

# Lecture 007

Trees 🌲🌴🌳

---

Edward Rubin

# Admin

# Admin

## Material

*Last time:* Classification

*Today!* Decision trees for regression and classification.

# Admin

## Upcoming

### Readings

- *Today* ISL Ch. 8.1
- *Next* ISL Ch. 8.2

### Problem sets

- *Penalized regression and classification:* Very soon!

### Project

- Group and topic **due today!**
- Start getting your data together!

# Decision trees

# Decision trees

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

# Decision trees

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**

# Decision trees

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**

# Decision trees

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**

# Decision trees

## Fundamentals

### Decision trees

- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**
4. often **underperform** relative to competing methods

# Decision trees

## Fundamentals

### Decision trees

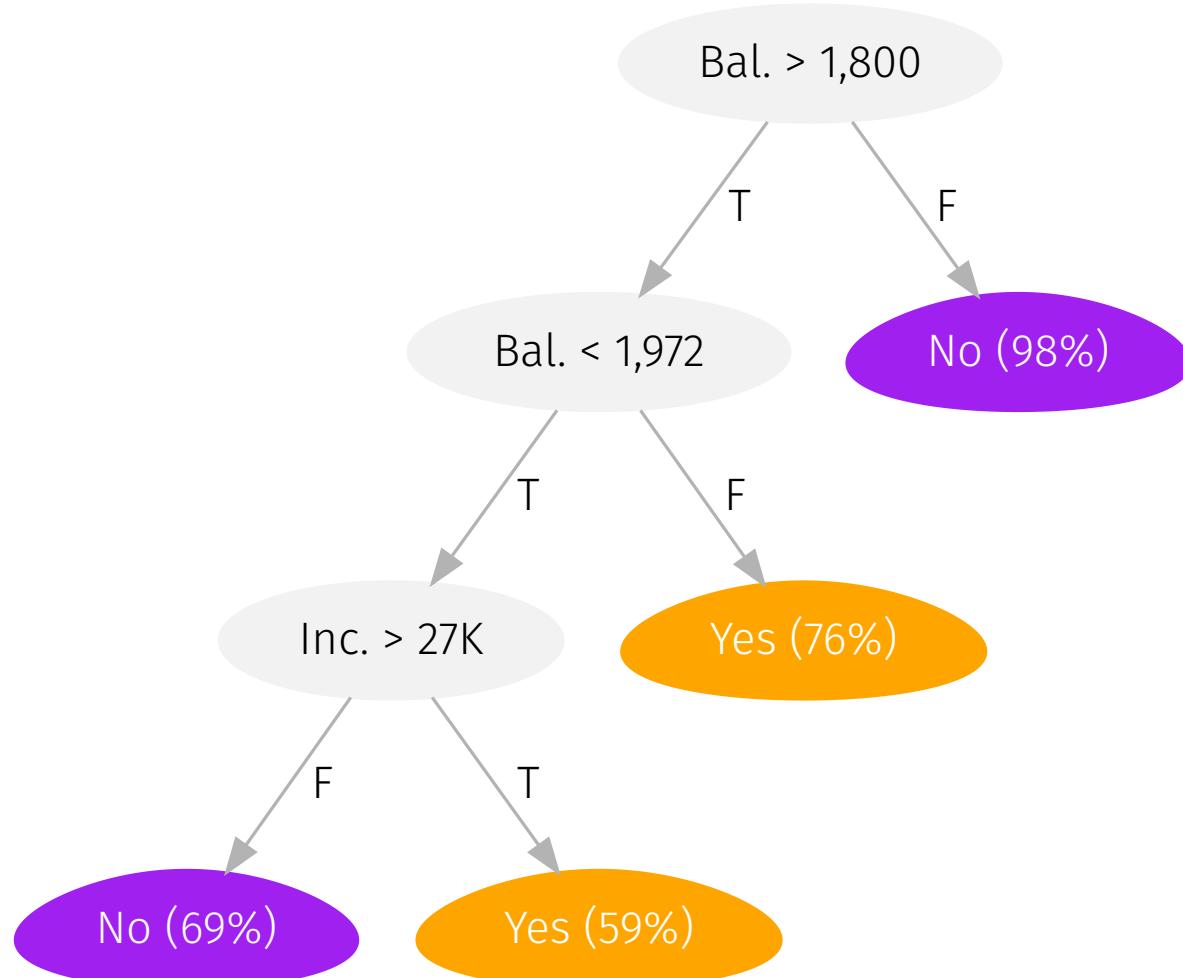
- split the *predictor space* (our  $\mathbf{X}$ ) into regions
- then predict the most-common value within a region

### Decision trees

1. work for **both classification and regression**
2. are inherently **nonlinear**
3. are relatively **simple** and **interpretable**
4. often **underperform** relative to competing methods
5. easily extend to **very competitive ensemble methods** (many trees) 

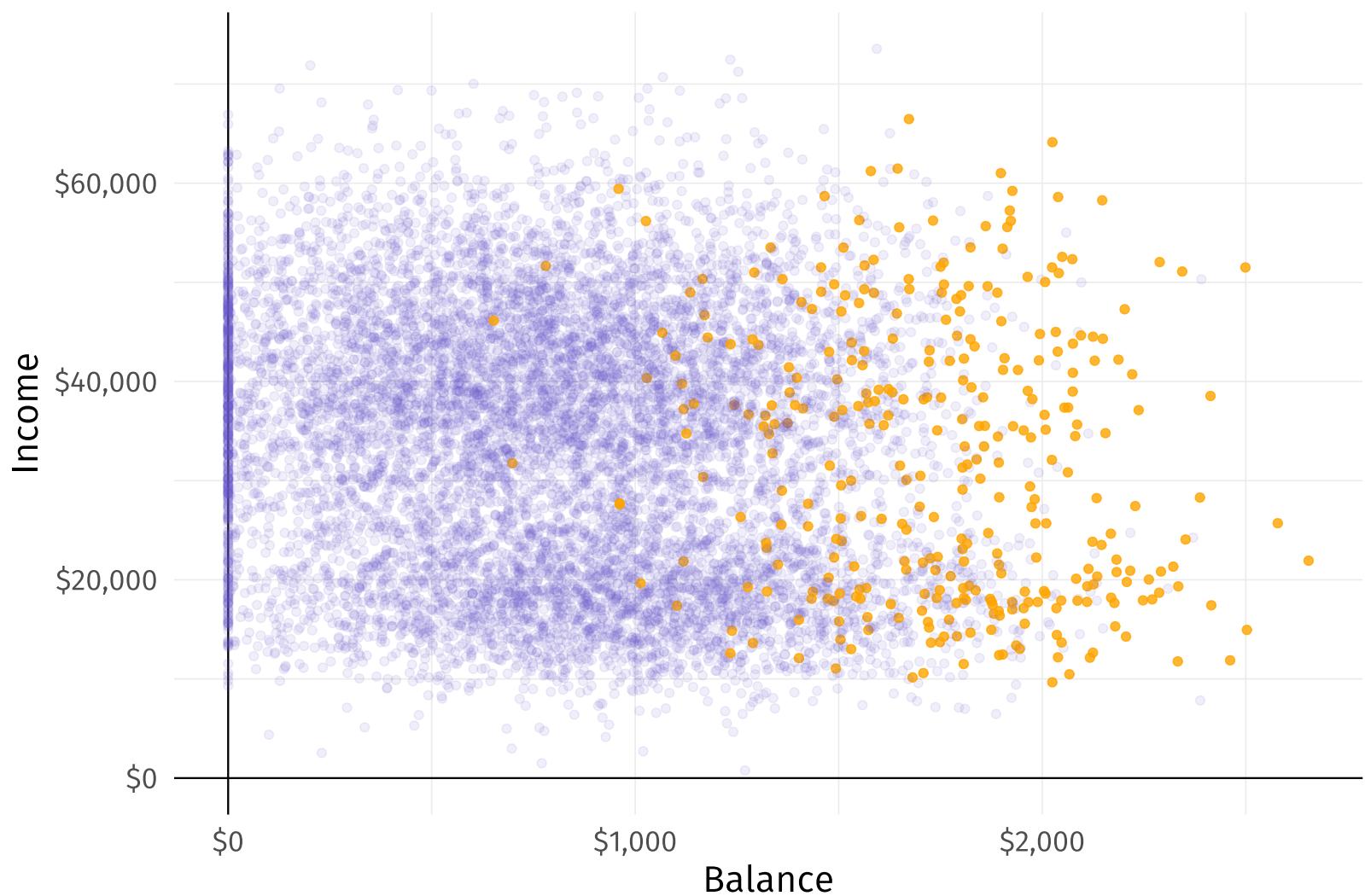
 Though the ensembles will be much less interpretable.

