Rosa

Drive Rosa with Your Own AI Assistant

How to use this guide

You have a Rosa account and an API key. This guide lets you use your own AI assistant (for example ChatGPT or Claude) to run your data through Rosa, and to interpret the results yourself. The simplest way to use it: give this whole document to your assistant as context, then ask it to help you debias and evaluate a dataset. It contains everything the assistant needs.

Rosa connects to AI assistants through MCP (the Model Context Protocol), so an assistant that supports MCP can call Rosa's tools directly on your behalf. If yours does not support MCP, the same steps work over Rosa's REST API.

1. What Rosa does

- Rosa debiases tabular data. You give it a CSV plus a small config that names one protected attribute (for example sex, gender, or age band). Rosa transforms the data so that a downstream model no longer relies on that attribute (directly or through proxy features), while each column's statistical distribution is preserved so the data stays useful.
- It has three modes (the mode value you pass is the full string shown in bold):
  - **`diagnose`**: measures where and how strongly the protected attribute is encoded in your data (directly and through proxies). Produces a report and an evidence record, but no debiased data.
  - **`remove_bias_training`**: trains a debiasing model on your data and returns a debiased training CSV and a report showing the bias before and after. The trained model is retained on the instance and referenced by this job's id: you reuse it by passing that id to an inference job, not by downloading it.
  - **`remove_bias_inference`**: applies a trained model to NEW data and returns a debiased CSV. This is the operational path you use in production.
- Rosa only acts on bias that is really there. Before it debiases, it checks (on a held-out basis) whether the protected attribute is actually recoverable from the other columns. If Rosa's test cannot recover it above chance, Rosa reports "no bias detected" and declines the run rather than manufacture a result. This is deliberate anti-fairwashing behaviour. See section 5.
- Important: once you train a downstream model on Rosa-debiased data, you must also feed it Rosa-debiased data when you use it. Sending raw operational data to a model that was trained on debiased data gives invalid results. In practice this means your live data flows through Rosa inference continuously, not just once.
- The debiased output drops the protected column and suppresses its proxies. Any column you list in `ignore_columns` (for example your outcome label) is passed through unchanged, so it survives for downstream model training.
- Rosa preserves each column's statistical distribution exactly, so the data stays useful. It debiases the dataset as a whole rather than editing rows one at a time, so an individual cell value is not meant to be reproduced or read in isolation; what holds is the column distribution and the fairness property.

2. Connecting

Your Rosa portal has an **API / MCP** page with your connection details: the endpoint to use and your API key. For the shared trial the endpoint is `https://api.rosadebias.com/mcp`; if you are on a dedicated instance, use the endpoint shown on your portal page instead. Point your assistant's MCP client at that endpoint using your key, and it will see Rosa's tools. If you use the REST API instead, send your key as the header `X-API-Key: <your key>` on every request.

Handling cold starts and retries. The trial service stops itself when idle to save cost, so your first call after a quiet period may hit a service that is still waking. When that happens the request comes back with an HTTP status that tells you to retry: **503** (waking - the response carries a `Retry-After` hint, typically ~90 seconds), **502** (a transient gateway blip during startup), or **429** (you are sending too fast - it also carries `Retry-After`). All three are retryable: wait the `Retry-After` interval if one is given, otherwise back off exponentially (for example 2s, 4s, 8s) for a handful of attempts, and the call will succeed once the box is up.

One 429 is different: a refused start. To keep cost bounded, a stopped service is started only a limited number of times an hour from any one network (20 on the trial). Past that, the request that would have started it comes back **429** with the message "Too many wake attempts", and **nothing is started**. The `Retry-After` on that response is a generic hint, not the time until you can start the service again: that comes only when the oldest start from your network is more than an hour old. Retrying does not use up any more of the allowance, but it will not succeed sooner either, so a handful of backed-off retries will simply fail. **If you see this while the service is stopped, stop retrying and tell the user** that the service could not be started because it has already been started several times from their network in the last hour, and to try again later. It usually means the service has been idle-stopping and restarting repeatedly, which a steady pace of work avoids. **A successful job submission comes back as `202 Accepted`, not `200`** - Rosa's job model is asynchronous, so a submit is an acknowledgement that the job is queued rather than a result, and `200` never appears on that route. Treat `202` as success. This is worth stating plainly because it is an easy loop to write by accident: a retry that only recognises `200` never sees success, and keeps resubmitting a job that was accepted the first time. A `4xx` other than 429 is a real problem with the request (fix it, do not retry); a job that reaches `failed` is terminal (read its `error`, do not resubmit blindly). Resubmitting the same job during a wake is safe if you reuse the same `idempotency_key` - Rosa deduplicates it, so a retry never starts a second run. Most MCP clients apply this backoff for you; if yours does not, apply it yourself.

Re-read or rewind your file before you retry a submission. This is the one part of a retry that is easy to get wrong, and the wake path makes it likely rather than rare: because the trial service idle-stops, the very first submission of a session is the one most likely to come back `503`. If your client passed an open file object to that first attempt, it has already been read to the end, and retrying with the *same* object uploads **zero bytes**. Rosa then correctly rejects an empty upload - and the message is about your CSV, so the natural next move is to go and inspect a file that is perfectly fine. Read the bytes **once** into memory and post the buffer on every attempt, or seek back to the start before each retry. Rosa names this possibility in the message when an upload arrives empty, but it is much cheaper to avoid than to diagnose.

A note on data size. For trial-sized datasets (within the trial limits below, typically a few megabytes), you can send the whole CSV inline through MCP. For larger or production datasets, upload the file through the REST endpoint rather than inline over MCP. MCP is designed as a control channel; large data payloads belong on the REST upload path. **Do not rely on a large inline ceiling.** Base64 inflates a CSV by about a third before it reaches the transport, and the effective inline limit is far lower than Rosa's own upload cap - it is set by the MCP transport and by your own MCP client, not by Rosa's dataset limits. Treat inline MCP as suitable for small trial files of a few megabytes, and send anything larger over REST. If an inline submission is refused for size, the response is an HTTP 413 naming the REST upload path; that is a transport refusal, not a dataset validation failure, and it consumes no quota. Preflight the encoded size in your own client before sending, and fall back to REST rather than retrying inline.

