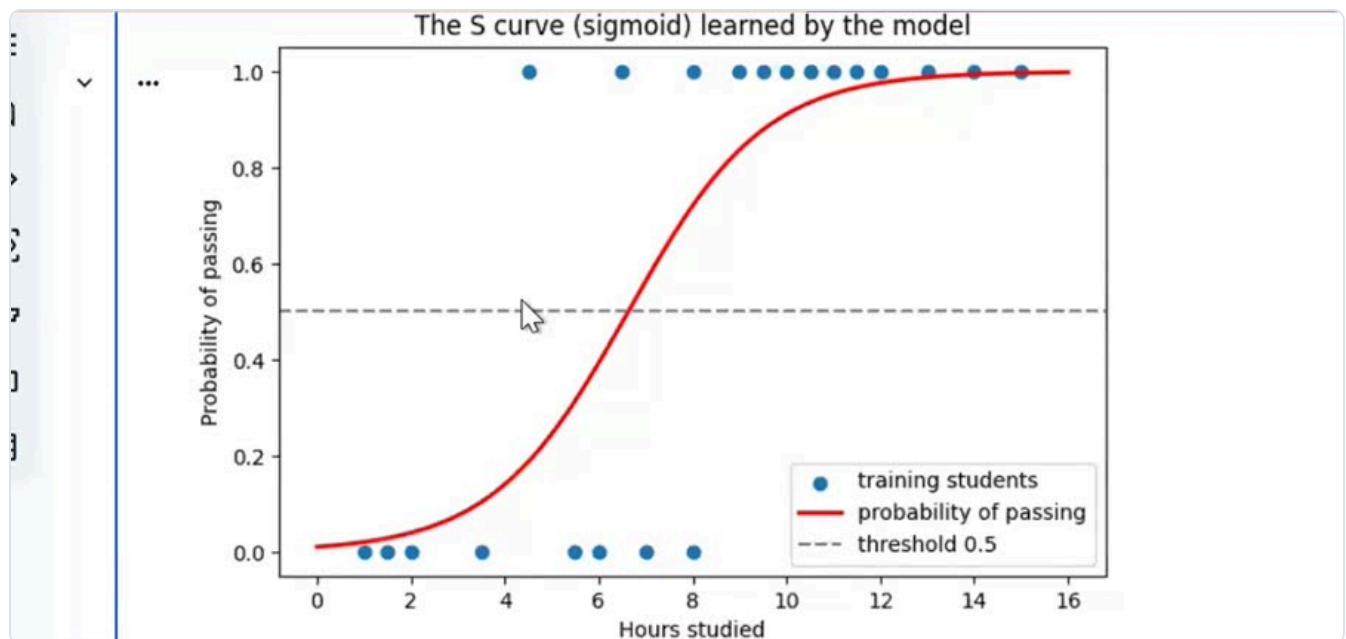
Why not just use linear regression? Plot the 30 students: fails sit along zero, passes along one. A straight line through them keeps climbing. After about 12 hours it rises past one, and at 15 hours it predicts about 1.3. A probability of 130% makes no sense, and with fewer hours the line can drop below zero.

The sigmoid: probability of passing



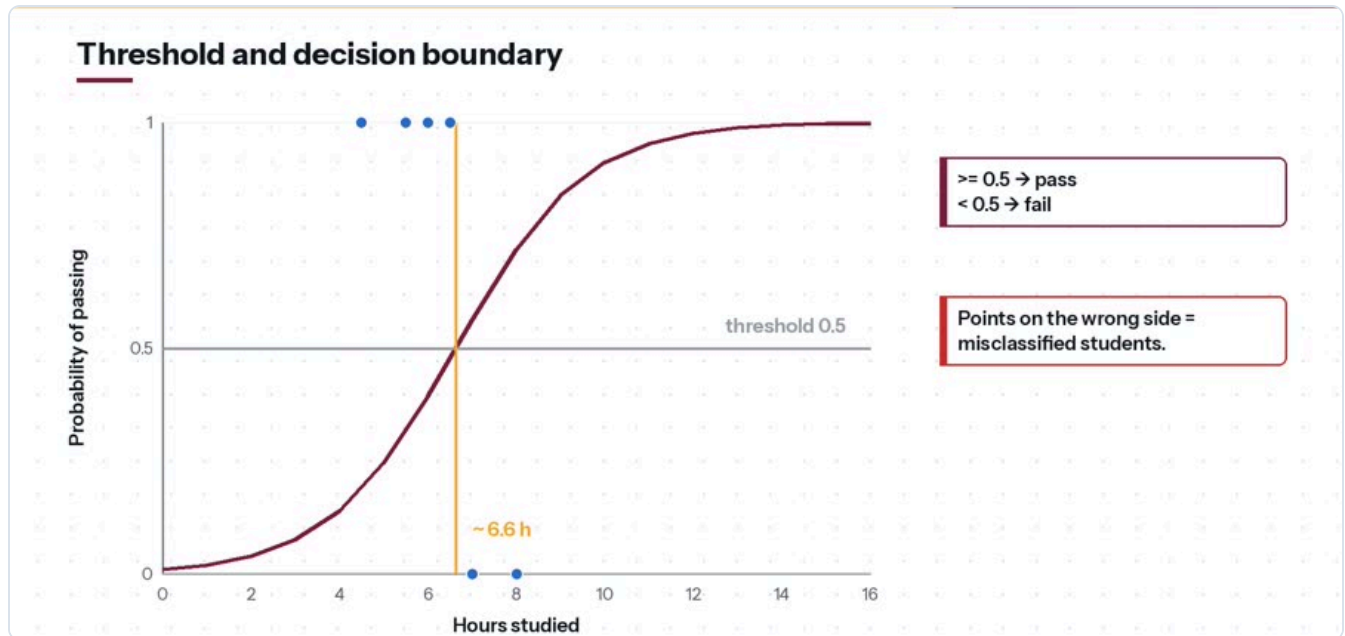
Logistic regression bends the line into an S-shaped curve, called the sigmoid. It takes the same linear score, slope times hours plus intercept, and squeezes it so the output always lies between zero and one. Read the curve as the probability of passing. At three hours, it's under 10%. At ten hours, it's over 90%. In the middle, the model is unsure.

Here's the real curve from the trainer's notebook



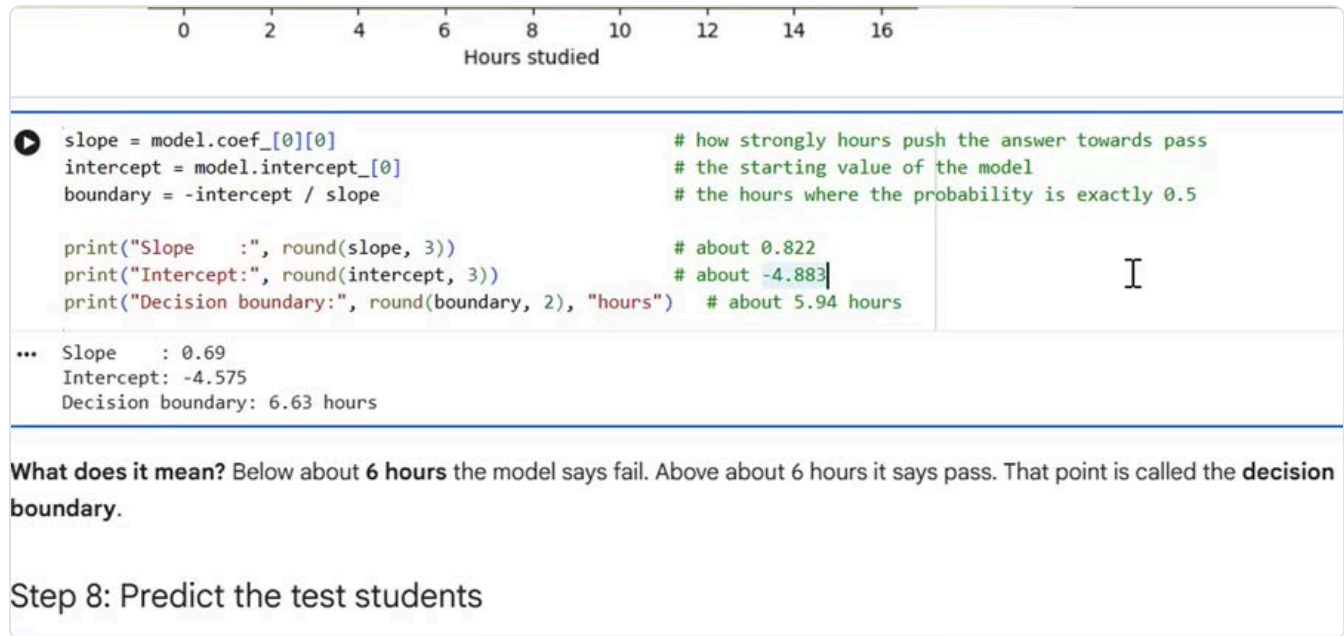
Here's the real curve from the trainer's notebook. The blue dots are the training students, at zero for fail and one for pass. The red S-curve is the probability of passing that the model learned. The dashed line is the threshold of 0.5, and the curve crosses it a little past six hours.

Threshold and decision boundary



To turn a probability into a yes or no, pick a threshold, usually 0.5. Above it, predict pass; below it, predict fail. The input where the curve crosses the threshold is the decision boundary. For this model, it's about six and a half hours. Six of the 30 students land on the wrong side. Those are the model's mistakes, which we'll count in AI-18.

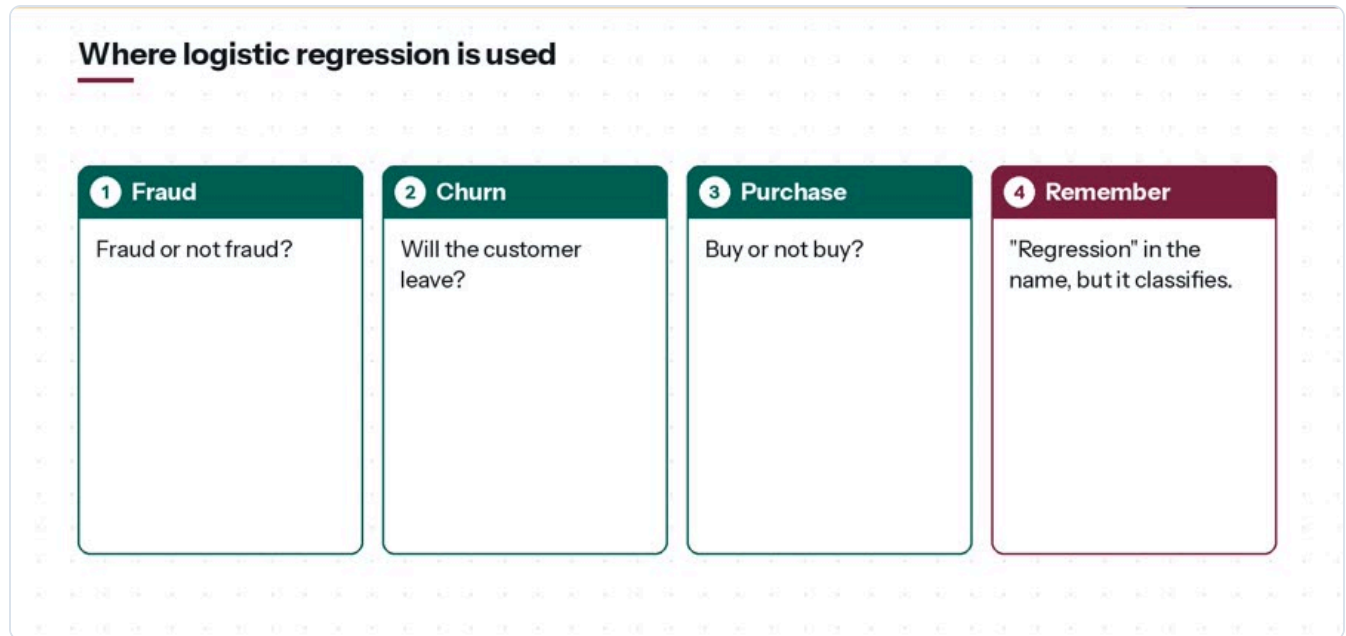
The notebook calculates it exactly



The notebook calculates it exactly. The decision boundary is minus the intercept divided by the slope. With a slope of 0.69 and an intercept of minus 4.575, the boundary is 6.63 hours. Below that, the model says fail. Above it, pass.

Tip Trust the printed output: $\text{boundary} = 4.575 \div 0.69 = 6.63 \text{ h}$. The '# about' comments are left over from an earlier run.

Where logistic regression is used



Logistic regression is a workhorse for binary problems everywhere. Banks use it to flag likely fraud. Subscription businesses use it to predict churn. Marketers use it to predict whether a customer will buy. And despite the word regression in its name, it's a classification model.