*Example:* **A simple decision tree** classifying credit-card default

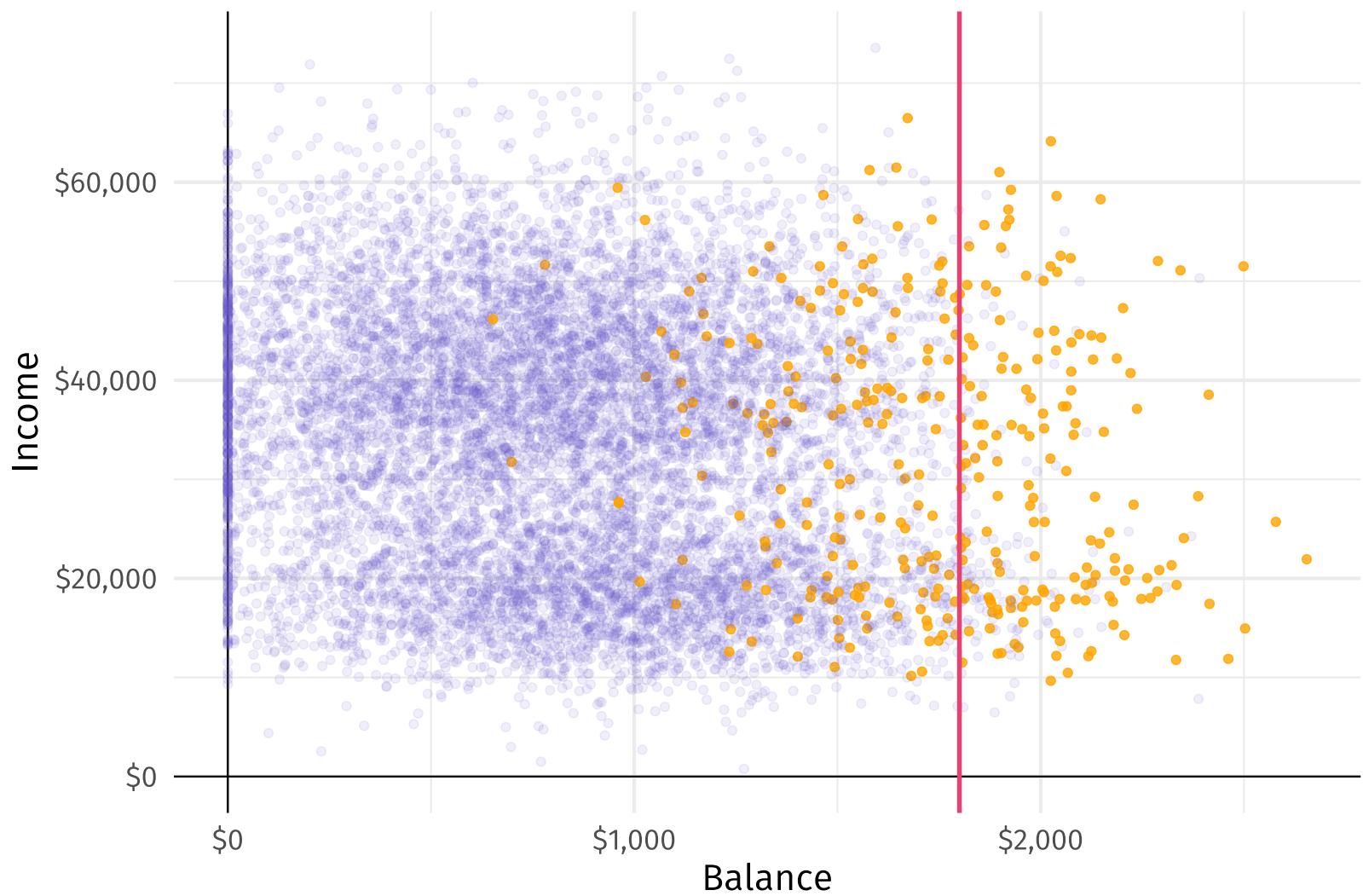


Let's see how the tree works

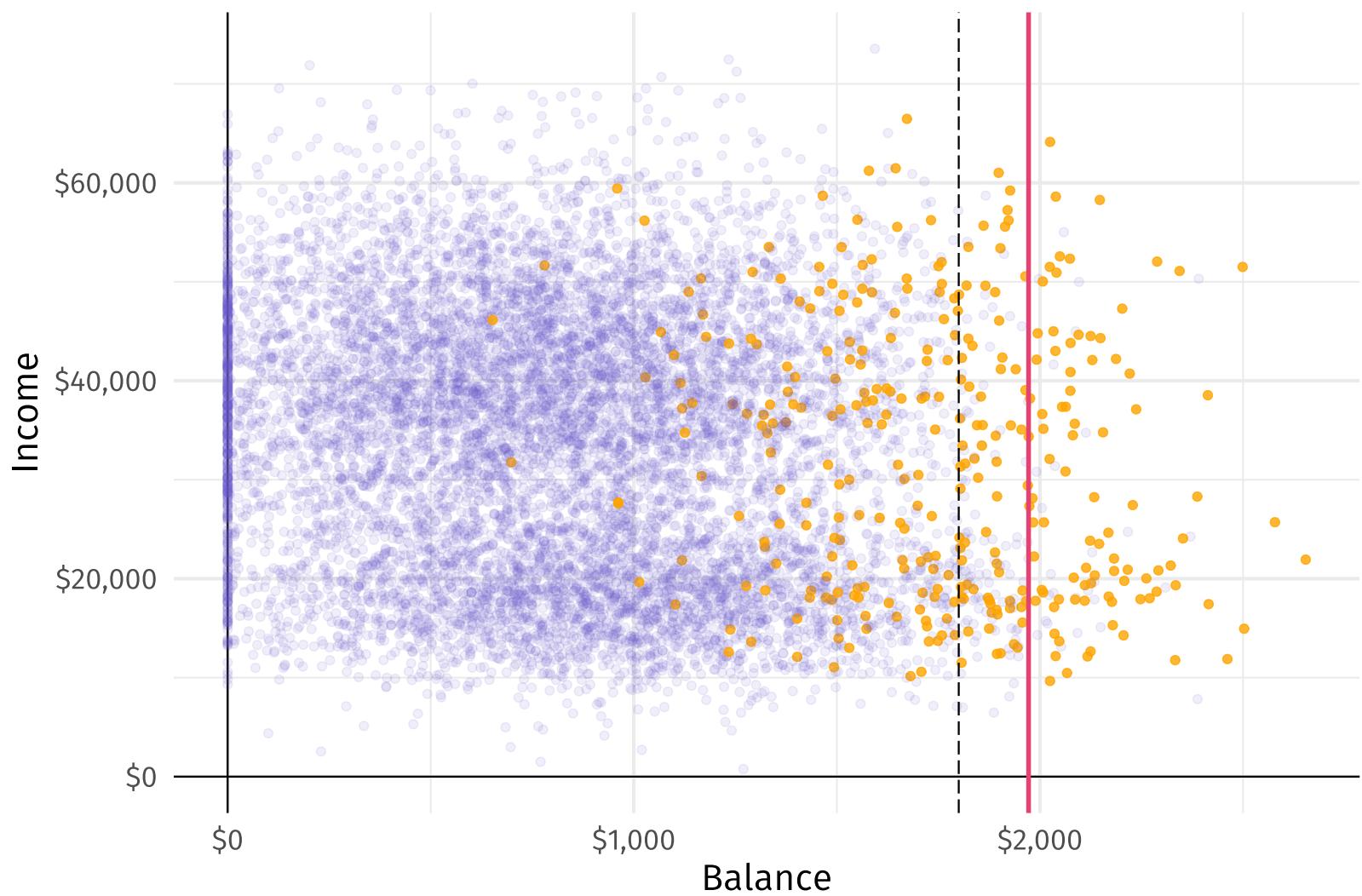
Let's see how the tree works—starting with our data (default: Yes vs. No).



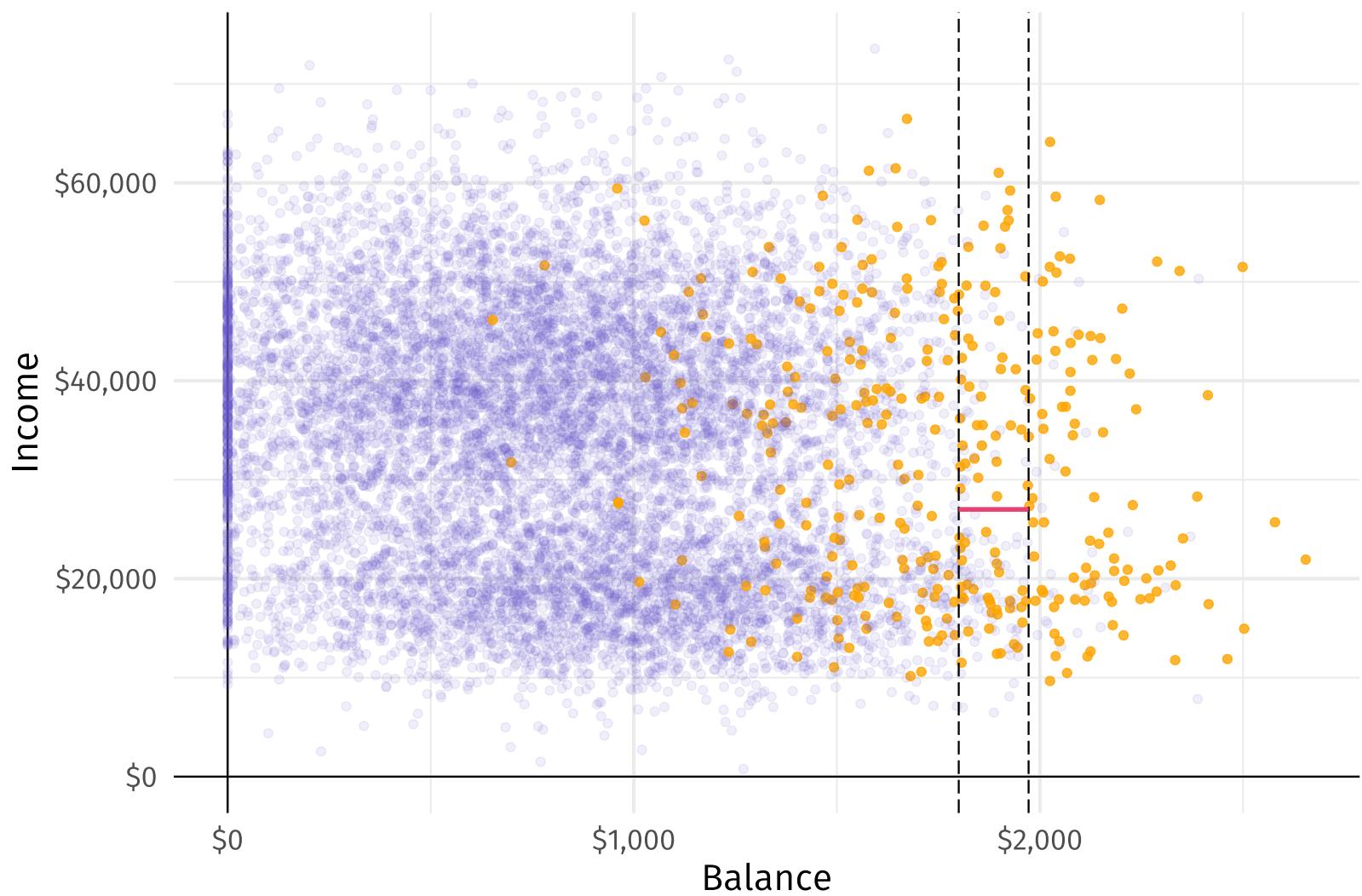
The **first partition** splits balance at \$1,800.



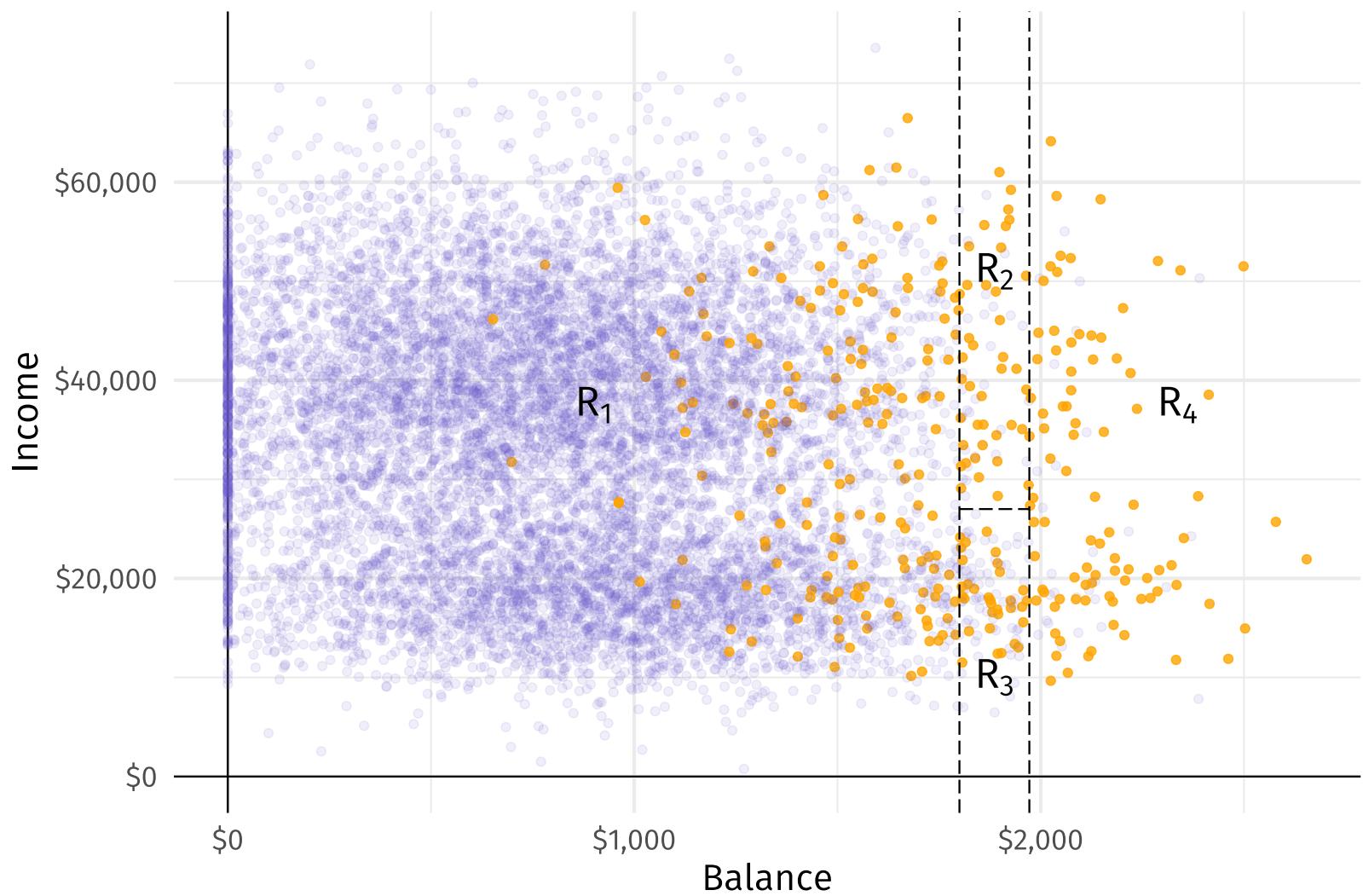
The **second partition** splits balance at \$1,972, (conditional on bal. > \$1,800).



