Chapter 11: Algorithmic Fairness and Ethics

Algorithmic Game Theory

Confusion matrix · Demographic parity · Equalized odds · Predictive parity · Calibration · Impossibility theorems · Cut and choose · Proportionality and envy-freeness · Ethics and auditing

Sercan KÜLCÜ · Giresun University · Computer Engineering

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides:
https://sercankulcu.github.io/files/agt/slides/Chapter_11_Algorithmic_Fairness_and_Ethics.html

Warm-up: A Fair Division Game

The rules

Two players take turns picking indivisible items, one at a time. Each item card shows how much each player values it. The four items on the right are our running example.

Think about it

- If your opponent always grabs their favorite item, should you always grab yours?
- Is there always a split in which everyone values their own share at least as much as the other's?

Cake	Book	Phones	Ball
P1: 5	P1: 4	P1: 3	P1: 1
P2: 1	P2: 9	P2: 8	P2: 1

Order: P1, P2, P1, P2

P2 (computer) always takes the item it values most among those left.

Learning Outcomes

Compute the criteria

Compute demographic parity, equalized odds, predictive parity and calibration from the confusion matrices of two groups.

Show the conflict

Show numerically why the criteria cannot all hold when base rates differ.

Fair division

Apply cut and choose; distinguish proportionality from envy-freeness.

Ethics and incentives

Discuss how transparency, accountability, auditing and strategic behavior shape design.

INTRODUCTION

Why Do Algorithmic Decisions Matter?

- Algorithms now make or steer high-stakes decisions: **lending, hiring, medical diagnosis, social media feeds, criminal justice**.
- Unfair or unethical decisions harm individuals and erode **public trust** in institutions and technology.
- The game-theory link: a decision rule is a **mechanism**, and its stakeholders (applicants, institutions, society) have conflicting interests.

Amazon hiring (2018)
A model trained on past résumés penalized applications containing the word "women's" and was scrapped.

COMPAS (2016)
A recidivism risk score and the debate over unequal error rates: with different base rates, fairness criteria conflict.

Credit scoring
Does the same score mean the same risk for everyone? Who never gets credit at all?

Where Do Bias and Discrimination Come From?

Historical data

Past inequalities end up in the labels, and the model learns them as "truth" (gender bias in hiring).

Underrepresentation

If a group is rare in the data, the model errs more on it (face recognition fails most on darker-skinned women: Gender Shades, 2018).

Measurement and proxies

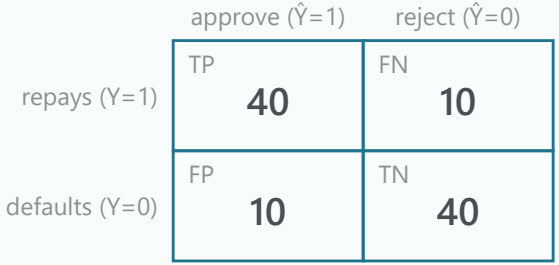
Measure a proxy ("health spending") instead of the target ("health need") and groups with less access look healthier.

Design and feedback

The wrong objective, proxies for a protected attribute (zip code), and decisions that reshape future data.

Dropping the protected attribute (gender, ethnicity) is not enough: other variables carry it indirectly. That is why outcomes must be **measured per group**.

The Confusion Matrix



Lending example: Y = does the applicant actually repay, $\hat{Y}$ = does the model approve; n = TP + FP + FN + TN

Rate	Definition	Question
Selection rate	(TP + FP) / n	What fraction is approved?
TPR (true positive rate)	TP / (TP + FN)	How many future repayers were approved?
FPR (false positive rate)	FP / (FP + TN)	How many future defaulters were approved?
PPV (precision)	TP / (TP + FP)	How many of the approved repay?
Base rate p	(TP + FN) / n	Share of repayers in the group

Each fairness criterion asks for one of these rates to be **equal across groups**. Which one to equalize is a value judgment.

Example: Two Groups, One Model

group	score	people	repay	default	decision (threshold 0.5)
A	0.80	50	40	10	approve
A	0.20	50	10	40	reject
B	0.80	20	16	4	approve
B	0.20	80	16	64	reject

100 applicants per group; the model outputs two scores: 0.8 ("low risk") and 0.2 ("high risk")

The share of repayers (the base rate) is 50% in group A and 32% in group B. The model uses a 0.5 threshold.

Example: Two Groups, One Model

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Score 0.8 is approved: in A, 40 repay (TP) + 10 default (FP); in B, 16 TP + 4 FP.

Example: Two Groups, One Model

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Score 0.2 is rejected: in A, 10 would-be repayers are rejected (FN) and 40 are correctly rejected (TN); in B, 16 FN + 64 TN.

Example: Two Groups, One Model

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Both confusion matrices are ready. Accuracy is 80% in both groups ((40+40)/100 and (16+64)/100). Is the model "fair"? Let us check the criteria one by one.

Demographic Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Demographic parity: $P(\hat{Y}=1 \mid A) = P(\hat{Y}=1 \mid B)$. Approved: 40 + 10 = 50 in A, 16 + 4 = 20 in B.

Demographic Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Selection rates: A = 50/100 = 0.50, B = 20/100 = 0.20. A gap of 0.30 ⇒ demographic parity **fails**.

Demographic Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Ratio: 0.20 / 0.50 = 0.40 < 0.8, below even the "four-fifths rule" used in US hiring. On its own this is not a legal finding of discrimination.

Demographic Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

But the groups differ: 50 actual repayers in A, 32 in B. Forcing demographic parity means approving defaulters in B or rejecting repayers in A.

Equalized Odds Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Equalized odds (Hardt et al., 2016): error rates must match conditional on the true outcome: $TPR_A = TPR_B$ and $FPR_A = FPR_B$.

Equalized Odds Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

TPR (approval among repayers): A = 40/(40+10) = 0.80, B = 16/(16+16) = 0.50. Half of B's creditworthy applicants are rejected!

Equalized Odds Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

FPR (approval among defaulters): A = 10/(10+40) = 0.20, B = 4/(4+64) = 0.059.

Equalized Odds Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

TPR gap 0.30, FPR gap 0.141 ⇒ equalized odds **fails**. Requiring equal TPR alone is called **equal opportunity**.

Predictive Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Predictive parity: the repayment rate among approved applicants (PPV) must be equal across groups. This is the lender's point of view.

Predictive Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

PPV: A = 40/(40+10) = 0.80, B = 16/(16+4) = 0.80 ⇒ predictive parity **holds**.

Predictive Parity Step by Step

Group A

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 40	FN 10
defaults ($Y=0$)	FP 10	TN 40

Group B

	approve ($\hat{Y}=1$)	reject ($\hat{Y}=0$)
repays ($Y=1$)	TP 16	FN 16
defaults ($Y=0$)	FP 4	TN 64

Rejections too (NPV: share of rejected who really default = TN/(TN + FN)): A = 40/50 = 0.80, B = 64/80 = 0.80. "Approve" and "reject" mean the same thing in both groups.

Calibration Step by Step

group	score s	people	repay	observed rate	equals s?
A	0.80	50	40	?	
A	0.20	50	10	?	
B	0.80	20	16	?	
B	0.20	80	16	?	

Calibration within groups: among people with score s, the repayment rate must be s **in every group**. A score should mean the same risk regardless of group.

Calibration Step by Step

group	score s	people	repay	observed rate	equals s?
A	**0.80**	**50**	**40**	**40/50 = 0.80**	**yes**
A	0.20	50	10	?	
B	0.80	20	16	?	
B	0.20	80	16	?	

Group A, score 0.8: 40 of 50 repay ⇒ 0.80.

Calibration Step by Step

group	score s	people	repay	observed rate	equals s?
A	0.80	50	40	40/50 = 0.80	**yes**
A	**0.20**	**50**	**10**	**10/50 = 0.20**	**yes**
B	0.80	20	16	?	
B	0.20	80	16	?	

Group A, score 0.2: 10 of 50 repay ⇒ 0.20.

Calibration Step by Step

group	score s	people	repay	observed rate	equals s?
A	0.80	50	40	40/50 = 0.80	**yes**
A	0.20	50	10	10/50 = 0.20	**yes**
B	**0.80**	**20**	**16**	**16/20 = 0.80**	**yes**
B	0.20	80	16	?	

Group B, score 0.8: 16 of 20 repay ⇒ 0.80.

Calibration Step by Step

group	score s	people	repay	observed rate	equals s?
A	0.80	50	40	40/50 = 0.80	**yes**
A	0.20	50	10	10/50 = 0.20	**yes**
B	0.80	20	16	16/20 = 0.80	**yes**
B	**0.20**	**80**	**16**	**16/80 = 0.20**	**yes**

Group B, score 0.2: 16 of 80 repay ⇒ 0.20. The model is **calibrated** in both groups: the score is an honest probability.

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

The function is called with group A's matrix.

Step 1 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
▶ A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

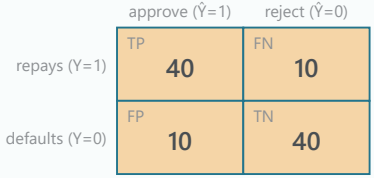
VARIABLES

—

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A



n = 40 + 10 + 10 + 40 = 100 applicants.

Step 2 / 11

```
▶ def metrics(TP, FP, FN, TN):
▶   n = TP + FP + FN + TN
3   sel = (TP + FP) / n      # demographic parity
4   tpr = TP / (TP + FN)     # equalized odds
5   fpr = FP / (FP + TN)
6   ppv = TP / (TP + FP)     # predictive parity
7   return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

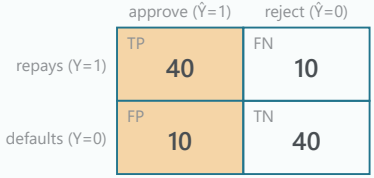
VARIABLES

`TP = 40` `FP = 10` `FN = 10` `TN = 40` `n = 100`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A



Selection rate = (40 + 10) / 100 = 0.5 (compared for demographic parity).

Step 3 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
▶    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`TP = 40` `FP = 10` `FN = 10` `TN = 40` `n = 100` `sel = 0.5`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

TPR = 40 / (40 + 10) = 0.8.

Step 4 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
▶    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`TP = 40` `FP = 10` `FN = 10` `TN = 40` `n = 100` `sel = 0.5` `tpr = 0.8`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

FPR = 10 / (10 + 40) = 0.2.

Step 5 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
▶    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`TP = 40` `FP = 10` `FN = 10` `TN = 40` `n = 100` `sel = 0.5` `tpr = 0.8` `fpr = 0.2`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

PPV = 40 / (40 + 10) = 0.8.

Step 6 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
▶    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`TP = 40` `FP = 10` `FN = 10`
`TN = 40` `n = 100` `sel = 0.5`
`tpr = 0.8` `fpr = 0.2`
`ppv = 0.8`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

The four rates are returned as a Python tuple and assigned to A.

Step 7 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
▶    return sel, tpr, fpr, ppv
8
▶  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

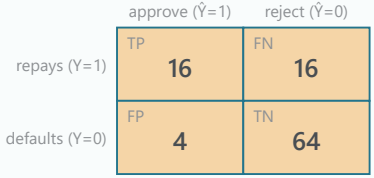
VARIABLES

`A = (0.5, 0.8, 0.2, 0.8)`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group B



Same computation for B: selection 20/100, TPR 16/32, FPR 4/68 ≈ 0.059, PPV 16/20.

Step 8 / 11

```
1  def metrics(TP, FP, FN, TN):
▶    n = TP + FP + FN + TN
▶    sel = (TP + FP) / n      # demographic parity
▶    tpr = TP / (TP + FN)     # equalized odds
▶    fpr = FP / (FP + TN)
▶    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`TP = 16` `FP = 4` `FN = 16` `TN = 64` `n = 100`

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group B

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 16	FN 16
defaults (Y=0)	FP 4	TN 64

B = (0.2, 0.5, 0.0588..., 0.8).

Step 9 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n     # demographic parity
4    tpr = TP / (TP + FN)    # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)    # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
▶  B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
12 for i, name in enumerate(NAMES):
13   print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

A = (0.5, 0.8, 0.2, 0.8)

B = (0.2, 0.5, 0.059, 0.8)

OUTPUT

Code Trace: `metrics(TP, FP, FN, TN)`

Group A

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 40	FN 10
defaults (Y=0)	FP 10	TN 40

No demographic parity: 0.5 ≠ 0.2.

Step 10 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
▶  NAMES = ["sel", "TPR", "FPR", "PPV"]
▶  for i, name in enumerate(NAMES):
▶    print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`i = 0` `name = sel`

OUTPUT

```
sel 0.5 0.2
```

Code Trace: `metrics(TP, FP, FN, TN)`

Group B

	approve (Ŷ=1)	reject (Ŷ=0)
repays (Y=1)	TP 16	FN 16
defaults (Y=0)	FP 4	TN 64

i = 1, 2, 3: TPR differs ⇒ no equal opportunity; FPR differs too ⇒ no equalized odds; PPV equal ⇒ predictive parity holds.