Key takeaways

- Yes/no outcomes need classification
- The sigmoid outputs a probability (0 to 1)
- A threshold turns probability into a decision
- The decision boundary is where the answer flips

Practice

1. List three yes/no questions from your own work that could be answered with logistic regression. For each, name the input columns you would use.
2. Using $p = 1 / (1 + e^{-(0.69 \times \text{hours} - 4.575)})$, calculate the probability of passing for 5 and 8 hours of study.

Check your understanding

1. Why can't linear regression predict probabilities well?
2. What range does the sigmoid output?
3. What is the decision boundary?
4. Is logistic regression used for regression or classification?

Answers at the back of this handbook.

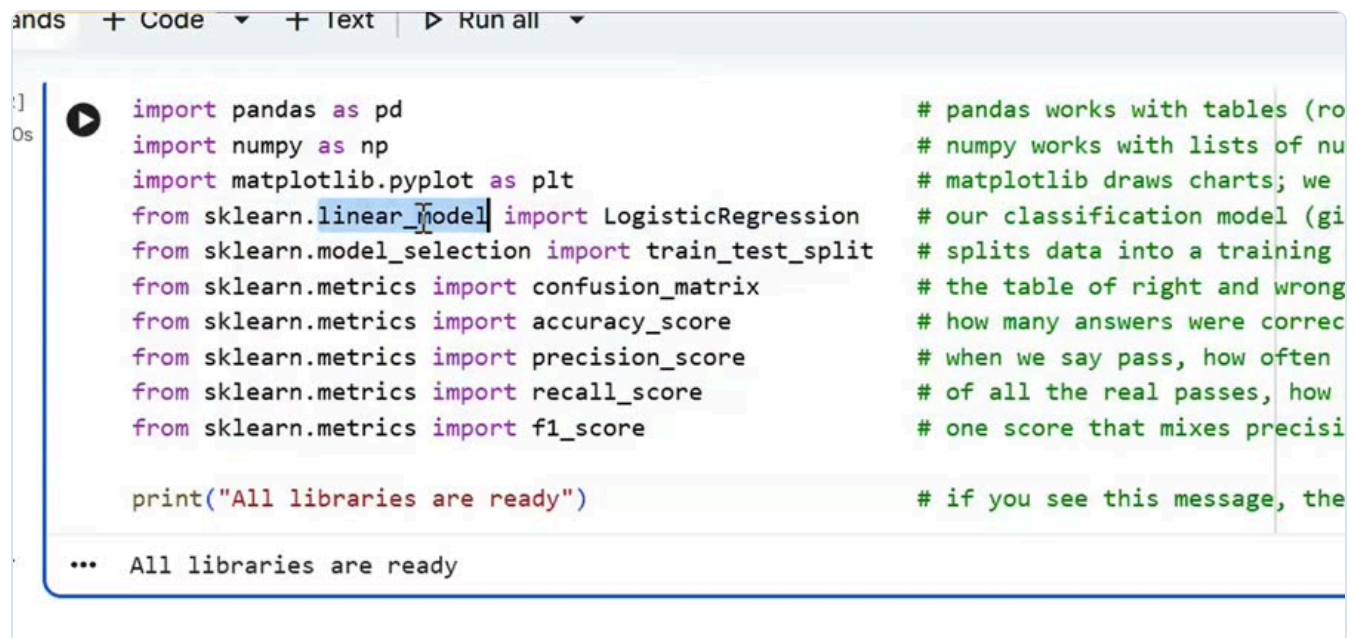
Hands-on: Pass/Fail Classifier & Evaluation Metrics

Video: 18_AI-18_Pass_Fail_Classifier_and_Metrics.mp4 · 4:28 · Stage: Modeling · Audience: Learners who completed AI-15 and AI-17

Learning objectives

- Fit a LogisticRegression classifier
- Read a confusion matrix; compute precision, recall, F1
- Tune the threshold for the cost of each error

The imports look familiar from AI-15, with a few new names



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

print("All libraries are ready")

... All libraries are ready
```

pandas works with tables (ro
numpy works with lists of nu
matplotlib draws charts; we
our classification model (gi
splits data into a training
the table of right and wrong
how many answers were correc
when we say pass, how often
of all the real passes, how
one score that mixes precisi
if you see this message, the

The imports look familiar from AI-15, with a few new names. LogisticRegression is the classification model. train_test_split divides the data, as before. And instead of regression errors, we bring in classification metrics: the confusion matrix, accuracy, precision, recall, and the F1 score.

Split, fit and predict

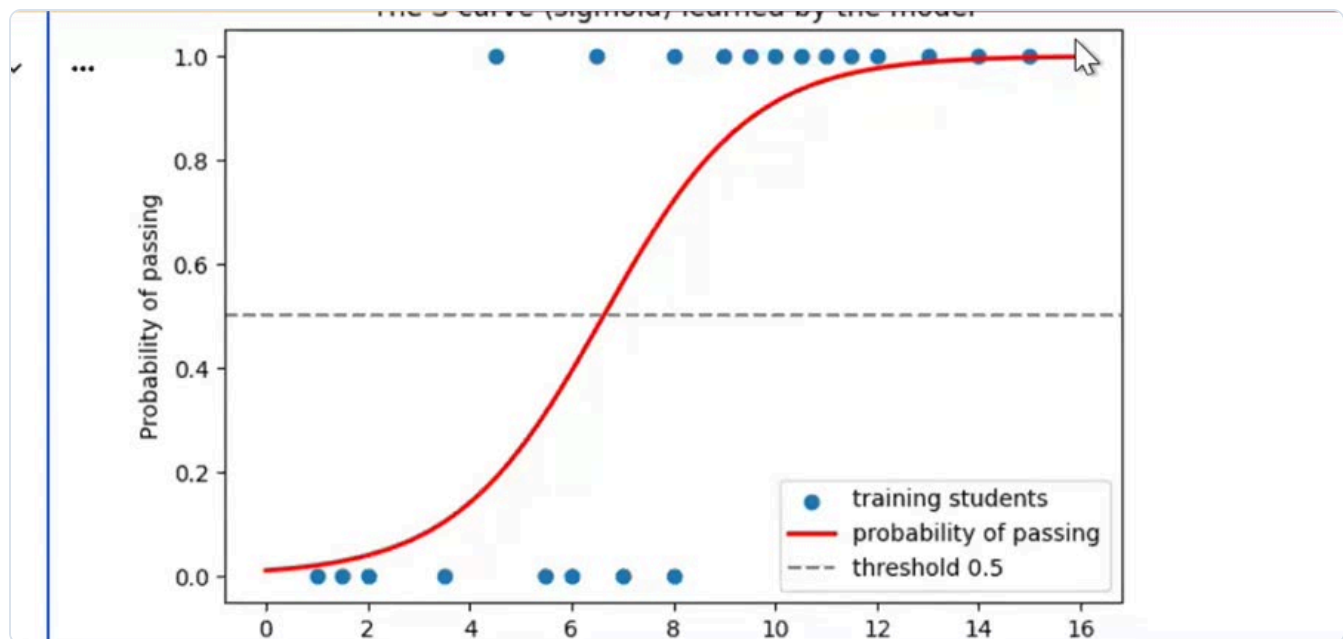
Split, fit and predict

```
1 X = data[["hours_studied"]]
2 y = data["passed"]          # 1 = pass, 0 = fail
3
4 X_train, X_test, y_train, y_test = train_test_split(
5     X, y, test_size=0.3, random_state=42)
6
7 model = LogisticRegression()
8 model.fit(X_train, y_train)
9
10 y_prob = model.predict_proba(X_test)[: , 1] # P(pass)
11 y_pred = model.predict(X_test)              # 0 or 1
```

predict_proba gives probabilities;
predict applies the 0.5 threshold.

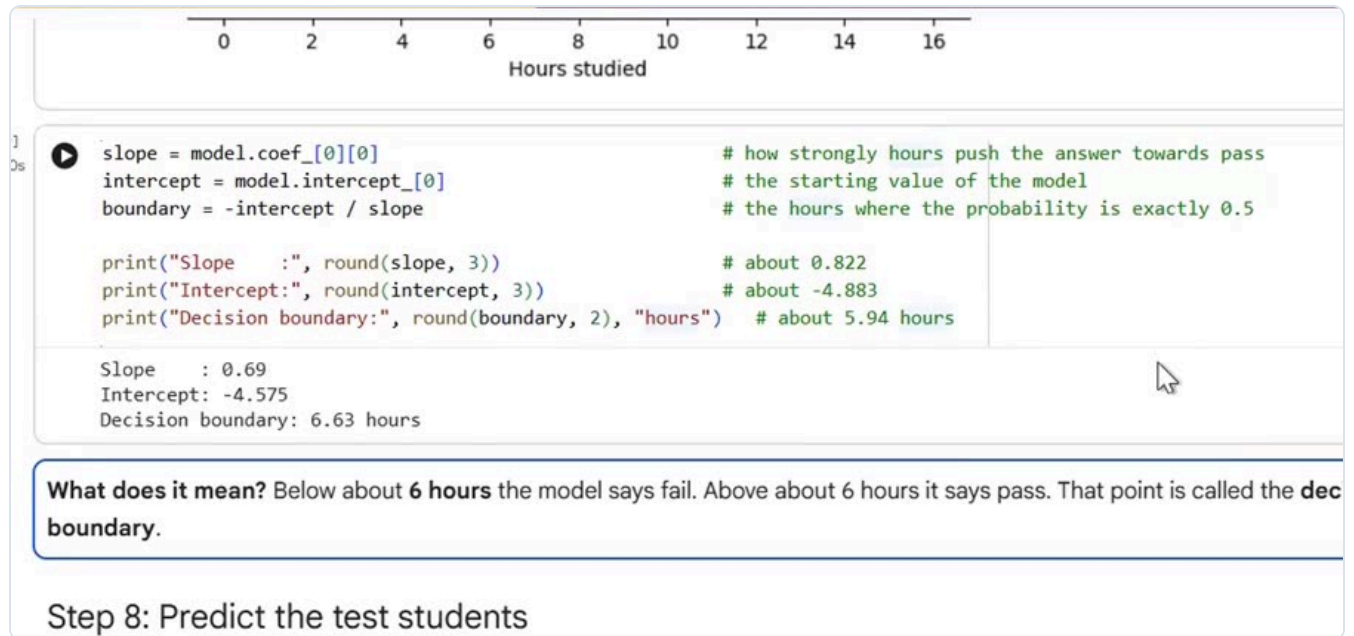
The steps match AI-15. Put hours studied in X, and pass or fail in y. Hold back thirty percent as a test set; the trainer's rule is eighty-twenty or seventy-thirty, never fifty-fifty. Then create the model and fit it. One new method matters here: predict underscore proba returns the probability of each class, not just the final answer.

Here's what the model learned



Here's what the model learned. The blue dots are the training students, at zero for fail and one for pass. The red S-curve is the probability of passing, and the grey dashed line is the zero point five threshold. Notice the overlap in the middle: some students who studied five to eight hours passed, and some failed.

From the slope and intercept, the notebook calculates the decision ...



From the slope and intercept, the notebook calculates the decision boundary: minus the intercept divided by the slope. The output says about six point six hours. Below that, the model predicts fail. Above it, pass.

Now the test students, sorted by hours

...	hours	actual	predicted	probability
3	2.5	0	0	0.05
4	3.0	0	0	0.08
6	4.0	0	0	0.14
8	5.0	0	0	0.24
10	5.5	1	0	0.31
12	6.0	1	0	0.39
15	7.0	1	1	0.56
16	7.5	1	1	0.65
19	8.5	1	1	0.78

TIP
Probabilities 0.31 and 0.39 are below 0.5, so both were predicted fail.

Now the test students, sorted by hours. Each row shows the actual result, the prediction, and the probability behind it. Most rows match. But two students who actually passed, at five and a half and six hours, got probabilities below zero point five, so the model predicted fail.

Tip Probabilities 0.31 and 0.39 are below 0.5, so both were predicted fail.

Confusion matrix (9 test students, threshold 0.5)

Confusion matrix (9 test students, threshold 0.5)

	Predicted fail	Predicted pass
Actually failed	TN = 4 (correct)	FP = 0 (said pass, failed)
Actually passed	FN = 2 (said fail, passed)	TP = 3 (correct)

TN, FP, FN and TP come from the predictions table in the notebook.

A confusion matrix counts every kind of answer. True negatives: the student failed, and we said fail. There are four. False positives: we said pass, but the student failed. There are none. False negatives: we said fail, but the student passed. There are two. And true positives: the student passed, and we said pass. There are three.

Metrics from the confusion matrix

Metrics from the confusion matrix			
1 Accuracy	2 Precision	3 Recall	4 F1
$(TP + TN) \div \text{all}$	$TP \div (TP + FP)$	$TP \div (TP + FN)$	$2 \times P \times R \div (P + R)$
$7 \div 9 \sim 0.78$	$3 \div 3 = 1.0$	$3 \div 5 = 0.6$	~ 0.75

From those four numbers come the key metrics. Accuracy is the share of all answers that were right: seven out of nine, about seventy-eight percent. Precision asks: when the model says pass, how often is it right? Three out of three, so one point zero. Recall asks: of the students who really passed, how many did we catch? Three out of five, so zero point six. F1 balances precision and recall, at about zero point seven five.

Which mistake costs more?

Which mistake costs more?		
Situation	False positive	False negative
Coaching at-risk students ("pass" = positive)	Said pass, student fails: no help given (costly)	Said fail, student passes: extra coaching not needed
Medical screening ("has condition" = positive)	Healthy person sent for more tests	Real case missed (usually costly)
Accuracy alone can hide the mistakes that matter.		

Which mistake costs more depends on the business. In the trainer's story, the teacher wants to find students who need extra coaching. Predicting pass for a student who will fail means a struggling student gets no help. In a medical lab test, missing a real case is usually the expensive error. Always ask which error hurts more before you judge a model.

You control the trade-off with the threshold

