ECO447 · WEEK 10

Model Trees & Ensemble Methods

Why many trees beat one — Random Forest, OOB, and XGBoost

Assoc. Prof. Dr. Onur Polat

Hacettepe University · Department of Economics

Fall 2026-2027 · Undergraduate ML for Economists · 28 slides

Today's Agenda

5 blocks · ~2 hours · hands-on with real economic data

01 **From Trees to Model Trees**

Linear predictions inside each leaf

02 **The Wisdom of Many Trees**

Why one tree is not enough

03 **Random Forest + OOB**

The classic economics-friendly ensemble

04 **Boosting & XGBoost**

Learning from mistakes, tree by tree

05 **Economic Applications**

Housing prices & credit scoring

BLOCK 1

From Trees to Model Trees

Making trees a little smarter

Quick Recap — the Regression Tree

What we learned last week

A regression tree asks a sequence of yes/no questions

Example: predicting house prices in California

1. "Is the median income > $5 000?" → Yes / No branches
2. For each branch, ask another question (e.g., number of rooms)
3. Keep splitting until each "leaf" contains similar houses
4. Predict the average price of those houses

✓ Easy to read and explain
✓ Handles nonlinear relationships naturally
✗ Predicts the SAME value for every house in a leaf — **blocky, step-shaped**
✗ Very sensitive to the training data — a single new house can change the tree

The Model Tree Idea

Instead of a single number in each leaf — fit a small linear regression

Prediction inside a leaf

$$\hat{y}(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \qquad \text{(inside each leaf)}$$

How this helps

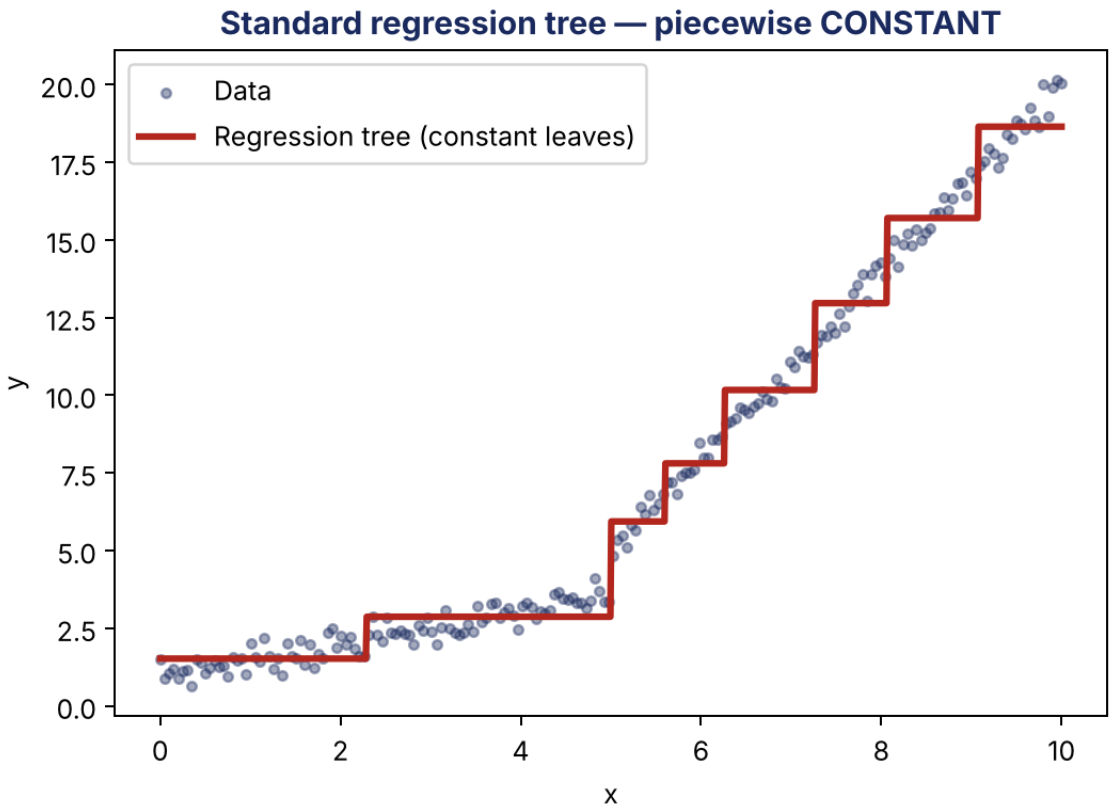
- The **tree decides the group** (which neighborhood, which income bracket)
- The **linear regression inside the leaf** estimates a local slope
- Think of it as a ***"different regression for different regions"*** — exactly what economists often do by hand!