Step 11 / 11

```
1  def metrics(TP, FP, FN, TN):
2    n = TP + FP + FN + TN
3    sel = (TP + FP) / n      # demographic parity
4    tpr = TP / (TP + FN)     # equalized odds
5    fpr = FP / (FP + TN)
6    ppv = TP / (TP + FP)     # predictive parity
7    return sel, tpr, fpr, ppv
8
9  A = metrics(40, 10, 10, 40)
10 B = metrics(16, 4, 16, 64)
11 NAMES = ["sel", "TPR", "FPR", "PPV"]
▶  for i, name in enumerate(NAMES):
▶    print(name, round(A[i], 3), round(B[i], 3))
```

VARIABLES

`i = 3` `name = PPV`

OUTPUT

```
sel 0.5 0.2
TPR 0.8 0.5
FPR 0.2 0.059
PPV 0.8 0.8
```

All Criteria, One Model: The Scorecard

Criterion	Must be equal	A	B	Holds?
Demographic parity	$P(\hat{Y}=1)$	0.50	0.20	**no**
Equal opportunity	TPR = $P(\hat{Y}=1 \mid Y=1)$	0.80	0.50	**no**
Equalized odds	TPR and FPR = $P(\hat{Y}=1 \mid Y=0)$	0.80; 0.20	0.50; 0.059	**no**
Predictive parity	PPV = $P(Y=1 \mid \hat{Y}=1)$	0.80	0.80	**yes**
Calibration	$P(Y=1 \mid \text{score} = s) = s$	0.8→0.8; 0.2→0.2	0.8→0.8; 0.2→0.2	**yes**

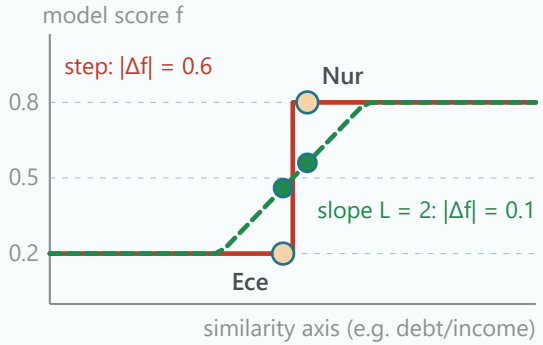
The applicant's view

"If I would repay, does my chance of approval depend on my group?" ⇒ TPR / FPR.

The lender's view

"Does the repayment chance of someone I approve depend on their group?" ⇒ PPV, calibration.

Beyond Group Fairness: Individual Fairness



distance between Ece and Nur d = 0.05; L·d = 0.1

- **Dwork et al. (2012)**: "treat similar people similarly": $|f(x) - f(y)| \leq L \cdot d(x, y)$ (the Lipschitz condition). f(x) is person x's score; d(x, y) measures how different two people are; L is the largest allowed slope. Two people's scores may differ by at most L times their distance.
- The red step model gives Ece and Nur, on either side of the threshold, 0.2 and 0.8: $|\Delta f| = 0.6 > 0.1 \Rightarrow$ **violation**. The green model satisfies the condition.
- Group criteria equalize **averages**: approving random people from B achieves demographic parity but is arbitrary for individuals.

The catch: who chooses the metric d? "Who counts as similar?" is itself a value judgment. Individual fairness does not remove the problem; it moves it into the choice of d.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

Chouldechova (2017): the criteria all come from the same four numbers, so they are linked.
Start with the base rate p.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

PPV = TP / (TP + FP) = p·TPR / (p·TPR + (1 − p)·FPR)

Write PPV, the subject of predictive parity, in terms of TPR, FPR and p.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

PPV = TP / (TP + FP) = p·TPR / (p·TPR + (1 − p)·FPR)

Clear the denominator: PPV·[p·TPR + (1 − p)·FPR] = p·TPR ⇒ (1 − p)·FPR·PPV = p·TPR·(1 − PPV)

Rearrange: **FPR = p / (1 − p) · (1 − PPV) / PPV · TPR**

Solving for FPR gives an **identity** linking the three rates (FPR, TPR, PPV) and the base rate.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

PPV = TP / (TP + FP) = p·TPR / (p·TPR + (1 − p)·FPR)

Clear the denominator: PPV·[p·TPR + (1 − p)·FPR] = p·TPR ⇒ (1 − p)·FPR·PPV = p·TPR·(1 − PPV)

Rearrange: **FPR = p / (1 − p) · (1 − PPV) / PPV · TPR**

A: p = 0.5 ⇒ 0.5/0.5 · 0.2/0.8 · 0.8 = 1 · 0.25 · 0.8 = **0.20** ✓

Check the identity on group A.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

PPV = TP / (TP + FP) = p·TPR / (p·TPR + (1 − p)·FPR)

Clear the denominator: PPV·[p·TPR + (1 − p)·FPR] = p·TPR ⇒ (1 − p)·FPR·PPV = p·TPR·(1 − PPV)

Rearrange: **FPR = p / (1 − p) · (1 − PPV) / PPV · TPR**

A: p = 0.5 ⇒ 0.5/0.5 · 0.2/0.8 · 0.8 = 1 · 0.25 · 0.8 = **0.20** ✓

B: p = 0.32 ⇒ 0.32/0.68 · 0.25 · 0.5 = **0.059** ✓

Check it on group B: it matches the 4/68 in the table.

Why Can't the Criteria Hold Together?

p = base rate, n people ⇒ TP = p·n·TPR, FP = (1 − p)·n·FPR

PPV = TP / (TP + FP) = p·TPR / (p·TPR + (1 − p)·FPR)

Clear the denominator: PPV·[p·TPR + (1 − p)·FPR] = p·TPR ⇒ (1 − p)·FPR·PPV = p·TPR·(1 − PPV)

Rearrange: **FPR = p / (1 − p) · (1 − PPV) / PPV · TPR**

A: p = 0.5 ⇒ 0.5/0.5 · 0.2/0.8 · 0.8 = 1 · 0.25 · 0.8 = **0.20** ✓

B: p = 0.32 ⇒ 0.32/0.68 · 0.25 · 0.5 = **0.059** ✓

PPV is equal (0.8) but p differs ⇒ the factor p/(1 − p) differs ⇒ TPR and FPR cannot **both** be equal.

Conclusion: with different base rates, predictive parity and equalized odds cannot both hold (except in extreme cases such as FPR = 0 and PPV = 1).

What If We Enforce Equalized Odds?

Goal: TPR = 0.8 and FPR = 0.2 in B as well (same as A). B has 32 repayers and 68 defaulters.

Imagine a (hypothetical) better model that achieves these error rates for B. Today's model (scores 0.8 and 0.2 only) cannot reach them by moving the threshold or by randomizing approvals.

What If We Enforce Equalized Odds?

Goal: TPR = 0.8 and FPR = 0.2 in B as well (same as A). B has 32 repayers and 68 defaulters.

TP = 32 × 0.8 = 25.6, FP = 68 × 0.2 = 13.6 (expected counts)

Fixing the error rates fixes the top and bottom rows of the confusion matrix.

What If We Enforce Equalized Odds?

Goal: TPR = 0.8 and FPR = 0.2 in B as well (same as A). B has 32 repayers and 68 defaulters.

TP = 32 × 0.8 = 25.6, FP = 68 × 0.2 = 13.6 (expected counts)

Approved = 25.6 + 13.6 = 39.2 ⇒ selection rate 0.39

B's approval rate rises from 0.2 to 0.39.

What If We Enforce Equalized Odds?

Goal: TPR = 0.8 and FPR = 0.2 in B as well (same as A). B has 32 repayers and 68 defaulters.

TP = 32 × 0.8 = 25.6, FP = 68 × 0.2 = 13.6 (expected counts)

Approved = 25.6 + 13.6 = 39.2 ⇒ selection rate 0.39

PPV_B = 25.6 / 39.2 = **0.65** < PPV_A = 0.80

The repayment rate among the approved falls.

What If We Enforce Equalized Odds?

Goal: TPR = 0.8 and FPR = 0.2 in B as well (same as A). B has 32 repayers and 68 defaulters.

TP = 32 × 0.8 = 25.6, FP = 68 × 0.2 = 13.6 (expected counts)

Approved = 25.6 + 13.6 = 39.2 ⇒ selection rate 0.39

PPV_B = 25.6 / 39.2 = **0.65** < PPV_A = 0.80

Equalized odds now holds, but predictive parity breaks: "approve" in B now signals a lower repayment probability (with only approve/reject outputs, the criterion that breaks is PPV).

Which criterion should we keep? Context (the kind of harm, the law) decides, not mathematics.

Calibration and Balance: Kleinberg's Theorem

Balance for the positive class: repayers should get the same average score in every group.

The score-level analogue of equalized odds: balance.

Calibration and Balance: Kleinberg's Theorem

Balance for the positive class: repayers should get the same average score in every group.

A, Y=1: (40·0.8 + 10·0.2)/50 = **0.68**; B, Y=1: (16·0.8 + 16·0.2)/32 = **0.50**

In our calibrated model, B's repayers get a lower average score.

Calibration and Balance: Kleinberg's Theorem

Balance for the positive class: repayers should get the same average score in every group.

A, Y=1: (40·0.8 + 10·0.2)/50 = **0.68**; B, Y=1: (16·0.8 + 16·0.2)/32 = **0.50**

A, Y=0: (10·0.8 + 40·0.2)/50 = **0.32**; B, Y=0: (4·0.8 + 64·0.2)/68 = **0.235**

No balance among defaulters either. Calibrated, yet unbalanced.

Calibration and Balance: Kleinberg's Theorem

Balance for the positive class: repayers should get the same average score in every group.

A, Y=1: (40·0.8 + 10·0.2)/50 = **0.68**; B, Y=1: (16·0.8 + 16·0.2)/32 = **0.50**

A, Y=0: (10·0.8 + 40·0.2)/50 = **0.32**; B, Y=0: (4·0.8 + 64·0.2)/68 = **0.235**

Theorem (Kleinberg, Mullainathan, Raghavan 2017): calibration and balance for both classes can hold together only with perfect prediction *or* equal base rates.

In our example the base rates differ (0.5 and 0.32) and prediction is imperfect: the criteria must conflict.

Change the Assumption: Which Fairness Criterion?

group	approve: repay / default	reject: repay / default	base rate
A	40 / 10	10 / 40	0.50
B	16 / 4	16 / 64	0.32

100 applicants per group, same model and threshold (0.5). **Is it fair?**

Change the Assumption: Which Fairness Criterion?

group	approve: repay / default	reject: repay / default	base rate
A	40 / 10	10 / 40	0.50
B	16 / 4	16 / 64	0.32

100 applicants per group, same model and threshold (0.5). **Is it fair?**

Criterion	A	B	Holds?
Demographic parity (approvals)	0.50	0.20	**no**
Equalized odds (TPR; FPR)	0.80; 0.20	0.50; 0.059	**no**
Predictive parity (PPV)	0.80	0.80	**yes**

Change the Assumption: Which Fairness Criterion?

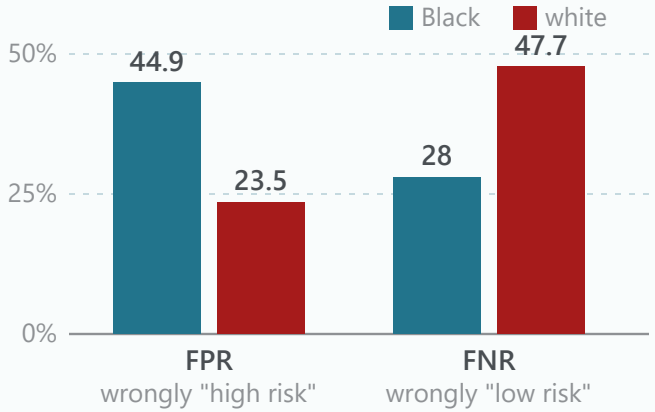
group	approve: repay / default	reject: repay / default	base rate
A	40 / 10	10 / 40	0.50
B	16 / 4	16 / 64	0.32

100 applicants per group, same model and threshold (0.5). **Is it fair?**

Criterion	A	B	Holds?
Demographic parity (approvals)	0.50	0.20	**no**
Equalized odds (TPR; FPR)	0.80; 0.20	0.50; 0.059	**no**
Predictive parity (PPV)	0.80	0.80	**yes**

It depends on the criterion: fair for the lender, unfair for the applicant. With unequal base rates, equalized odds and predictive parity cannot both hold (Chouldechova, 2017; Kleinberg et al., 2017). Choosing one is a value judgment.

A Real Case: The COMPAS Debate



ProPublica (2016), Broward County data: percentages

ProPublica

Among defendants who did not reoffend, 44.9% of Black and 23.5% of white defendants were labeled "high risk": **unequal FPR**. For FNR (the share of reoffenders labeled “low risk”) the picture reverses.

Northpointe (the vendor)

The same risk score corresponds to similar reoffense rates in both groups: **calibration / predictive parity holds**.

Both sides computed correctly. With different base rates, the two criteria cannot hold together (Chouldechova's identity): the real dispute is over **which definition of fairness** to adopt. Report error rates per group, not a single accuracy number.

The Fairness–Accuracy Trade-off

- Accuracy alone does not show **on whom the errors fall**: in our example accuracy is 80% in both groups, yet TPR is 0.8 vs 0.5.
- A fairness constraint may raise total error; the real choice is **which errors are acceptable** (wrongful rejection or wrongful approval?).

Pre-processing
Reweight the data and fix underrepresentation.

In-processing
Add a fairness constraint to the training loss (giving up some accuracy).

Post-processing
Group-specific thresholds (Hardt et al.): may be legally contested.

The context of use, legal requirements, regulation and the views of affected groups should guide the choice of criterion. Example: credit scoring that treats low-income applicants fairly.