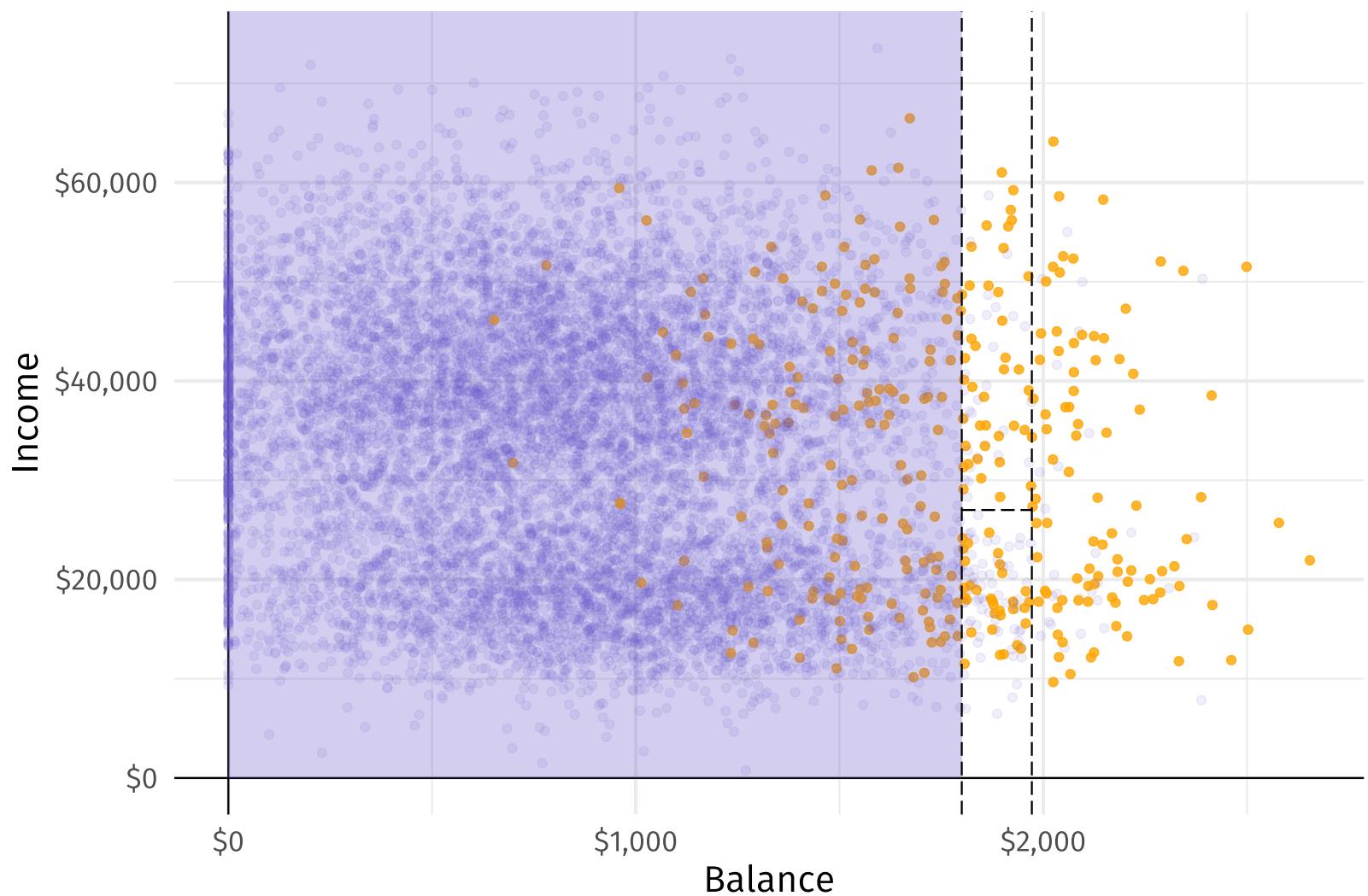
The **third partition** splits income at \$27K **for** bal. between \$1,800 and \$1,972.



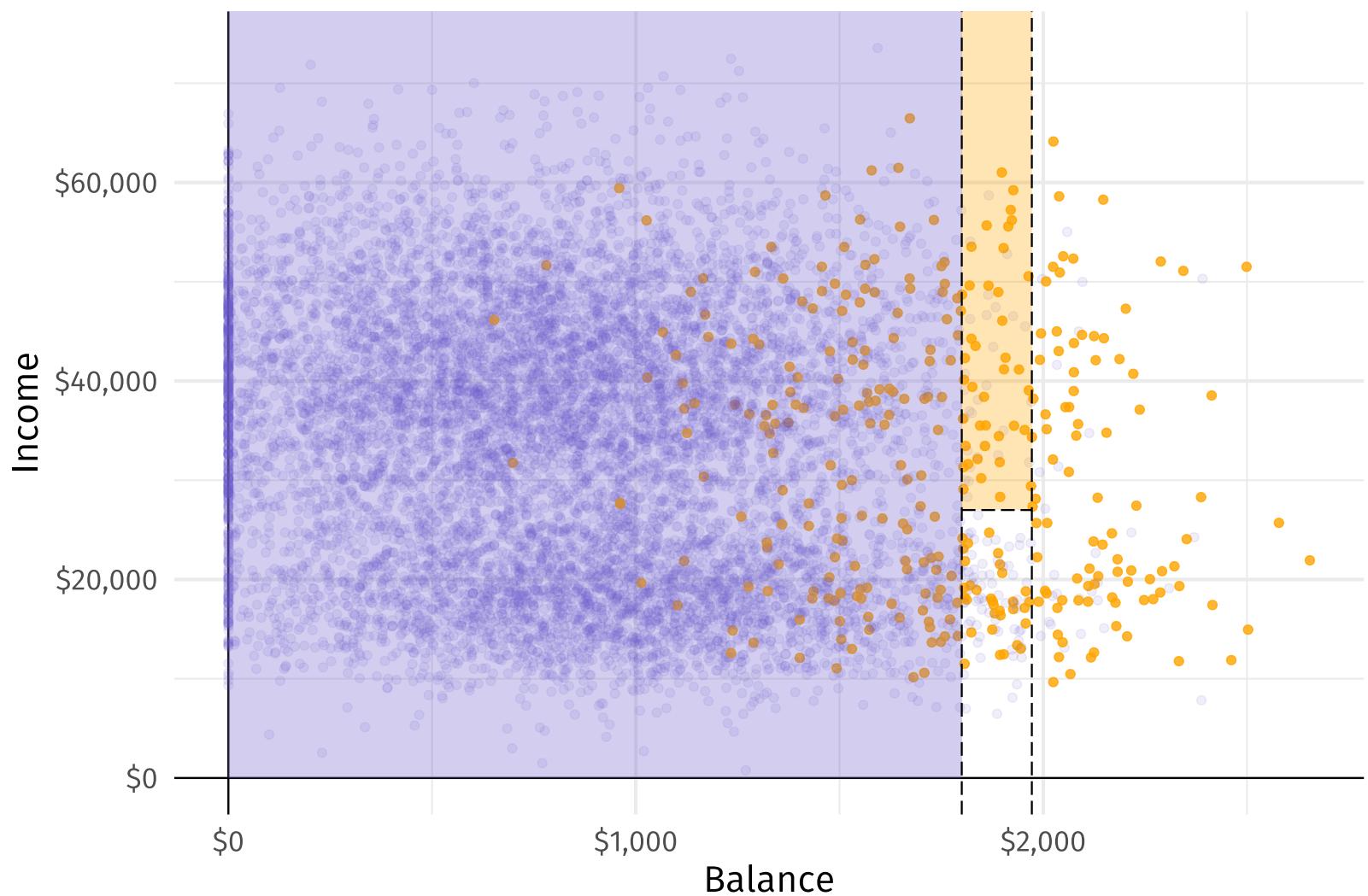
These three partitions give us four **regions**...