```
print("At 0.5 -> recall:", round(recall_score(y_test, y_pred), 2),  
      " precision:", round(precision_score(y_test, y_pred), 2))      # the old scores  
print("At", threshold, "-> recall:", round(recall_score(y_test, y_pred_new), 2),  
      " precision:", round(precision_score(y_test, y_pred_new), 2))  # the new scores  
compare                      # show the comparison
```

```
... At 0.5 -> recall: 0.6 precision: 1.0  
At 0.2 -> recall: 1.0 precision: 0.83
```

	hours	probability	answer_at_0.5	answer_at_new	actual
3	2.5	0.05	0	0	0
4	3.0	0.08	0	0	0
6	4.0	0.14	0	0	0
8	5.0	0.24	0	1	0
10	5.5	0.31	0	1	1
12	6.0	0.39	0	1	1
15	7.0	0.56	1	1	1

You control the trade-off with the threshold. The trainer lowered it from zero point five to zero point two, so any student with at least a twenty percent chance is predicted to pass. Recall rose to one point zero, catching every student who passed, while precision fell to zero point eight three. Lowering the threshold catches more positives, at the cost of more false alarms. Raise it instead when a false alarm is the costly mistake, as in the coaching story.

Tip At 0.2 the 5-hour student (probability 0.24) is now predicted pass but actually failed: one false positive.

Key takeaways

- Fitted LogisticRegression; read the S-curve
- Counted TP, TN, FP, FN in a confusion matrix
- Calculated accuracy, precision, recall, F1
- Tuned the threshold to the cost of each mistake

Practice

1. Using the predictions table in the video, recompute the confusion matrix and all four metrics at a threshold of 0.2.
2. Pick a classification problem from your work (fraud, churn, approvals). Decide whether a false positive or a false negative costs more, and whether you would raise or lower the threshold.

Check your understanding

1. What does precision measure?
2. What does recall measure?
3. Recall was 0.6 at threshold 0.5. What happened when the threshold dropped to 0.2?
4. Why can accuracy be misleading?

Answers at the back of this handbook.

Clustering & K-Means Concepts

Video: 19_AI-19_Clustering_and_K-Means_Concepts.mp4 · 3:44 · Stage: Modeling · Audience: Learners who completed the supervised-learning modules (AI-14 to AI-18)

Learning objectives

- Contrast clustering with classification
- Describe the four K-Means steps
- Explain why distance-based models need scaling

In supervised learning, every row comes with a label, and the model...

Supervised: every row has a label, the model learns the line

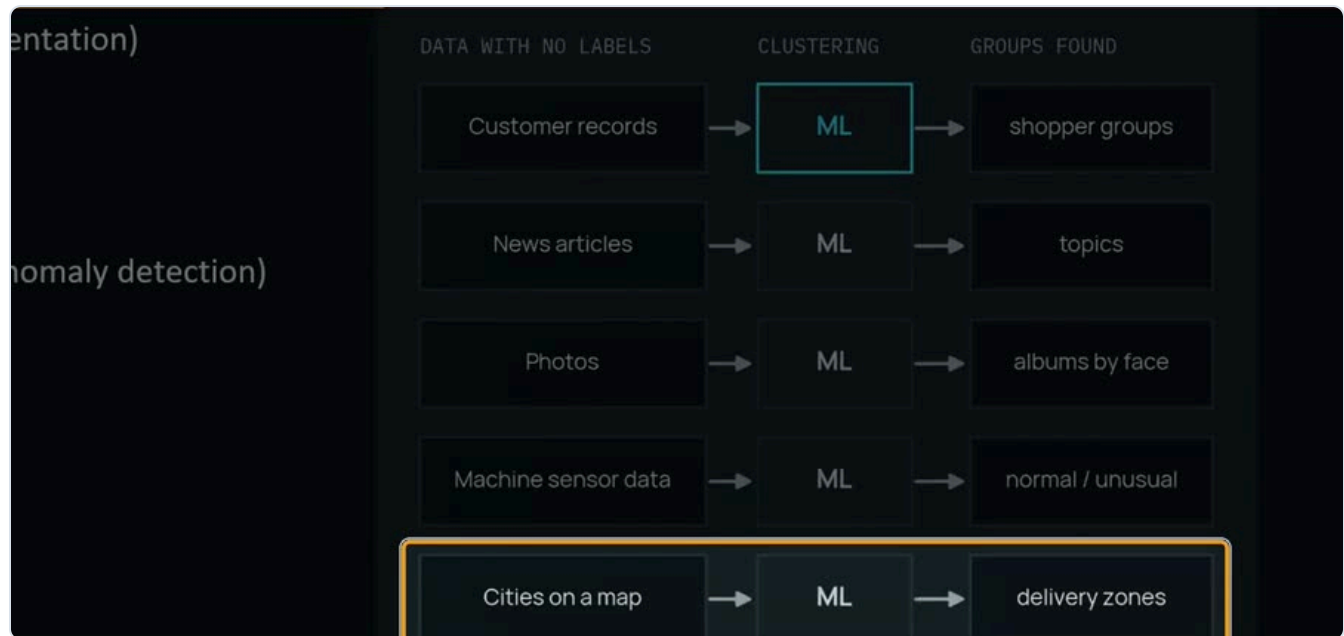
Unsupervised: no labels, the model finds the structure

New question: not "what is the answer" but "what groups exist"

No accuracy score here: there is nothing to compare with

In supervised learning, every row comes with a label, and the model learns the line. In unsupervised learning there are no labels at all; the model finds the structure. The question changes from what is the answer, to what groups exist. And there's no accuracy score to check against, because there's no right answer to compare with.

Clustering is everywhere



Clustering is everywhere. Shops group customers into segments for targeted offers. News sites group articles into topics. Photo apps group pictures by face. Factories flag unusual sensor readings, which is anomaly detection. And delivery firms group addresses into zones.

Don't confuse clustering with classification

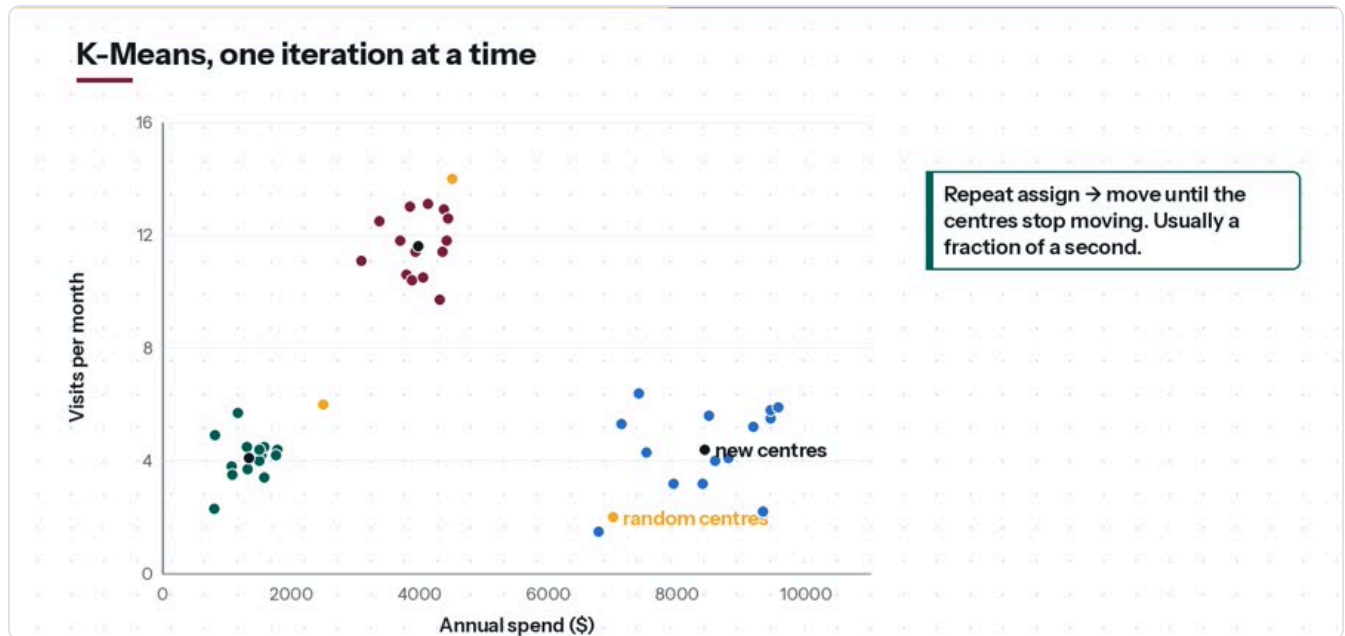
Classification: the groups are already known

Clustering: the groups have to be discovered

They work together: cluster once to find the segments, then classify new rows into them

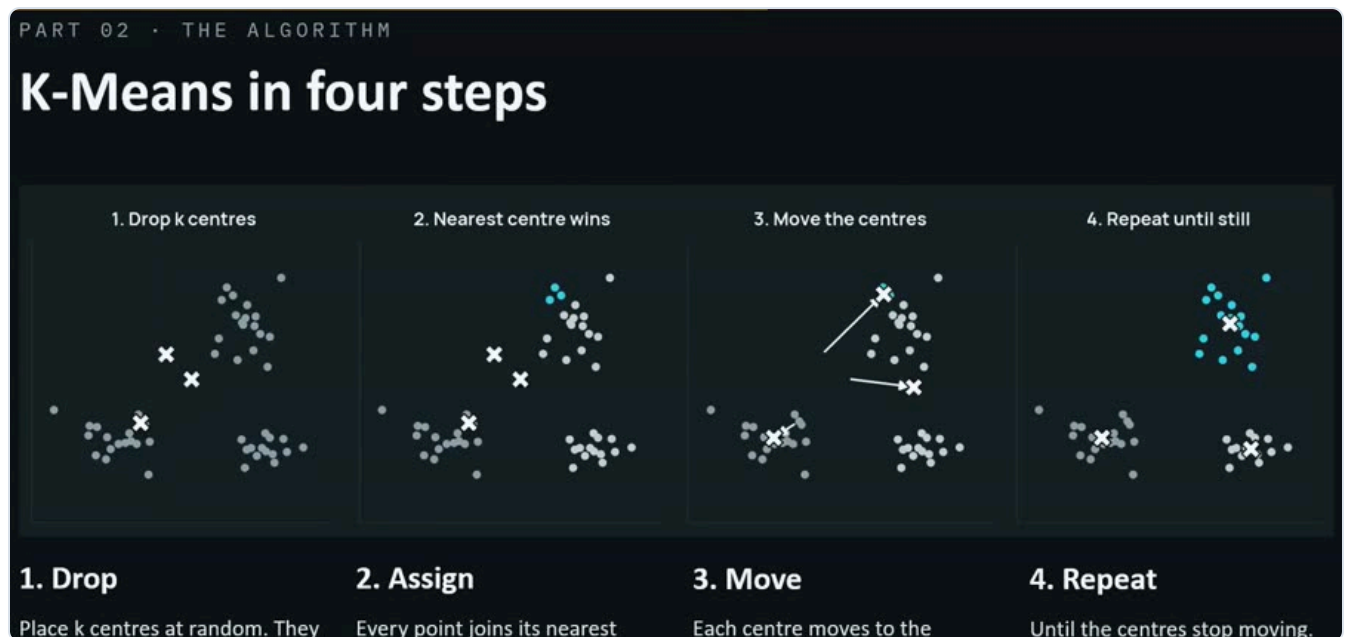
Don't confuse clustering with classification. In classification, the groups are already known, and the model puts each new item into an existing box. In clustering, the groups have to be discovered first: clustering builds the boxes. They also work together: cluster once to find the segments, then train a classifier to place new customers into them.

K-Means, one iteration at a time



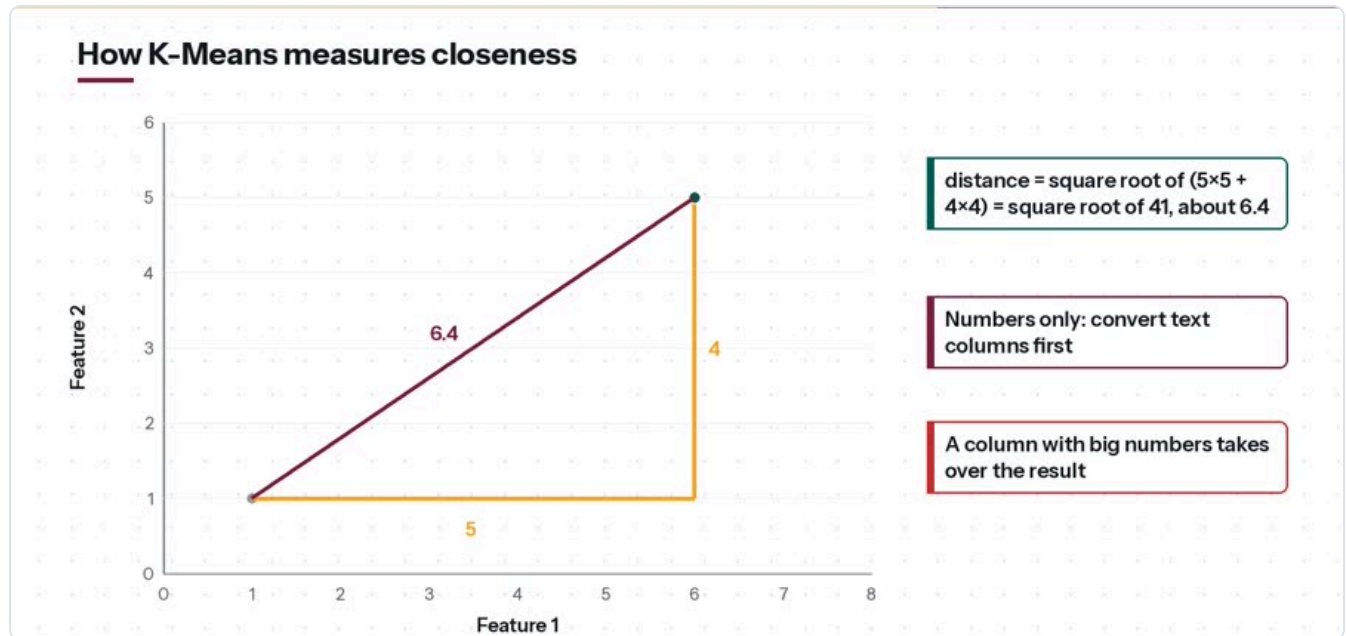
Here's K-Means on forty-two customers, plotted by annual spend and visits per month. At first the points are just unlabelled points. Step one: drop three centres at random. Step two: each customer joins its nearest centre. Step three: each centre moves to the middle of its group. Then repeat, until the centres stop moving.

The trainer's slide sums it up in four steps



The trainer's slide sums it up in four steps. Drop k centres. Nearest centre wins. Move the centres. Repeat until still. Notice that you choose k, the number of groups. K-Means never decides that for you.

How K-Means measures closeness



How does K-Means decide what's nearest? It uses plain straight-line distance, the formula you learned at school. Here, one point is five across and four up from another, so the distance is the square root of five squared plus four squared, about six point four. That has two consequences. Every column must be a number, so text must be converted first. And everything depends on the size of your numbers.

Here's why that matters

With scaling: both columns count, and a third group appears

THE RULE

Always scale before you cluster.

One line: `StandardScaler()`

Here's why that matters. Spend runs from about four hundred to ten thousand four hundred dollars, while visits run from two to fourteen. Without scaling, spend decides everything and visits are ignored. With scaling, both columns count, and a third group appears. The rule: always scale before you cluster, with one line of `StandardScaler`.

Four K-Means traps

Four K-Means traps

1 You pick k

Use the elbow method and silhouette score (AI-20).

2 Random starts

Run several starts (`n_init`) and keep the best result.

3 Outliers

Extreme points drag centres away. Check and clean first.

4 Round blobs

Odd shapes or densities? Try DBSCAN or hierarchical clustering.

Keep four traps in mind. You must choose `k` yourself; the next module shows the elbow and silhouette methods. Random starting centres can give different answers, so run several starts and keep the best. Outliers pull centres towards them. And K-Means expects round, similar-sized blobs; for odd shapes, methods like DBSCAN work better.

Key takeaways

- Clustering finds groups without labels
- Drop → assign → move → repeat
- Distance-based: always scale first

Practice

1. List three business problems in your organisation that could use clustering, and say what columns you would cluster on.
2. On paper, run one K-Means iteration for six points and two starting centres: assign each point, then move each centre to the mean.

Check your understanding

1. What is the key difference between clustering and classification?
2. Name the four K-Means steps.
3. Why must you scale before K-Means?
4. Who decides k in K-Means?

Answers at the back of this handbook.

Hands-on: Customer Segmentation with K-Means

Video: 20_AI-20_Customer_Segmentation_with_K-Means.mp4 · 5:17 · Stage: Modeling · Audience: Learners who completed AI-19 (clustering concepts) and the Python modules

Learning objectives

- Standardise features with StandardScaler
- Choose k with the elbow and silhouette methods
- Name segments and assign new customers

Here's the scenario

Simple Clustering: Who Are My Customers?

The story: Meera runs an online store. She wants to send offers by email, but sending the same offer to everybody does not work. spenders ignore small discounts, and budget shoppers ignore expensive products.

She has a list of 42 customers with two facts about each one: **how much they spent last year**, and **how many times they visit the shop in a month**.

Nobody has told her which customer belongs to which group. There are **no labels** in this data.

Our job: let the computer find the natural groups by itself, and then give each group a name.

- This is **unsupervised learning**. There is no right answer to learn from.
- The method we use is called **K-Means clustering**.

In Python, a comment starts with `#`. Everything after `#` on a line is only for humans. Python ignores it. Every line of code below has a comment that explains it.

Run each cell with **Shift + Enter**, from top to bottom.

Here's the scenario. An online store owner wants to send different email offers to different kinds of customers, because one offer for everybody doesn't work. She has forty-two customers, with two facts about each: how much they spent last year, and how many times a month they visit. Nobody has labelled them. That makes this unsupervised learning.

Step one imports the libraries

Step 1: Import the libraries