Code Trace: Equal Opportunity by Randomization



q = 0
TP = 16 FP = 4

q = 0: only score 0.8 is approved (today's model). TP = 16, FP = 4.

Step 1 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
▶    TP = 16 + 16 * q
▶    FP = 4 + 64 * q
4    tpr = TP / 32
5    fpr = FP / 68
6    hits = TP + (68 - FP)    # TP + TN
7    return tpr, fpr, round(hits / 100, 3)
8
▶  for q in [0, 0.2, 0.4, 0.6]:
▶    tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

VARIABLES

q = 0 | TP = 16 | FP = 4

OUTPUT

Code Trace: Equal Opportunity by Randomization



q = 0
TP = 16 FP = 4
TPR_B = 0.5
FPR_B = 0.06
accuracy_B = 0.8

TPR = 16/32 = 0.5; FPR = 4/68 ≈ 0.06; accuracy (16 + 64)/100 = 0.80.

Step 2 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
2    TP = 16 + 16 * q
3    FP = 4 + 64 * q
▶    tpr = TP / 32
▶    fpr = FP / 68
▶    hits = TP + (68 - FP)    # TP + TN
▶    return tpr, fpr, round(hits / 100, 3)
8
9  for q in [0, 0.2, 0.4, 0.6]:
10   tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

VARIABLES

q = 0 | tpr = 0.5 | fpr = 0.059 | hits = 80

OUTPUT

Code Trace: Equal Opportunity by Randomization



q = 0.2
TP = 19.2 FP = 16.8

q = 0.2: 20% of the 80 people scored 0.2 are approved. TP = 16 + 16·q = 19.2, FP = 4 + 64·q = 16.8.

Step 3 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
▶    TP = 16 + 16 * q
▶    FP = 4 + 64 * q
4    tpr = TP / 32
5    fpr = FP / 68
6    hits = TP + (68 - FP)    # TP + TN
7    return tpr, fpr, round(hits / 100, 3)
8
▶  for q in [0, 0.2, 0.4, 0.6]:
▶    tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

VARIABLES

`q = 0.2` `TP = 19.2` `FP = 16.8`

OUTPUT

```
0 0.5 0.06 0.8
```

Code Trace: Equal Opportunity by Randomization



q = 0.2
TP = 19.2 FP = 16.8
TPR_B = 0.6
FPR_B = 0.25
accuracy_B = 0.704

TPR rose to 0.6 but accuracy fell to 0.704: four of every five new approvals are defaulters.

Step 4 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
2    TP = 16 + 16 * q
3    FP = 4 + 64 * q
▶    tpr = TP / 32
▶    fpr = FP / 68
▶    hits = TP + (68 - FP)    # TP + TN
▶    return tpr, fpr, round(hits / 100, 3)
8
9  for q in [0, 0.2, 0.4, 0.6]:
10   tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

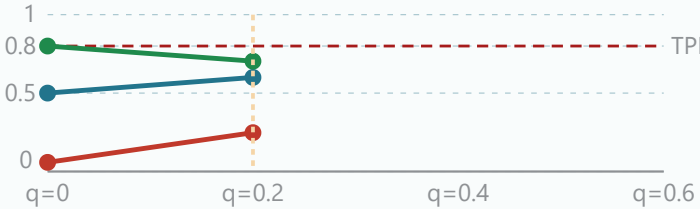
VARIABLES

q = 0.2 | tpr = 0.6 | fpr = 0.247 | hits = 70.4

OUTPUT

```
0 0.5 0.06 0.8
```

Code Trace: Equal Opportunity by Randomization



q = 0.4
TP = 22.4 FP = 29.6

q = 0.4: 40% of the 80 people scored 0.2 are approved. TP = 16 + 16·q = 22.4, FP = 4 + 64·q = 29.6.

Step 5 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
▶    TP = 16 + 16 * q
▶    FP = 4 + 64 * q
4    tpr = TP / 32
5    fpr = FP / 68
6    hits = TP + (68 - FP)    # TP + TN
7    return tpr, fpr, round(hits / 100, 3)
8
▶  for q in [0, 0.2, 0.4, 0.6]:
▶    tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

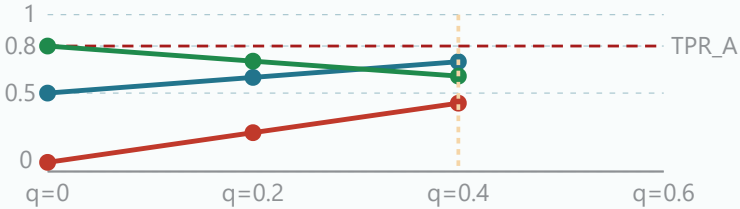
VARIABLES

q = 0.4 | TP = 22.4 | FP = 29.6

OUTPUT

```
0 0.5 0.06 0.8
0.2 0.6 0.25 0.704
```

Code Trace: Equal Opportunity by Randomization



q = 0.4
TP = 22.4 FP = 29.6
TPR_B = 0.7
FPR_B = 0.44
accuracy_B = 0.608

TPR 0.7; FPR jumps to 0.44, accuracy 0.608.

Step 6 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
2    TP = 16 + 16 * q
3    FP = 4 + 64 * q
▶    tpr = TP / 32
▶    fpr = FP / 68
▶    hits = TP + (68 - FP)    # TP + TN
▶    return tpr, fpr, round(hits / 100, 3)
8
9  for q in [0, 0.2, 0.4, 0.6]:
10   tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

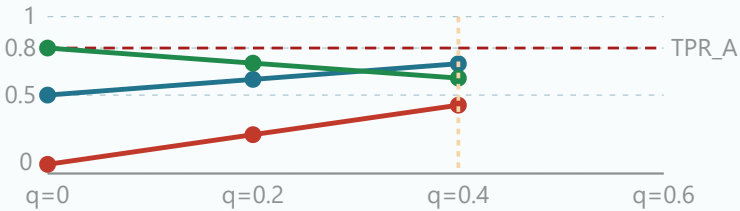
VARIABLES

q = 0.4 | tpr = 0.7 | fpr = 0.435 | hits = 60.8

OUTPUT

```
0 0.5 0.06 0.8
0.2 0.6 0.25 0.704
```

Code Trace: Equal Opportunity by Randomization



q = 0.6
TP = 25.6 FP = 42.4

q = 0.6: 60% of the 80 people scored 0.2 are approved. TP = 16 + 16·q = 25.6, FP = 4 + 64·q = 42.4.

Step 7 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
▶    TP = 16 + 16 * q
▶    FP = 4 + 64 * q
4    tpr = TP / 32
5    fpr = FP / 68
6    hits = TP + (68 - FP)    # TP + TN
7    return tpr, fpr, round(hits / 100, 3)
8
▶  for q in [0, 0.2, 0.4, 0.6]:
▶    tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

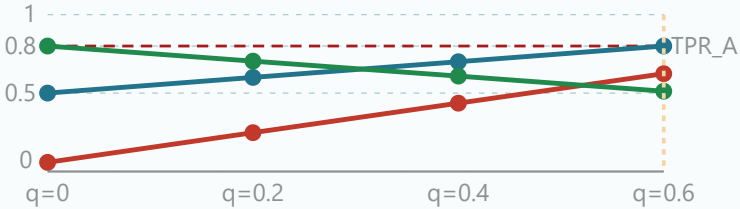
VARIABLES

q = 0.6 | TP = 25.6 | FP = 42.4

OUTPUT

```
0 0.5 0.06 0.8
0.2 0.6 0.25 0.704
0.4 0.7 0.44 0.608
```

Code Trace: Equal Opportunity by Randomization



q = 0.6
TP = 25.6 FP = 42.4
TPR_B = 0.8
FPR_B = 0.62
accuracy_B = 0.512

TPR_B = 0.8 = TPR_A: **equal opportunity achieved**. The price: accuracy 0.512, FPR 0.62.

Step 8 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
2    TP = 16 + 16 * q
3    FP = 4 + 64 * q
▶    tpr = TP / 32
▶    fpr = FP / 68
▶    hits = TP + (68 - FP)    # TP + TN
▶    return tpr, fpr, round(hits / 100, 3)
8
9  for q in [0, 0.2, 0.4, 0.6]:
10   tpr, fpr, d = group_b(q)
11   print(q, round(tpr, 2), round(fpr, 2), d)
```

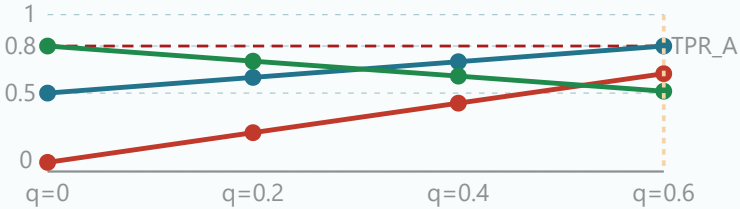
VARIABLES

q = 0.6 | tpr = 0.8 | fpr = 0.624 | hits = 51.2

OUTPUT

```
0 0.5 0.06 0.8
0.2 0.6 0.25 0.704
0.4 0.7 0.44 0.608
```

Code Trace: Equal Opportunity by Randomization



q = 0.6
TP = 25.6 FP = 42.4
TPR_B = 0.8
FPR_B = 0.62
accuracy_B = 0.512

For equal opportunity, 60% of those scored 0.2 were approved at random: about 38 of the 64 defaulters get loans. FPR is more than three times A's (0.2), so equalized odds still fails.

Step 9 / 9

```
1  def group_b(q):   # approve score 0.2 w.p. q
2    TP = 16 + 16 * q
3    FP = 4 + 64 * q
4    tpr = TP / 32
5    fpr = FP / 68
6    hits = TP + (68 - FP)    # TP + TN
7    return tpr, fpr, round(hits / 100, 3)
8
9  for q in [0, 0.2, 0.4, 0.6]:
10   tpr, fpr, d = group_b(q)
▶    print(q, round(tpr, 2), round(fpr, 2), d)
```

VARIABLES

q = 0.6 | tpr = 0.8 | d = 0.512

OUTPUT

