

Exercise Sheet: From Data Analysis to Communication

Data, Data Storage, Data Collection

Exercise 1 – Twelve Builds

A continuous integration (CI) service compiles and tests a project's code at every push. The durations of its last twelve builds, in minutes, are:

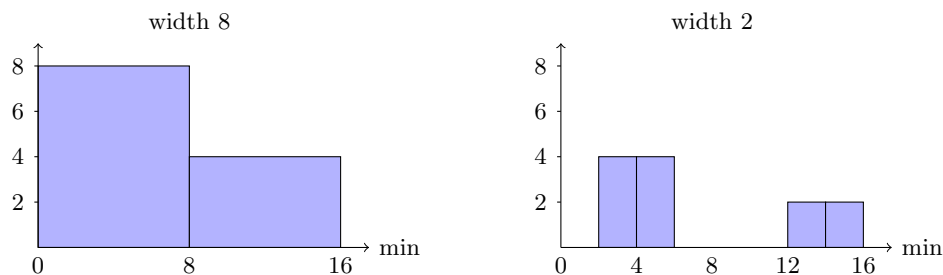
2, 3, 3, 3, 4, 4, 4, 5, 13, 13, 15, 15.

- (Q1) Compute the mean and the median of the durations. The team's dashboard displays “a typical build takes 7 minutes”. Comment.

The mean is $84/12 = 7$. The median is the average of the 6th and 7th values, both equal to 4, so the median is 4.

No build took 7 minutes: the mean falls in the gap between two groups of builds (2 to 5 minutes, and 13 to 15 minutes). The median describes the eight fast builds but says nothing about the four slow ones, a third of all builds. No single number describes a “typical” build here, because there are two kinds of builds.

- (Q2) Below are two histograms of the same durations, with bins of width 8 and of width 2, both starting at 0. For each histogram, what would you conclude about the shape of the distribution, and which summary would you then report?

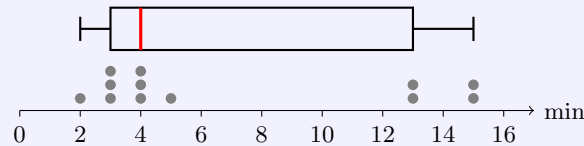


- **Width 8:** two bars, 8 and 4. The distribution looks right-skewed, and the usual advice would be to report the median, 4 minutes.
- **Width 2:** two separate groups, with empty bins from 6 to 12 minutes. There are two peaks, possibly two subpopulations: split the data before summarizing.

The important point is that the bin width is a choice of the analyst, and here it changes the diagnosis. Bins that are too wide merge separate groups; bins that are too narrow show noise. Try several widths before concluding.

- (Q3) Sketch the box plot of the durations. Compute the quartiles with the linear interpolation approach: Q_1 is the value at position $1 + 0.25 \times (n - 1)$ in the sorted list, and Q_3 the value at position $1 + 0.75 \times (n - 1)$, interpolating between two neighbours if needed. Does the box plot show what you found in the previous question?

No: the box plot hides the gap. With $n = 12$, Q_1 is at position 3.75, between the 3rd and 4th values, both equal to 3: $Q_1 = 3$. Q_3 is at position 9.25, between the 9th and 10th values, both equal to 13: $Q_3 = 13$. The median is 4 and the IQR is 10. The fences are $3 - 15 = -12$ and $13 + 15 = 28$, so there are no outliers, and the whiskers reach the minimum 2 and the maximum 15.



(The raw values are drawn as grey dots below the box plot for comparison: they are not part of the box plot.)

The box stretches from 3 to 13, which suggests durations spread over this range, while no build lies between 5 and 13. The median near the left end of the box suggests a skewed distribution, the same misleading diagnosis as the histogram with wide bins. The important point is that a box plot shows five numbers: it cannot show a gap or two peaks. Look at a histogram, or at the values, before trusting a box plot.

- (Q4) Each build record also has a field **cache**: **hit** if the build reused files compiled by an earlier build, **miss** otherwise. The four slowest builds are exactly the misses. Which summaries and which chart would you now report?

Split by **cache** and summarize each group: the hits (2 to 5 minutes) have median 3.5, the misses (13 to 15 minutes) have median 14, and the share of misses is $4/12 = 1/3$. The chart compares a quantitative feature across the groups of a qualitative one: box plots of the duration per **cache** value, or a histogram with one colour per value. The single number 7 was a mixture of two very different builds.

Exercise 2 – A Canary Release

An online shop deploys a new release (v2) of its checkout service as a *canary*: for one week, the load balancer sends 10% of the requests to v2 and the others to the current release v1. Every request is logged as a JSON record, for instance:

```
{ "time": "2026-10-03T14:02:11", "endpoint": "/pay", "release": "v2",
  "status": 200, "latency_ms": 64, "rating": 4 }
```

The field **status** is the HTTP status code (200: success, 404: not found, 500: server error). After paying, some users rate the speed of the checkout from 1 (very slow) to 5 (very fast). During the week, v1 served 450 000 requests and v2 served 50 000.