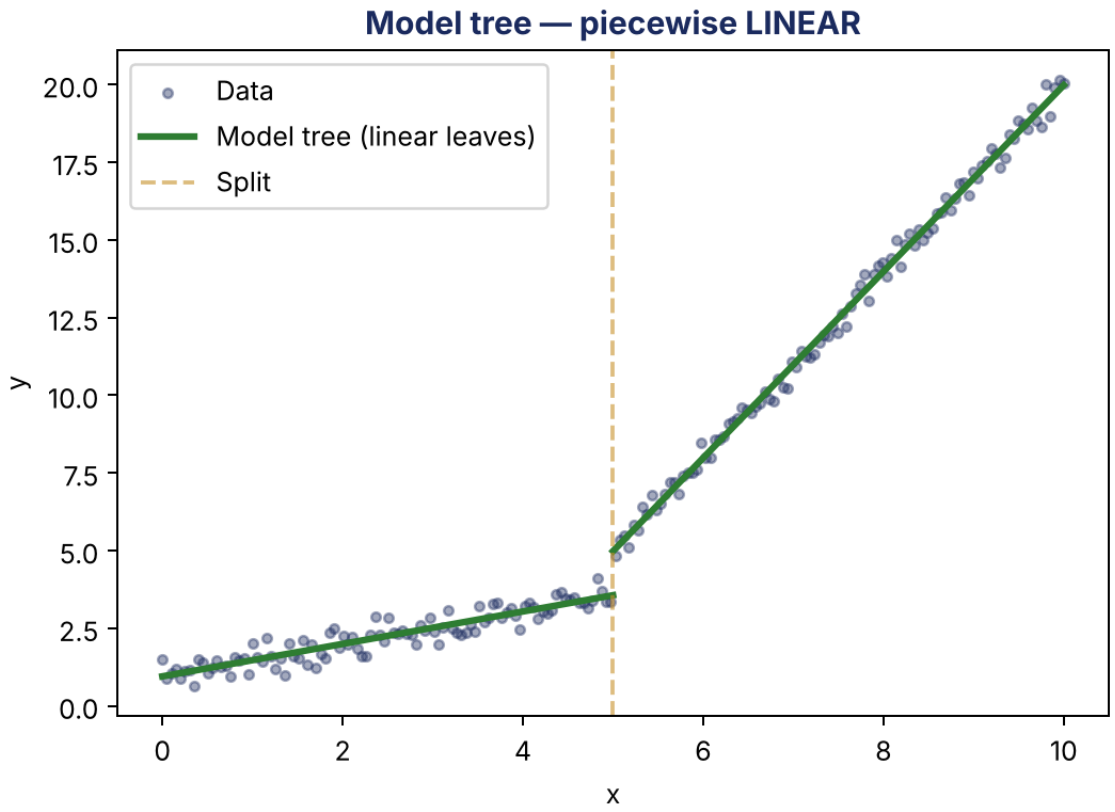
Example — Mincer earnings equation:

- Returns to education are different for different industries
- A model tree can automatically discover the industries and fit a separate return-to-education slope within each

Model Tree vs Regression Tree — in Pictures

Same data, two different fits





- **Left:** the regular tree draws flat horizontal segments
- **Right:** the model tree fits a line in each region — cleaner, fewer splits needed
- *Especially useful when the true relationship is roughly linear within each group — common in economics*

BLOCK 2

The Wisdom of Many Trees

Why one tree is rarely enough

The Problem with a Single Tree

It gets a different answer every time you shuffle the data

A thought experiment

Suppose you have 1 000 house-price observations. You randomly pick 800 to train a tree. You get a tree with certain splits. Now you randomly pick a DIFFERENT 800 → you might get a completely different tree, with different splits, different leaves, different predictions. Single trees are UNSTABLE.