```
0 0.5 0.06 0.8
0.2 0.6 0.25 0.704
0.4 0.7 0.44 0.608
0.6 0.8 0.62 0.512
```

Fair Division: Cake Cutting

- A divisible resource (the cake = the interval [0, 1]: time, land, compute) is to be shared among n people.
- Each person has a **valuation function** v_i: the whole cake is worth 1, but people value different parts differently.
- The algorithm may only ask queries (the Robertson–Webb model): "what is this piece worth to you?" and "where does a piece of a given value end?". X_i is the piece person i receives; $v_i(X_i)$ is its value to person i.

Proportionality

Everyone gets at least **1/n** by their own measure: $v_i(X_i) \geq 1/n$.

Envy-freeness

No one prefers someone else's piece to their own: $v_i(X_i) \geq v_i(X_j)$.

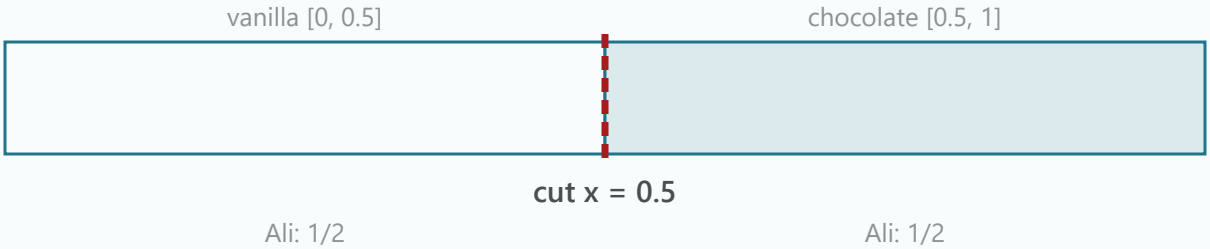
The game-theoretic view: people may be strategic. A good protocol gives every honest participant a **guarantee**, whatever the others do.

Cut and Choose Step by Step



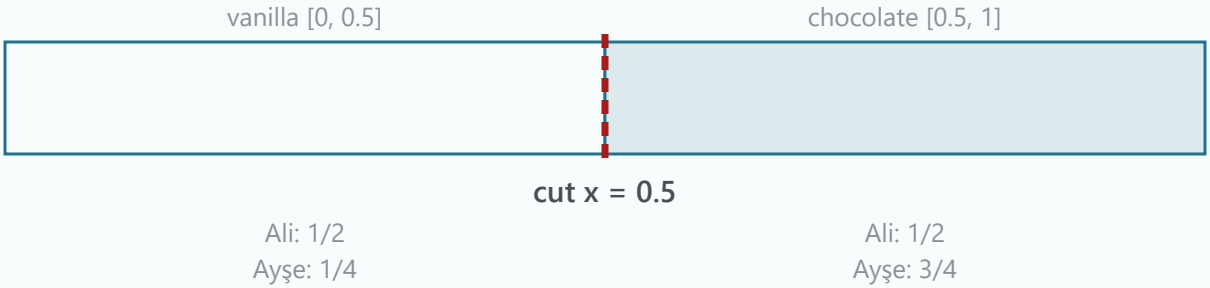
Ali values the whole cake uniformly. Ayşe likes chocolate **3 times** as much as vanilla: $v_{Ayşe}(\text{vanilla}) = 1/4$, $v_{Ayşe}(\text{chocolate}) = 3/4$.

Cut and Choose Step by Step



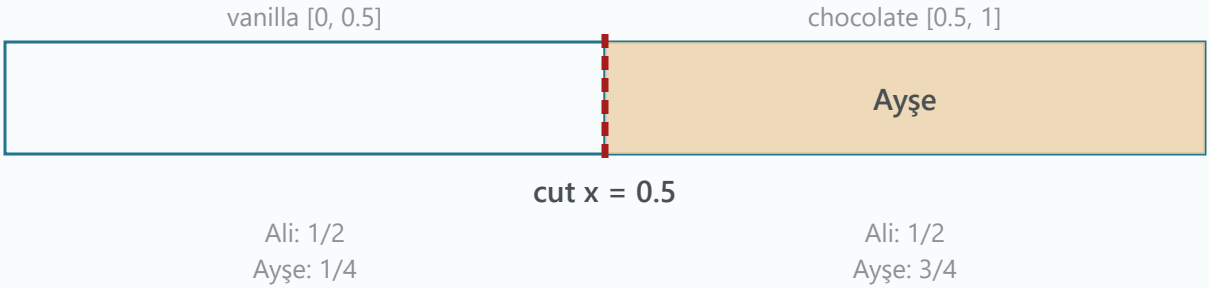
1. Ali cuts into two pieces equal in his eyes ⇒ x = 1/2, each piece worth 1/2 to Ali.

Cut and Choose Step by Step



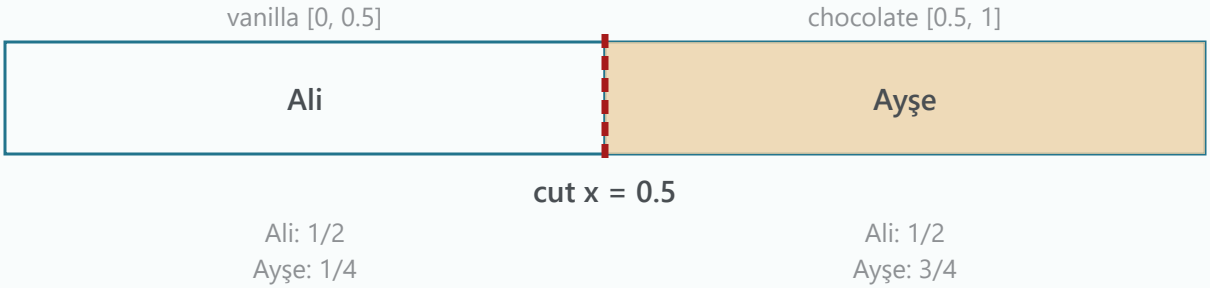
2. Ayşe evaluates: left piece 1/4, right piece 3/4.

Cut and Choose Step by Step



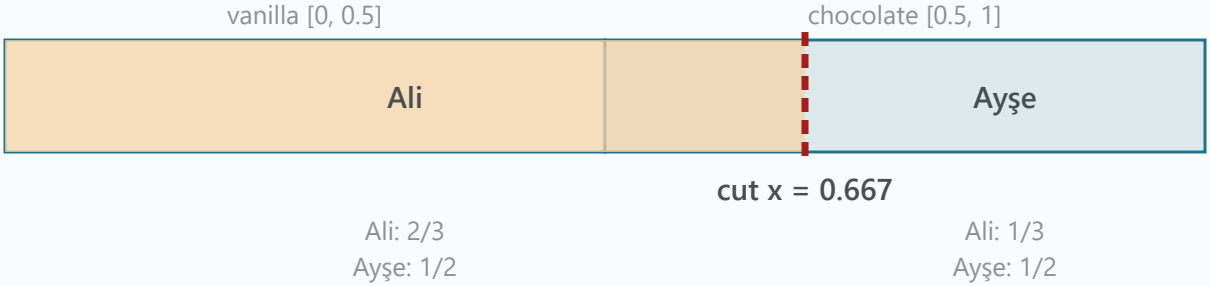
3. Ayşe chooses the more valuable right piece (chocolate).

Cut and Choose Step by Step



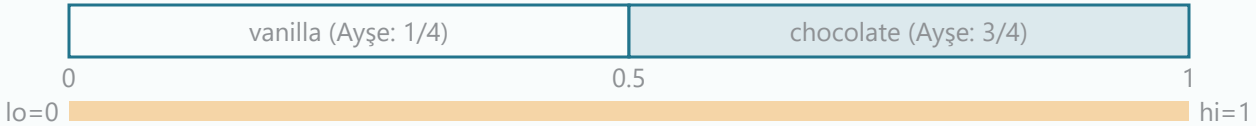
Result: Ali 1/2 ≥ 1/2, Ayşe 3/4 ≥ 1/2 ⇒ **proportional**; Ali 1/2 = 1/2, Ayşe 3/4 > 1/4 ⇒ **envy-free**.

Cut and Choose Step by Step



Swap roles: Ayşe cuts at x = 2/3 (1/2 – 1/2 in her eyes). Ali picks the left piece and gets 2/3; Ayşe gets 1/2. The chooser has the edge over the cutter.

Code Trace: A Cut Query by Bisection



A Robertson–Webb *cut query*: "where is x so that [0, x] is worth 1/2 to you?" Ayşe's valuation is increasing in x, so bisection finds it.

Step 1 / 10

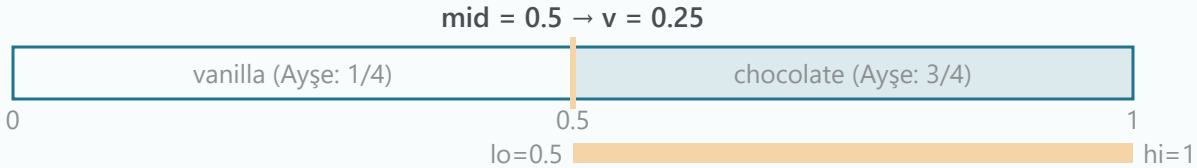
```
1  def v_ayse(x):          # value of [0, x]
2    if x <= 0.5: return x / 2
3    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
▶    lo, hi = 0.0, 1.0
6    for _ in range(steps):
7      mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
9      else: hi = mid
10   return (lo + hi) / 2
▶  print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`target = 0.5` `lo = 0.0` `hi = 1.0`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 1: mid = 0.5: all of the vanilla is worth 1/4 = 0.25 < 0.5 to Ayşe ⇒ lo = 0.5. The interval halves (width 1/2).

Step 2 / 10

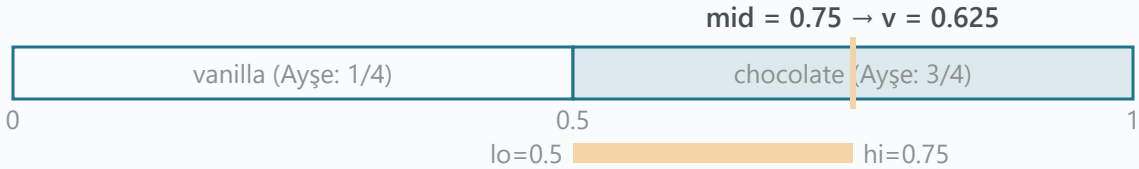
```
1  def v_ayse(x):           # value of [0, x]
▶    if x <= 0.5: return x / 2
3    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
▶      if v(mid) < target: lo = mid
9      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 0` `mid = 0.5` `v(mid) = 0.25` `lo = 0.5` `hi = 1.0`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 2: v(0.75) = 0.625 ≥ 0.5 ⇒ the cut is here or to the left: hi = mid. The interval halves (width 1/4).

Step 3 / 10

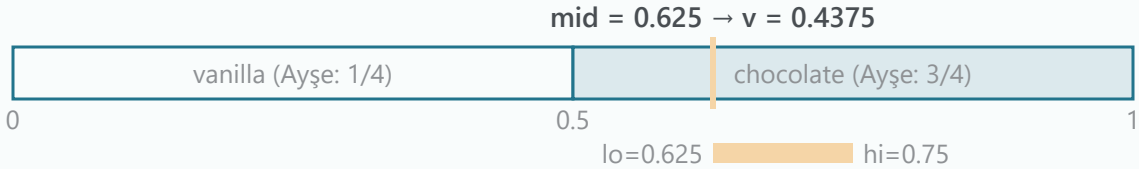
```
1  def v_ayse(x):            # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
▶      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 1` `mid = 0.75` `v(mid) = 0.625` `lo = 0.5` `hi = 0.75`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 3: v(0.625) = 0.4375 < 0.5 ⇒ the cut is further right: lo = mid. The interval halves (width 1/8).

Step 4 / 10

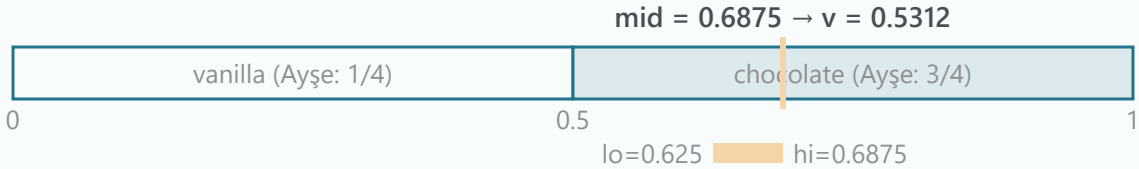
```
1  def v_ayse(x):           # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
▶      if v(mid) < target: lo = mid
9      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 2` `mid = 0.625` `v(mid) = 0.4375` `lo = 0.625` `hi = 0.75`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 4: v(0.6875) = 0.5312 ≥ 0.5 ⇒ the cut is here or to the left: hi = mid. The interval halves (width 1/16).

Step 5 / 10

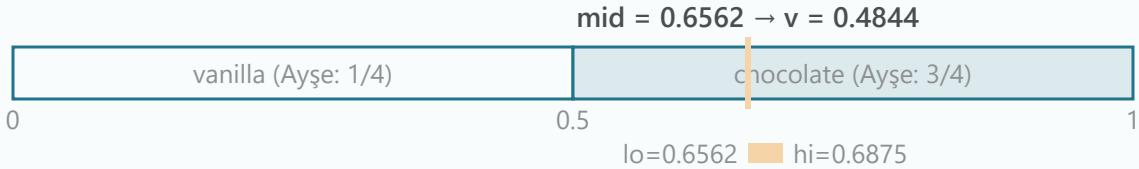
```
1  def v_ayse(x):            # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
▶      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 3` `mid = 0.6875` `v(mid) = 0.53125` `lo = 0.625` `hi = 0.6875`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 5: v(0.6562) = 0.4844 < 0.5 ⇒ the cut is further right: lo = mid. The interval halves (width 1/32).

Step 6 / 10

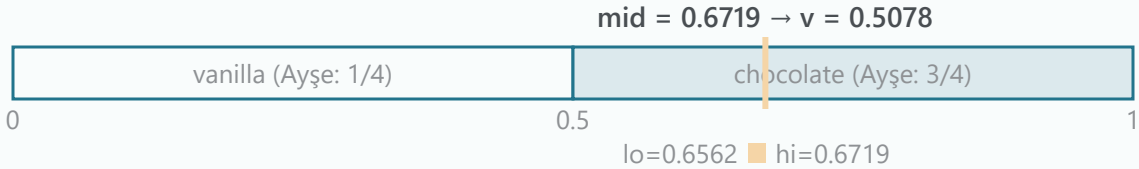
```
1  def v_ayse(x):            # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
▶      if v(mid) < target: lo = mid
9      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 4` `mid = 0.65625` `v(mid) = 0.48438` `lo = 0.65625` `hi = 0.6875`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 6: v(0.6719) = 0.5078 ≥ 0.5 ⇒ the cut is here or to the left: hi = mid. The interval halves (width 1/64).

Step 7 / 10

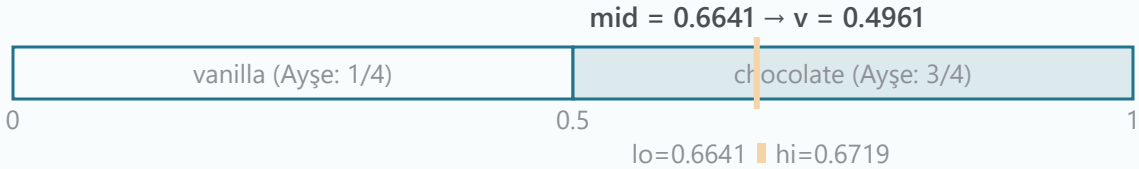
```
1  def v_ayse(x):            # value of [0, x]
     if x <= 0.5: return x / 2
     return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
     for _ in range(steps):
       mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
       else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 5` `mid = 0.671875` `v(mid) = 0.50781` `lo = 0.65625` `hi = 0.671875`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 7: v(0.6641) = 0.4961 < 0.5 ⇒ the cut is further right: lo = mid. The interval halves (width 1/128).