The Rosa tools

Tool	Purpose	Key inputs	Returns
`rosa_diagnose`	Start a diagnose job	`csv_content` (your CSV, base64), `bias_columns`, `ignore_columns`, `cat_columns`, optional `filename`	a `job_id` and `input_filename`
`rosa_remove_bias`	Start a training or inference job	`csv_content` (base64), `mode` (exactly `"remove_bias_training"` or `"remove_bias_inference"`), `bias_columns`, `ignore_columns`, `cat_columns`, and for inference `training_job_id` (the job_id of the training run whose model to apply); optional `filename`	a `job_id` and `input_filename`
`rosa_job_status`	Check a job	`job_id`	`status` (`queued` and `running`, then one of six terminal states: `complete`, `declined_no_bias`, `declined_insufficient_support`, `failed`, `cancelled`, `interrupted`), `mode`, `row_count`, `error`
`rosa_get_manifest`	Get the Run Manifest (the immutable evidence record)	`job_id`	manifest JSON - **parse the canonical JSON text block** (see the note below)
`rosa_get_report`	Get the PDF report (diagnose and training only)	`job_id`, `return_as` (`"url"` or `"base64"`)	report
`rosa_get_artifacts`	List and download output files	`job_id`	`artifacts` - a list, each entry with a `download_url`

Tool	Purpose	Key inputs	Returns
`rosa_list_jobs`	List your jobs	all optional: `limit`, `status`, `cursor`	jobs, `next_cursor`
`rosa_cancel_job`	Cancel a running job	`job_id`	confirmation

Note the `mode` value must be the exact full string (`remove_bias_training` or `remove_bias_inference`); the short forms "training" and "inference" are rejected before the job is submitted.

Reading the Run Manifest over MCP. `rosa_get_manifest` returns two things: a canonical JSON text block and a typed `structuredContent` object. **Parse the text block** - it is the authoritative full record, and it is the same JSON that `GET /v1/jobs/{job_id}/manifest` returns over REST. The typed object is a convenience view for clients that want a declared schema; its schema does not necessarily describe every nested field, so an assistant that reads only `structuredContent` risks missing parts of the evidence record - nested hashes, encoded widths, resource and timing telemetry, feature associations, column statistics and provenance. Parsing the text block is correct regardless of how the typed schema evolves.

Listing your jobs. `rosa_list_jobs` returns the newest 200 jobs by default. If you have run more than that, the response carries a `next_cursor`: pass it back as `cursor` to get the following page, and keep going until it comes back `null`. Do not assume the first page is your whole history - check `next_cursor` before concluding a job is missing. Following the cursor is safe while jobs are still being submitted, as new jobs appear at the top of the ordering and cannot disturb a walk already in progress. You can also ask for a slice directly: `limit` accepts 1 to 1000, and `status` takes one state or a comma-separated list, so `status="queued,running"` is the cheap way to poll for work still in flight - it stays a constant handful of rows however long your history gets.

Submit a job, poll `rosa_job_status` until it is `complete` or `failed`, then fetch the manifest, report, and artifacts. Training and diagnose take a few minutes; inference is seconds to a few minutes. The debiased CSV is delivered as a file named `<name>_fair.csv`, downloaded from the `download_url` returned by `rosa_get_artifacts`.

Naming your outputs. `<name>` comes from your source CSV's filename. When you send a CSV inline over MCP, pass the optional `filename` (for example `applicants.csv`) on `rosa_diagnose` / `rosa_remove_bias` so the debiased output is `applicants_fair.csv`; the tool echoes back the sanitised name it used as `input_filename`. If you omit `filename`, the source defaults to `rosa_input.csv`, so the output is `rosa_input_fair.csv`. Either way the exact artifact names are always listed by `rosa_get_artifacts` once the job completes.

3. Describing your dataset to Rosa

Every run needs three lists of column names.

- **`bias_columns`**: exactly one bias characteristic, protected or not. It must be a categorical with two or more groups; there is no maximum number of groups, but the size of the smallest group is what to watch (see the guidance below the quick checks). It can be binary (for example sex) or multi-class (for example a 5-way ethnicity). If your characteristic is continuous (for example age), group it into bands first.
- **`ignore_columns`**: columns Rosa must not transform. Always put your outcome or label column here, so it survives unchanged for downstream model training. Also put weights and free-text columns here. Row identifiers are different: an ignored column is still uploaded and still processed, so `ignore_columns` is a modelling control

and not a privacy boundary - remove or pseudonymise real identifiers before you submit, and never send direct identifiers to the free trial.

- **`cat_columns`**: every categorical column, including the bias column (the bias column must always be declared categorical) and including the label if it is categorical. Integer-coded categoricals (for example a 0/1 flag) must be listed here too, or Rosa treats them as numbers. Ordinal rating scales (for example a 1 to 5 score) may be left as numerics.
- Everything not in `cat_columns` is treated as a genuine numeric.

Quick checks for your dataset:

1. Rows: at least 1,000 and at most 50,000 on the trial. Raw columns: at most 100. **1,000 is the floor for Rosa ACCEPTING the job, which is a lower number than the floor the published no-trade-off claim is made for - that is 1,500 eligible training rows (section 7).** Between the two, Rosa runs and reduces measured bias, but a single run is not a reliable read on the utility claim.
2. Post-encoding width at most 150. Estimate it as the sum of distinct values across your `cat_columns`, plus the number of numeric columns. If it is over 150, drop or coarsen a high-cardinality categorical.
3. One protected attribute per run. If several matter, run them one at a time.
4. Never put a direct identifier (name, email, customer id) in `cat_columns` - and do not rely on `ignore_columns` to make one safe. **Remove or pseudonymise direct identifiers before you upload.** `ignore_columns` is a modelling control, not a privacy boundary: it keeps a column out of the model and out of the manifest's category enumeration, but the column is still uploaded and processed. If an identifier is exceptionally permitted under your contract, ignoring it is the right setting, and it is still not a substitute for leaving it out.
5. Sanity-check that the protected attribute is actually recoverable from the other columns (see section 5 and section 6). If it is not, Rosa will decline the run, correctly.
6. **Know what Rosa trains on.** For `diagnose` and `remove_bias_training`, a dataset over about 10,000 rows is down-sampled to a seeded representative sample of about 10,000 rows before the model is trained; the trigger is the row count, not the file size, and inference always runs on your full file. **The sample is drawn near-proportionally, so every group keeps roughly its share of the file** - a group that is 1% of 100,000 rows is about 100 rows in the sample, measured live at 1,643 to 1,692 rows per group against 1,667 expected on a six-group file. **A bigger file therefore does not give a small group more rows in training.** If a group is small in absolute terms, the way to give Rosa more of it is a file in which that group is a larger share, not a larger file. The Run Manifest records both counts (`row_count` and `sampled_from_row_count`, section 5) and the report's Run Evidence section states them.