- (Q1) A monitoring tool computes the mean of every numeric field of the records. Which of these means are meaningless or misleading, and what should the tool report instead?

The meaning of a feature, not its data type, decides the summary.

- **status** is stored as an integer, but the codes are labels: 404 is not “twice” 200, and a mean status of 227 means nothing. A rise of this mean could come from more 404 (clients asking for missing pages) or from more 500 (server failures), and only the second is the release’s fault. Report the proportion of each code.
- **rating** is ordinal: the order is meaningful, the gaps are not (is “2 to 3” the same step as “4 to 5”?). A mean assumes equal gaps and can hide a change of distribution (e.g., more 1s *and* more 5s). Report the proportion of each level, or

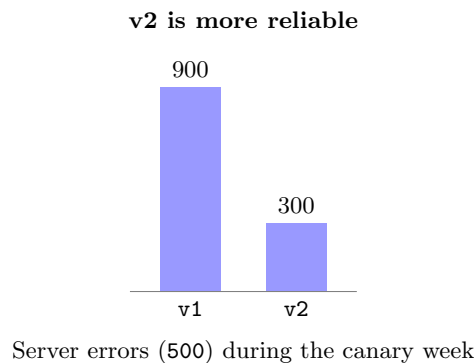
the median.

The mean of `latency_ms` is meaningful, but mixes typical requests and timeouts.

(Q2) To compare the server errors of the two releases, a teammate processes the logs in two steps:

1. keep the records whose `status` is 500;
2. group these records by `release`, and count the records of each group.

They post the chart of the result in the team's channel. Is the caption justified? At which step is the information needed for the comparison lost? Change the pipeline, and describe the chart you would post instead.



No. The chart compares counts, but v2 served nine times fewer requests than v1. The comparison needs a rate:

$$v1 : \frac{900}{450\,000} = 0.2\%, \quad v2 : \frac{300}{50\,000} = 0.6\%.$$

v2 returns a server error three times as often as v1: the caption says the opposite of the data.

The information is lost at step 1: it drops the successful requests *before* the grouping, so each group only contains errors, and the denominator is gone. Group first, then compute two counts in each group:

1. group the records by `release`;
2. in each group, compute the proportion of records whose `status` is 500.

(Q3) The dashboard reports a mean latency of 224 ms for v1 and 296 ms for v2, and someone concludes that “v2 is about 30% slower”. Assume that, in both releases, requests take between 150 and 250 ms, about 200 ms on average, except those that hit a timeout, which are cut off at 5000 ms. Let t be the share of requests that hit a timeout. Express the mean latency as a function of t , then estimate t for each release.

A share $1 - t$ of the requests take about 200 ms and a share t take 5000 ms, so the mean latency is

$$200(1 - t) + 5000t = 200 + 4800t,$$

Isolating for t , we obtain

$$t = \frac{\text{mean} - 200}{4800}.$$

So

$$\begin{aligned} \text{v1 : } t &= \frac{224 - 200}{4800} = \frac{24}{4800} = \frac{1}{200} = 0.5\%, \\ \text{v2 : } t &= \frac{296 - 200}{4800} = \frac{96}{4800} = \frac{1}{50} = 2\%. \end{aligned}$$

“About 30% slower” is not what happened: typical requests did not change, but time-outs were multiplied by four, and these few requests pull the mean up. One mean mixes two phenomena that have different causes and different fixes.

- (Q4) Engineers who run web services report latency by percentiles rather than by the mean: the p50 (the median) and the p99, the latency below which 99% of the requests fall. Give the p50 and the p99 of each release. Which one shows the change?

The p50 is about 200 ms in both releases: the median ignores the timeouts. In v1, only 0.5% of requests time out, so 99% of them take at most 250 ms: the p99 is at most 250 ms. In v2, 2% of requests time out, more than the 1% above the p99, so the p99 is 5000 ms. The p99 shows the change, and says what it is: the slowest requests, not the typical ones. This is why service-level objectives are usually written on a high percentile, e.g. “99% of requests in less than 300 ms”.

- (Q5) Write the sentence, and choose the single chart, that you would show to the product director, who has two minutes in a meeting to decide whether to roll v2 out to every user.

For the director, one message without statistical vocabulary, tied to the decision: “v2 is as fast as v1 for a typical request, but four times as many requests time out: 1 in 50 instead of 1 in 200. We should fix this before rolling it out.” The single chart: two bars with the share of timed-out requests per release, labelled with their values. The server errors found earlier could be added to the same sentence. The important point is that the distributions and percentiles are for the engineers, who need to find the cause; the director needs the fact that drives the decision.

Exercise 3 – Does a Second Reviewer Prevent Bugs?

A software company with five teams wants to know whether reviewing code with two reviewers, rather than one, reduces bugs. A pull request (PR) is called *buggy* if a bug ticket is linked to it within 30 days of its merge. Two reviewers are mandatory in the platform team, which maintains the payment code and has the most automated tests. In the mobile team, the author of a PR chooses one or two reviewers. The three other teams always use one reviewer. In March 2025, the platform team moved to a new bug tracker, and the links between its tickets and the 100 PRs it merged that month were lost.