**Predictions** cover each region (e.g., using the region's most common class).



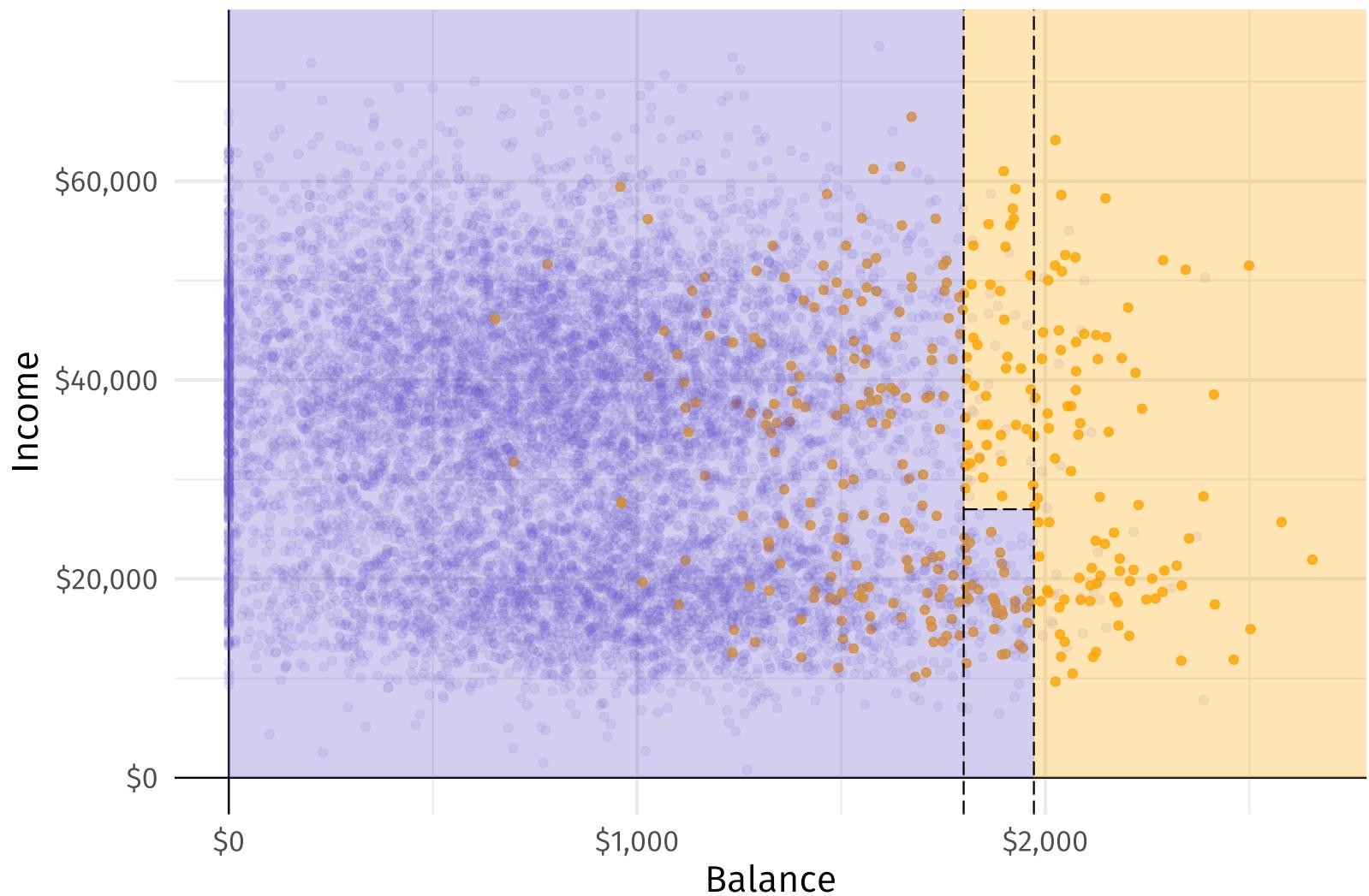
**Predictions** cover each region (e.g., using the region's most common class).



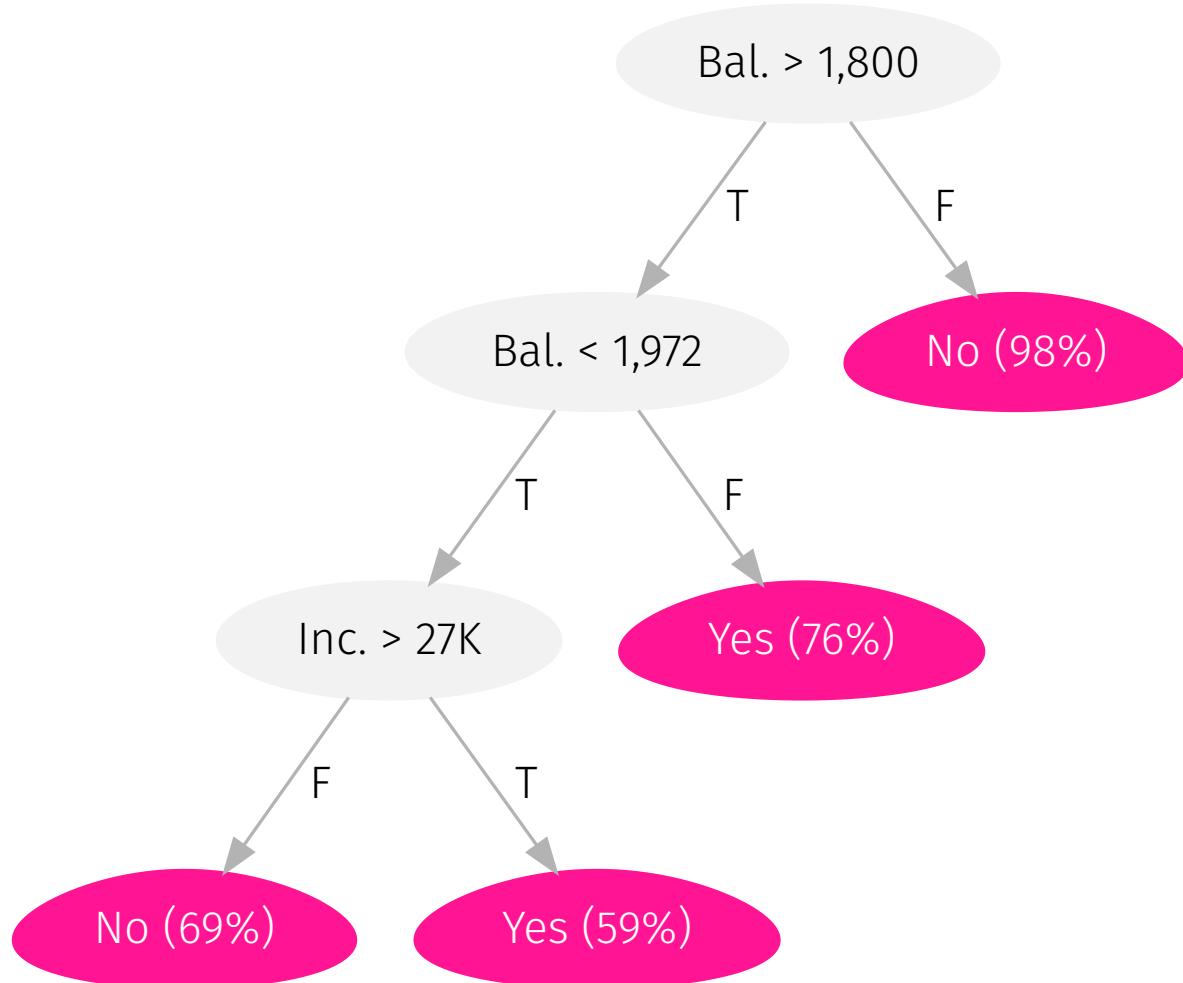
**Predictions** cover each region (e.g., using the region's most common class).



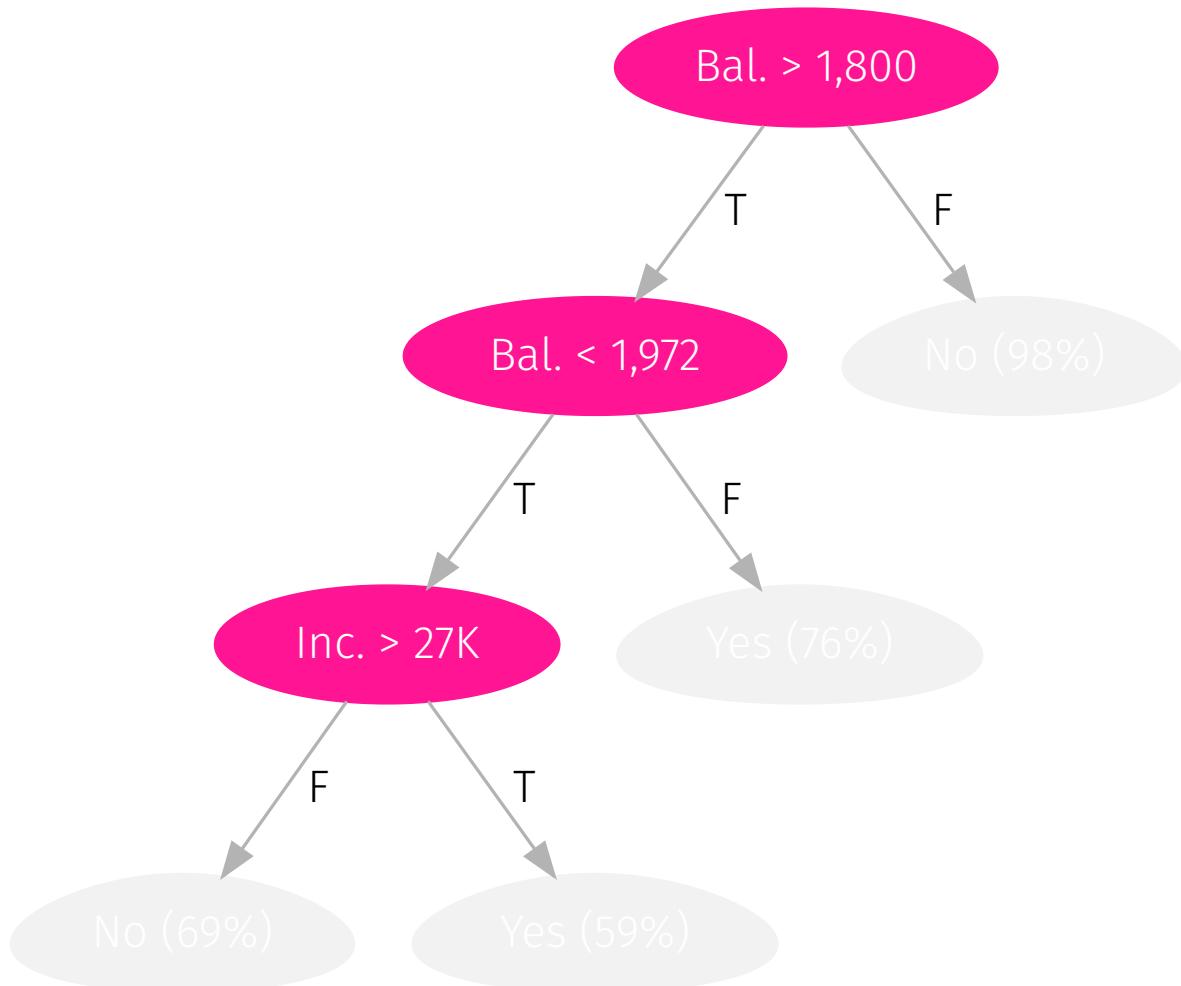
**Predictions** cover each region (e.g., using the region's most common class).