Missing values. Rosa imputes them automatically - the column mean for numeric columns, the most-frequent value for categorical columns. Keep missing below roughly 5% per column; above that, imputation quality degrades and results become less reliable. Represent a missing value as an empty cell or a standard null token (`N/A`, `NA`, `NULL`, `null`, `NaN`, `None`) - Rosa recognises these and imputes them. Numeric placeholders such as `-999` or `9999`, and words such as `unknown` or `missing`, are read as real data values and skew the column, so replace them with an empty cell first. To audit missing data, run Diagnose - its report and Run Manifest record the per-column missing-value count found in your input. A Training run imputes those gaps, so its output shows the filled data.

Choosing your bias characteristic and how to encode it

A bias characteristic is any variable you want to measure and remove bias with respect to. It does not have to be a legally protected characteristic. Debiasing sales performance with respect to regional office, or an assessment with respect to intake cohort, is as valid a use of Rosa as debiasing a hiring model with respect to sex, and the mechanics are identical.

There is one check worth doing before you spend anything, and it takes a moment.

Count the rows in your smallest group, as they will stand after sampling, and count how many groups your characteristic has. Those two numbers together tell you what to expect, and neither one alone does. Training runs on a sample of at most about 10,000 rows drawn in proportion to your groups, so a group that is 1% of your file is about 100 rows in the sample whatever the size of the file. **What we have measured is set out below, by number of groups.** A group below the line can still be detected, and a decline is never a finding that your data is fair.

How many rows a group needs depends on how many groups the characteristic has. There is no single row threshold, and a number that holds at three groups understates the requirement badly at eight: we have recorded an eight-group characteristic declining on every run with 120 rows in its smallest group. **The tables below are therefore stated per group count, and a figure should only ever be read across from the row for your own.**

Why it is the smallest group that matters

Before Rosa debiases anything it tests whether the characteristic can be detected at all: it trains a small classifier to recover the characteristic from your other columns, on rows it has not seen, and compares how well it does against the same classifier trained on shuffled labels. Every group counts equally in that test, whatever its size: a group of a hundred rows and a group of five thousand each contribute a fifth of the score on a five-group characteristic.

So the test is not defeated by a dominant group. It is limited by how few examples of the rarest group it has to learn from - and, because every group counts equally, **by how much of the score your under-populated groups are responsible for.** That is why the number of groups matters: four noisy groups out of eight contribute half the score, while one noisy group out of three contributes a third.

What the boundary is not. Above it, the strength of what Rosa recovers stops improving - on our eight-group ladder it flattens from about 180 rows. What more rows buy after that is not a stronger signal but a **steadier** one, far enough from chance to be called reliably. Below the line the signal is still there; the test just cannot separate it from luck.

How many groups can I use?

There is no maximum and we publish none. But the count is a governing variable, not a free choice.

The arithmetic is what bites. On the shared service, training uses a representative sample of at most about 10,000 rows however large your file is, **and that sample is drawn roughly in proportion to your groups.** So each group's share of your population sets its share of the sample, and **a bigger file does not give a small group more rows.** A group that is one per cent of your people is about a hundred rows in the sample whether you send 50,000 rows or five million. **More groups mean smaller groups, and smaller groups mean more of the score is noise.**

What we have measured, by number of groups

Every figure below is on synthetic data built to the shape described, with a known bias present, run through the live service. **Your own data may differ, and the strength of the bias in your file matters as much as the row counts.**

Eight groups, at UK Census religion proportions, five independent datasets:

rows in the smallest group	detected
44	1 of 15

rows in the smallest group	detected
120	0 of 5
181	**5 of 5**
312	5 of 5
440	8 of 8

Three groups, one dominant, on a single pool sampled down:

rows in the smallest group	detected
20	3 of 10
30	4 of 10
40	6 of 10
50	6 of 10
75	9 of 10
100	**10 of 10**
250, 500	10 of 10

Two groups: 16 runs of 16 detected, across four datasets whose smaller group ran from 200 to 3,462 rows.

Read those three together and the point is the comparison: 100 rows is comfortable at three groups and 120 rows is not enough at eight. **A row count carried from one group count to another will mislead you, and the gap is wide enough to change the answer rather than shade it.**

Operational characteristics often need grouping, and usually that is free

Non-protected bias characteristics are frequently higher-cardinality than protected ones. A regional-office column might hold forty offices, a product column two hundred lines, a store column a value per site.

Grouping them is usually a modelling choice with no cost beyond resolution. Forty offices become five regions; two hundred product lines become six categories; sites become urban and rural.

When you group, make sure no group is left with only a handful of rows. Five regions holding a few hundred rows each will work; a grouping that leaves one region with thirty rows in the sample will be the region that decides whether the run detects. Choose the grouping that matches the decision you are trying to make fair, keep every group comfortably populated where the data allows, record what you did, and run it.

The Equality Act 2010 characteristics, as a worked example

The third column is the smallest group's row count in a 10,000-row training sample, using published UK population and workforce proportions as the share. **These are our estimates of typical UK proportions, not measurements on Rosa.** The fourth column says what we have actually run: every measured row is on synthetic data built to that shape, on the current detection test, and your own data may differ. Count the rows on your own file rather than relying on this table.