```
import pandas as pd # pandas works with tables (rows and columns), 1
import numpy as np # numpy works with lists of numbers
import matplotlib.pyplot as plt # matplotlib draws charts; we call it plt for sh
from sklearn.cluster import KMeans # the clustering model that finds the groups
from sklearn.preprocessing import StandardScaler # puts both columns on the same scale
from sklearn.metrics import silhouette_score # a score that tells us how clean the groups are

print("All libraries are ready") # if you see this message, the imports worked
```

All libraries are ready

Step 2: Create the data

Step one imports the libraries. pandas handles the table, and matplotlib draws the charts. From scikit-learn we bring in KMeans, the clustering model, StandardScaler, which puts both columns on the same scale, and silhouette_score, which measures how clean the groups are.

Step two builds the data

```
data = pd.DataFrame({ # make a table from the two lists
    "annual_spend": annual_spend, # first column
    "visits_per_month": visits_per_month # second column
}) # close the table

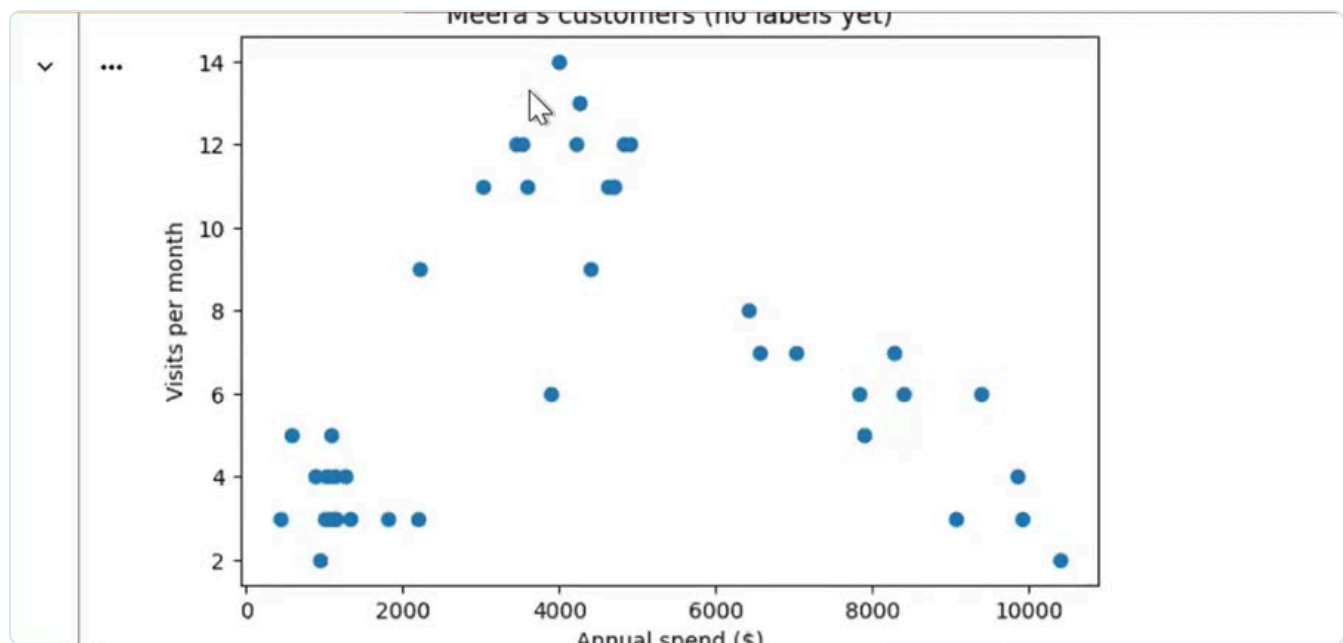
print(data.shape) # (42, 2) -> 42 customers, 2 columns
data.head() # show the first 5 customers
```

(42, 2)

	annual_spend	visits_per_month
0	1030	4
1	1330	3
2	3530	12
3	8400	6

Step two builds the data. Two lists, annual spend and visits per month, become a pandas DataFrame. The shape confirms forty-two rows and two columns, and there's no answer column at all.

Plot the customers before any modelling



Plot the customers before any modelling. You can already see the shape: a group with low spend and few visits, a group in the middle that visits often, and a group of high spenders who visit less. K-Means will find these groups for us.

Step four scales the data, and in clustering this step isn't optional

```
scaler = StandardScaler() # create the scaler
scaled = scaler.fit_transform(data) # learn the average and spread, then rescale both columns

print("Before scaling (first customer):", data.iloc[0].values) # spend in thousands, visits in single digits
print("After scaling (first customer):", scaled[0].round(2)) # both columns are now small numbers
```