The data is stored in two tables: `prs`, with one row per merged PR, and `tickets`, with one row per bug ticket, linked to its PR by `pr_id`.

```
prs(id, team, reviewers, merged_at)
tickets(id, pr_id, created_at)
```

A data analyst writes the following findings.

- (1) We analysed the 2000 PRs merged in 2025: 400 had two reviewers (300 in the platform team, 100 in the mobile team) and 1600 had one.
- (2) We flagged each PR as buggy or not with the query below.

```

1 SELECT p.id, p.reviewers, COUNT(t.id) > 0 AS buggy
2 FROM prs p
3 LEFT JOIN tickets t
4     ON t.pr_id = p.id
5     AND t.created_at <= p.merged_at + INTERVAL '30 days'
6 GROUP BY p.id, p.reviewers;

```

- (3) 10% of the two-reviewer PRs were buggy, against 20% of the one-reviewer PRs.
- (4) The mobile team confirms this: 10% against 20%, with 100 PRs of each kind.
- (5) A second reviewer therefore halves the risk of a bug.
- (6) We recommend requiring two reviewers on every PR.

(Q1) Label each of the findings (3) to (6) as an observation or an interpretation, in one line each.

- (3) Observation: it describes the data.
- (4) The numbers are an observation; the word “confirms” is an interpretation.
- (5) Interpretation: a causal claim (“halves the risk”) drawn from observational data, since nobody assigned the number of reviewers at random.
- (6) Neither: a recommendation. It goes beyond description, relies on (5) being true, and ignores the cost (reviewer time).

(Q2) Using the rough check of the lecture (a margin of ± 2 SE, with $SE(\hat{p}) = \sqrt{\hat{p}(1 - \hat{p})/n}$), decide whether the gap in (3) is bigger than noise. Same question for (4).

For (3):

$$SE_2 = \sqrt{\frac{0.1 \times 0.9}{400}} = \frac{0.3}{20} = 0.015, \quad SE_1 = \sqrt{\frac{0.2 \times 0.8}{1600}} = \frac{0.4}{40} = 0.01.$$

The margins are ± 3 and ± 2 points: [7%, 13%] and [18%, 22%] do not overlap, so the gap is beyond noise.

For (4):

$$SE_2 = \sqrt{\frac{0.1 \times 0.9}{100}} = 0.03, \quad SE_1 = \sqrt{\frac{0.2 \times 0.8}{100}} = 0.04.$$

The margins are ± 6 and ± 8 points: [4%, 16%] and [12%, 28%] overlap, so the gap is within noise. The same rates do not carry the same evidence: (4) does not “confirm” anything. Note that this is not a statistical test, only a rough check.

(Q3) Using finding (1) and the description of the teams, explain why (3) does not support (5). Which finding avoids this problem, and what does it allow you to conclude about (5)?

Because of confounding by team, and the within-team comparison (4) is too small to conclude. By (1), 300 of the 400 two-reviewer PRs, three quarters, come from the platform team, while none of the one-reviewer PRs do. The team influences both the number of reviewers (mandatory two in the platform team) and the bug rate (most tests, a different kind of code). The tests alone could explain the low bug rate of two-reviewer PRs, with or without a second reviewer.

Only (4) compares PRs within one team, where the team effect is the same for both groups. But this comparison is within noise. Besides, in the mobile team, authors choose the number of reviewers, so they may ask for two on riskier PRs: another confounder, which could even hide a real effect. The data is compatible with (5), but does not support it: the comparison free of the team effect is too small to conclude. Only assigning the number of reviewers at random, e.g. by a coin flip for each PR, would rule out every confounder at once.

(Q4) The query in (2) makes a cleaning choice that no finding mentions. Which part of the query makes it, and which PRs does it affect? Redo the comparison of (3) with the other reasonable choice. Does the observation survive? (*Hint: COUNT on a column ignores NULL*).

The `LEFT JOIN` keeps every PR, including those that match no ticket. For these, `COUNT(t.id)` counts no ticket, so `buggy` is false. This is right for a PR without a bug, but not for the 100 platform PRs of March: for them, “no ticket” means “unknown”, and the query silently replaces a missing value by “no bug”.

The other choice is to drop these 100 PRs, since their outcome is unknown. They are all two-reviewer PRs (the platform team always uses two reviewers), so only the two-reviewer group changes:

- In (3), 10% of the 400 two-reviewer PRs were buggy: 40 PRs. The query counted the 100 PRs of March as not buggy, so these 40 buggy PRs are all among the 300 other PRs.
- Dropping the 100 PRs leaves 40 buggy PRs out of 300:

$$\frac{40}{300} = \frac{2}{15} \approx 13\%.$$

- The one-reviewer group contains no platform PR, so it is unchanged: 320 buggy PRs out of 1600, that is 20%.

The observation survives (two-reviewer PRs are still less often buggy), but the gap shrinks from $20 - 10 = 10$ points to about $20 - 13 = 7$ points: a third of it came from the cleaning choice.

Remember to never hide a cleaning choice: state it in the report, with the result under both choices, as a limitation. Here the cleaning choice hides in the join.