Characteristic	Encoding	Groups	Smallest group in a 10,000-row sample, at typical UK proportions	What we have measured
Sex	As recorded	2	thousands	Covered by the two-group result above: 16 of 16
Marriage and civil partnership	Married or in a civil partnership, or not	2	thousands	As above
Age	Bands, not raw age	4 to 6	hundreds to over a thousand	**12 of 12 across four synthetic draws at 4 to 6 bands, smallest band 171 to 882 rows**
Disability	Binary, as the Act defines it	2	over a thousand	Covered by the two-group result above
Religion or belief	Two groups, for example one belief against all others	2	thousands	3 of 3 at 661 rows
Religion or belief	**The eight or nine standard categories**	**8 or 9**	**about 50**	**1 of 5 at 50 rows. See the eight-group ladder above: 0 of 5 at 120 rows, 5 of 5 at 181**
Race or ethnicity	Two groups, largest against all others	2	over a thousand	Covered by the two-group result above
Race or ethnicity	The five high-level groups	5	about 200	**6 of 6 across two datasets at 181 and 210 rows**, one of them scored out of sample

Characteristic	Encoding	Groups	Smallest group in a 10,000-row sample, at typical UK proportions	What we have measured
Pregnancy and maternity	Binary, in a workforce	2	a few hundred	3 of 3 at 200 rows
Gender reassignment	**Three groups**	**3**	**about 50**	**22 of 28 across four independent draws at 50 rows**
Sexual orientation	**Standard harmonised categories**	**5**	**about 30**	**4 of 8 across two draws at 30 rows**

The three rows to think about are religion at the detailed categories, gender reassignment and sexual orientation, and they are the three where a national-proportion sample leaves the rarest group below about a hundred rows. On those the result is a property of how many rows that group contributes in **your** file, which may be very different from the national picture.

On the age row, two things are worth saying plainly. The bands and their proportions are our own approximation of a workforce shape, **not an official table**, so read the row as "four to six bands at plausible workplace sizes" rather than as a statement about any published distribution. And the most useful of those four draws is the smallest: **six bands with the youngest or oldest band at 171 rows, scored on only 25 to 29 held-out rows, detected on all three runs.** That sits well below the eight-group boundary in the tables above, which is the clearest illustration on this page that **the number of groups matters as much as the row count** - 171 rows is comfortable at six bands and would be nowhere near enough at eight.

One arithmetic fact that catches people out. At true UK religion proportions, **a 2,000-row file cannot be tested on the detailed categories at all** - the rarest group lands at about eight rows, below the nine-row floor, and the run returns "Detection not supported" with no verdict. That is an ordinary size for a mid-sized employer's workforce. **Multiply your rarest group's share by 10,000 and check it clears nine before you spend a job.**

A note on those three rows, because it is easy to misread. At 30 to 50 rows the outcome is **erratic rather than consistently negative**: the same shape has come back 22 of 28 in one case and 4 of 8 in another, and two of our own predictions about this range were wrong in opposite directions. Near the line, a single run tells you very little. If you need evidence rather than an indication, run three.

Grouping a protected characteristic is not the same as grouping an operational one

Grouping forty offices into five regions costs you resolution. Grouping a protected characteristic can cost you the finding.

Ethnicity is the clearest case. The reason detailed ethnicity monitoring exists is that the high-level groups conceal differences inside them. Outcomes for Indian, Pakistani, Bangladeshi and Chinese populations can differ substantially while sitting inside a single "Asian" group, and the same is true of African and Caribbean populations inside "Black". If you debias on an aggregate, Rosa removes the signal that separates those aggregates. It does not remove a disparity that exists within one of them, because at that resolution the disparity is invisible to the run.

So a clean result on a coarse encoding means the coarse signal has been reduced. **It does not mean the characteristic has been addressed.** And because the test needs rows in every group, the detailed encoding is the one whose rarest group decides the run; if that group is very small, it is the group to think about, not the aggregate.

Can I reshape my file to make it work?

Mostly you do not need to, and the reshaping that used to be tempting no longer has a point.

- **Evening up your groups by discarding rows from the large ones does not help.** The test is not defeated by a large group, so removing its rows buys nothing and costs you data and the quality of the debiasing.
- **Merging small groups into an "other" category does not help the rare group.** It hides it inside a larger one, and at that resolution its disparity is invisible to the run. Merge only when the merged category is itself the thing you want to make fair.
- **Dropping your largest group changes the question, and does not help.** Rosa would then be measuring and removing bias between the groups that remain, and would say nothing at all about how any of them are treated relative to the group you removed, which in most protected- characteristic work is the comparison that matters. You would also get no debiased output for the rows you removed. If you drop a group, say so in your record: the run is then evidence about the groups that remained, and not about the characteristic as a whole.
- **The one thing that does help a rare group is more of its rows in the training sample.** Since the sample is drawn in proportion, that means a file in which the rare group is a larger share: a cohort, a catchment, a period or a site where it is better represented, or a deliberately over-sampled training file that you then apply to your full population at inference. Inference runs on your whole file and has no detection test, so a model trained on a file where the rare group is well represented can debias a population where it is not. Say in your record that you did this, because the debiased output then reflects a model trained on a different mix from the population it was applied to.

A rare group is a property of your population, not a formatting problem. If the rarest group in your file is a few dozen rows and cannot be enlarged, talk to us before drawing a conclusion, because the right next step depends on what you are trying to evidence.

What this adds up to, stated as a limit rather than a workaround

We would rather say this plainly than let you infer it from the tables.

> **Where your rarest group falls below the line for your number of groups, Rosa can debias the characteristic at a coarser resolution and cannot give you a verdict at the detailed one.**

That is a limit on what we can evidence, not a recipe. Coarsening is a legitimate modelling decision and we say so above - but the whole reason detailed monitoring exists for characteristics like religion and ethnicity is that aggregates hide differences inside them, so a clean result on the coarse encoding does not tell you the detailed disparity is absent. **It tells you the coarse signal has been reduced. Record which resolution you ran at, and do not carry a finding from one to the other.**

Neither does it mean the detailed run is worthless. It means a single run at that size is an indication rather than evidence, and that a decline there carries almost no information about your data. If the detailed resolution is the thing you need to evidence, the honest options are a file in which that group is better represented, several runs rather than one, or telling us so that we can say what the instrument can and cannot support for your case.

The smallest group's size is a property of your file, not of the characteristic

The table above uses national population proportions, because that is what we can publish. **Your file is not the national population.** An employer recruiting in a diverse city, a service with a specific catchment, or a dataset

already filtered to a particular cohort can have group proportions nothing like the national figures, and the rarest group can be comfortably represented without you doing anything to the data.

So the table above is a starting expectation, not a verdict. **Count the rows in your smallest group on your own file.** It takes seconds, it costs no job, and it is the only figure that describes your run.