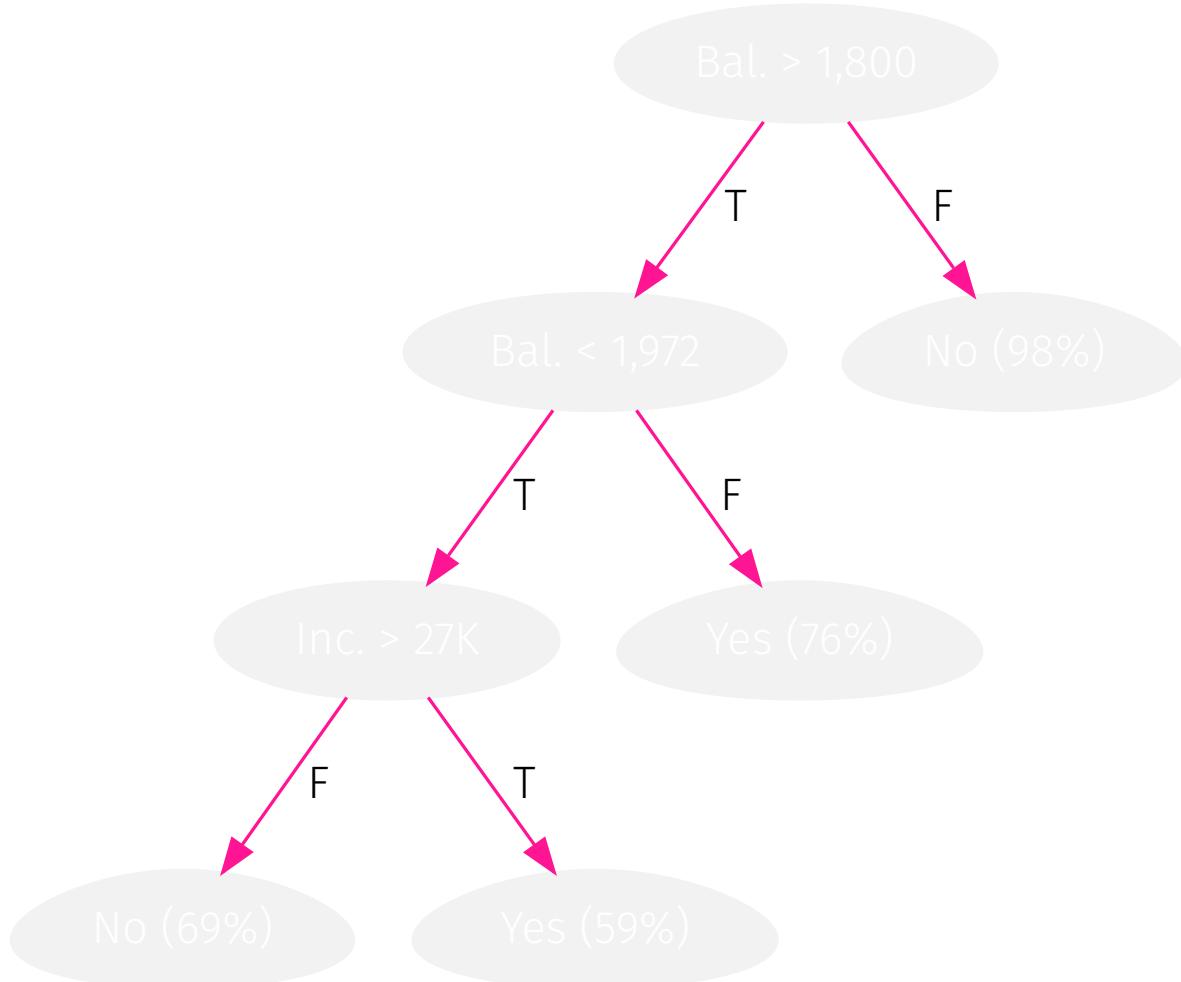
The **regions** correspond to the tree's **terminal nodes** (or **leaves**).



The graph's **separating lines** correspond to the tree's **internal nodes**.



The segments connecting the nodes within the tree are its **branches**.



You now know the anatomy of a decision tree.

But where do trees come from—how do we train a tree?

 grow

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

As we saw, the task of **growing a tree** involves two main steps:

1. **Divide the predictor space** into  $J$  regions (using predictors  $\mathbf{x}_1, \dots, \mathbf{x}_p$ )

# Decision trees

## Growing trees

We will start with **regression trees**, i.e., trees used in regression settings.

As we saw, the task of **growing a tree** involves two main steps:

1. **Divide the predictor space** into  $J$  regions (using predictors  $\mathbf{x}_1, \dots, \mathbf{x}_p$ )
2. **Make predictions** using the regions' mean outcome.

For region  $R_j$  predict  $\hat{y}_{R_j}$  where

$$\hat{y}_{R_j} = \frac{1}{n_j} \sum_{i \in R_j} y$$

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

**Problem:** Examining every possible partition is computationally infeasible.

# Decision trees

## Growing trees

We **choose the regions to minimize RSS** across all  $J$  regions, i.e.,

$$\sum_{j=1}^J \left( y_i - \hat{y}_{R_j} \right)^2$$

**Problem:** Examining every possible partition is computationally infeasible.

**Solution:** a *top-down, greedy* algorithm named **recursive binary splitting**

- **recursive** start with the "best" split, then find the next "best" split, ...
- **binary** each split creates two branches—"yes" and "no"
- **greedy** each step makes *best* split—no consideration of overall process

# Decision trees

## Growing trees: Choosing a split

*Recall* Regression trees choose the split that minimizes RSS.

To find this split, we need

1. a predictor,  $\mathbf{x}_j$
2. a cutoff  $s$  that splits  $\mathbf{x}_j$  into two parts: (1)  $\mathbf{x}_j < s$  and (2)  $\mathbf{x}_j \geq s$

# Decision trees

## Growing trees: Choosing a split

*Recall* Regression trees choose the split that minimizes RSS.

To find this split, we need

1. a predictor,  $\mathbf{x}_j$
2. a cutoff  $s$  that splits  $\mathbf{x}_j$  into two parts: (1)  $\mathbf{x}_j < s$  and (2)  $\mathbf{x}_j \geq s$

Searching across each of our predictors  $j$  and all of their cutoffs  $s$ , we choose the combination that **minimizes RSS**.

# Decision trees

## Example: Splitting

*Example* Consider the dataset

i	y	x <sub>1</sub>	x <sub>2</sub>
1	0	1	4
2	8	3	2
3	6	5	6

# Decision trees

## Example: Splitting

*Example* Consider the dataset

i	y	x <sub>1</sub>	x <sub>2</sub>
1	0	1	4
2	8	3	2
3	6	5	6

With just three observations, each variable only has two actual splits. 

 You can think about cutoffs as the ways we divide observations into two groups.

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	y	$x_1$	$x_2$
1	0	1	4
2	8	3	2
3	6	5	6

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

Note<sub>1</sub> Splitting  $x_1$  at 2 yields the same results as 1.5, 2.5—anything in (1, 3).

# Decision trees

## Example: Splitting

One possible split:  $x_1$  at 2, which yields (1)  $x_1 < 2$  vs. (2)  $x_1 \geq 2$

i	pred.	y	$x_1$	$x_2$
1	0	0	1	4
2	7	8	3	2
3	7	6	5	6

This split yields an RSS of  $0^2 + 1^2 + (-1)^2 = 2$ .

Note<sub>1</sub> Splitting  $x_1$  at 2 yields the same results as 1.5, 2.5—anything in (1, 3).

Note<sub>2</sub> Trees often grow until they hit some number of observations in a leaf.

# Decision trees

## Example: Splitting

An alternative split:  $x_1$  at 4, which yields (1)  $x_1 < 4$  vs. (2)  $x_1 \geq 4$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

# Decision trees

## Example: Splitting

An alternative split:  $x_1$  at 4, which yields (1)  $x_1 < 4$  vs. (2)  $x_1 \geq 4$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

Previous: Splitting  $x_1$  at 4 yielded RSS = 2. (*Much better*)

# Decision trees

## Example: Splitting

Another split:  $x_2$  at 3, which yields (1)  $x_1 < 3$  vs. (2)  $x_1 \geq 3$

i	pred.	y	$x_1$	$x_2$
1	3	0	1	4
2	8	8	3	2
3	3	6	5	6

This split yields an RSS of  $(-3)^2 + 0^2 + 3^2 = 18$ .

# Decision trees

## Example: Splitting

Final split:  $x_2$  at 5, which yields (1)  $x_1 < 5$  vs. (2)  $x_1 \geq 5$

i	pred.	y	$x_1$	$x_2$
1	4	0	1	4
2	4	8	3	2
3	6	6	5	6

This split yields an RSS of  $(-4)^2 + 4^2 + 0^2 = 32$ .

# Decision trees

## Example: Splitting

Across our four possible splits (two variables each with two splits)

- $x_1$  with a cutoff of 2: **RSS** = 2
- $x_1$  with a cutoff of 4: **RSS** = 32
- $x_2$  with a cutoff of 3: **RSS** = 18
- $x_2$  with a cutoff of 5: **RSS** = 32

our split of  $x_1$  at 2 generates the lowest RSS.

Note: Categorical predictors work in exactly the same way.

We want to try **all possible combinations** of the categories.

*Ex:* For a four-level categorical predictor (levels: A, B, C, D)

- Split 1: A|B|C vs. D
- Split 2: A|B|D vs. C
- Split 3: A|C|D vs. B
- Split 4: B|C|D vs. A
- Split 5: A|B vs. C|D
- Split 6: A|C vs. B|D
- Split 7: A|D vs. B|C

we would need to try 7 possible splits.

# Decision trees

## More splits

Once we make our a split, we then continue splitting, **conditional** on the regions from our previous splits.

So if our first split creates  $R_1$  and  $R_2$ , then our next split searches the predictor space only in  $R_1$  or  $R_2$ . 

 We are no longer searching the full space—it is conditional on the previous splits.

# Decision trees

## More splits

Once we make our a split, we then continue splitting, **conditional** on the regions from our previous splits.

So if our first split creates  $R_1$  and  $R_2$ , then our next split searches the predictor space only in  $R_1$  or  $R_2$ . 

The tree continue to **grow until** it hits some specified threshold, e.g., at most 5 observations in each leaf.

 We are no longer searching the full space—it is conditional on the previous splits.

# Decision trees

## Too many splits?

One can have too many splits.

Q Why?

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

**Q** So what can we do?

# Decision trees

## Too many splits?

One can have too many splits.

**Q** Why?

**A** "More splits" means

1. more flexibility (think about the bias-variance tradeoff/overfitting)
2. less interpretability (one of the selling points for trees)

**Q** So what can we do?

**A** Prune your trees!

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

*Updated strategy:* Grow big trees  $T_0$  and then trim  $T_0$  to an optimal **subtree**.

# Decision trees

## Pruning

**Pruning** allows us to trim our trees back to their "best selves."

*The idea:* Some regions may increase **variance** more than they reduce **bias**. By removing these regions, we gain in test MSE.

*Candidates for trimming:* Regions that do not **reduce RSS** very much.

*Updated strategy:* Grow big trees  $T_0$  and then trim  $T_0$  to an optimal **subtree**.

*Updated problem:* Considering all possible subtrees can get expensive.

# Decision trees

## Pruning

Cost-complexity pruning<sup>▲</sup> offers a solution.

Just as we did with lasso, **cost-complexity pruning** forces the tree to pay a price (penalty) to become more complex.

Complexity here is defined as the number of regions  $|T|$ .



Also called: *weakest-link pruning*.

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, i.e.,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha(\geq 0)$ , we get a subtree  $T \subset T_0$ .

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, i.e.,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha (\geq 0)$ , we get a subtree  $T \subset T_0$ .

$\alpha = 0$  generates  $T_0$ , but as  $\alpha$  increases, we begin to cut back the tree.

# Decision trees

## Pruning

Specifically, **cost-complexity pruning** adds a penalty of  $\alpha|T|$  to the RSS, i.e.,

$$\sum_{m=1}^{|T|} \sum_{i:x \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$$

For any value of  $\alpha (\geq 0)$ , we get a subtree  $T \subset T_0$ .

$\alpha = 0$  generates  $T_0$ , but as  $\alpha$  increases, we begin to cut back the tree.

We choose  $\alpha$  via cross validation.

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

### Regression trees

- **Predict:** Region's mean
- **Split:** Minimize RSS
- **Prune:** Penalized RSS

# Decision trees

## Classification trees

Classification with trees is very similar to regression.

### Regression trees

- **Predict:** Region's mean
- **Split:** Minimize RSS
- **Prune:** Penalized RSS

### Classification trees

- **Predict:** Region's mode
- **Split:** Min. Gini or entropy
- **Prune:** Penalized error rate

An additional nuance for **classification trees**: We typically care about the **proportions of classes in the leaves**—not just the final prediction.

 Defined on the next slide.  ... or Gini index or entropy

# Decision trees

## The Gini index

Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

# Decision trees

## The Gini index

Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

The **Gini index** tells us about a region's "purity" 

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

if a region is very homogeneous, then the Gini index will be small.

