

Which past student is closest?

Task 1 | Lecture slides 4-9 | 25 minutes

Work locally in Python. Create one script or notebook. Use the printed data below. Complete each TODO, run your code, and explain the output to the TA. No routine email or notebook submission.

Plan: Tasks 1-6: 80 minutes; break: 10 minutes; Task 7: 30 minutes. All records are fictional. Predicting withdrawal means predicting a later outcome; it does not mean a student should withdraw.

Student	Early score	Minutes	Later outcome
A (past)	70	60	Stay
B (past)	80	180	Withdraw
NEW	79	100	Unknown

1. Use only the early score

Calculate $|79 - 70|$ and $|79 - 80|$. The smaller difference identifies the closer student. Copy that student's outcome as the toy prediction. Check: the differences are 9 and 1.

Small example: For A, $\text{abs}(79 - 70)$ returns 9: the score gap is 9 points.

2. Use score and minutes together

```
from math import sqrt

def distance(x, y):
    return sqrt(sum((a-b)**2 for a, b in zip(x, y)))

A = [70, 60]
B = [80, 180]
new = [79, 100]
print("Distance to A:", distance(new, A))
# TODO: print distance to B and the predicted outcome
```

Check: raw distance selects A. Explain why the minutes differences have a large effect on the result.

Small example: NEW to A: $\sqrt{9^2 + 40^2} = 41$. Compare this with your distance to B.

3. Standardise and repeat the same rule

Fit using A and B only. For each feature, compute the mean and population standard deviation (divide by n). Score: mean 75, SD 5. Minutes: mean 120, SD 60.

```
from statistics import mean, pstdev
score_mean = mean([70, 80])
score_sd = pstdev([70, 80])
# TODO: compute minutes_mean and minutes_sd
# TODO: z = (value - mean) / sd for A, B and NEW
# TODO: use distance(...) on the two z values
```

Show the TA: A = [-1, -1], B = [1, 1]; calculate NEW yourself. The scaled rule selects B. Explain why NEW must not influence the fitted mean or SD.

Small example: NEW's score becomes $(79 - 75) / 5 = 0.8$. Now calculate its minutes value.

Apply the saved transformations

Task 2 | Lecture slides 10-16 | 12 minutes

1. Min-max scaling: keep the training reference

Use the exact lecture login counts: **[4, 12, 16]**. The training minimum is 4 and maximum is 16. Formula: $(x - \text{minimum}) / (\text{maximum} - \text{minimum})$.

```
training = [4, 12, 16]
minimum, maximum = min(training), max(training)

def minmax(x):
    return (x - minimum) / (maximum - minimum)

print(minmax(12))
# TODO: print minmax(4), minmax(16), minmax(22)
```

Check: 4 becomes 0; 12 becomes about 0.667; 16 becomes 1; future value 22 becomes 1.5. Explain why you must keep 1.5 rather than refitting the scale with 22.

Small example: For 12 logins: $(12 - 4) / (16 - 4) = 8 / 12 = 0.667$, rounded.

2. Natural logarithm: preserve the order

Use the exact lecture counts **[0, 9, 99]**. Apply $\text{log1p}(x) = \ln(1 + x)$. This is the natural logarithm.

```
from math import log1p
counts = [0, 9, 99]
# TODO: loop through counts
# TODO: print x and round(log1p(x), 3)
```

Check: results are 0, 2.303 and 4.605. Which gap is reduced most? Explain why zero is allowed and why the ordering is unchanged.

Small example: $\text{round}(\text{log1p}(9), 3)$ returns 2.303. The transformed gaps are equal; the raw gap from 9 to 99 shrinks most in absolute size.

3. Follow the lecture's decision tree

The toy tree asks: **early score < 75?** If yes, predict Stay; otherwise, predict Withdraw. Implement this with if/else for scores 70, 80 and 79.

Repeat with standardised scores, using $z < 0$. The predictions should match. Explain why this threshold rule behaves differently from a distance rule.