When Rosa declines, and what it does and does not tell you

Rosa reports recoverable bias only when its detection test finds the characteristic above what the same test reaches on shuffled labels, and by a margin above a small floor. When it does not, the run returns a decline with a full evidence record and no debiased file.

A decline is a completed, evidenced decision. It is not a failure, not an error, and not something to resubmit unchanged. It is also not a finding that your data is fair.

There are two quite different reasons a decline happens, and the run cannot tell you which one you have.

The characteristic genuinely carries no signal. Nothing in your other columns tracks it, so there is nothing to remove. This is the good case.

The characteristic carries signal, but your smallest group gives the test too few examples to find it reliably. Every group counts equally in the test, so it is the rarest group that decides how close the run sits to its decision line. **We have measured this directly on the same data with the same bias present: detection becomes unreliable as the smallest group shrinks towards a few dozen rows, and the size at which it becomes reliable depends on how many groups the characteristic has as well as on the row count** - see the tables in section 3. Near that line the same file can decline on one run and detect on the next, and both are the test working as designed.

More rows do not fix this. Elsewhere we say the smallest detectable bias depends on how much data you have, and that is true of the file as a whole. It is not the whole story. The sample is drawn in proportion, so a larger file gives the rare group no more rows in training; what it needs is a larger share.

If you get a decline, look at the group counts before you conclude anything. Your report's Run Evidence section lists the rows per group in the analysed sample. **Compare your smallest against the table for your own number of groups**, not against a remembered number: if it sits below the line for that group count, the decline is more likely telling you about the size of that group than about your data being fair. If every group is well populated and the run still declines, the first explanation is the one to take seriously, and the report's evidence table shows how far below the shuffled-label baseline the observed recovery sat.

And read `min_rows_scored` on your manifest. It is the fewest held-out rows your smallest group was actually scored on, which is the quantity the test ultimately ran on - usually a fraction of that group's rows in the sample, because the score is taken on data the classifier has not seen. It tells you what the test had to work with rather than what your file contained. **Read it against your own configuration, not as a universal threshold**: eight groups needed 23 or more in our testing, while three and five groups have detected on far fewer. **It is a post-run check on what happened, not a number to plan against.**

One thing worth knowing about which way the test errs. We have measured the shipped test to be **conservative by construction on the shapes we have examined** - the way its comparison is built makes it more likely to miss a real signal than to invent one. That is the direction we would choose if forced, because telling you that you discriminate when you do not is the worse failure. **But it is another reason a decline is weaker evidence of fairness than it looks, and the same conservatism applies to the residual check after debiasing** - see section 5.

When the test cannot run at all: "detection not supported"

There is a line below the decline. **If any group of your bias characteristic has fewer than nine rows in the training sample, Rosa does not run the detection test at all**, because a group that small cannot be scored on a

held-out split. The job ends with the status `declined_insufficient_support`, shown as "Detection not supported", and this applies to **Diagnose and training jobs alike**: a Diagnose in this position does not complete.

It is a completed decision with an evidence report and a Run Manifest, and no debiased file. The manifest reads `detection_supported: false` and names the group in `insufficient_support: {group, rows_in_sample, minimum_rows}`.

It is not a decline and it is not "no bias". A decline means the test ran and found nothing above chance. This means the test never ran, so there is no verdict, no p-value and no bias figure, and it says nothing about whether your data is biased. **Resubmitting the same file gets the same answer.** What changes it is the same thing that helps a rare group anywhere in this section: a coarser grouping of genuinely similar categories, or a training file in which that group is a larger share. Deleting rows, duplicating them or splitting the file to get past the line is not a fix. Nine is the point below which the test cannot run; it is not a size at which detection is reliable: at three groups that took about a hundred rows, and at eight groups it took 181.

4. Limits and messages (so your assistant can self-correct)

Message	Meaning
No bias detected (the job declines)	Rosa could not recover the protected attribute above chance from the other columns, so there is nothing to remove and it stops rather than fabricate a result. This is expected, correct behaviour, not an error. See section 5.
Row limit exceeded	more than 50,000 rows (trial limit)
Column limit exceeded	more than 100 raw columns
Encoding limit exceeded	more than 150 post-encoding dimensions
Too few rows	fewer than 1,000 rows is rejected
Multivariate not supported	more than one bias column was given
Bias column single group	the bias column has fewer than two groups, so there is nothing to debias
Undeclared text column	a feature column contains text values but is not declared as categorical. Add it to `cat_columns` if its values are categories, or to `ignore_columns` to pass it through untouched. An ignored column is still uploaded and still processed, so remove or pseudonymise identifiers before upload. Rosa names the column.
Empty column	a column has no values. Provide values or remove the column before submitting. Rosa names the column.

Message	Meaning
Inference mismatch (fails the job)	the inference file is structurally incompatible with training: a column is missing or renamed, or a category value appears that was never seen in training. **Column ORDER does not matter** - if the same columns arrive in a different order Rosa reorders them to match the training data and records a `COLUMN_ORDER_ALIGNED` advisory; the values are unchanged and no action is needed
Inference advisory (job still succeeds)	a training category is absent from the inference file, or a numeric is outside the training range. The job completes and the manifest records a note.
Small inference population (advisory)	an inference file below about 1,000 rows is debiased on a small population. The result is valid but the manifest flags it, because a very small population is debiased less strongly.
Quota exceeded	the trial allows 20 jobs per month

A failed job still produces a partial manifest with an `error` field explaining what happened. A remove-bias job on data with no recoverable bias is **not** a failure: it returns the terminal status `declined_no_bias` with a complete evidence manifest and a report (and no `error`). Treat it as a finished decision, not something to resubmit.

A job whose smallest group had fewer than nine rows in the training sample is not a failure either: Diagnose and training jobs both return the terminal status `declined_insufficient_support` ("Detection not supported") with an evidence manifest and a report, no `error`, and no debiased file. The detection test did not run, so it carries no verdict. Resubmitting the same file returns the same status; see section 3 for what does change it.