Homogenous regions are easier to predict.

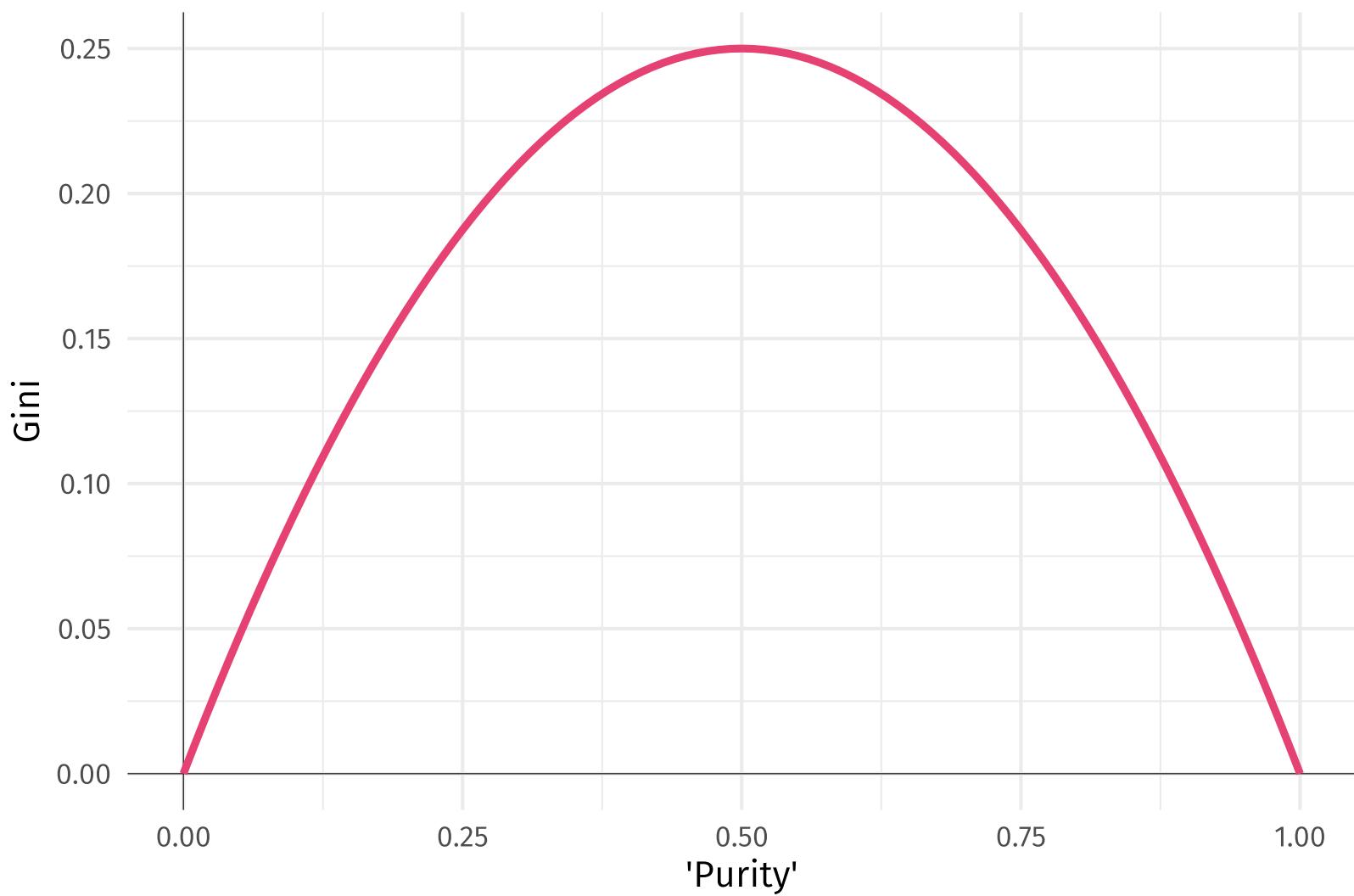
Reducing the Gini index yields to more homogeneous regions

.  
∴ We want to minimize the Gini index.



This vocabulary is Voldemort's contribution to the machine-learning literature.

## Gini as a function of 'purity'



# Decision trees

## Entropy

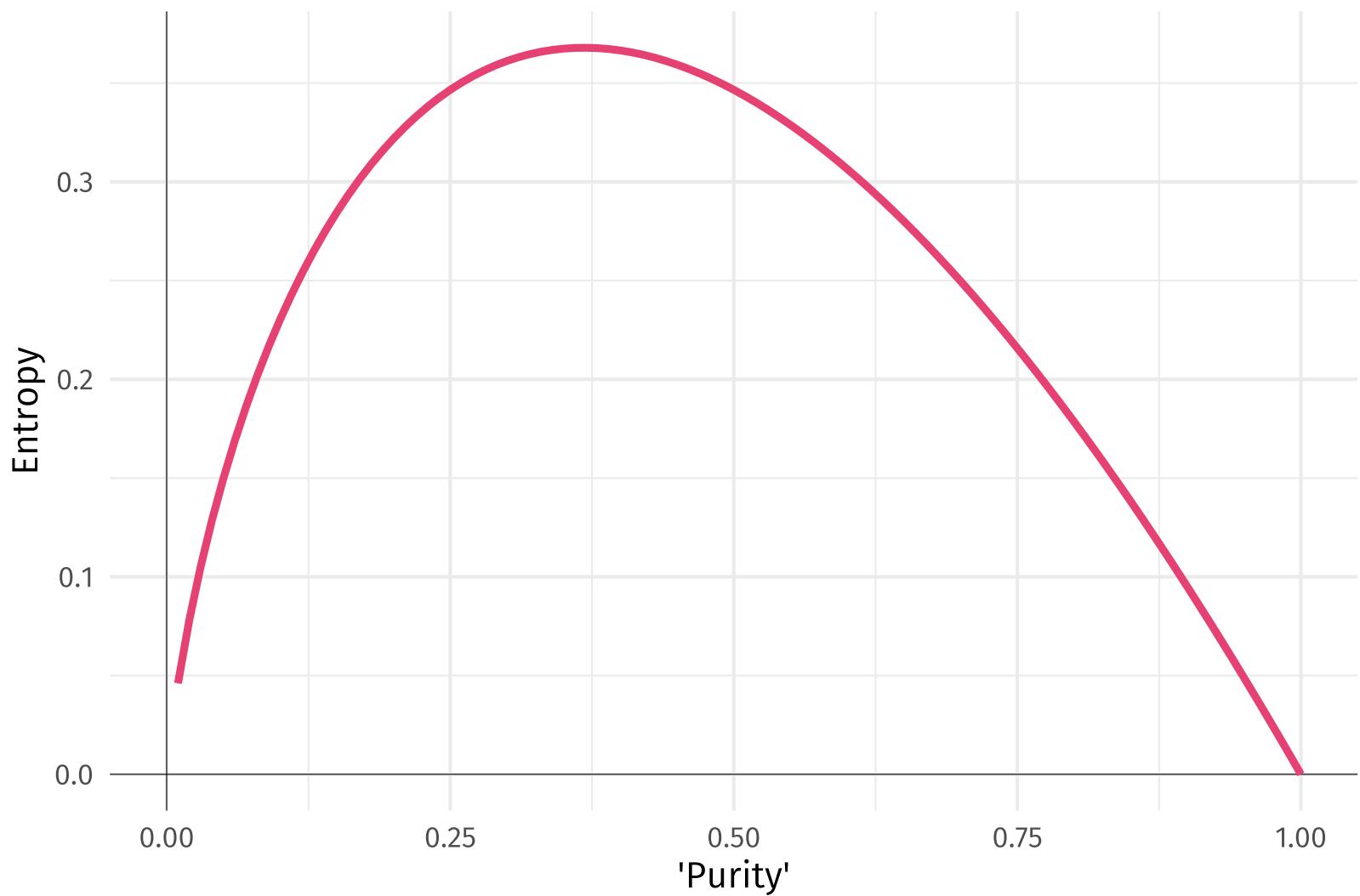
Let  $\hat{p}_{mk}$  denote the proportion of observations in class  $k$  and region  $m$ .

**Entropy** also measures the "purity" of a node/leaf

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$$

**Entropy** is also minimized when  $\hat{p}_{mk}$  values are close to 0 and 1.

## Entropy as a function of 'purity'



# Decision trees

## Rational

Q Why are we using the Gini index or entropy (vs. error rate)?

# Decision trees

## Rational

**Q** Why are we using the Gini index or entropy (vs. error rate)?

**A** The error rate isn't sufficiently sensitive to grow good trees.

The Gini index and entropy tell us about the **composition** of the leaf.

# Decision trees

## Rational

**Q** Why are we using the Gini index or entropy (vs. error rate)?

**A** The error rate isn't sufficiently sensitive to grow good trees.

The Gini index and entropy tell us about the **composition** of the leaf.

*Ex.* Consider two different leaves in a three-level classification.

### Leaf 1

- **A:** 51, **B:** 49, **C:** 00
- **Error rate:** 49%
- **Gini index:** 0.4998
- **Entropy:** 0.6929

### Leaf 2

- **A:** 51, **B:** 25, **C:** 24
- **Error rate:** 49%
- **Gini index:** 0.6198
- **Entropy:** 1.0325

The **Gini index** and **entropy** tell us about the distribution.

# Decision trees

## Classification trees

When **growing** classification trees, we want to use the Gini index or entropy.

However, when **pruning**, the error rate is typically fine—especially if accuracy will be the final criterion.

# Decision trees

## In R

To train decision trees in R, we can use `parsnip`, which draws upon `rpart`.

In `parsnip`, we use the aptly named `decision_tree()` function.

# Decision trees

## In R

To train decision trees in R, we can use `parsnip`, which draws upon `rpart`.