The ensemble idea

Instead of trusting ONE tree, grow **HUNDREDS of trees** and average their predictions. The random "noise" in individual trees cancels out, while the real "signal" is preserved.

Analogy: *asking 100 people to guess the weight of a bull. Any single guess is off by a lot. But the AVERAGE of 100 guesses is surprisingly accurate — "wisdom of crowds" (Galton, 1907).*

Why Averaging Works

A simple statistical fact you already know

$$\mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{y}^{(b)}\right) = \frac{\sigma^2}{B} \qquad \text{(if independent)}$$

Translation

- If each tree has prediction variance σ^2, averaging B independent trees reduces the variance to σ^2/B
- With 100 trees, the "wobbliness" of predictions shrinks by a factor of 100
- In practice, trees are not perfectly independent — but they are DIFFERENT enough for averaging to help a lot

✓ Variance goes DOWN (good)

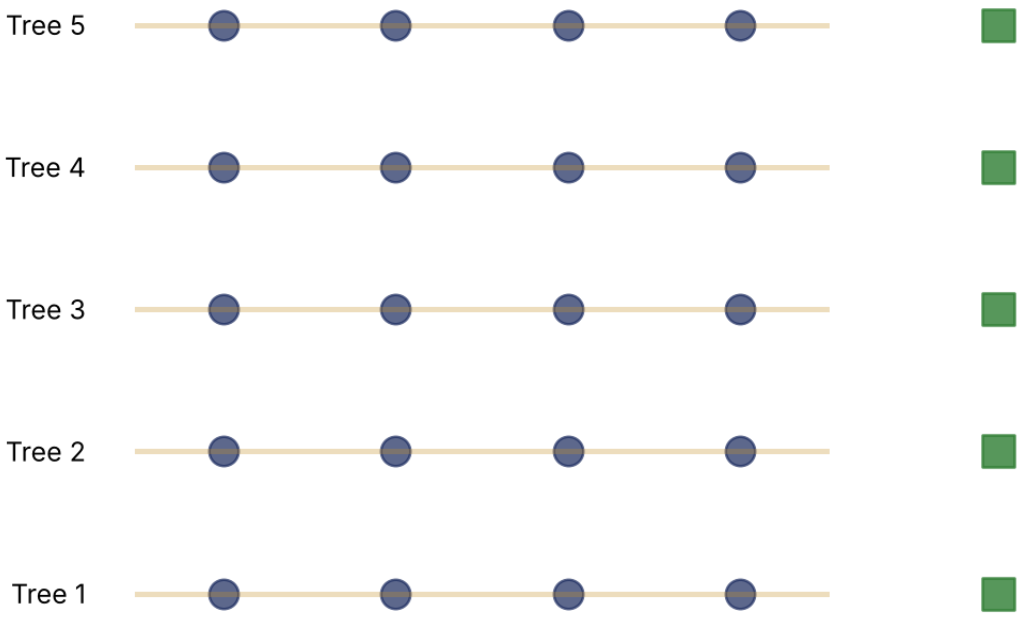
= The average tree bias is unchanged (not a free lunch, but very useful!)

This is the main reason ensembles beat single trees on almost every economic dataset.