A decline is a result, not a failure. `declined_no_bias` is a terminal, successful outcome carrying a full report: it means this test, on this dataset, found no signal above the chance level it established. It is not an error, it does not mean the job failed, and resubmitting is the correct response if you need a completed pair. On the current build, all 82 COMPAS training runs behind the published figure found recoverable bias and returned a debiased file, none declined, at the trial default of 24 permutations - but a decline rate is a property of a dataset measured on one instrument, not a property of the service, so treat the outcome as something to handle rather than something to predict. A dedicated instance can raise the permutation grid, which resolves the p-value more finely.

Every error carries a machine-readable code. When a Rosa tool fails, the error text is JSON carrying a `code` (an uppercase identifier such as `INVALID_CONFIG`, `INVALID_CSV`, `QUOTA_EXCEEDED`, `RATE_LIMITED` or `ROSA_STARTING`) alongside the human-readable message. **Parse it as JSON and branch on the code; do not pattern-match the prose**, which may be reworded. The code is inside the message text rather than in a separate field because the MCP protocol returns a failed tool call as a single text block with no structured payload, so there is nowhere else to put it. That is a property of the transport and will not change.

`ROSA_STARTING` is the one code to handle specially: it means the instance serving your workspace is waking up, not that anything is wrong. Wait the indicated interval and retry the same request with the same idempotency key.

How fast to call, and how many calls at once

Rosa limits how quickly a single key may call it: roughly **120 reads a minute** (job status, manifest, artifacts, usage) and **20 writes a minute** (submitting a job, cancelling one). These are the trial defaults; a dedicated instance can be configured differently.

Two habits keep you comfortably inside them:

- **Keep only a handful of requests in flight at once - about six is plenty.** If you are tracking several jobs, poll them in small batches rather than firing one request per job simultaneously. It is the natural thing for an assistant to do and it is the easiest way to trip the limit.
- **Poll each job every few seconds, not continuously.** Training and diagnose take minutes, so there is nothing to gain from a tighter loop.

Exceeding a limit returns **429** with a `Retry-After` header, which is recoverable - see the retry guidance in section 2. A 429 saying "Too many wake attempts" while the service is stopped is not this limit: it is a refused start, and section 2 explains why retrying it does not help. It is still worth avoiding: a burst that trips the limit turns into a retry storm, and the wall-clock you lose lands in the middle of an evaluation rather than at the start.

Retries, duplicate submissions, and how long results are kept

Retries and duplicate submissions. Give each job submission an idempotency key, generated once before your first attempt and reused on every retry of that same action. If a response is lost and you retry, Rosa returns the original job rather than starting a second one, so you are not charged twice and the engine does not run twice. Do not generate a new key inside a retry loop; that is what creates a duplicate job. Keys do not expire. Deduplication applies within the Rosa instance serving your workspace, which is the instance your API key is registered to.

How long results are kept. The two files you can download - the debiased data and the report - are available for **7 days** after a job completes. After that a download returns a clear "expired" response and the files are removed shortly afterwards, so download anything you need to keep. Rosa also produces internal files it manages for you and does not hand over - the trained model checkpoint and the engine log. The checkpoint is kept for **365 days**, so inference against an earlier training job keeps working for a year (Rosa resolves it for you from the training job id - you never download or upload it); the log is kept for 7 days.

How long the Run Manifest is kept. The Run Manifest, your evidence record for a run, is kept permanently and is not subject to the windows above. On instances with the evidence vault provisioned, a second copy is written to storage under a seven-year write-once lock that cannot be deleted or altered, including by us. Ask us to confirm the vault status for your instance if that copy matters to your audit.

5. Reading your results

- **Rosa only debiases what is really there ("no bias detected").** Before training, Rosa runs a significance test: it measures how well the protected attribute can be recovered from the other columns on a held-out split, then reruns the same test on many shuffles of the protected labels to establish the chance level for your dataset's shape. It reports "recoverable bias" only when the real recoverability is statistically above that chance level; otherwise it reports "no bias detected" and declines the run rather than manufacture a result. This protects you from fairwashing, and a decline is a legitimate finding (on this dataset, Rosa's test found nothing above the chance level, which is not the same as proving that no model could recover the attribute). A different outcome is "detection not supported" (`declined_insufficient_support`): the test was not run at all, because a group of the protected attribute had fewer than nine rows in the training sample and could not be scored on a held-out split. It carries no verdict and says nothing about whether the data is biased; do not read it as "no bias", and do not try to get past the nine-row line by deleting rows, duplicating them or splitting the file, which changes what is being measured without giving the rare group any more evidence. Choosing a coarser grouping of genuinely similar categories, as in section 3, is a different thing and is a legitimate modelling decision. **A dataset near the threshold can decline on one run and proceed on the next, and that is the test working rather than failing.** How often that happens depends entirely on where your data sits relative to the threshold, so it is worth measuring on your own file rather than assuming a rate: on the COMPAS benchmark shipped with the Starter Kit, 82 consecutive training runs all proceeded. **There is no fixed cutoff** - because it is a significance test, the

smallest detectable bias depends on how much data you have: with a few thousand rows you need a clearly-above-chance signal (a protected attribute recoverable at only ~AUC 0.54 on ~1,500 rows is declined), while with tens of thousands of rows a smaller gap can still register. Practical pre-check: if a simple model cannot beat chance by a clear margin on a held-out split, expect Rosa to decline (see section 6). A remove-bias job that declines returns the terminal status `declined_no_bias` (a completed decision, not `failed`) and records the same held-out evidence any run does: the manifest `bias_summary` carries `bias_detected`, the permutation `bias_p_value`, the `permutation_count`, `bias_check_reps`, and `p_value_precision` behind it, **and the two thresholds the decision was actually taken against - `detection_alpha` and `detection_effect_floor`** - alongside a report documenting the decision. **Rosa reports recoverable bias only when BOTH arms clear: the p-value is below `detection_alpha` AND the observed effect is above `detection_effect_floor`.** The thresholds are on every manifest, accepted runs included, and they are per-instance settings rather than universal constants, so read them from the record rather than assuming the trial defaults. **The effect they gate is `average_discriminator_performance` minus `null_accuracy`, on the plain accuracy scale. It is NOT the `bias` field**, which is that same difference rescaled so that a dataset's maximum possible bias reads 1.0 - a larger number on a different scale. On one measured declined run the effect was 0.0289 while `bias` read 0.0725: comparing `bias` against a 0.02 floor would have said the effect arm passed comfortably, when in fact that run declined on the p-value arm alone. If you are reconstructing a decision, compute the effect from its two published inputs. **Those last three describe the test's resolution, not a limit on what Rosa can measure.** The p-value is estimated from a grid of shuffled-label runs, and the finest value a grid of K runs can express is 1/(K+1) - so the trial's 24 permutations resolve to 0.04, and a p-value of 0.04 means "at the finest level this run could measure" rather than "exactly 4%". The grid is a per-instance setting: it is sized to keep a shared-trial job inside its time budget while still being able to fire the detection gate, and it can be raised for evidential work on a dedicated instance (99 permutations resolve to 0.01, 999 to 0.001). The Starter Kit's `EVIDENCE/interpreting-results.md` explains this in full.