Small example: Score 70: $70 < 75$ is true, so predict Stay. Its $z = -1$ also satisfies $z < 0$.

What each calculation means

Transformation	Purpose in this lab
Standardisation	Compare numbers using their training spread.
Min-max	Express a value relative to training endpoints.
log1p	Compress the gaps between large non-negative counts.

Show the TA: your scaled counts, log values and three tree predictions. Scaling alone does not guarantee a better prediction.

Encode categories and build features

Task 3 | Lecture slides 17-22 | 15 minutes

1. One-hot encode the exact programme categories

Keep the column order **[CS, Economics, Psychology]**. Use one indicator per category: 1 means a match, 0 means no match. Programme names have no higher/lower order.

```
categories = ["CS", "Economics", "Psychology"]
def encode(programme):
    return [int(programme == c) for c in categories]
print(encode("CS"))
# TODO: encode Economics and Psychology
# TODO: encode Physics without changing categories
```

Check: CS becomes [1, 0, 0]. Physics becomes [0, 0, 0] under our chosen ignore-unknown policy. Explain this policy. Also state why Low, Medium, High workload has an order while programme does not.

Small example: CS matches the first column only: [1, 0, 0]. Low < Medium describes an order; CS < Economics does not.

2. Total activity from completed Weeks 1-4

Week	W1	W2	W3	W4
Logins	3	4	2	5

```
weekly_logins = [3, 4, 2, 5]
# TODO: print the total using sum(...)
```

Check: total = 14. This is a count of logins, not hours or attendance. It may be used only after W4 is complete and the values are accessible.

Small example: The first two weeks contribute $3 + 4 = 7$ logins. Add W3 and W4 to finish.

3. Calculate minutes per login safely

Student	Total minutes	Logins
R	200	10
S	100	5

Calculate the ratio for R and S. Both equal 20. Keep minutes and login totals too: the same ratio does not imply the same amount of activity.

```
def minutes_per_login(minutes, logins):
    if logins == 0:
        return None
    return minutes / logins
print(minutes_per_login(200, 10)) # R: 20.0
# TODO: run for S and (40 minutes, 0 logins)
# Example indicator: int(0 == 0) returns 1
# TODO: print the zero-login indicator for R and S
```

4. Derive elapsed days from verified dates

Snapshot = 5 September 2026. A first registered on 1 September; B on 4 September. Use `datetime.date` and subtract each registration date from the snapshot. Check: 4 days and 1 day. A record-update date cannot replace the first-registration date.

Small example: After from `datetime` import `date`, run `(date(2026, 9, 5) - date(2026, 9, 1)).days`. It returns 4. Repeat for B.

Fit once, then transform a new student

Task 4 | Lecture slides 29-32 | 15 minutes

Use this exact training table. Fit means learning and saving a rule from training rows.

Transform means applying that saved rule to a row.

Training row	Score	Logins	Programme
1	60	2	CS
2	60	2	Economics
3	80	6	Psychology
4	80	6	CS

1. Learn and save the training rules

```
from statistics import mean, median, pstdev
scores = [60, 60, 80, 80]
logins = [2, 2, 6, 6]
login_median = median(logins)
# TODO: compute score mean and population SD
# TODO: compute login mean and population SD
# Keep category order: CS, Economics, Psychology
```

Check: login median = 4; score mean = 70, SD = 10; login mean = 4, SD = 2. All rules come from the four training rows.

Small example: Sorted logins are [2, 2, 6, 6]. The two middle values give median = $(2 + 6) / 2 = 4$.

2. Transform the exact future student

Score	Logins	Programme
78	Missing (None)	CS

Step A: replace the missing login count using the saved training median.

Step B: standardise the score and completed login count using the saved means and SDs.

Step C: one-hot encode CS using the saved category order.

```
new_score, new_logins, new_programme = 78, None, "CS"
if new_logins is None:
    new_logins = login_median
# TODO: compute score_z and logins_z
# TODO: final = [score_z, logins_z] + encode(new_programme)
# TODO: print final
```

Check: the final vector is **[0.8, 0, 1, 0, 0]**. State what each of its five positions means.