Two Ways to Build an Ensemble

Bagging and Boosting — two different strategies

Bagging / Random Forest — parallel



Boosting — sequential, each tree fixes errors



- **Bagging / Random Forest:** grow many trees INDEPENDENTLY on random samples; average at the end (parallel teamwork)
- **Boosting / XGBoost:** grow trees ONE AT A TIME, each new tree correcting the errors of the previous ones (sequential teamwork)

BLOCK 3

Random Forest + Out-of-Bag

The default ensemble for economic data

Step 1 — Bagging (Bootstrap Aggregating)

The simplest tree ensemble

$$\hat{y}_{\text{bag}}(x) = \frac{1}{B}\left[\hat{y}^{(1)}(x) + \hat{y}^{(2)}(x) + \ldots + \hat{y}^{(B)}(x)\right]$$

The recipe

1. From your training set of n observations, draw a **bootstrap sample** (n draws WITH replacement — some rows appear more than once, some none at all)

2. Grow a tree on this sample — no pruning, let it grow deep

3. Repeat B times (typically B = 100 to 500) → you have B different trees

4. **To predict a new observation:** send it through every tree, take the average (regression) or majority vote (classification)

Step 2 — Random Forest adds a Clever Twist

Breiman (2001)

The problem with plain bagging

If one feature (say, "median income") is very strong, every tree will use it near the top. All the trees look similar → averaging them doesn't help much.

🎯 Breiman's fix: randomize at every split

- At each split, the tree can only use a **random subset** of the features (not all of them)
- Typical choice: `√p features for classification, p/3 for regression`
- This forces trees to use DIFFERENT features at different splits → trees become genuinely different
- Averaging many different trees gives the best variance reduction

⟹ *Random Forest is just Bagging + "random subset of features at each split". That's the whole secret.*

The Clever Part — OOB Validation

You get cross-validation for FREE

$$\left(1 - \frac{1}{n}\right)^n \longrightarrow e^{-1} \approx 0.368 = 36.8\%$$

The insight

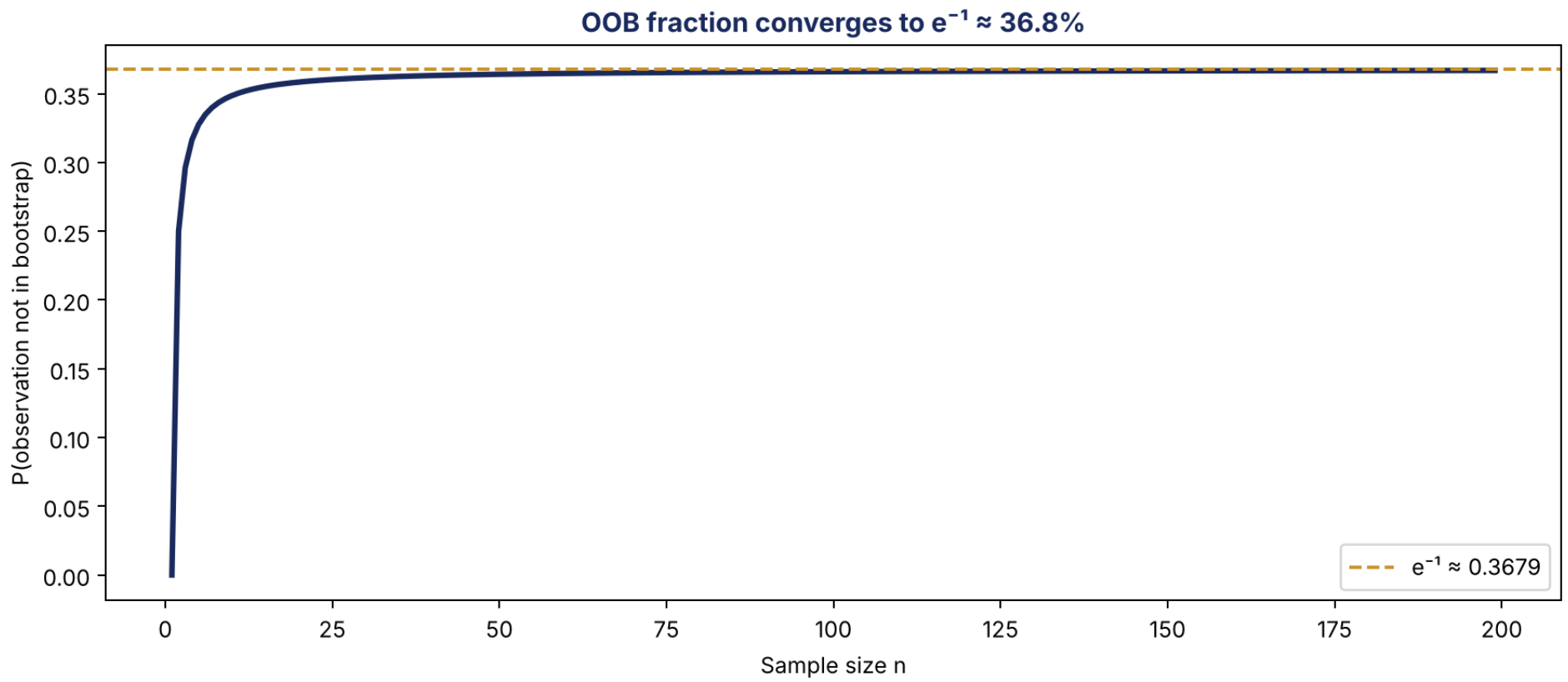
- When we draw a bootstrap sample of size n with replacement, about **63% of the original rows** show up (some multiple times)
- The remaining **~37% are "out-of-bag"** for that tree
- For each observation i, use only the trees that did NOT see it in training to predict it
- Compare with the true y_i, average across all i → OOB score