```
Before scaling (first customer): [1030  4]
After scaling (first customer): [-1.06 -0.74]
```

do we need this? K-Means measures distance between points.

spend goes from about 400 to 10,400. Visits go from 2 to 14. Without scaling, a difference of 2,000 dollars looks huge and a difference of 1 visit looks like nothing, so the spend column would decide everything and the visits column would be ignored.

`StandardScaler` rescales each column so that the average becomes 0 and the spread becomes 1. Now both columns count equally.

Step 5: How many groups? The elbow method

TIP

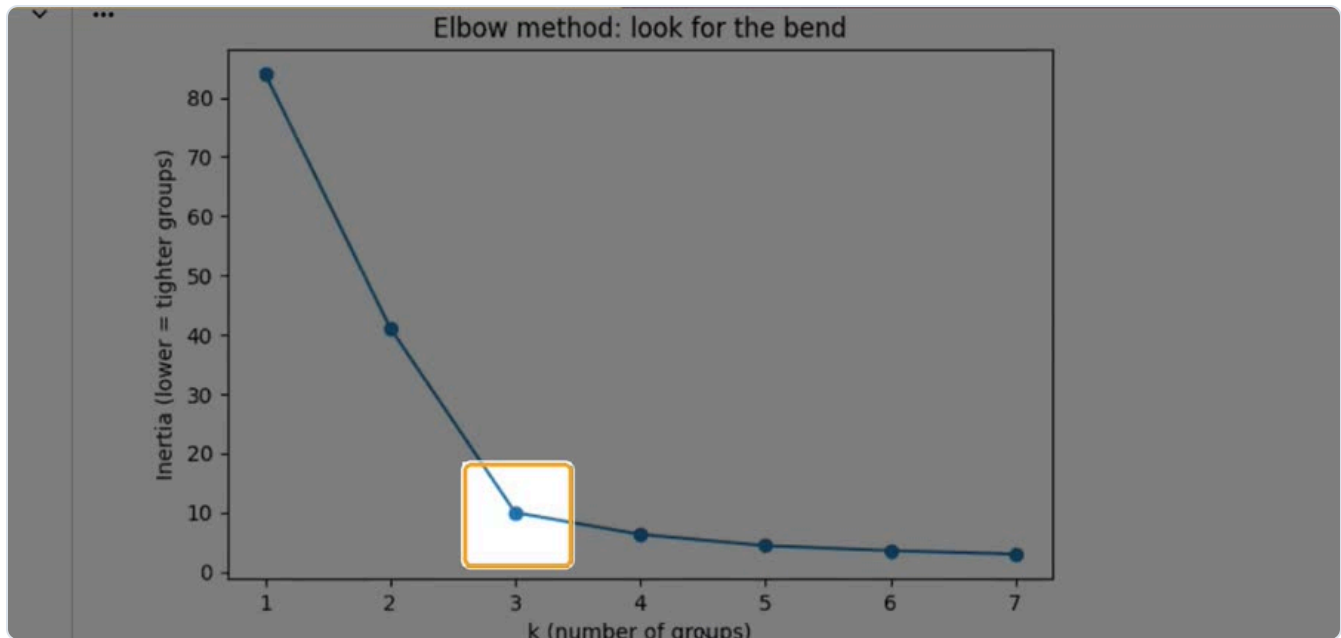
Scaled value = (value - column average) ÷ column standard deviation.

Algorithms cannot decide **k**, the number of groups. We choose it. The elbow chart helps.

Step four scales the data, and in clustering this step isn't optional. K-Means measures distance. Spend runs into thousands of dollars, while visits are single digits, so without scaling, spend would decide everything. `StandardScaler` rescales each column to an average of zero and a spread of one. The first customer, one thousand and thirty dollars and four visits, becomes minus one point zero six and minus point seven four.

Tip Scaled value = (value - column average) ÷ column standard deviation.

K-Means can't choose k, the number of groups



K-Means can't choose k, the number of groups. We choose it, with help from the elbow method. Fit the model for k from one to seven and plot the inertia, which measures how tight the groups are. Inertia always falls as k grows, so look for the bend. Here the curve drops sharply until three, then flattens.

For a second opinion, the silhouette score compares how close each ...

p 6: A second opinion, the silhouette score

```
for k in range(2, 7):  
    km = KMeans(n_clusters=k, n_init=10, random_state=42)  # the silhouette score needs at least 2 groups  
    labels = km.fit_predict(scaled)  # a model with k groups  
    score = silhouette_score(scaled, labels)  # find the groups and get the group number of each customer  
    print("k =", k, "-> silhouette", round(score, 3))  # how clean are the groups? between -1 and 1  
                                                    # print one line per k
```

```
k = 2 -> silhouette 0.544  
k = 3 -> silhouette 0.714  
k = 4 -> silhouette 0.688  
k = 5 -> silhouette 0.62  
k = 6 -> silhouette 0.617
```

to read it: the silhouette score compares how close a point is to its own group against the next nearest group. Closer to 1 is better, and above point five usually means clear grouping.

best score is at **k = 3** (about 0.71). Both methods agree, so we are confident.

p 7: Build the final model with k = 3

For a second opinion, the silhouette score compares how close each customer is to its own group versus the next nearest group. Closer to one is better, and above point five usually means clear groups. The best score here is at k equals three, about point seven one. Both methods agree, so we choose three.

Step seven builds the final model with three clusters, ten restarts...

```
model = KMeans(n_clusters=3, n_init=10, random_state=42) # 3 groups, 10 restarts, fixed randomness
data["cluster"] = model.fit_predict(scaled) # find the groups and save the group number in a new column

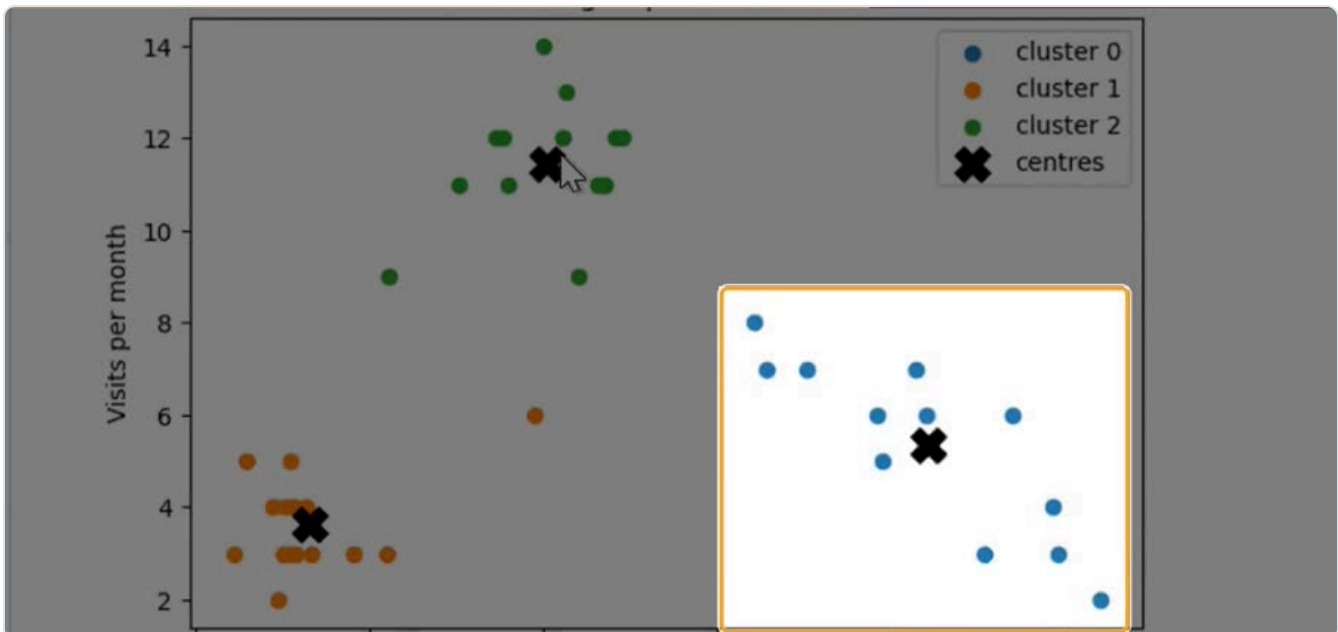
print(data["cluster"].value_counts().sort_index()) # how many customers landed in each group
data.head() # look at the table with the new column
```

```
cluster
0    12
1    16
2    14
Name: count, dtype: int64
```

	annual_spend	visits_per_month	cluster
0	1030	4	1
1	1330	3	1
2	3530	12	2
3	8400	6	0
4	4820	12	2

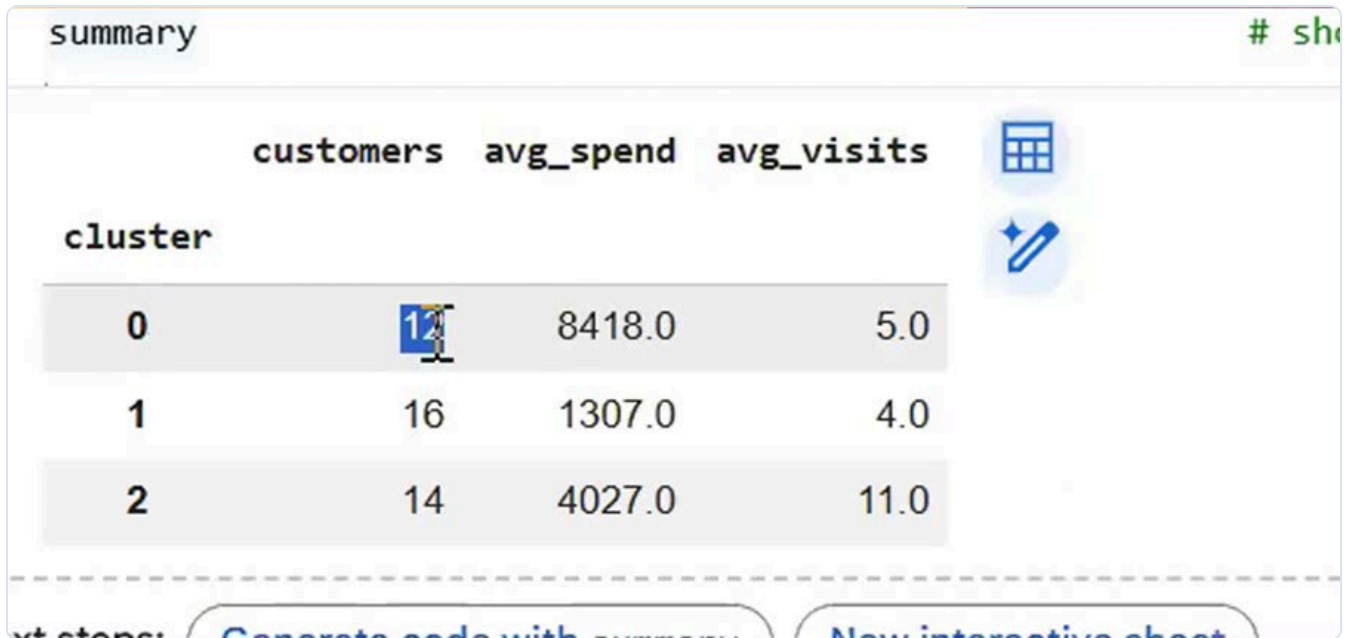
Step seven builds the final model with three clusters, ten restarts, and a fixed random state so everyone gets the same answer. `fit_predict` assigns every customer a cluster number. The counts show twelve, sixteen, and fourteen customers, and the new cluster column appears in the table.

Now plot the result



Now plot the result. Each colour is one cluster, and each black cross is a cluster centre, converted back into dollars and visits. One group spends little and visits rarely. One visits often with moderate spend. And one spends the most.

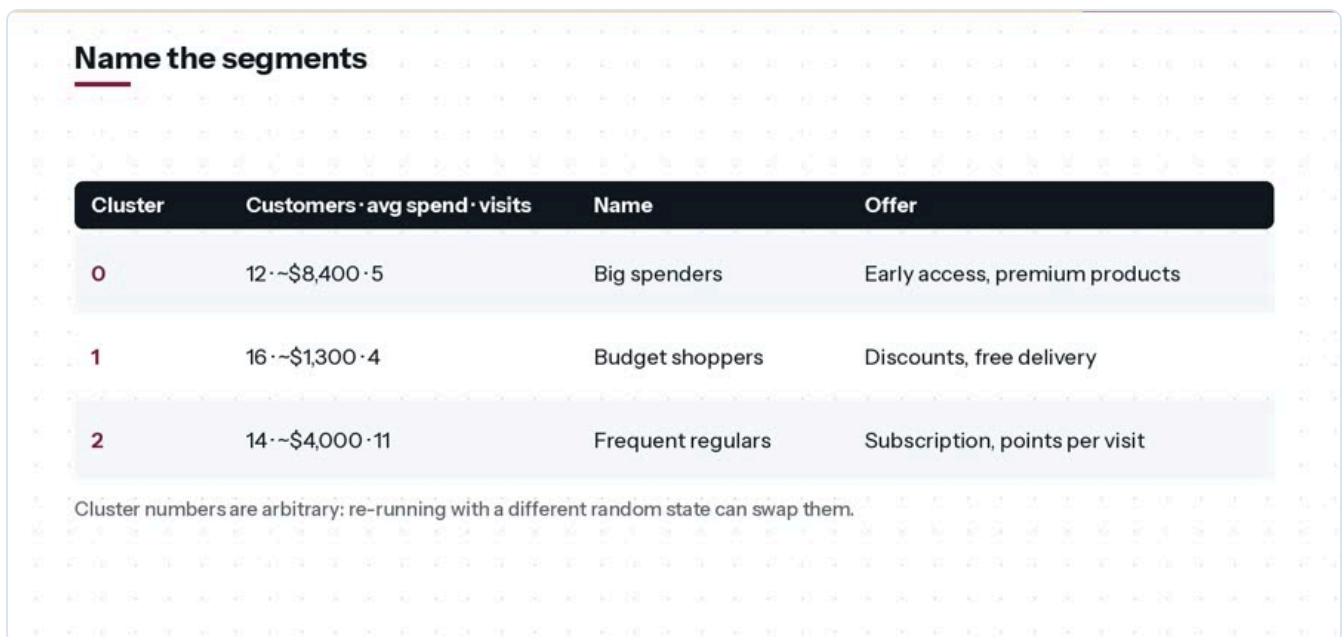
A summary table gives the facts for each cluster



cluster	customers	avg_spend	avg_visits
0	12	8418.0	5.0
1	16	1307.0	4.0
2	14	4027.0	11.0

A summary table gives the facts for each cluster. Cluster zero: twelve customers, averaging about eight thousand four hundred dollars and five visits. Cluster one: sixteen customers, about thirteen hundred dollars and four visits. Cluster two: fourteen customers, about four thousand dollars and eleven visits.

Name the segments



Cluster	Customers · avg spend · visits	Name	Offer
0	12 · ~\$8,400 · 5	Big spenders	Early access, premium products
1	16 · ~\$1,300 · 4	Budget shoppers	Discounts, free delivery
2	14 · ~\$4,000 · 11	Frequent regulars	Subscription, points per visit

Cluster numbers are arbitrary: re-running with a different random state can swap them.

Numbers alone don't help the business, so a person who knows the business names each group. Cluster zero are the big spenders: offer early access and premium products. Cluster one are budget shoppers: discounts and free delivery work best. Cluster two are frequent regulars: a subscription or points for every visit. The computer finds the groups; people explain what they mean.

Assign a new customer

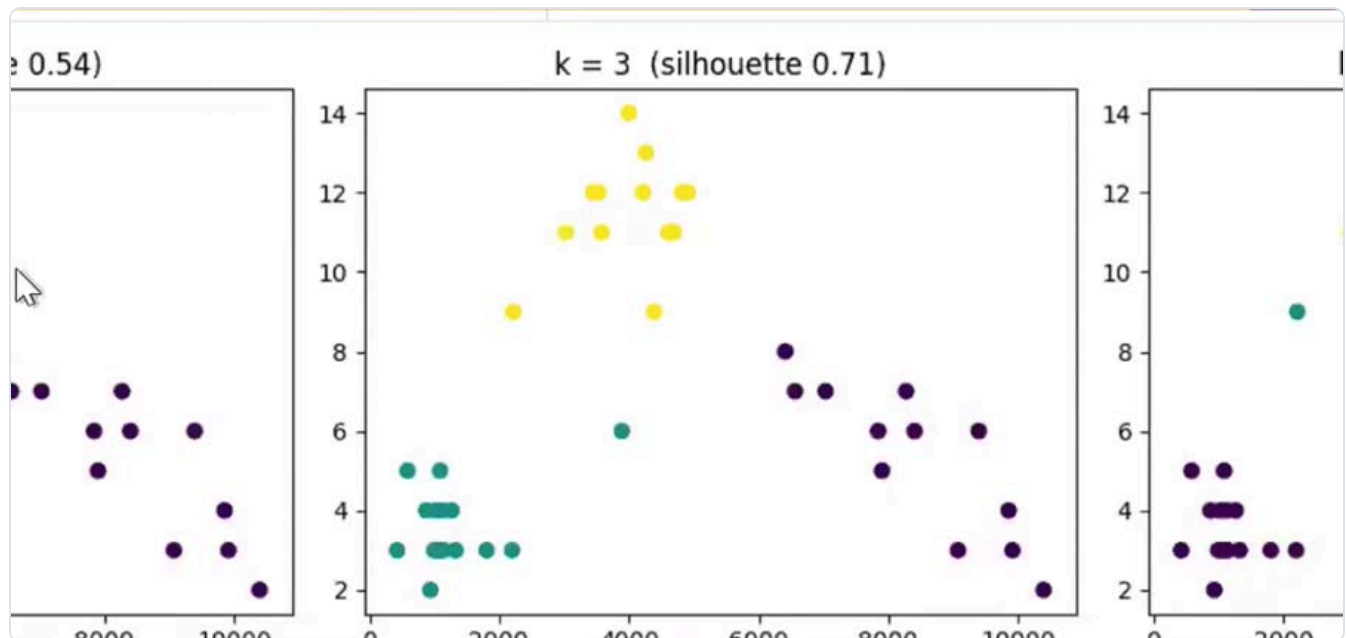
Assign a new customer