Step 8 / 10

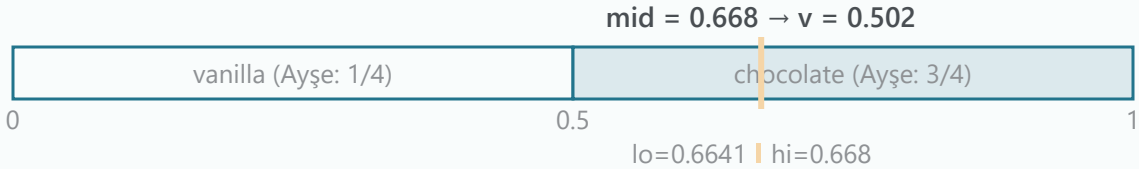
```
1  def v_ayse(x):            # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
▶      if v(mid) < target: lo = mid
9      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 6` `mid = 0.6640625` `v(mid) = 0.49609` `lo = 0.6640625` `hi = 0.671875`

OUTPUT

Code Trace: A Cut Query by Bisection



Query 8: v(0.668) = 0.502 ≥ 0.5 ⇒ the cut is here or to the left: hi = mid. The interval halves (width 1/256).

Step 9 / 10

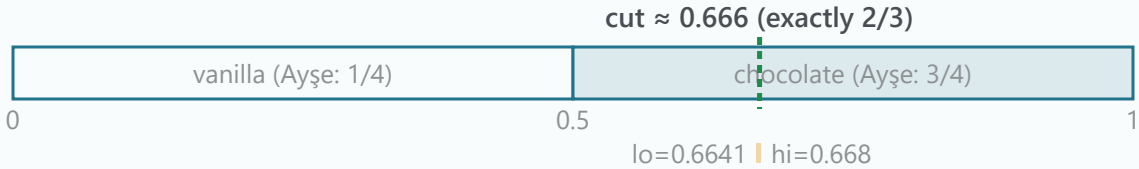
```
1  def v_ayse(x):            # value of [0, x]
▶    if x <= 0.5: return x / 2
▶    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
▶    for _ in range(steps):
▶      mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
▶      else: hi = mid
10   return (lo + hi) / 2
11 print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`_ = 7` `mid = 0.66796875` `v(mid) = 0.50195` `lo = 0.6640625` `hi = 0.66796875`

OUTPUT

Code Trace: A Cut Query by Bisection



After 8 queries the interval is 1/256 wide; its midpoint is 0.666. The true cut is 2/3: where Ayşe would cut if she were the cutter.

Step 10 / 10

```
1  def v_ayse(x):          # value of [0, x]
2    if x <= 0.5: return x / 2
3    return 0.25 + 1.5 * (x - 0.5)
4  def cut_point(v, target, steps):
5    lo, hi = 0.0, 1.0
6    for _ in range(steps):
7      mid = (lo + hi) / 2
8      if v(mid) < target: lo = mid
9      else: hi = mid
▶    return (lo + hi) / 2
▶  print(round(cut_point(v_ayse, 0.5, 8), 3))
```

VARIABLES

`lo` = **0.6640625**

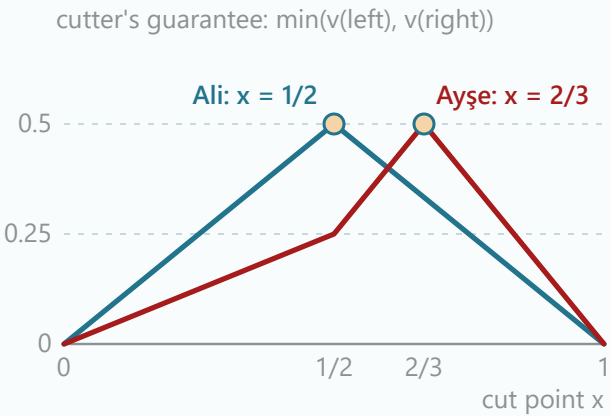
`hi` = **0.66796875**

`result` = **0.66602**

OUTPUT

```
0.666
```

Why Is Cut and Choose Fair?



- The **cutter** does not know which piece will be left, so the guarantee is the smaller piece. It peaks at the cutter's own halfway point (1/2): cutting honestly is the **maximin** strategy.
- The **chooser** takes the piece that is larger in her eyes: worth ≥ 1/2 and no less than the other, wherever the cut is.
- A cutter who knows the chooser's preferences can shift the cut and get more than 1/2, but the *guarantee* is 1/2.

With two people, proportionality ⇔ envy-freeness: if $v(X_1) \geq 1/2$, then $v(X_2) = 1 - v(X_1) \leq 1/2 \leq v(X_1)$. With three or more, envy-freeness is stronger.

Proportional but Not Envy-Free: Three People

people	piece X	piece Y	piece Z	gets	≥ 1/3?
Ali	0.40	0.30	0.30		
Ayşe	0.20	0.35	0.45		
Cem	0.30	0.30	0.40		

Three people, three pieces (X, Y, Z). The table shows each person's value for each piece; each row sums to 1.

Step 1 / 3

Last diminisher
A proportional division for n people.

Selfridge–Conway
Envy-free for 3 people, at most 5 cuts.

Aziz–Mackenzie (2016)
Envy-free for n people; finitely many but astronomically many queries.

Proportional but Not Envy-Free: Three People

people	piece X	piece Y	piece Z	gets	≥ 1/3?
Ali	**0.40**	0.30	0.30	X: 0.40	**yes**
Ayşe	0.20	**0.35**	0.45	Y: 0.35	**yes**
Cem	0.30	0.30	**0.40**	Z: 0.40	**yes**

Allocation: Ali → X, Ayşe → Y, Cem → Z. Everyone gets at least 1/3 ≈ 0.33 by their own measure ⇒ **proportional**.

Last diminisher
A proportional division for n people.

Selfridge–Conway
Envy-free for 3 people, at most 5 cuts.

Aziz–Mackenzie (2016)
Envy-free for n people; finitely many but astronomically many queries.

Proportional but Not Envy-Free: Three People

people	piece X	piece Y	piece Z	gets	≥ 1/3?
Ali	**0.40**	0.30	0.30	X: 0.40	**yes**
Ayşe	0.20	**0.35**	**0.45**	Y: 0.35	**yes**
Cem	0.30	0.30	**0.40**	Z: 0.40	**yes**

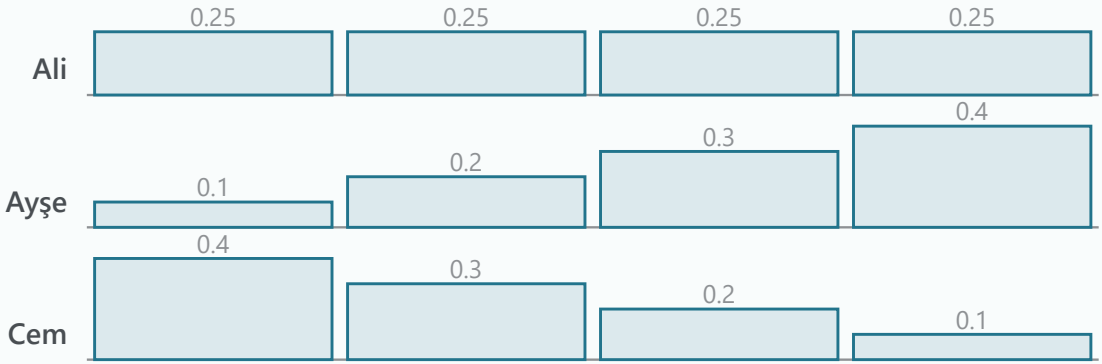
But to Ayşe, Cem's piece Z (0.45) beats her own Y (0.35) ⇒ Ayşe **envies** Cem. Proportional, but not envy-free.

Last diminisher
A proportional division for n people.

Selfridge–Conway
Envy-free for 3 people, at most 5 cuts.

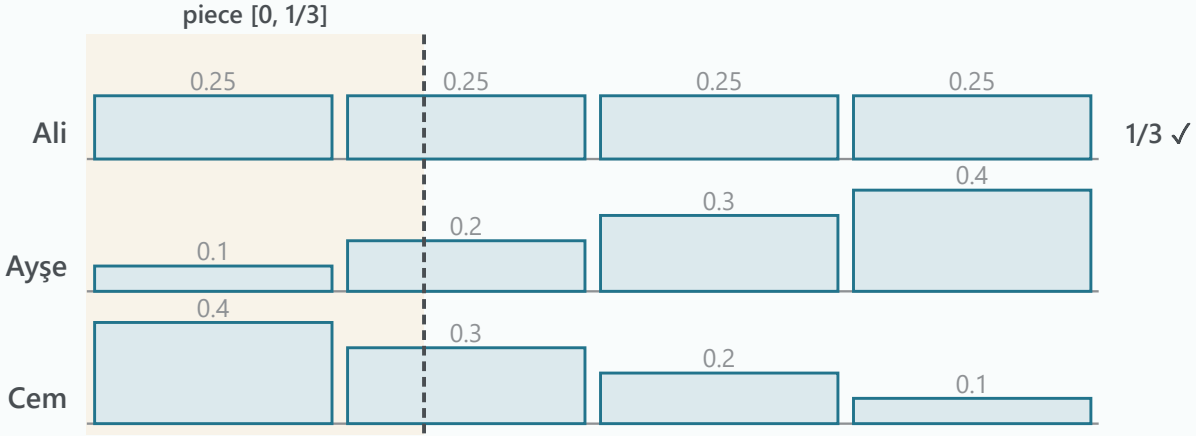
Aziz–Mackenzie (2016)
Envy-free for n people; finitely many but astronomically many queries.

Last Diminisher Step by Step



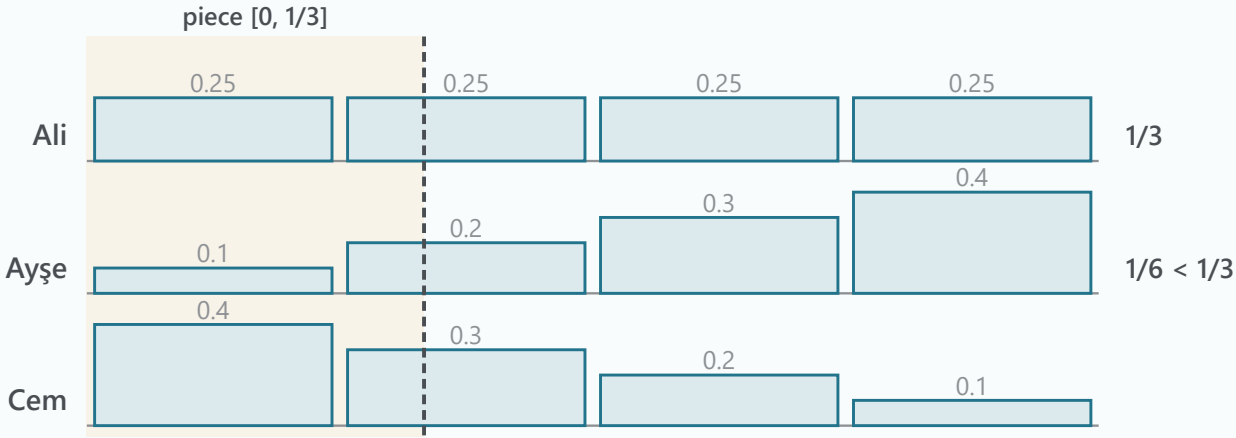
Three people; the cake has four quarters. Bars show each person's value per quarter (total 1). Ali is uniform; Ayşe likes the right, Cem the left.

Last Diminisher Step by Step



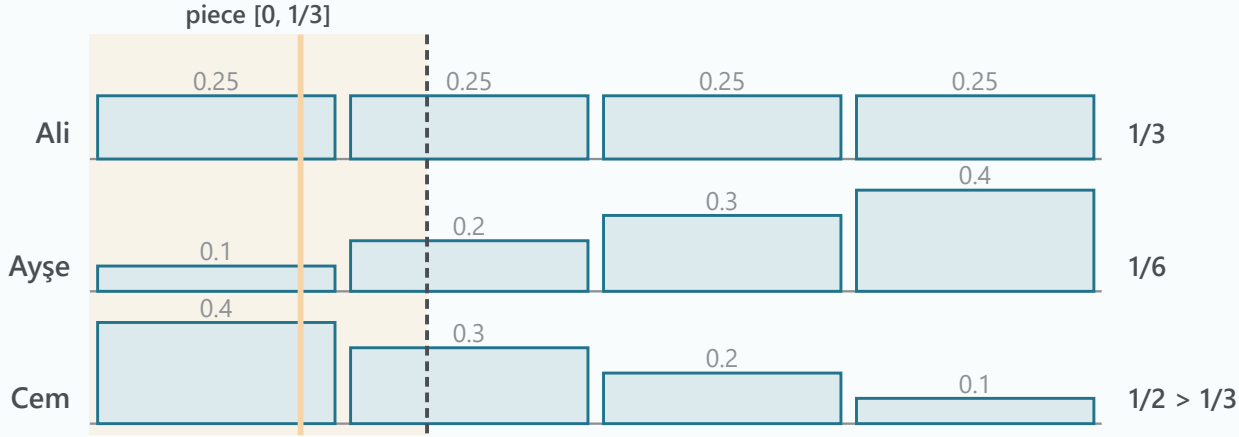
Ali cuts a piece worth 1/3 to him: [0, 1/3].