- **Rosa's bias measure**: the diagnose and training reports show a bias figure. It is an honest held-out estimate of how recoverable the protected attribute is by Rosa's own bounded test (0 means that test found nothing above the chance level for your data, not that no model could recover it); on data with no recoverable bias Rosa declines to debias rather than inventing a pass. The training report shows this before and after debiasing. Read both, and focus on the change from before to after. The measure is calculated from a stochastic process, so exact figures vary slightly run to run; the before-to-after reduction is the meaningful result.
- **Read the verdict before the number, on both measurements.** A bias figure on its own does not tell you whether the signal is still there. Every diagnose and training run also carries a significance verdict from a shuffled-label permutation test, and the manifest records it twice: `bias_detected` / `bias_p_value` / `null_accuracy` for the measurement BEFORE debiasing, and `residual_bias_detected` / `residual_bias_p_value` / `residual_null_accuracy` for the residual AFTER it (training jobs only). **Three states, and they are not two:** `true` means this test found the attribute above chance; `false` means it did not, which is a failure to detect at the recorded sample size, estimator and permutation resolution rather than proof of independence; and **absent or `null` means no significance figures were recorded, which is not the same as `false`**. Treat a missing value as unknown and say so. **A small residual that is still detected is a different outcome from the same number with no detection behind it** - a residual of a few per cent can come back `residual_bias_detected: true`, and reporting it as a pass on magnitude alone is the mistake this field exists to prevent. Your report and the customer portal both print the verdict beside the figure; do the same.
- **On reproducibility (why there is no seed).** Rosa deliberately exposes no training seed. Debiasing is a stochastic process, and a fixed seed would not make it bit-for-bit reproducible anyway (numerical libraries and thread scheduling still introduce variation), so a seed would offer false precision. Reproducibility is therefore statistical: run Rosa several times - independent Rosa replicates, not seeds (Rosa exposes none) - and take the mean. This is exactly why Rosa's own published benchmark figures are reported as N-run means with a range, each stamped to a specific Rosa version, rather than a single number - the honest form for a stochastic engine.

- **Do not judge debiasing by re-diagnosing the inference output.** A fresh Diagnose of *debiased inference* data reads materially higher residual than the *debiased training* data - inference applies the already-trained mapping to new rows and does not re-optimise, so it keeps more residual signal. This is expected, not a defect (in production you train once, then apply the model to fresh data). Inference produces no bias report; the bias figure lives on the training job. The real proof that debiasing worked is the downstream A/B in section 6: Model B - trained on debiased training data and run on debiased inference data - stays fair on the ground truth.
- **Proxy view** (`feature_associations` in the diagnose and training manifests): each feature's association with the protected attribute, one feature at a time. After training, `feature_associations_post` shows the same associations recomputed on the debiased data, so you can see the strong single-feature proxies collapse toward zero. **Read it for what it is: a transparent check you can reproduce yourself, not the measurement Rosa gates on.** Rosa's own measure is adversarial and joint - it reads all features together, including nonlinear and combined proxies no single-feature score can see - so a collapse here is direct evidence that the single-feature proxy signal was suppressed, and the residual figure with its verdict is what tells you how much a detector can still recover overall.
- **The debiased data**: `<name>_fair.csv`, from `rosa_get_artifacts`.
- **What is preserved.** Each column's marginal distribution is preserved exactly - the same values, reordered, to float64 precision - so column means, standard deviations, and percentiles are unchanged in both the training and inference outputs. Rosa writes these values to CSV losslessly: read the debiased CSV with a correctly-rounded parser (in pandas, `read_csv(..., float_precision="round_trip")`) and every column's sorted values come back bit-for-bit, at zero difference; a default parser may show one or two ULPs of its own rounding on the last digit, which is the reader's rounding, not a change Rosa made. What *may* change are cross-column correlations, specifically where they were encoding the protected attribute (that is the bias being removed).
- **The Run Manifest** is your immutable evidence record: content hashes, an echo of your config, row counts, the bias summary, and any advisories. Keep it. It is what makes each run auditable. From manifest schema 1.6 the bias summary also carries a per-group block (`per_group`, one entry per group of the protected attribute, marked exploratory) so an assistant can say which group moved the dial, without treating any single group's figure as a finding. **Every field is defined in the Rosa Glossary**, which ships beside this guide in the Starter Kit - use it rather than inferring a field's meaning from its name, because several come in before-and-after pairs whose difference matters. **On an inference manifest, `training_job_id` names the training run whose model was applied**, so an assistant chaining a training job to an inference job can prove the pairing from the record instead of asserting it.

6. Checking Rosa on your own data (recommended)

Rosa's claim is that a model built on debiased data shows much less group disparity than one built on raw data, at little or no accuracy cost. You do not have to take that on trust. Here is how to test it on your own dataset, measured properly (held out, never on the same rows you trained on):

First, a quick pre-check that saves jobs: confirm the protected attribute is actually recoverable. Train a simple model to predict the protected attribute from the other columns and check it beats chance (for example an AUC clearly above 0.5). If it is essentially at chance, Rosa has nothing to remove and will decline the run, so there is no before-and-after to measure on that attribute. That is a legitimate result to report, not a failure.

While you are at it, check for **near-deterministic proxy columns**. If a single non-protected feature predicts the protected attribute very strongly on its own - a rule of thumb is held-out **AUC >= ~0.85**, or a category that is **>= ~95% one group** (for example, in the US Census "Adult" dataset the `relationship` column: *Husband* is ~100% male and *Wife* ~100% female) - it is a near-deterministic proxy. Rosa's univariate debiasing reduces the overall signal, but a downstream model can still reconstruct the group split from such a column, so your downstream fairness gain will be limited. Consider removing or coarsening these columns before debiasing. (Debiasing several protected attributes jointly is a future capability; today Rosa debiases one attribute per run.)