```
1 new_customer = pd.DataFrame({
2     "annual_spend": [7000],
3     "visits_per_month": [4]
4 })
5
6 new_scaled = scaler.transform(new_customer)
7 segment = model.predict(new_scaled)
8 print(segment) # → [0] big spenders
```

predict returns the nearest cluster centre.

To place a new customer, put their numbers in a one-row table, scale them with the same scaler you already fitted, and call predict. A customer who spent seven thousand dollars and visits four times a month is closest to the big spenders' centre, so they get the premium offer.

Finally, what if we pick the wrong k? With two groups, the regulars...



Finally, what if we pick the wrong k ? With two groups, the regulars and the big spenders are merged, and the silhouette drops to point five four. With five, natural groups get split into small pieces, scoring point six two. Three groups, at point seven one, is the clearest answer.

Key takeaways

- Scaled both features with StandardScaler
- Chose $k = 3$ with elbow and silhouette
- Named the segments for the business
- Assigned a new customer with transform + predict

Practice

1. Re-run the notebook with `random_state=7`. Do the cluster numbers change? Do the groups change?
2. Add a third feature (for example average order value) and repeat the elbow and silhouette steps.
3. Score three new customers of your own and write the offer each should get.

Check your understanding

1. Why must the data be scaled before K-Means?
2. What does the elbow chart plot?
3. What silhouette score usually indicates clear groups?
4. To place a new customer, should you call `fit_transform` or `transform`?

Answers at the back of this handbook.

Decision Trees, Overfitting & Random Forests

Video: 21_AI-21_Decision_Trees_and_Random_Forests.mp4 · 6:15 · Stage: Modeling · Audience: Learners who completed AI-15 to AI-20

Learning objectives

- Describe a tree as yes/no splits chosen by Gini impurity
- Diagnose overfitting with train vs test accuracy by depth
- Explain ensembles and majority voting

Here's the trainer's slide

Each box asks one yes or no question

Each path ends in an answer

No computer needed to follow it

And it explains itself: "your loan is large and the income does not support it"

WHY THIS MATTERS
In banking, insurance and health, a model that cannot explain itself often cannot be used.

A DECISION TREE IS JUST A CHAIN OF YES

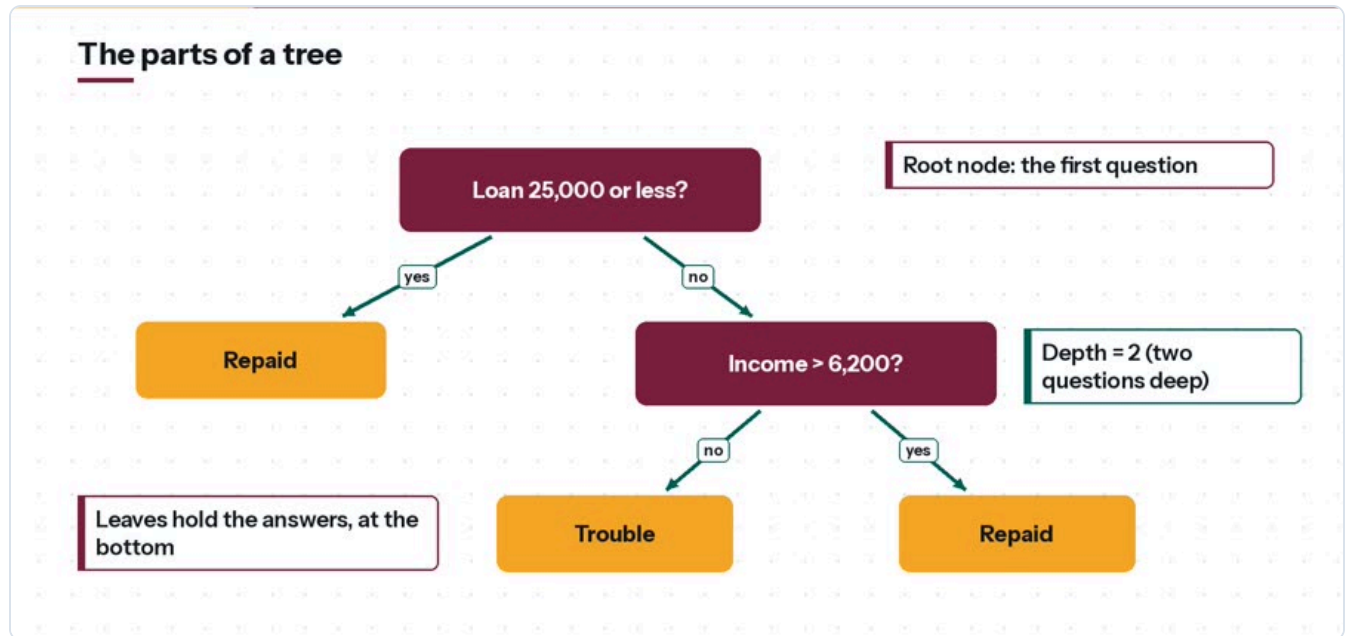
```