Last Diminisher Step by Step



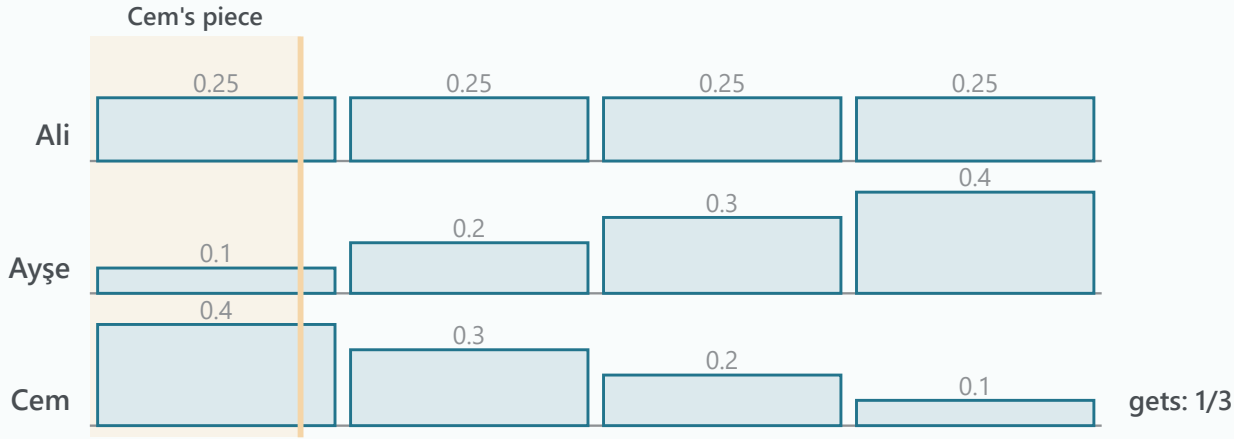
Ayşe values it at 1/6, less than 1/3, so she leaves it alone ("passes").

Last Diminisher Step by Step



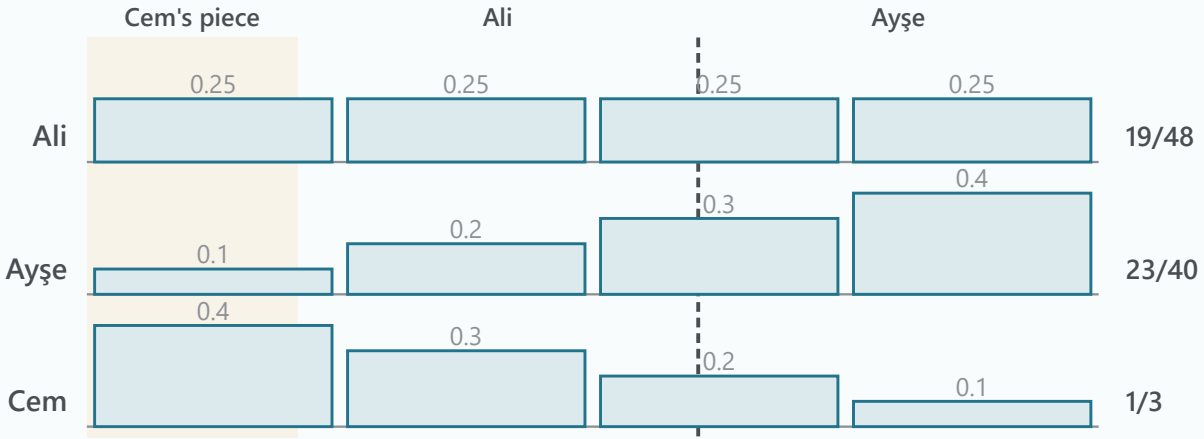
To **Cem** the piece is worth 1/2 > 1/3: he trims it, moving the cut to 5/24 (exactly 1/3 for Cem).

Last Diminisher Step by Step



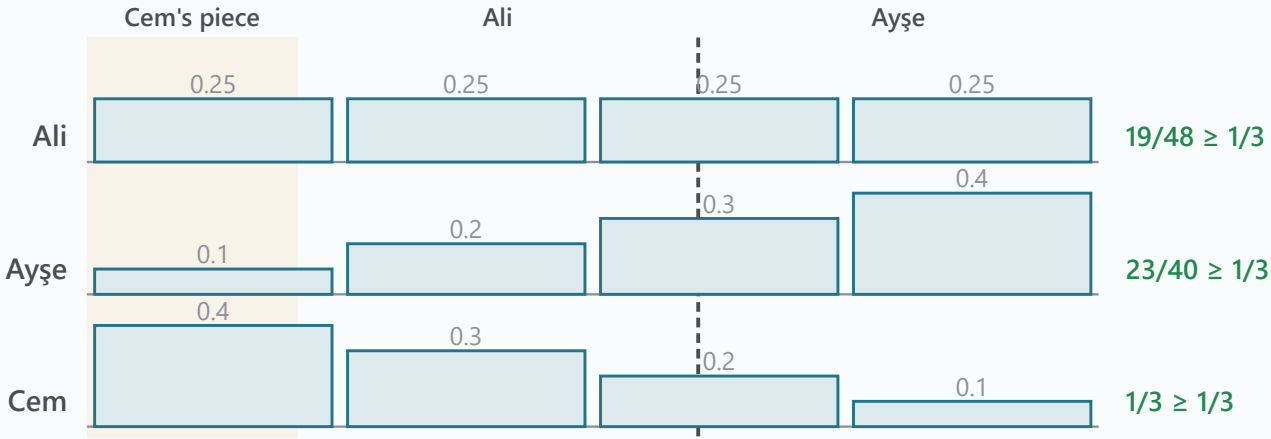
The **last trimmer** (Cem) takes the piece and leaves. For those who did not trim, the piece is worth ≤ 1/3, so the rest of the cake is worth ≥ 2/3 to them.

Last Diminisher Step by Step



The remaining [5/24, 1] is split between two by cut and choose: Ali cuts at 29/48, Ayşe picks the more valuable right piece (23/40).

Last Diminisher Step by Step



Everyone got ≥ 1/3 by their own measure ⇒ **proportional**. O(n^2) queries for n people. Not envy-free, though: Cem values Ali's piece at 9/20 (his own 1/3).

Moving Knife (Dubins–Spanier) and Even–Paz

The rule

A knife slides slowly from left to right across the cake. When the piece left of the knife is worth 1/n to someone, that person calls "stop!", takes the piece and leaves; the knife continues with the rest, and the last person takes what remains (Dubins and Spanier, 1961).

people	round 1: v([0, x]) = 1/3	round 2: v([5/24, x]) = 1/3	gets
Cem	**x = 5/24 ≈ 0.21 (stop!)**	—	[0, 5/24]: 1/3
Ali	x = 1/3 ≈ 0.33	**x = 13/24 ≈ 0.54 (stop!)**	[5/24, 13/24]: 1/3
Ayşe	x = 19/36 ≈ 0.53	x = 43/72 ≈ 0.60	[13/24, 1]: 13/20

Values as in the last diminisher example (per quarter: Ali 0.25; Ayşe 0.1–0.4; Cem 0.4–0.1). The first piece matches last diminisher: Cem stops the knife first. Everyone gets ≥ 1/3 ⇒ **proportional**. Not envy-free: Cem values Ali's piece at 2/5 > 1/3.

Even–Paz (1984): finds a proportional division with far fewer queries by halving the cake at each step.

Indivisible Goods: Round-Robin and EF1

people	m1	m2	m3	m4	own	other's
Ali	7	5	3	1	0	0
Ayşe	8	6	1	1	0	0

4 indivisible goods, 2 people; the table shows values. Round-robin: on each turn, the next person takes the most valuable remaining good.

Step 1 / 5

With indivisible goods, envy-freeness is not always possible (one car, two siblings). With additive values, round-robin in which everyone picks their favorite remaining good always yields an EF1 allocation.

Indivisible Goods: Round-Robin and EF1

people	m1	m2	m3	m4	own	other's
Ali	**7**	5	3	1	7	0
Ayşe	8	6	1	1	0	8

Ali takes m1 (7).

Step 2 / 5

With indivisible goods, envy-freeness is not always possible (one car, two siblings). With additive values, round-robin in which everyone picks their favorite remaining good always yields an EF1 allocation.

Indivisible Goods: Round-Robin and EF1

people	m1	m2	m3	m4	own	other's
Ali	**7**	5	3	1	7	5
Ayşe	8	**6**	1	1	6	8

Ayşe takes m2 (6) from what is left.

Step 3 / 5

With indivisible goods, envy-freeness is not always possible (one car, two siblings). With additive values, round-robin in which everyone picks their favorite remaining good always yields an EF1 allocation.

Indivisible Goods: Round-Robin and EF1

people	m1	m2	m3	m4	own	other's
Ali	**7**	5	**3**	1	10	5
Ayşe	8	**6**	1	1	6	9

Ali takes m3 (3).

Step 4 / 5

With indivisible goods, envy-freeness is not always possible (one car, two siblings). With additive values, round-robin in which everyone picks their favorite remaining good always yields an EF1 allocation.

Indivisible Goods: Round-Robin and EF1

people	m1	m2	m3	m4	own	other's
Ali	**7**	5	**3**	1	10	6
Ayşe	8	**6**	1	**1**	7	9

Ayşe takes m4. Ayşe: her own bundle 7, Ali's bundle 9 in her eyes ⇒ envy! But removing m1 from Ali's bundle gives 1 ≤ 7: **EF1** (envy-free up to one good).

With indivisible goods, envy-freeness is not always possible (one car, two siblings). With additive values, round-robin in which everyone picks their favorite remaining good always yields an EF1 allocation.

Back to the Game: Fair Division

Greedy picking is not a best response

The opening items (P1 / P2 value): Cake 5/1, Book 4/9, Headphones 3/8, Ball 1/1.

P1's plan	Pick order	P1	P2
always the best	Cake, Book, Phones, Ball	8	10
Book first	Book, Phones, Cake, Ball	**9**	9

P2 values the Cake at 1, so it will still be there later. In round-robin, taking your favorite item every turn is not always the best strategy.

What about fairness?

Under the second plan, P2 values P1's bundle at 9 + 1 = 10 and its own at 9: **envy**. Remove the Book and 1 ≤ 9: the allocation is still EF1. The theory's guarantee: if both sides always pick their favorite, round-robin yields EF1.

In practice: data from Harvard Business School's draft-based course allocation showed students making exactly this kind of strategic pick, at a cost in welfare (Budish and Cantillon, 2012).

Even a fair protocol is open to strategic players: the EF1 guarantee holds when everyone picks their favorite on their turn.

Which Allocation? Three Social Welfare Criteria

Utilitarian
Maximize social welfare SW = $\Sigma\, v_i$: efficiency.

Egalitarian
Maximize the smallest v_i (maximin).

Nash social welfare
Maximize $v_{Ali} \times v_{Ayşe}$; scaling one person's values by 10 does not change the answer.

Allocation	Ali	Ayşe	sum	min	product	envy-free?
Round-robin: Ali {m1, m3}, Ayşe {m2, m4}	10	7	17	7	70	no, EF1
Utilitarian: Ali {m3, m4}, Ayşe {m1, m2}	4	14	**18**	4	56	no, not even EF1
Nash / egalitarian (not unique): Ali {m2, m3, m4}, Ayşe {m1}	9	8	17	**8**	**72**	yes

Under the utilitarian allocation Ali values Ayşe's bundle at 12 (his own 4); even without m1 it is 5 > 4. Theorem (Caragiannis et al., 2016): with nonnegative additive values, a Nash-welfare-maximizing allocation is both **Pareto optimal** and **EF1** (with a tie-breaking rule if the product is always 0).

The Price of Fairness and Pareto Efficiency

Pareto efficiency

No other allocation makes someone better off without hurting anyone. 11 of the 16 allocations here are, including round-robin's (10, 7).

Price of fairness

Maximum welfare over maximum welfare among fair allocations. Near 1: fairness is cheap.

Allocation	Ali	Ayşe	sum	envy-free?
Ali {m3, m4}, Ayşe {m1, m2}	4	14	**18**	no
Ali {m3}, Ayşe {m1, m2, m4}	3	15	**18**	no
Ali {m2, m3, m4}, Ayşe {m1}	9	8	17	**yes**
Ali {m2, m3}, Ayşe {m1, m4}	8	9	17	**yes**

Only two of 16 allocations are envy-free (total 17; with the tie on m4, both are Nash and egalitarian optima). Price of fairness = 18 / 17 ≈ **1.059**: about 5.6% of welfare. Like the PoA, but the loss comes from the fairness constraint, not selfishness. For a divisible cake and n people, the price of proportionality grows like √n (Caragiannis et al., 2012).

Ethical Principles and Standards

Transparency and explainability

Decision processes should be understandable; people should be able to learn why a decision was made.

Accountability

Someone must be responsible for a wrong decision; appeal and human review must be possible.

Fairness and equality

Social impact and group-level outcomes must be taken into account.

Privacy

Collect only the data needed, and only for its stated purpose.

GDPR (EU, 2018), Art. 22	Right to contest, and to human intervention in, significant decisions based solely on automated processing
EU AI Act (2024)	Risk-based: credit scoring and hiring are "high risk"
KVKK (Turkey), OECD AI Principles	Rules for processing personal data; shared principles for trustworthy AI

Strategic Design and Incentives

- When an algorithm sets a rule, people **respond** to it: those being classified are players too (strategic classification).
- **Goodhart's law:** when a measure becomes a target, it ceases to be a good measure (click counts → clickbait).
- Giving users the right incentives and preventing malicious manipulation (fake reviews, bot accounts) is part of the design.

Gameable features

Easy-to-change features such as "number of cards" in a credit score move the score without changing the real risk.