⟹ You get a reliable out-of-sample performance estimate **without doing any cross-validation** — the Random Forest gives it to you automatically.

In sklearn: `RandomForestRegressor(oob_score=True)`

Where Does the 37% Come From?

A short mathematical curiosity



- For any observation, the probability of NOT being picked in one draw is (1 − 1/n)
- Across n draws: $(1 - 1/n)^n \to e^{-1} \approx 0.368$
- Even for small n, this is already close to 37% — so OOB validation works even on modest datasets

Random Forest in Practice

Three settings that matter + sensible defaults

Setting	What it does	Starting value
`n_estimators`	How many trees to grow	300 – 500
`max_features`	Features to try at each split	sqrt(p) (classification) p/3 (regression)
`max_depth`	How deep each tree can be	None (grow fully)
`oob_score`	Report the free validation score?	True

💡 ***In practice: try the defaults first! RF is one of the most forgiving algorithms in machine learning.***

Feature Importance — Which Variables Matter?

RF gives you this almost for free

Two simple methods

① Impurity-based (MDI)

- Each time a feature is used to split a node, it reduces impurity by some amount
- Sum these reductions across all trees → importance
- Fast (free during training) but biased toward features with many unique values
- In sklearn: `model.feature_importances_`

② Permutation importance

- Shuffle one feature's values randomly on the test set
- See how much the model's performance drops
- Big drop → important feature. Small drop → useless feature.
- Slower but more reliable than impurity-based
- In sklearn: `sklearn.inspection.permutation_importance`

BLOCK 4

Boosting & XGBoost

Learning from mistakes, one tree at a time

The Boosting Idea

Grow trees one by one, each one fixing the errors of the previous ones

The step-by-step story

1. Start with a simple prediction (e.g., the average of y)
2. Compute the **residuals** (what the current model got wrong — the errors)
3. Fit a **small tree** to predict those errors
4. Add a small fraction (v) of this tree to the model → errors become smaller
5. Repeat 100s of times. Each tree is weak on its own — together they are very strong.

$$r_i = y_i - F_{m-1}(x_i) \qquad \text{(the error of the current model)}$$

$$F_m(x) = F_{m-1}(x) + \nu \cdot h_m(x)$$

Why the "Small Fraction v"?

The learning rate controls how fast we learn

An analogy

Climbing down a hill in fog. BIG steps (large v) → fast but you might overshoot the valley. SMALL steps (small v) → slow but careful; more likely to reach the true bottom.