graph TD
    A[Is the loan 25,000 or less] -- yes --> B[Expect it to be repaid]
    A -- no --> C[Trouble]
  
```

YOU CAN PRINT THIS AND FOLLOW IT BY HAND

Here's the trainer's slide. A tree is a chain of yes or no questions. The first question: is the loan twenty-five thousand or less? If yes, expect it to be repaid. If not, ask a second question: is the income above six thousand two hundred? No leads to trouble, and yes leads to repaid. The tree explains itself, which is why banking, insurance, and health teams like trees.

The parts of a tree



A few words to know. The first question is the root node. Each answer follows a branch to the next question. Where a path stops, we reach a leaf, which holds the final answer. And depth counts how many questions deep the tree goes. Trees grow downwards, with the leaves at the bottom.

How is this different from logistic regression? Logistic regression...

Logistic regression: one straight line, one rule for everybody

A tree: boxes, like steps, one column at a time

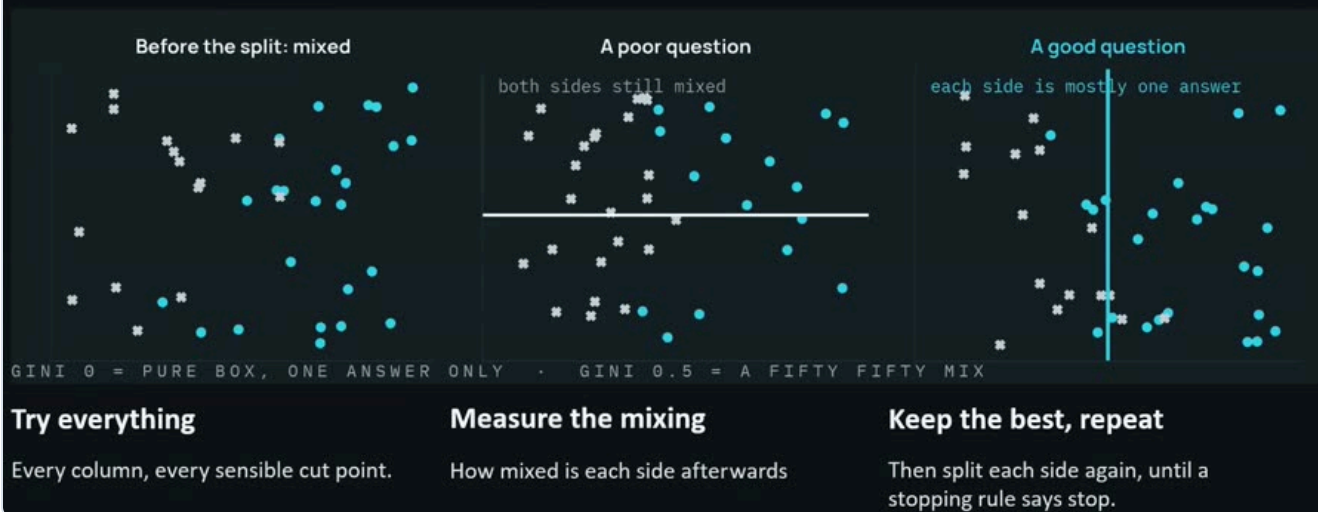
Different rules for different parts of the data

That flexibility is the strength, and also the danger

How is this different from logistic regression? Logistic regression draws one straight line, one rule for everybody. A tree cuts the data into boxes, like steps, using one column at a time. So it can apply different rules to different parts of the data. That flexibility is its strength, and also its danger.

So who picks the questions? The computer does, in three steps

Choosing the best question



So who picks the questions? The computer does, in three steps. First, try everything: every column, and every sensible cut point. Second, measure the mixing. A poor question leaves both sides still mixed. A good question leaves each side mostly one answer. Third, keep the best question, then split each side again, until a stopping rule says stop.

Gini impurity: how mixed is a box?

Gini impurity: how mixed is a box?

Box contents	Calculation	Gini
10 repaid, 0 trouble	$1 - (1.0 \times 1.0 + 0 \times 0)$	0 (pure)
5 repaid, 5 trouble	$1 - (0.5 \times 0.5 + 0.5 \times 0.5)$	0.5 (worst)
8 repaid, 2 trouble	$1 - (0.8 \times 0.8 + 0.2 \times 0.2)$	0.32

Gini = $1 -$ the sum of each answer's share, squared. The best question gives the lowest weighted Gini on its two sides.

The usual mixing score is called Gini impurity. Gini is one, minus the squared share of each answer. A pure box, where every loan was repaid, scores zero. A fifty-fifty mix scores zero point five, the worst it gets with two answers. A box with eight repaid and two in trouble scores zero point three two. The tree picks the question whose two sides have the lowest Gini, weighted by size.

Here's a practical gift

So: no scaling step, mixed column types are fine

COMPARE WITH CLUSTERING

There, an unscaled column took over the whole result. Here it cannot happen.

Here's a practical gift. A tree never measures distance. It only asks: is this value above that number? Whether a column is in dollars or thousands, the same rows go each way. So unlike K-Means, there's no scaling step, and you can mix numbers and categories. Trees are also readable, and fast to train. Compare that with clustering, where one unscaled column took over the whole result.

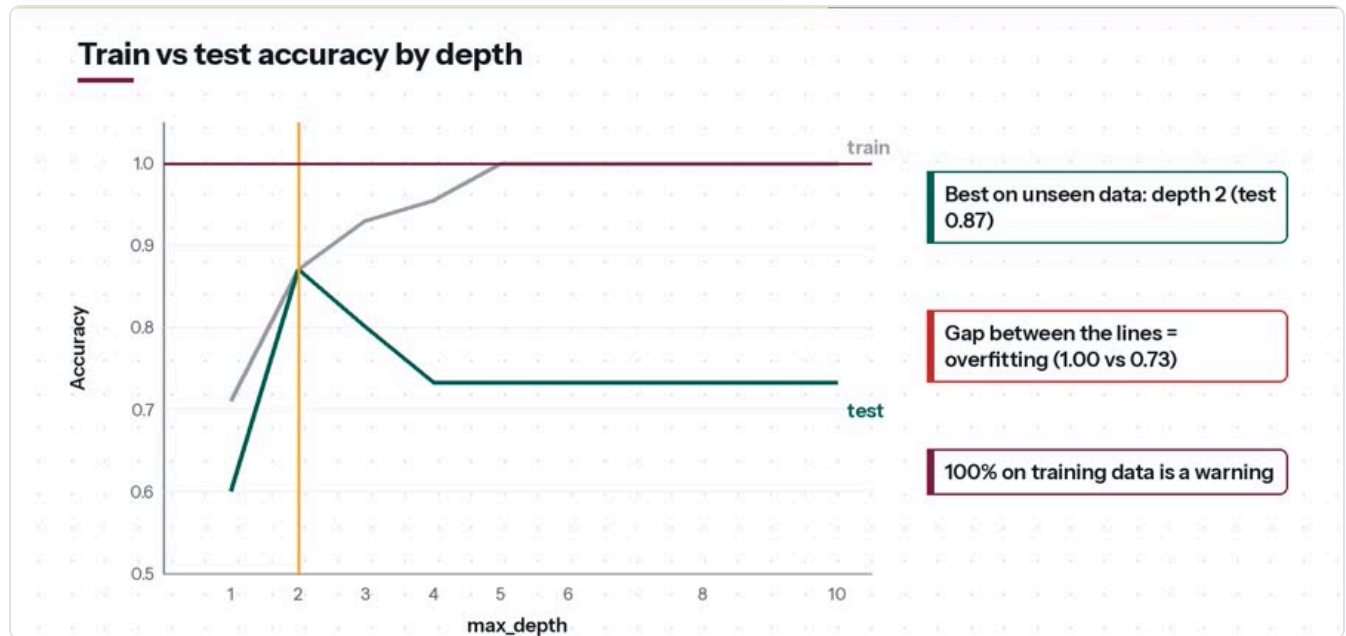
Now the danger



Now the danger. Like a student who memorises last year's exam answers, a deep tree can memorise its training data. With depth one, many rows land on the wrong side: train zero point seven one, test zero point six. That's underfitting. Depth two matches the real pattern: both scores are zero point eight seven. With no limit, the tree draws thin strips around single points: a perfect one on training data, but zero point seven three on test.

Tip The trainer's analogy: change the exam questions a little, and the memoriser fails.

Train vs test accuracy by depth



Here's the same story as a chart. As we allow more depth, training accuracy only goes up, until it reaches one hundred percent. Test accuracy rises, peaks at depth two, then falls and stays low. The gap between the two lines is the overfitting. So remember: one hundred percent on training data is a warning, not a win.

How do you keep a tree under control? In scikit-learn, the main han...

How to keep a tree under control

TIP

Pruning in scikit-learn: the `ccp_alpha` parameter.

`max_depth`

How many questions are allowed. Start at 3 or 4. This is the main one.

`min_samples_leaf`

The smallest number of rows in a leaf. Stops leaves built for one odd customer.

`min_samples_split`

A box needs this many rows before it may be split again.

`max_leaf_nodes`

A direct limit on the total number of leaves.

How do you keep a tree under control? In scikit-learn, the main handle is max depth, the number of questions allowed. Start at three or four. Min samples leaf sets the smallest number of rows in a leaf, so no leaf is built for one odd customer. Min samples split says how many rows a box needs before it may be split again. And pruning grows the full tree, then cuts back what doesn't help.

Tip Pruning in scikit-learn: the `ccp_alpha` parameter.

One tree is readable, but nervous: small changes in the data can ch...

Build hundreds of trees, then take the majority answer

Every tree is different: a random sample of rows (bagging)

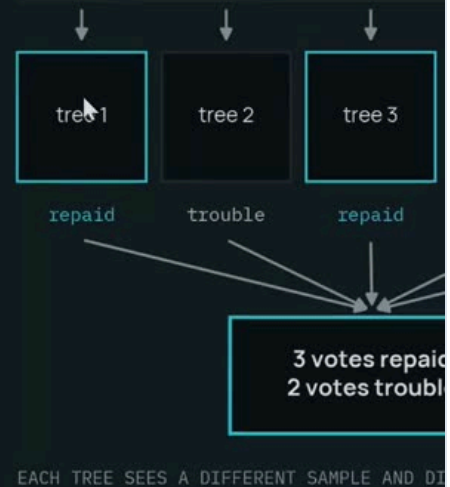
And at each question, only a random handful of columns

Different mistakes cancel each other out in the vote

ANALOGY

**One person guesses the sweets in a jar badly. A
A hundred guesses, averaged, land close.**

ONE NEW APPLICANT



One tree is readable, but nervous: small changes in the data can change it a lot. A random forest builds hundreds of trees, then takes the majority answer. Each tree sees a random sample of rows, called bagging, and a random handful of columns at each question. For one new applicant, three trees say repaid and two say trouble, so the majority wins. Different mistakes cancel each other out in the vote, like a hundred guesses of the sweets in a jar, averaged.

What does the forest give you? You gain accuracy, and a model far less sensitive



What does the forest give you? You gain accuracy, and a model far less sensitive to small changes. You lose the ability to read it: a hundred trees can't be printed on one page. On this small loan dataset, the forest of one hundred trees scored zero point eight seven on test data, the same as the best single tree. On bigger, messier data, forests usually pull ahead. Boosting methods like XGBoost go further, building trees one after another to fix earlier mistakes.

Trees and forests in scikit-learn

Trees and forests in scikit-learn

```
1 from sklearn.tree import DecisionTreeClassifier
2 from sklearn.ensemble import RandomForestClassifier
3
4 tree = DecisionTreeClassifier(max_depth=3, random_state=42)
5 tree.fit(X_train, y_train)
6
7 for d in [1, 2, 3, 5, None]:
8     t = DecisionTreeClassifier(max_depth=d).fit(X_train, y_train)
9     print(d, t.score(X_train, y_train), t.score(X_test, y_test))
10
11 forest = RandomForestClassifier(n_estimators=100, random_state=42)
12 forest.fit(X_train, y_train)
13 print(forest.score(X_test, y_test))
```

n_estimators = number of trees.
Each sees a random sample of rows and columns.

In code, both models follow the fit and score pattern you know. Create a decision tree classifier with a max depth, and fit it. To find a good depth, loop over a few values and print train and test accuracy side by side. Then try a random forest with one hundred trees, and compare.

Key takeaways

- Tree = yes/no questions chosen by Gini
- No scaling needed; rules you can read
- Limit depth; watch the train-test gap
- Forest = many trees vote; steadier, less readable

Practice

1. Draw a three-question decision tree for a yes/no decision at work (for example, approve an expense). Label the root, branches, leaves and depth.
2. A box holds 6 'yes' and 4 'no' rows. Calculate its Gini impurity. Then calculate it for 9 'yes' and 1 'no'. Which box is purer?
3. Using the depth table from the video, explain which depth you would choose and why.

Check your understanding

1. What does a Gini impurity of 0 mean?
2. Why don't decision trees need feature scaling?
3. Training accuracy is 1.00 and test accuracy is 0.73. What is happening?
4. What makes each tree in a random forest different?
5. What do you give up when you move from one tree to a forest?

Answers at the back of this handbook.

Feature Engineering & Data Cleaning

Video: 22_AI-22_Feature_Engineering_and_Data_Cleaning.mp4 · 7:12 · Stage: Data preparation · Audience: Learners who completed AI-15 to AI-21

Learning objectives

- Handle missing values and outliers sensibly
- Derive features from dates, text and distances; scale
- Prevent data leakage: split first, then fit

Here's an honest picture of the job

model

WHY THIS IS GOOD NEWS

Anyone can call fit and predict. Few people can read a raw export and know what it means.

Here's an honest picture of the job. Finding and cleaning data takes about forty-five percent of a data scientist's time. Building features takes about thirty percent. Training models is only about ten percent. So three quarters of the work happens before the model. That's good news: few people can read a raw export and know what it means.

Our example predicts food delivery times

Two columns of words: weather and area

WHAT BEGINNERS DO

Delete the difficult columns. Today we prove those are the valuable ones.

Our example predicts food delivery times. Here's what the system really exports. The order time is a date stored as text. The distance column has nine values missing. One prep time says a hundred and fifty minutes: an order stuck in the system. And weather and area are words. Three of these five columns can't enter a model as they are. Beginners delete the difficult columns, but those are the valuable ones.

Here's the story in one slide



Here's the story in one slide. The model is linear regression, and it never changes. Step one uses raw numbers only and drops the gaps: an average error of about five minutes. Cleaning properly and keeping every row brings it to four point seven seven. Adding time features from the timestamp: three point nine six. And adding weather and area: two point eight three minutes. That's forty-four percent less error, from better features alone.

Six checks before you model

Six checks before you model

```
1 df.shape           # rows, columns
2 df.isna().sum()    # gaps per column
3 df.dtypes          # dates stored as text?
4 df.duplicated().sum() # repeated rows
5 df.describe()      # impossible values
6 df["city"].value_counts() # one city, 3 spellings
```

Standardise spelling and case so one city is one category.

Before you model, run six quick checks in pandas. Shape: did you get all the rows? Is NA sum: missing values per column. D types: is a date stored as text? Duplicated sum, for repeated rows. Describe, to spot a negative age or a distance of zero. And value counts, to catch one city spelled three ways.

Missing values: three honest choices

Missing values: three honest choices

Choice	How	Trade-off
Drop the rows	<code>df.dropna()</code>	Quick, but you lose real orders, and you can't drop a live customer
Fill with the median	Numbers: median. Text: most common or "unknown"	Keeps every row; safe default. Never fill IDs
Flag it	<code>distance_missing = 1 or 0</code> (plus a fill)	Often the best: missingness itself can carry signal

Rule of thumb: fill numbers with the median, text with "unknown", and add a missing flag when it may matter.

Missing values give you three honest choices. Dropping the rows is quick, but you lose real orders, and in production you can't drop a live customer. Filling with the median keeps every row, and it's the safe default for numbers, though never for ID columns. For text, use an unknown category. Flagging adds a one-or-zero column saying the value was missing, and it's often the best of all.

Outliers raise a business question before a technical one: is the v...

good

REMEMBER THE HOUSE PRICE LAB

One price typed with an extra zero pulled the whole regression line.

Outliers raise a business question before a technical one: is the value real, or a mistake? A wholesale customer is real, so keep it. Here, four orders took more than ninety minutes because they were stuck in the system. That's a mistake, so mark the value missing, then fill it. Don't delete the whole row: the other columns are still good. After the fix, prep times look sensible. Remember the house price lab, where one extra zero pulled the whole regression line.

Dates are gold

A raw timestamp says nothing to a model

The hour carries the traffic, the kitchen load, the couriers on shift

Also useful: days since last order, days until renewal, the month

Dates are gold. A raw timestamp says nothing to a model, but the hour carries the traffic and the kitchen load. One timestamp becomes five useful columns: the hour, nineteen, the evening rush. The day of the week. Whether it's a weekend. A rush-hour flag, for the dinner peak. And a holiday flag from a holiday list. Days since the last order is another favourite.

Words must become numbers too

Text into numbers: one-hot encoding

WEATHER, AS WORDS

clear
rain
fog
rain

one-hot
→

AS NUMBERS THE MODEL CAN USE

weather_rain	weather_fog
0	0
1	0
0	1
1	0

Never do this

`clear = 1, rain = 2, fog = 3`
the model then believes
fog is three times clear

One line in pandas