Recommender systems

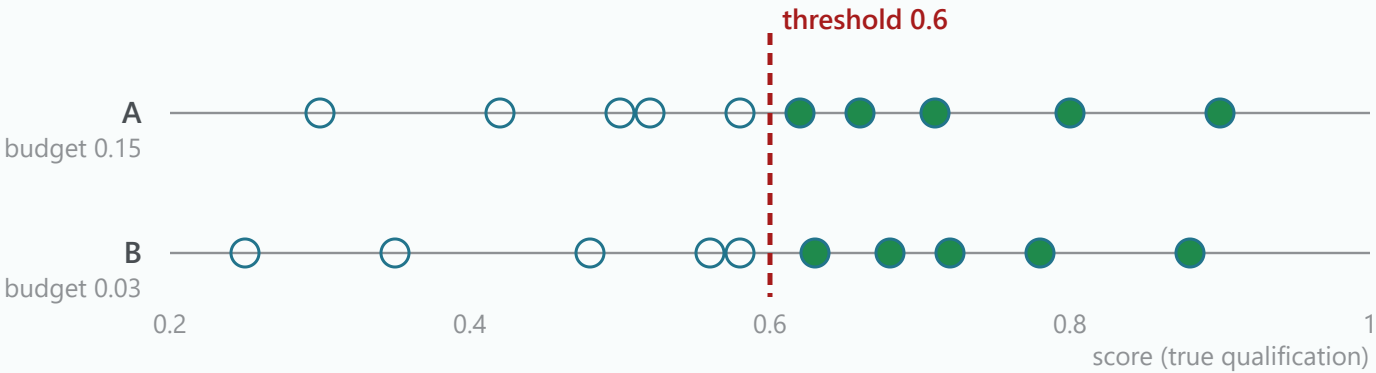
An objective that rewards the user's long-term benefit, not engagement.

Unequal costs

If gaming costs differ across groups, strategic behavior creates new unfairness.

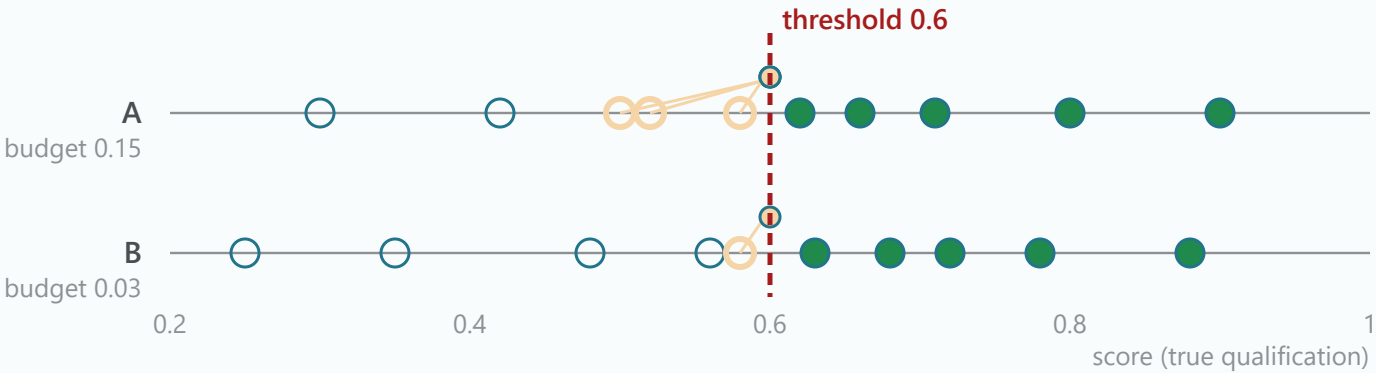
The mechanism design question applies here too: honest, rule-following behavior should be in each person's own interest.

Strategic Classification Step by Step



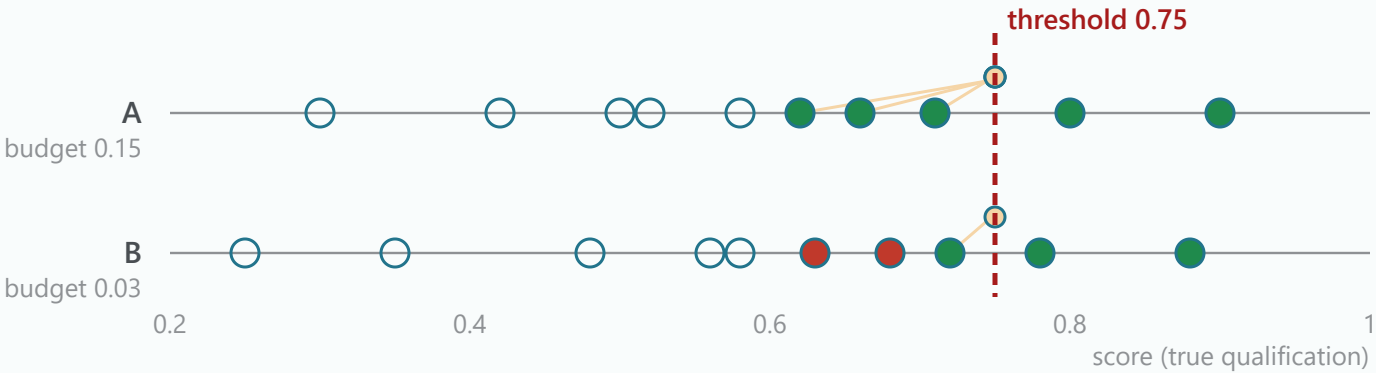
Threshold 0.6; filled circle = truly qualified (score ≥ 0.6). People can raise their score at a **cost** (e.g. number of cards). A's budget is 0.15, B's 0.03.

Strategic Classification Step by Step



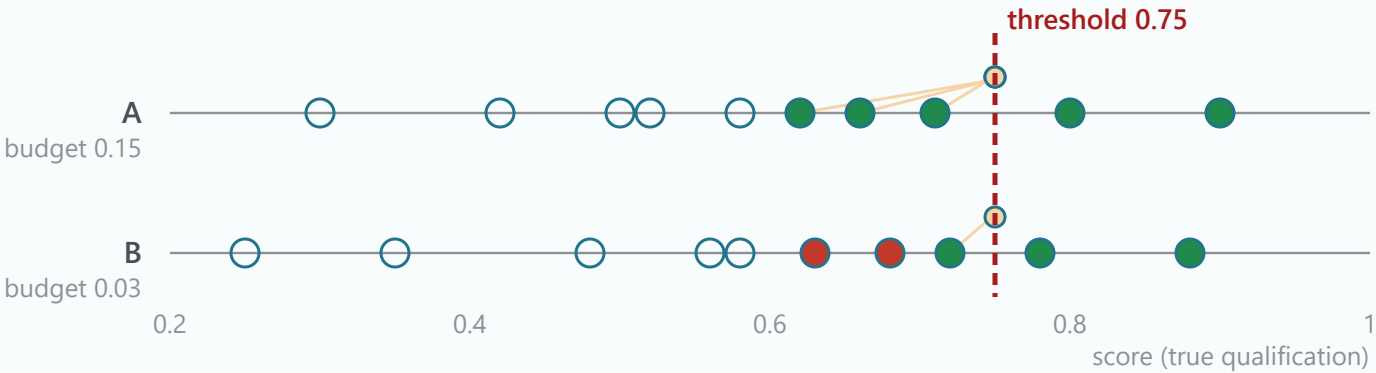
Gaming: anyone within their budget of the threshold moves up to it. 3 unqualified people from A and 1 from B get through (orange ring).

Strategic Classification Step by Step



The institution responds by raising the threshold to 0.75 (accounting for A's budget). Now no unqualified person gets through.

Strategic Classification Step by Step



But only 3 of B's 5 qualified people pass, given B's small budget (red); in A, 5 of 5. The defense against strategic behavior created a new **inequality** (Hu, Immorlica, Vaughan 2019).

ETHICS AND GOVERNANCE

Algorithmic Auditing

What is audited?

Data sources, the target variable, per-group performance and human oversight, examined together.

Documentation

Model cards and datasheets make assumptions, intended use and known limitations visible.

After deployment

Data drift, appeals and unexpected harms are monitored regularly.

Participation

Affected groups are consulted; a right of appeal and a clear chain of responsibility are defined.

Audit checklist: base rates, selection rate, TPR, FPR, PPV and calibration are reported **separately for each group**, with a written justification of which criterion was prioritized and why.

Discussion: Can Algorithms Be Neutral?

- Algorithms are not independent of the values of the institutions that choose their objectives, data, criteria and thresholds.
- Historical inequalities can seep into training data and target variables.
- Different fairness criteria give conflicting verdicts on the same decision (the impossibility theorems).
- After deployment a model can change behavior and create new feedback loops.

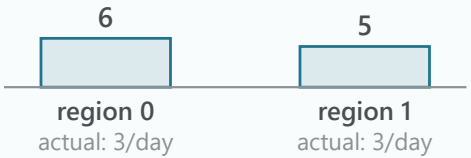
Core challenges

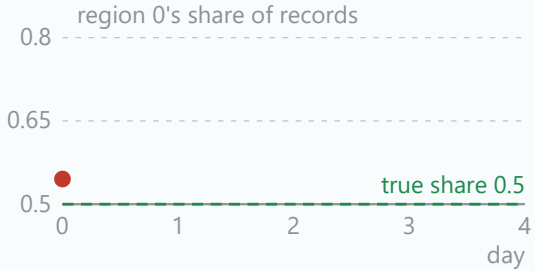
Fairness depends on context; trade-offs among accuracy, privacy and group fairness; cultural differences.

The response

Instead of claiming perfect neutrality: measure impact, document choices, provide an appeals process.

Code Trace: Self-Confirming Data





The two regions have **equal** true incident counts; past records show one more in region 0 (6 vs 5), perhaps by chance.

Step 1 / 9

```
records = [6, 5]    # past records (region 0, 1)
actual = [3, 3]     # true daily incidents: equal!
for day in range(1, 5):
  target = 0 if records[0] >= records[1] else 1
  records[target] += actual[target]  # only there
  share = records[0] / sum(records)
  print(day, target, records, round(share, 2))
```

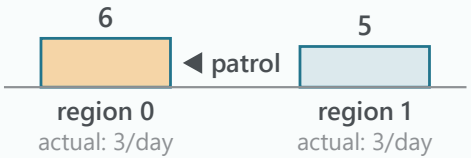
VARIABLES

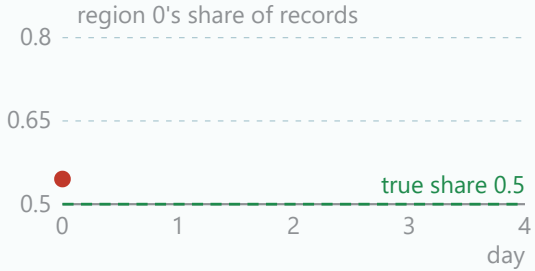
`records = [6, 5]`

`actual = [3, 3]`

OUTPUT

Code Trace: Self-Confirming Data





Day 1: the patrol goes to the region with more records ⇒ target = 0.

Step 2 / 9

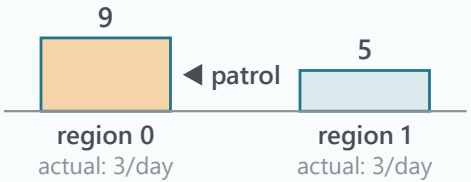
```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
▶ for day in range(1, 5):
▶   target = 0 if records[0] >= records[1] else 1
5   records[target] += actual[target]  # only there
6   share = records[0] / sum(records)
7   print(day, target, records, round(share, 2))
```

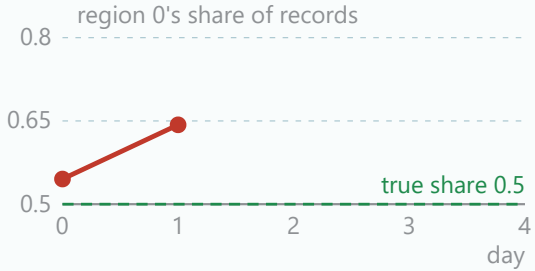
VARIABLES

`day = 1` `records = [6, 5]` `target = 0`

OUTPUT

Code Trace: Self-Confirming Data





Incidents are **recorded** only where the patrol goes: records = [9, 5]. The 3 incidents in region 1 stay invisible. Share 0.64.