Small example: Missing logins become 4, then $(4 - 4) / 2 = 0$. The five positions are score_z, logins_z, CS, Economics, Psychology.

3. Explain the order

Why fill a missing numerical value before scaling it? Why must the new student not create a new median, mean, SD or set of programme columns?

Show the TA: the saved training values and final vector. This table has no outcome labels, so the lecture example prepares inputs; it does not yet produce a withdrawal prediction.

Keep unavailable information out

Task 5 | Lecture slides 24-28, 33-35 | 8 minutes

Prediction time: end of Week 4. For each row, write USE or DO NOT USE and one short reason. Decide based on when the information became accessible.

Candidate input	When available / what it contains
W1-W4 logins	Already visible at the end of W4
Programme at enrolment	Historical value already recorded
Exam date announced in W1	The schedule is known in W1
W3 quiz score	First released in W5
Current GPA updated in W6	Only the later version is available
Whole-semester attendance	Includes classes after W4

Small example: W1-W4 logins: USE, because they are already visible at the cutoff. Decide the other rows yourself.

Audit three mistakes from the lecture

1. Training values are [2, 4]; a held-out test value is [100]. Compute the training mean and the combined mean. Check: 3 versus about 35.33. Which mean may be saved?

Small example: The safe mean is $(2 + 4) / 2 = 3$. Applying it later does not add the test value to training.

2. A W4 prediction uses W5 logins. Name the unavailable information and remove it.

3. The input includes "will withdraw later". Keep that value as the historical training label, not as an input feature.

Write the safe order: define the question; separate training and test; fit preprocessing on training; transform both with the same rules; fit the prediction rule on training; check predictions on test.

Task 6 | Check the exact lecture predictions

Lecture slides 36-40 | 5 minutes

Test student	True label	Predicted label
T1	Withdraw	Withdraw
T2	Stay	Withdraw
T3	Withdraw	Withdraw

```
actual = ["Withdraw", "Stay", "Withdraw"]
predicted = ["Withdraw", "Withdraw", "Withdraw"]
# TODO: count mismatches using a loop
# TODO: error_rate = mistakes / len(actual)
# TODO: accuracy = 1 - error_rate
```

Check: one mistake; error = 1/3; accuracy = 2/3. Explain why a perfect training result cannot replace this held-out check. A more complex rule need not work better on new students.

Small example: T1: true = predicted, so add 0 mistakes. T2: Stay differs from Withdraw, so add 1.

Show the TA: your availability decisions, both means, and the error calculation. Take a 10-minute break before the combined exercise.

One slightly harder student-table exercise

Task 7 | New practice using Week 4 skills | 30 minutes

Goal: predict later withdrawal using information accessible at the end of W4. This is a new fictional practice table, not an additional lecture example. All listed scores and W1-W4 activity were accessible at the cutoff.

Minutes = total platform minutes across W1-W4. W1-W4 columns = login counts. Later outcome is known for historical students only. Missing minutes are recorded as None.

ID / set	Score	Minutes	W1	W2	W3	W4	Programme	Later
S1 / train	60	60	1	1	0	0	CS	Withdraw
S2 / train	65	90	1	1	1	1	Economics	Withdraw
S3 / train	80	180	3	3	3	3	Psychology	Stay
S4 / train	85	None	4	4	4	4	CS	Stay
T1 / test	62	70	1	0	1	0	CS	Withdraw
T2 / test	82	170	3	4	3	4	Economics	Stay
N1 / new	78	None	2	3	2	3	Physics	Unknown