```
pd.get_dummies(df,  
columns=[...],  
drop_first=True)
```

The exception

Small, medium, large have a real order.
There 1, 2, 3 is correct.

Words must become numbers too. One-hot encoding gives each category its own zero-or-one column: weather rain, and weather fog. One line in pandas does it: get dummies, with drop first set to true. Clear needs no column, because if rain and fog are both zero, it was clear. Never number them one, two, three: the model would believe fog is three times clear, unless there's a real order, like small, medium, and large.

Which models need scaling? Distance-based models can't work without...

Distance based models cannot work without it

Trees and forests never need it

Linear models prefer it, and need it with gradient descent

When in doubt, scale: it costs one line and never hurts

BUT MIND THE ORDER

Learn the scaling on the training data only. See the next part.

DOES THIS MODEL NEED SCALING?

- Linear / logistic regression
- K-Means and KNN**
- Decision tree
- Random forest
- Neural networks**

WHEN IN DOUBT, SCALING NEVER HURTS THE

Which models need scaling? Distance-based models can't work without it: K-Means and KNN are essential cases, and so are neural networks. Trees and forests never need it. Linear models prefer it. When in doubt, scale: it costs one line. But mind the order: learn the scaling on the training data only.

Now the creative part: inventing new columns

Inventing new columns

RAW COLUMN	FEATURE YOU BUILD	WHERE IT IS USED
date of birth	→ age, age band	banking, insurance, health
last purchase date	→ days since last order	retail churn, telecom
salary and loan	→ loan to income ratio	credit scoring
GPS points	→ distance, average speed	delivery, ride hailing
30 daily readings	→ rolling 7 day average, trend	sensors, demand forecast
product description	→ word counts, text length	e-commerce, support tickets

THE MODEL CANNOT INVENT THESE, ONLY A PERSON WHO KNOWS THE BUSINESS CAN

TIP

Four patterns cover most features: ratios, durations, aggregations and flags.

Now the creative part: inventing new columns. Date of birth becomes age, or an age band. Last purchase date becomes days since the last order, for churn. Salary and loan become a loan to income ratio, for credit scoring. GPS points become distance and average speed, for delivery. Thirty daily readings become a rolling seven-day average, and a product description becomes word counts. The model can't invent these. Only a person who knows the business can.

Tip Four patterns cover most features: ratios, durations, aggregations and flags.

Now the trap: data leakage

And the model is useless in the live app

WARNING SIGN

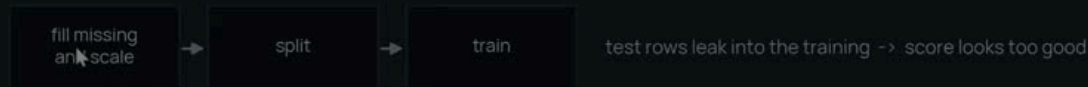
A suspiciously perfect score is not a success. Go and find the leak.

Now the trap: data leakage. Leakage is a feature you wouldn't have at prediction time. Usually it's the answer in disguise. In the lab, one leaking column gave a perfect R squared of one, and was useless in the live app. Test every feature with one question: would I know this value at the moment I have to make the prediction? If yes, use it. If not, drop it. Classic leaks are anything filled in after the event. A suspiciously perfect score is not a success: go and find the leak.

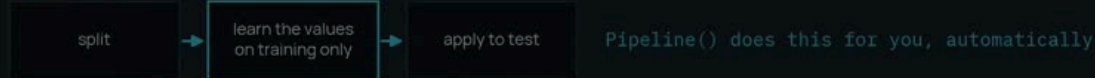
Leakage also hides in the order of your steps

Split first, then fit

WRONG ORDER



RIGHT ORDER



Fill, scale and encode using the training rows only, then apply to the test rows.

Pipeline() does it for you and is the professional way to package these steps.

Leakage also hides in the order of your steps. The wrong order fills missing values and scales the whole dataset, then splits it. The test rows have already shaped the training, so the score looks too good. The right order splits first. Then learn the fill values and the scaling on the training rows only, and apply them to the test rows.

Split first, then fit: a leakage-safe pipeline

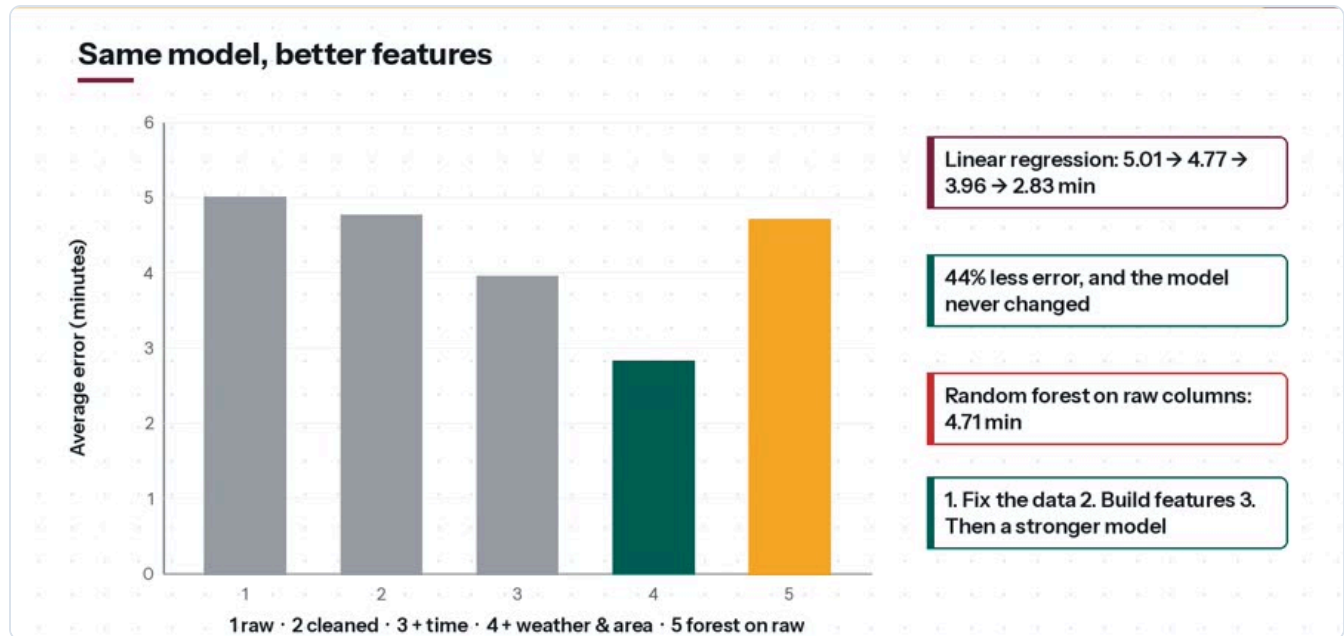
Split first, then fit: a leakage-safe pipeline

```
1 X_train, X_test, y_train, y_test = train_test_split(
2     X, y, test_size=0.2, random_state=42) # split first
3
4 pipe = Pipeline([
5     ("fill", SimpleImputer(strategy="median")),
6     ("scale", StandardScaler()),
7     ("model", LinearRegression()),
8 ])
9
10 pipe.fit(X_train, y_train) # learns from train
11 print(pipe.score(X_test, y_test)) # honest score
```

The test set is treated as data you have never seen.

In scikit-learn, a pipeline packages these steps for you. Split the data first. Then chain the steps: a simple imputer that fills gaps with the median, a standard scaler, and the model. When you call fit on the training data, every step learns from training rows only. Scoring on the test set then gives you an honest number.

Same model, better features



Here's the result. Same linear regression, better features. The average error fell from five point zero one minutes to two point eight three: forty-four percent less. A random forest on the raw columns managed only four point seven one. So fix the data first, build features second, and only then try a stronger model.

Key takeaways

- Six checks before you model
- Median fill + missing flag; outliers: ask the business
- Dates, words and locations become features; scale for distance
- Split first, then fit: no leakage

Practice

1. Take any export you work with. Run the six checks (`shape`, `isna().sum()`, `dtypes`, `duplicated().sum()`, `describe()`, `value_counts()`) and list every problem you find.
2. From a single timestamp column, create hour, day of week, is_weekend and a rush-hour flag in pandas.
3. For three columns in a dataset you know, apply the leakage test: would you know this value at the moment of prediction?

Check your understanding

1. What is the safe default for filling a missing numeric value?
2. Why not encode clear, rain and fog as 1, 2 and 3?
3. Which models need scaling: K-Means, decision trees, or random forests?
4. What is data leakage?
5. What is the right order: fill and scale, then split; or split, then fill and scale?

Answers at the back of this handbook.

Answer key

AI-12 • Arguments, Lambdas, Filter & Map

1. Positional arguments
2. Call `f1()` without an argument; Java is used by default
3. A tuple
4. A filter object (an iterator)
5. filter keeps items that pass a test; map transforms every item
6. It holds one type and does maths on every element at once (vectorised), including 2-D tables

AI-13 • Classes & Objects: How scikit-learn Models Work

1. `__init__`
2. The object the method was called on
3. Once, when the object is created
4. An object (instance) of the `LinearRegression` class
5. A method, because it belongs to the model's class

AI-14 • Linear Regression Explained

1. The change in y when x increases by 1 (the slope)
2. Actual value minus predicted value
3. So positive and negative errors can't cancel, and big errors count more
4. The error spread grows with x (unequal variance)

AI-15 • Hands-on: Predicting Ice-Cream Demand with scikit-learn

1. So it stays a table (2-D), which scikit-learn expects
2. Holds back 4 rows as the test set
3. About 6 more ice creams per extra degree
4. RMSE

AI-16 • Multiple Regression & Prediction Intervals

1. It's the effect of that one input when all other inputs stay the same
2. Deliveries held information the first model was missing
3. Prediction, because individual cases vary around the average
4. A straight-line model may be wrong

AI-17 • From Regression to Classification: Logistic Regression

1. Its straight line goes below 0 and above 1
2. Between 0 and 1
3. The input value where the probability crosses the threshold
4. Classification

AI-18 • Hands-on: Pass/Fail Classifier & Evaluation Metrics

1. Of the cases predicted positive, the share that really were positive
2. Of the real positives, the share the model caught
3. Recall rose to 1.0 and precision fell to about 0.83
4. It treats all mistakes equally and hides which kind of error the model makes

AI-19 • Clustering & K-Means Concepts

1. Classification uses known groups; clustering discovers them
2. Drop centres, assign to nearest, move centres, repeat until stable
3. It uses distance, so the column with the biggest numbers dominates
4. You do

AI-20 • Hands-on: Customer Segmentation with K-Means

1. K-Means uses distance, so the column with bigger numbers would dominate
2. Inertia (group tightness) for each k
3. Above 0.5, closer to 1 is better
4. transform, reusing the scaler fitted on the training data

AI-21 • Decision Trees, Overfitting & Random Forests

1. The box is pure: every row has the same answer
2. They never measure distance; each split only asks whether a value is above a threshold
3. Overfitting: the tree memorised the training data
4. Each sees a random sample of rows (bagging) and a random subset of columns at each split
5. Readability: a forest can't be printed and followed by hand

AI-22 • Feature Engineering & Data Cleaning

1. The median, learned from the training data
2. The model would treat them as ordered quantities (fog = 3 × clear)
3. K-Means (distance-based); trees and forests don't
4. Using information you would not have at prediction time, often the answer in disguise
5. Split first, then learn fill values and scaling on the training rows only