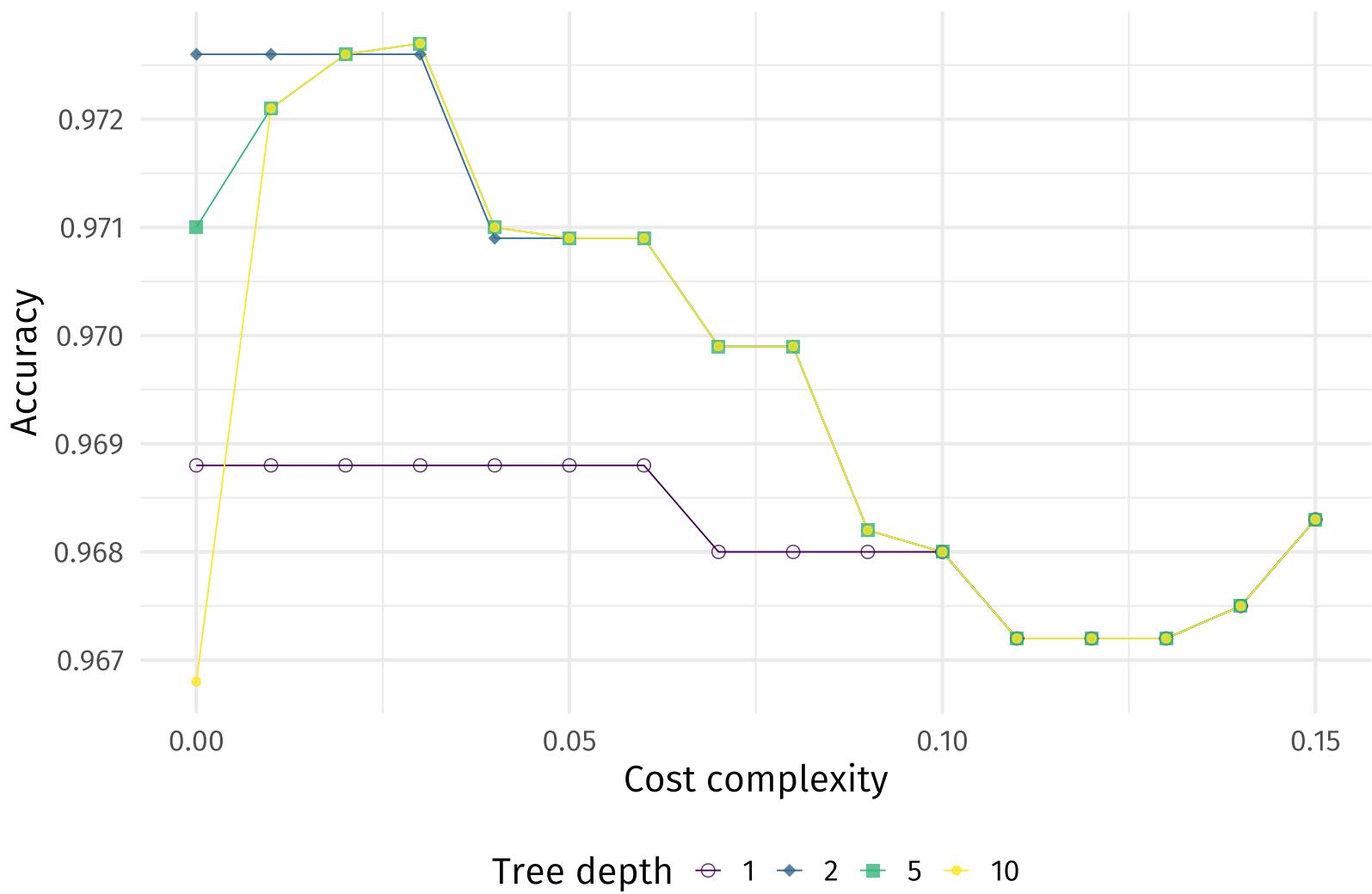
In `parsnip`, we use the aptly named `decision_tree()` function.

The `decision_tree()` model (with `rpart` engine) wants four inputs:

- `mode`: "regression" or "classification"
- `cost_complexity`: the cost (penalty) paid for complexity
- `tree_depth`: max. tree depth (max. number of splits in a "branch")
- `min_n`: min. # of observations for a node to split

```
# Define our CV split
set.seed(12345)
default_cv = default_df %>% vfold_cv(v = 5)
# Define the decision tree
default_tree = decision_tree(
  mode = "classification",
  cost_complexity = tune(),
  tree_depth = tune(),
  min_n = 10 # Arbitrarily choosing '10'
) %>% set_engine("rpart")
# Define recipe
default_recipe = recipe(default ~ ., data = default_df)
# Define the workflow
default_flow = workflow() %>%
  add_model(default_tree) %>% add_recipe(default_recipe)
# Tune!
default_cv_fit = default_flow %>% tune_grid(
  default_cv,
  grid = expand_grid(
    cost_complexity = seq(0, 0.15, by = 0.01),
    tree_depth = c(1, 2, 5, 10),
  ),
  metrics = metric_set(accuracy, roc_auc)
)
```

# Accuracy, complexity, and depth



## To plot the CV-chosen tree...

1. **Fit** the chosen/best model.

```
best_flow =  
  default_flow %>%  
  finalize_workflow(select_best(default_cv_fit, metric = "accuracy")) %>%  
  fit(data = default_df)
```

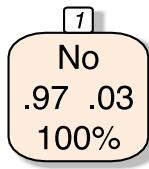
2. **Extract** the fitted model, e.g., with `extract_fit_parsnip`.

*Old/deprecated way:* `pull_workflow_fit()`

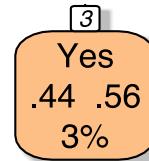
```
best_tree = best_flow %>% extract_fit_parsnip()
```

3. **Plot** the tree, e.g., with `rpart.plot()` from `rpart.plot`.

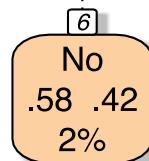
```
best_tree$fit %>% rpart.plot()
```



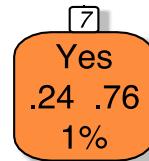
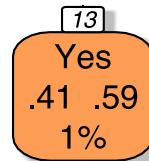
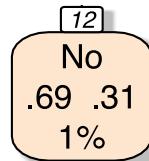
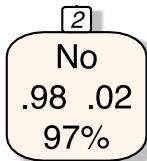
yes · **balance < 1800** · no



**balance < 1972**



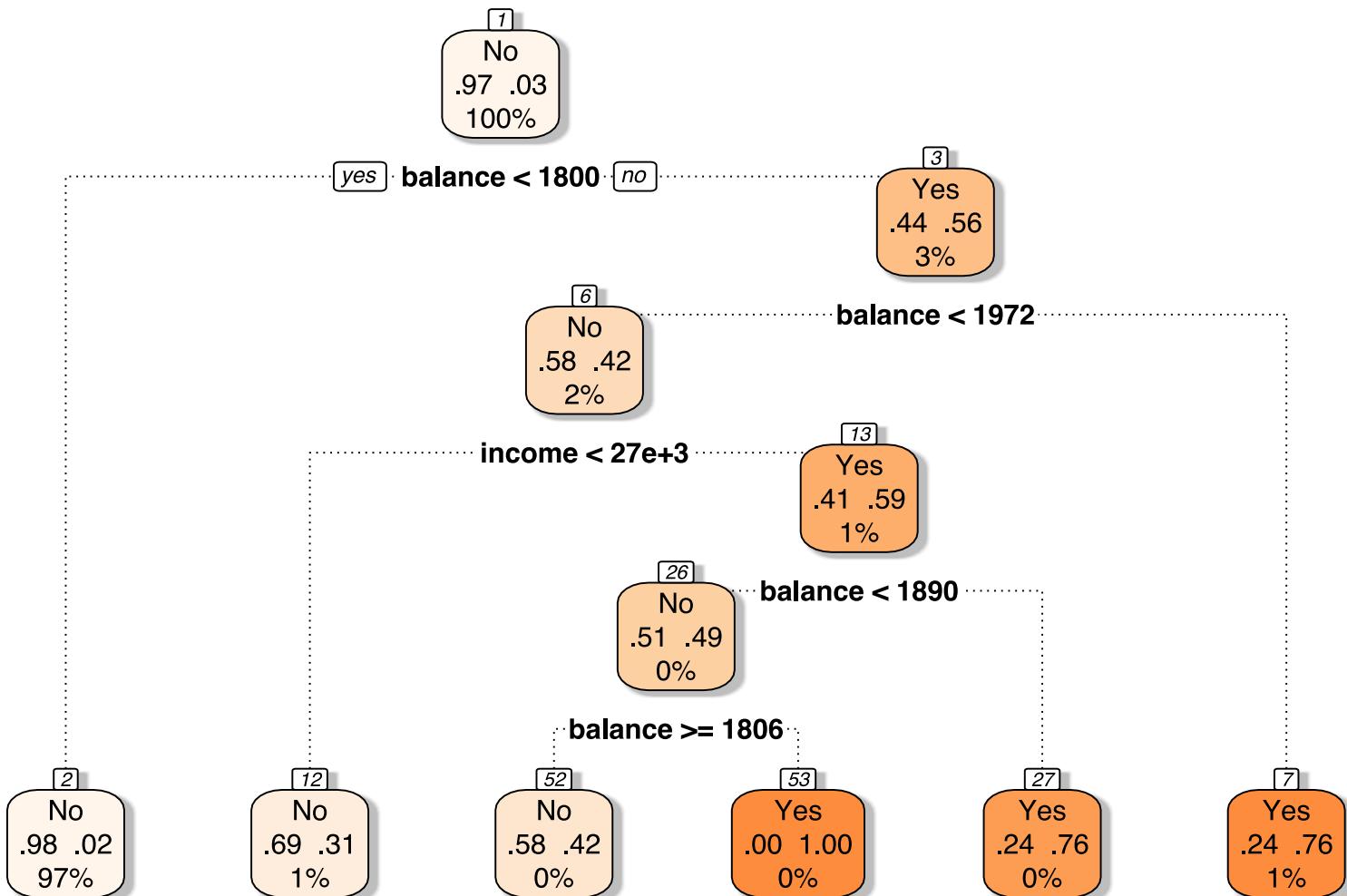
**income < 27e+3**



The previous tree has cost complexity of 0.03 (and a max. depth of 5).

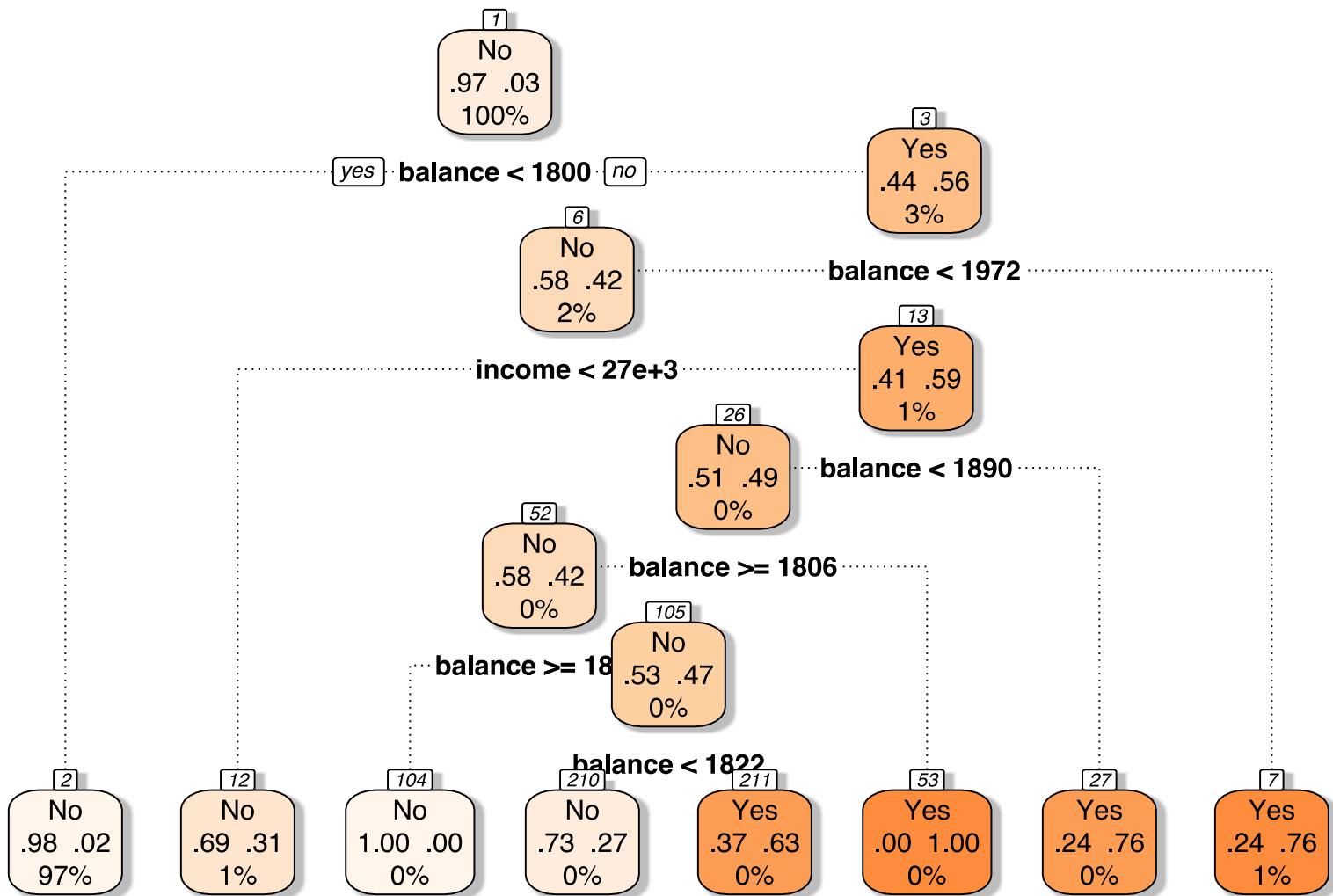
We can compare this "best" tree to a less pruned/penalized tree

- `cost_complexity = 0.005`
- `tree_depth = 5`



What if we hold the cost complexity constant but increase the max. depth?