Two parameters that work together

- v (learning rate) — typical values: `0.05 or 0.1`
- M (number of trees) — typical values: `300 to 2 000`

Rule of thumb: small v + many trees + **early stopping** (stop when the validation score stops improving)

XGBoost — "eXtreme Gradient Boosting"

Chen & Guestrin (2016) — the most popular tool on Kaggle

What makes XGBoost special

- **Regularization built in:** penalizes overly complex trees → less overfitting than plain boosting
- **Handles missing values:** automatically learns the best way to route observations with NaN values
- **Fast:** parallel tree construction, GPU support, scales to millions of rows
- **Best for:** tabular / structured data — exactly the kind of data economists work with

Python one-liner: `from xgboost import XGBRegressor, XGBClassifier`

XGBoost — Which Settings to Change?

Only 4 or 5 really matter at first

Setting	What it does	Starting value
`n_estimators`	Maximum number of trees	1000 (with early stopping)
`learning_rate`	v (step size)	0.05 or 0.1
`max_depth`	Depth of each tree	4 to 6
`subsample`	Share of rows per tree	0.8
`colsample_bytree`	Share of features per tree	0.8
`early_stopping_rounds`	Stop when val score plateaus	30 or 50

BLOCK 5

Economic Applications

Housing prices and credit scoring

Application 1 — California Housing

A classic hedonic pricing problem

The dataset

- 20 640 California census tracts, 1990 census
- 8 features: median income, house age, average rooms, average bedrooms, population, occupants per household, latitude, longitude
- Target: median house value (in $100 000)
- Available in sklearn: `from sklearn.datasets import fetch_california_housing`

Typical results you should see in your Colab

Method	Test RMSE	Test R^2
Single tree	~0.70	~0.63
Bagging (200 trees)	~0.55	~0.77
Random Forest (500)	~0.51	~0.81
XGBoost (with early stop)	**~0.45**	**~0.85**

$\Longrightarrow$ *XGBoost gives about 35% lower prediction error than a single tree.*

Application 2 — German Credit Scoring

A classic binary classification problem in economics

The dataset

- 1 000 bank loan applicants (German dataset, widely used in credit scoring research)
- 20 features: credit history, loan purpose, amount, duration, savings, employment, age, housing, etc.
- Target: good credit risk (1) or bad credit risk (0)
- Available online: `fetch_openml("credit-g")`

Why interpretation is as important as accuracy

- Banks must **explain** why they declined a loan (EU GDPR Article 22, Turkey KVKK Art. 11)
- Random Forest and XGBoost are more accurate than a logistic regression — but less immediately interpretable
- **SHAP values** provide a feature-by-feature explanation for EVERY prediction → usable for regulatory compliance
- Also useful for understanding what drives the model economically — is it credit history? employment? loan amount?

SHAP — Explaining Ensemble Predictions

Give every prediction a clear breakdown

The idea in one sentence

SHAP decomposes each prediction into "this is the baseline + feature X pushed it up by this much + feature Y pulled it down by that much". The contributions always add up to the final prediction.

How to use it

1. Fit your Random Forest or XGBoost model as usual
2. `explainer = shap.TreeExplainer(model)`
3. `shap_values = explainer.shap_values(X_test)`
4. `shap.summary_plot(shap_values, X_test)` → global importance
5. `shap.force_plot(...)` → local explanation for one customer

Takeaways & Homework

What to remember and practice

🧠 Key takeaways

1. **Model trees** use a linear regression inside each leaf — like running "a different regression for different groups"
2. **Bagging** = grow many trees on bootstrap samples, average their predictions → more stable, better accuracy
3. **Random Forest** = Bagging + random subset of features at each split → the standard tabular ML tool
4. **OOB score** = free validation estimate — no need for cross-validation
5. **Boosting** (XGBoost) = grow trees sequentially, each one correcting the previous errors
6. **SHAP** explains predictions feature-by-feature — essential for credit scoring and regulated decisions

💻 Colab practice

- Run the Colab end-to-end — don't skip the exercises inside each block
- California Housing: single tree vs Random Forest vs XGBoost (3 models, same split)
- Try n_estimators = 50, 100, 300, 500 on Random Forest — plot OOB score
- German Credit: fit an RF, print the top 10 features by importance
- (Optional challenge) Use SHAP to explain one specific loan applicant's prediction