Start with this table in Python

```
rows = [
    ["S1", "train", 60, 60, 1, 1, 0, 0, "CS", "Withdraw"],
    ["S2", "train", 65, 90, 1, 1, 1, 1, "Economics", "Withdraw"],
    ["S3", "train", 80, 180, 3, 3, 3, 3, "Psychology", "Stay"],
    ["S4", "train", 85, None, 4, 4, 4, 4, "CS", "Stay"],
    ["T1", "test", 62, 70, 1, 0, 1, 0, "CS", "Withdraw"],
    ["T2", "test", 82, 170, 3, 4, 3, 4, "Economics", "Stay"],
    ["N1", "new", 78, None, 2, 3, 2, 3, "Physics", None]
]
columns = ["id", "set", "score", "minutes", "w1", "w2",
           "w3", "w4", "programme", "outcome"]
import pandas as pd
df = pd.DataFrame(rows, columns=columns)
print(df)
```

If needed, install Pandas in your local terminal: `python -m pip install pandas`. Do not include ID, set or outcome in a distance vector.

Fixed split: S1-S4 train; T1-T2 test; N1 is a future student. Keep this split throughout. Begin below, then continue on the next page.

1. Derive early activity features

Add `early_logins = w1 + w2 + w3 + w4`. **Small example:** $S1 = 1 + 1 + 0 + 0 = 2$. Repeat for every row.

```
df["early_logins"] = df[["w1", "w2", "w3", "w4"]].sum(axis=1)
train = df[df["set"] == "train"].copy()
# TODO: create separate copies named test and future
```

2. Fill missing minutes using training only

Training minutes [60, 90, 180] have median 90. **Small example:** fill S4's missing minutes with 90. Save this median for all sets. In Pandas, None becomes NaN; use `.fillna(90)`, not `is None` check.

Build, predict and explain

Task 7 continued | Continue the student-table exercise

3. Fit and reuse the scale

Use numerical features **[score, minutes, early_logins]**, in that order. After filling minutes, compute each training mean and population SD. Pandas uses sample SD by default; use **.std(ddof=0)** here.

```
saved_median = train["minutes"].median()
train["minutes"] = train["minutes"].fillna(saved_median)
# TODO: fill test and future minutes with saved_median
numeric = ["score", "minutes", "early_logins"]
mu = train[numeric].mean()
sd = train[numeric].std(ddof=0)
train_z = (train[numeric] - mu) / sd
# TODO: transform test and future using SAME mu and sd
```

Check: means = [72.5, 105, 8.5]. No test or future row may influence the median, means or SDs.

Small example: Training score SD is about 10.308. S1 score_z = (60 - 72.5) / 10.308, about -1.213. Use the unrounded SD in code.

4. Encode and calculate a safe ratio

Use Task 3's encode function. Keep [CS, Economics, Psychology]; Physics remains [0, 0, 0]. Also calculate minutes per login and a zero-login indicator. Mark ratios using filled minutes as based on replacement values. Keep ratios for explanation; the prediction vector uses only the three scaled numbers and programme indicators.

Small example: S1 is CS, so its indicators are [1, 0, 0]. Its ratio = 60 / 2 = 30 minutes/login; zero-login indicator = 0.

5. Copy the closest training outcome

For each row, create **[score_z, minutes_z, early_logins_z, CS, Economics, Psychology]**. Use Task 1's distance function to compare T1, T2 and N1 with S1-S4. Print all four distances, the closest training ID and its copied label. If there is a tie, choose the smaller training ID.

Small example: T1 to S1 is about 0.295. Calculate the other three distances before choosing a label.

6. Check the held-out predictions

Use T1 and T2 outcomes only now, to count mistakes and calculate error. Report N1's prediction separately: its true later outcome is unknown. Two test cases are too few to establish reliable performance; do not change the rule after inspecting them.

Small example: If one of two predictions is wrong, error = 1 / 2 = 0.5. Calculate your own result.

7. Show the TA and explain in plain words

Show the completed table, saved median/means/SDs, vectors, distances, two test predictions and N1 prediction. Explain one scaling choice, one missing-value choice, the unknown programme, and why later outcomes stayed outside the inputs. A support conversation requires human judgement beyond this toy prediction.

Small example: Explain: "I filled S4's minutes with 90, the training median, and used that same saved rule for N1."