Step 3 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
3 for day in range(1, 5):
4   target = 0 if records[0] >= records[1] else 1
▶   records[target] += actual[target]  # only there
▶   share = records[0] / sum(records)
▶   print(day, target, records, round(share, 2))
```

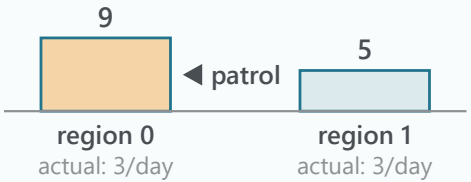
VARIABLES

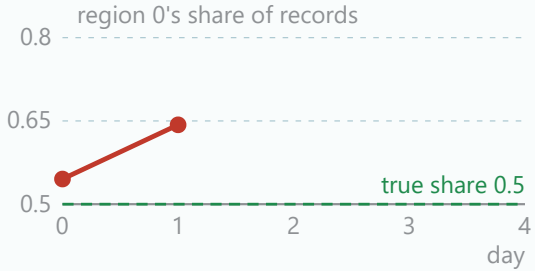
`records = [9, 5]` `share = 0.643`

OUTPUT

```
1 0 [9, 5] 0.64
```

Code Trace: Self-Confirming Data





Day 2: the patrol goes to the region with more records ⇒ target = 0.

Step 4 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
▶ for day in range(1, 5):
▶   target = 0 if records[0] >= records[1] else 1
5   records[target] += actual[target]  # only there
6   share = records[0] / sum(records)
7   print(day, target, records, round(share, 2))
```

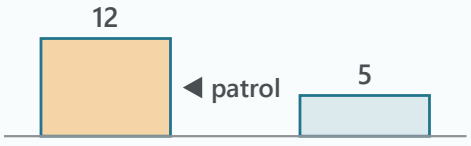
VARIABLES

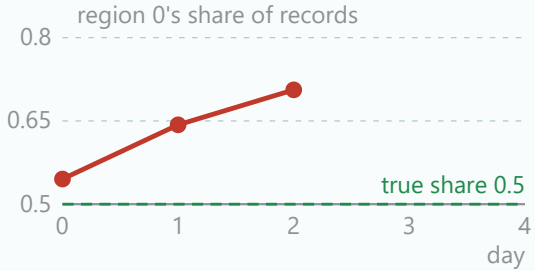
`day = 2` `records = [9, 5]` `target = 0`

OUTPUT

```
1 0 [9, 5] 0.64
```

Code Trace: Self-Confirming Data





Incidents are **recorded** only where the patrol goes: records = [12, 5]. The 3 incidents in region 1 stay invisible. Share 0.71.

Step 5 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
3 for day in range(1, 5):
4   target = 0 if records[0] >= records[1] else 1
▶   records[target] += actual[target]  # only there
▶   share = records[0] / sum(records)
▶   print(day, target, records, round(share, 2))
```

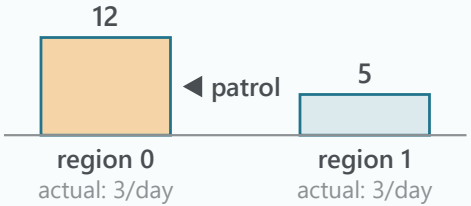
VARIABLES

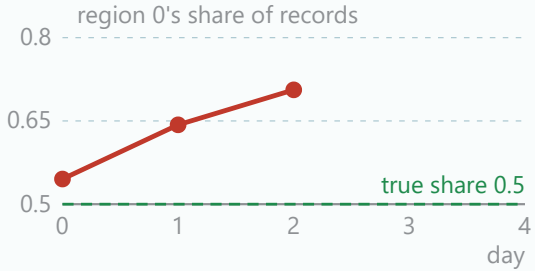
`records = [12, 5]` `share = 0.706`

OUTPUT

```
1 0 [9, 5] 0.64
2 0 [12, 5] 0.71
```

Code Trace: Self-Confirming Data





Day 3: the patrol goes to the region with more records ⇒ target = 0.

Step 6 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
▶ for day in range(1, 5):
▶   target = 0 if records[0] >= records[1] else 1
5   records[target] += actual[target]  # only there
6   share = records[0] / sum(records)
7   print(day, target, records, round(share, 2))
```

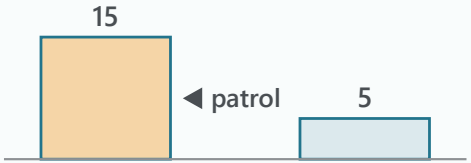
VARIABLES

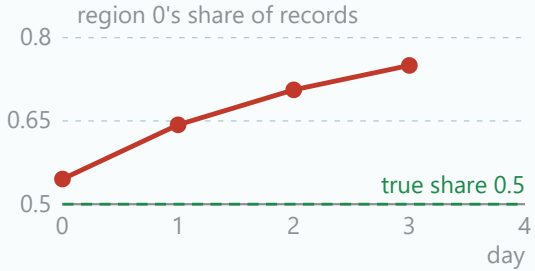
`day = 3` `records = [12, 5]` `target = 0`

OUTPUT

```
1 0 [9, 5] 0.64
2 0 [12, 5] 0.71
```

Code Trace: Self-Confirming Data





Incidents are **recorded** only where the patrol goes: records = [15, 5]. The 3 incidents in region 1 stay invisible. Share 0.75.

Step 7 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
3 for day in range(1, 5):
4   target = 0 if records[0] >= records[1] else 1
▶   records[target] += actual[target]  # only there
▶   share = records[0] / sum(records)
▶   print(day, target, records, round(share, 2))
```

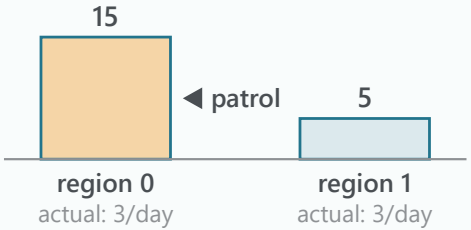
VARIABLES

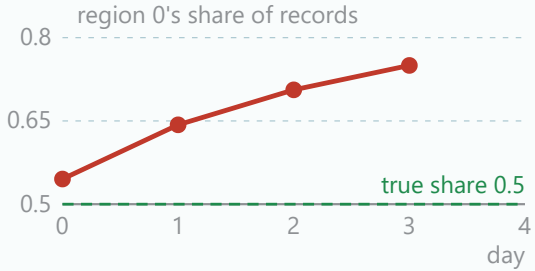
`records = [15, 5]` `share = 0.75`

OUTPUT

```
1 0 [9, 5] 0.64
2 0 [12, 5] 0.71
3 0 [15, 5] 0.75
```

Code Trace: Self-Confirming Data





Day 4: the patrol goes to the region with more records ⇒ target = 0.

Step 8 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
▶ for day in range(1, 5):
▶   target = 0 if records[0] >= records[1] else 1
5   records[target] += actual[target]  # only there
6   share = records[0] / sum(records)
7   print(day, target, records, round(share, 2))
```

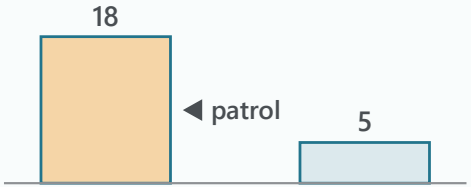
VARIABLES

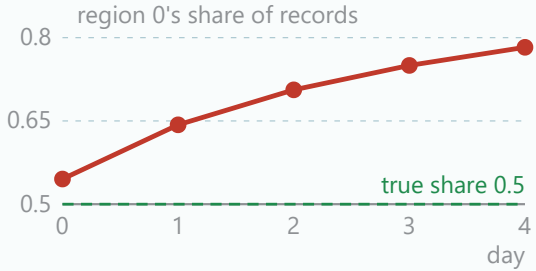
`day = 4` `records = [15, 5]` `target = 0`

OUTPUT

```
1 0 [9, 5] 0.64
2 0 [12, 5] 0.71
3 0 [15, 5] 0.75
```

Code Trace: Self-Confirming Data





Share 0.55 → 0.78: the data "confirm" the model because they were collected by the model's own decisions (Ensign et al., 2018).

Step 9 / 9

```
1 records = [6, 5]   # past records (region 0, 1)
2 actual = [3, 3]    # true daily incidents: equal!
3 for day in range(1, 5):
4   target = 0 if records[0] >= records[1] else 1
▶   records[target] += actual[target]  # only there
▶   share = records[0] / sum(records)
▶   print(day, target, records, round(share, 2))
```

VARIABLES

`records = [18, 5]` `share = 0.783`

OUTPUT

```
1 0 [9, 5] 0.64
2 0 [12, 5] 0.71
3 0 [15, 5] 0.75
4 0 [18, 5] 0.78
```

Real-World Applications and an Exercise

Credit and banking
Scoring; choosing between calibration and equal opportunity.

Hiring and HR
Résumé screening; bias in historical data, the four-fifths rule.

Healthcare
Diagnosis and treatment recommendations; proxy targets (spending ≠ need).

Exercise: Design a threshold classifier for a small dataset. (1) Build the confusion matrices of two groups. (2) Compute the four group rates (selection rate, TPR, FPR, PPV) with the `metrics` function. (3) Vary the threshold and report which criterion you can satisfy and what you lose.

Summary

Criteria

Demographic parity (selection rate), equalized odds (TPR, FPR), predictive parity (PPV), calibration.

Our example

A calibrated model with equal PPV; still, selection rate 0.5/0.2 and TPR 0.8/0.5.

Incompatibility

FPR = p/(1−p)·(1−PPV)/PPV·TPR: with different base rates the criteria cannot all hold (Chouldechova; Kleinberg et al.).

Fair division

Cut and choose: proportional and envy-free for 2 people. For 3+, proportional ≠ envy-free; EF1 for indivisible goods.

Ethics

Transparency, accountability, privacy; GDPR, the EU AI Act.

Incentives

People respond to algorithms: Goodhart's law, strategic classification.

SUMMARY

Sample Problems

1. Group C: TP = 30, FP = 20, FN = 10, TN = 40. Find the selection rate, TPR, FPR and PPV.
2. Which criteria hold between group C from Problem 1 and group A from this chapter?
3. If p = 0.2, PPV = 0.5 and TPR = 0.6, what is FPR?
4. In cut and choose, Ali cuts and knows Ayşe likes chocolate 3 times as much. If Ayşe finds the two pieces equally valuable, she picks the chocolate (right) piece. Where does Ali cut, and what does he get in his own eyes?
5. Show that with two people, proportionality implies envy-freeness.
6. In round-robin, if Ayşe went first (same values), who would get what? A person indifferent between two goods takes the lower-numbered one. Is the allocation envy-free?

▶ **Show answers**

SUMMARY

Exercises: New Topics

1. In the `group_b` trace, if A's TPR were 0.7, what q would give equal opportunity in B? What would B's accuracy be?
2. After running `cut_point(v_ayse, 0.5, 3)`, what is [lo, hi] at the end of the loop? How many queries are needed at least so that the returned midpoint is strictly within 0.001 of the true cut point?
3. In the last diminisher example, if Cem's values were like Ayşe's (0.1, 0.2, 0.3, 0.4), who would get the first piece?
4. With L = 2, d(x, y) = 0.1 and f(x) = 0.8, what range does individual fairness allow for f(y)?
5. Run the feedback code with `records = [5, 6]`. What is region 0's share after 4 days?

▶ **Show answers**

References (1/2)

- S. Barocas, M. Hardt, A. Narayanan, *Fairness and Machine Learning: Limitations and Opportunities*, MIT Press, 2023.
- A. Chouldechova, "Fair prediction with disparate impact: a study of bias in recidivism prediction instruments", *Big Data* 5(2), 2017.
- J. Kleinberg, S. Mullainathan, M. Raghavan, "Inherent trade-offs in the fair determination of risk scores", *ITCS*, 2017.
- M. Hardt, E. Price, N. Srebro, "Equality of opportunity in supervised learning", *NeurIPS*, 2016.
- J. Angwin, J. Larson, S. Mattu, L. Kirchner, "Machine Bias", *ProPublica*, 2016.
- J. Buolamwini, T. Gebru, "Gender Shades", *Proc. FAT**, 2018.
- S. J. Brams, A. D. Taylor, *Fair Division: From Cake-Cutting to Dispute Resolution*, Cambridge University Press, 1996.
- C. Dwork, M. Hardt, T. Pitassi, O. Reingold, R. Zemel, "Fairness through awareness", *ITCS*, 2012.

References (2/2)

- D. Ensign et al., "Runaway feedback loops in predictive policing", *Proc. FAT**, 2018.
- L. Hu, N. Immorlica, J. W. Vaughan, "The disparate effects of strategic manipulation", *Proc. FAT**, 2019.
- A. D. Procaccia, "Cake cutting algorithms", in *Handbook of Computational Social Choice*, Chapter 13, 2016.
- I. Caragiannis, D. Kurokawa, H. Moulin, A. D. Procaccia, N. Shah, J. Wang, "The unreasonable fairness of maximum Nash welfare", *ACM EC*, 2016.
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (fair division chapter). Course pages: **Procaccia, CMU 15-896** · **Shah, Toronto CSC304**.
- E. Budish, E. Cantillon, "The multi-unit assignment problem: theory and evidence from course allocation at Harvard", *American Economic Review* 102(5), 2012.
- H. Aziz, S. Mackenzie, "A discrete and bounded envy-free cake cutting protocol for any number of agents", *FOCS*, 2016.
- L. E. Dubins, E. H. Spanier, "How to cut a cake fairly", *American Mathematical Monthly* 68(1), 1961; S. Even, A. Paz, "A note on cake cutting", *Discrete Applied Mathematics* 7(3), 1984.
- I. Caragiannis, C. Kaklamanis, P. Kanellopoulos, M. Kyropoulou, "The efficiency of fair division", *Theory of Computing Systems* 50(4), 2012.

References: Regulation and News

- J. Dastin, "Amazon scraps secret AI recruiting tool that showed bias against women", *Reuters*, 10 October 2018.
- European Parliament and Council, General Data Protection Regulation (GDPR), Regulation (EU) 2016/679, Art. 22 (in force 2018).
- European Parliament and Council, Artificial Intelligence Act, Regulation (EU) 2024/1689.
- Law No. 6698 on the Protection of Personal Data (KVKK), Official Gazette of Turkey, 7 April 2016.
- OECD, *Recommendation of the Council on Artificial Intelligence* (OECD AI Principles), 2019.

The End

Chapter 11: Algorithmic Fairness and Ethics

Previous: Chapter 10: Multi-Agent Systems · Next: Chapter 12: Behavioral and Collective Decisions