- `cost_complexity = 0.005`
- `tree_depth = 10` (moved up from 5)



What if we ratchet up complexity constant?

- `cost_complexity = 0.1` (increased from `0.005`)
- `tree_depth = 10`

**1**  
No  
.97 .03  
100%

**yes** balance < 1800 **no**

**2**  
No  
.98 .02  
97%

**3**  
Yes  
.44 .56  
3%

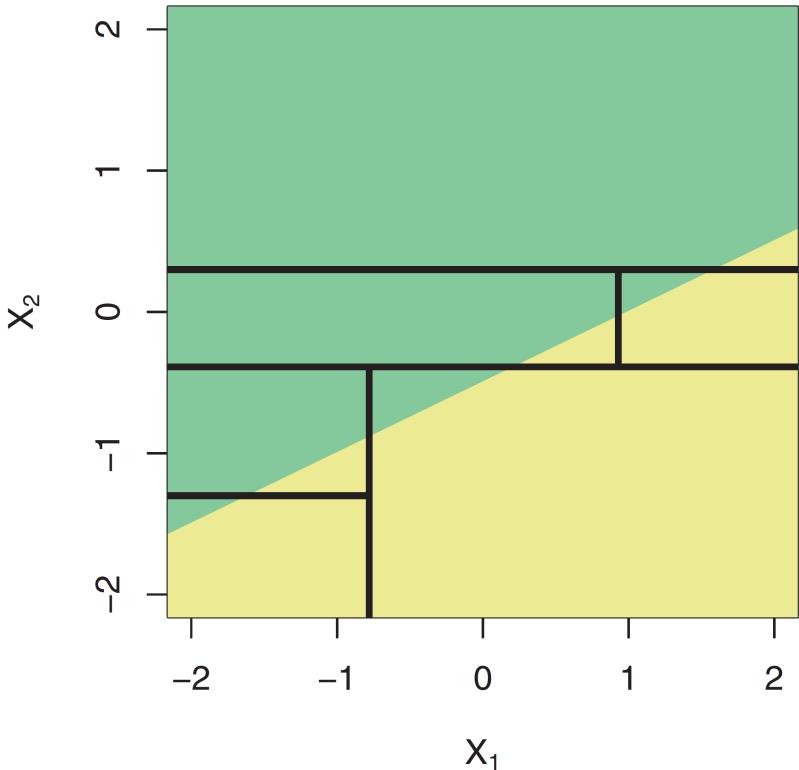
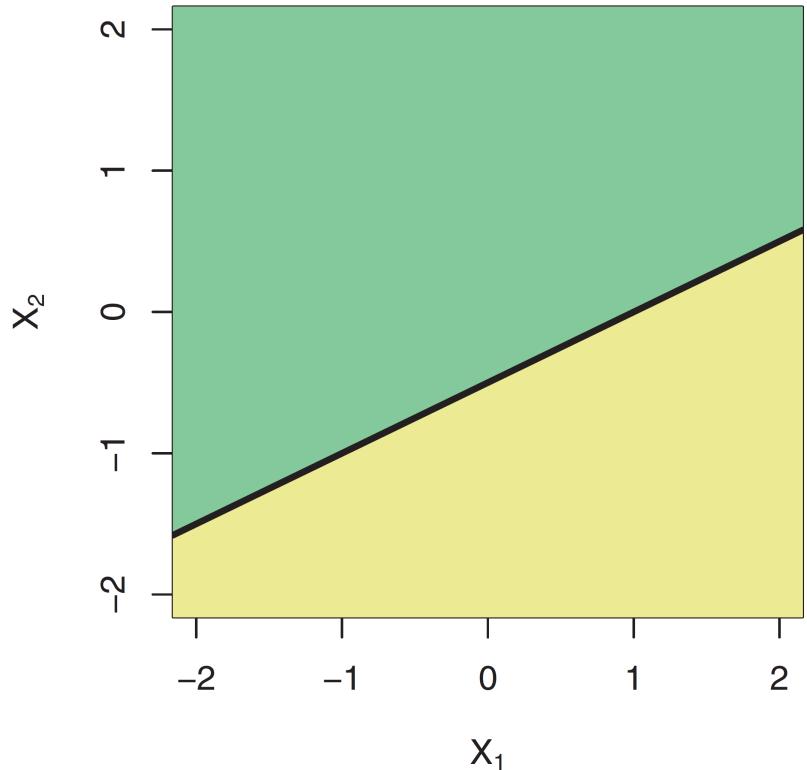
*Fun tool* Grant McDermott's `parttree` package.

**Q** How do trees compare to linear models?

**Q** How do trees compare to linear models?

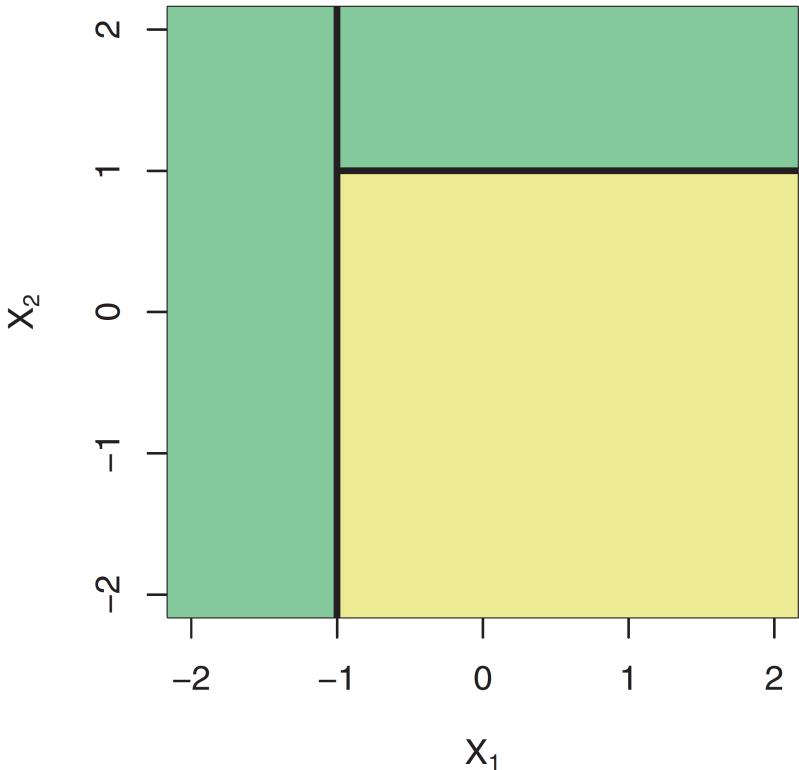
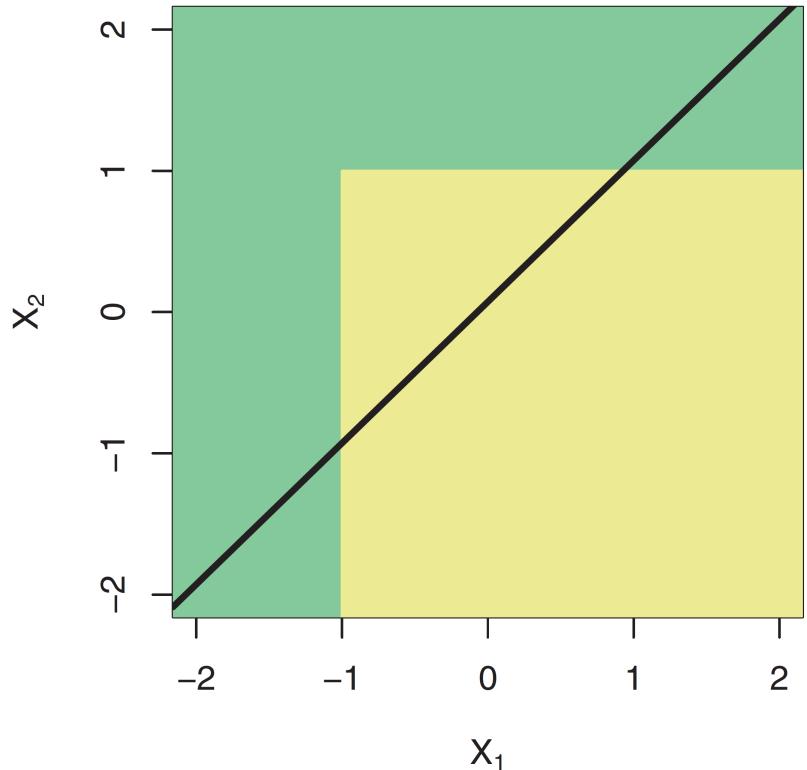
**A** It depends how linear the true boundary is.

**Linear boundary:** trees struggle to recreate a line.



Source: ISL, p. 315

**Nonlinear boundary:** trees easily replicate the nonlinear boundary.



Source: ISL, p. 315

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS 

-  Without needing to create lots of dummy variables!

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS 

### Weaknesses

- Outperformed by other methods
- Struggle with linearity
- Can be very "non-robust"

**Non-robust:** Small data changes can cause huge changes in our tree.

 Without needing to create lots of dummy variables!

# Decision trees

## Strengths and weaknesses

As with any method, decision trees have tradeoffs.

### Strengths

- + Easily explained/interpreted
- + Include several graphical options
- + Mirror human decision making?
- + Handle num. or cat. on LHS/RHS 

### Weaknesses

- Outperformed by other methods
- Struggle with linearity
- Can be very "non-robust"

**Non-robust:** Small data changes can cause huge changes in our tree.

Next: Create ensembles of trees  to strengthen these weaknesses. 

 Without needing to create lots of dummy variables!

 Forests!  Which will also weaken some of the strengths.

# Sources

These notes draw upon

- An Introduction to Statistical Learning (*ISL*)  
James, Witten, Hastie, and Tibshirani

# Table of contents

## Admin

- Today
- Upcoming

## Other

- Sources/references

## Decision trees

1. Fundamentals
2. Partitioning predictors
3. Definitions
4. Growing trees
5. Example: Splitting
6. More splits
7. Pruning
8. Classification trees
  - The Gini index
  - Entropy
  - Rationale
9. In R
10. Linearity
11. Strengths and weaknesses