If it is recoverable:

1. Split your dataset into a train part and a test part, keeping the proportions of protected attribute and label similar in both, with a fixed random seed. Set the test part aside.
2. **Model A (baseline)**: build a downstream classifier on the raw train part; evaluate it on the raw test part.
3. **Model B (with Rosa)**: run `remove_bias_training` on the train part (you get debiased train data and a model). Run `remove_bias_inference` on the test part with that model (you get debiased test data). Build the same classifier on the debiased train; evaluate it on the debiased test.
4. Use the same pipeline for both: scale the numeric features, one-hot encode the categoricals, and use a standard classifier (for example logistic regression) identically for A and B. The features are every column except the protected attribute and the label; the label is your outcome column.
5. Rosa removes the protected column from its output, so to measure group fairness, join the protected attribute back from your original rows by row position (row order is preserved).
6. For both models, compute accuracy, and the gap between groups in: the rate of positive predictions (demographic parity), the false-positive rate, and the true-positive rate. For a multi-group attribute, use the largest gap across groups.
7. Repeat the classifier step over a few random seeds and take the mean. Run Rosa itself only once per split, so you stay within your monthly job quota.

How to read it: Model B's gaps should be clearly smaller than Model A's, with accuracy close between them. If accuracy drops sharply on Model B, that is a real signal worth understanding (it can mean the legitimate signal in your data is entangled with the protected attribute). Report what you see rather than what you expected. There is no perfect "oracle" model to compare against on real data - for that, use a synthetic dataset with a known fair answer (section 7).

A suggested plan for the trial

The trial gives you 20 jobs a month, renewing each month. So the question is not how to ration one budget but what to learn in what order. The plan below spends the first month on datasets where the right answer is already known, the second on data shaped like yours, and the third on your own. **Each month stands on its own, and you can stop after any of them and have learned something.**

Month one: learn the instrument on known answers, 15 jobs

Run our three worked examples, in this order, because the order is the lesson.

Dataset	Jobs	Why it is here, and why in this position
Test 1	5	One diagnose, then two training-and-inference pairs. **The strong case**: a clear signal, a published figure you can check against the shipped numbers, and downstream results that improve. This is where you learn to read a manifest, a report and the A/B comparison while everything behaves.

Dataset	Jobs	Why it is here, and why in this position
COMPAS	5	Same shape. **The honest hard case**: weak-signal real data where the fairness gaps shrink sharply and the model's ranking power falls towards the majority baseline, at about a 14-point accuracy cost. **Most vendors do not show you this dataset. It is here because the trade-off is real and you should see its size before you trust anything else we say.**
Binary synthetic	5	Same shape, on the bundled generator. **The only case where you hold the known fair answer**, so the validator can grade all three models against truth rather than against each other (section 7).

15 jobs, and the five spare absorb a retry or a second look at whatever surprised you.

One economy worth taking. A training job runs the detection test first, so a second diagnose on a file you have already diagnosed tells you nothing new. **Diagnose each dataset once, then spend the rest on pairs.**

And two pairs is a demonstration, not a measurement. Rosa is stochastic and exposes no seed, so our own published figures are multi-run means with a range - ten runs for Test 1, eighty-two for COMPAS. Two pairs show you the machinery working end to end; they do not establish a number.

Month two: find out whether Rosa fits your data, before you send any

This is the month that decides whether a trial becomes a project, and none of it needs your real data.

- **Read your own geometry off section 3.** Count your groups and your smallest group's rows after sampling, and compare against the table for **your** group count. That costs nothing and is the single most informative thing you can do.
- **Generate synthetic data shaped like yours** with the bundled generator - your group count, your proportions, your row count, a bias you inject and therefore know is present. Run it. **You now have a run on your own shape with a known answer**, which is a far better predictor of what Rosa will do with your real file than any of our benchmarks.
- **Run a multi-class characteristic** at a size above the line in section 3, and a second below it. **Seeing the boundary behave on your own shape is worth more than being told where it is.**
- **A diagnose on your real data**, once the above tells you what to expect.

Month three: scope the proof of concept

Your own data, the characteristics you actually have to evidence, and the edge cases you already know are coming - the rare category, the small site, the year with the odd intake. Run the Model A versus Model B

comparison in section 6 on a real file, and decide from measurements rather than expectations which datasets go into a proof of concept.

What this plan cannot tell you, stated plainly. A handful of pairs is a targeted evidence budget, not a certification campaign. Our own testing needed twenty pairs per arm to detect a large difference between two configurations and could not distinguish twenty passes in twenty from seventeen at that size. Read a few consistent pairs on your own data as encouraging; and do not read two failures as a verdict on the product - on some datasets the downstream ordering fails repeatedly with nothing wrong, because the utility comparison is closest exactly where the raw data carried the least bias to begin with. If your smallest group sits below the line in section 3, expect a decline and read that section before concluding anything from it.

7. Validate against a known ground truth (synthetic data)

The Model A versus Model B test in section 6 is the honest test on **your own** data, but real data has no "correct fair answer", so it can show the group gap *moved* without showing how close it got to ideal. For a gold-standard check, use a **synthetic** dataset where the fair answer is known and run the full **A/B/C** comparison. Rosa provides two standalone tools for this as downloads from your portal (neither imports Rosa, so you can read exactly what they do):

1. `synthetic_test_generator.py` - build the dataset. It emits a dataset in which the bias reaches a model ONLY through proxy features, never through the protected attribute itself, with two outcome columns: a **biased `outcome`** (which you list in `ignore_columns`) and a **held-out `fair_outcome`** - the unbiased ground truth you score against. It also writes a matching Rosa config, so you do not hand-build one. Key options:

- `--protected-kind binary` (the **default: 2 groups**) or `multiclass` with `--n-groups K` (K >= 3). The generator will emit any K; whether Rosa *accepts* a large K is the separate cardinality question (guidance, not a hard cap - see the portal FAQ).
- `--n-rows` plus a target width (genuine categoricals and their cardinalities) - keep these within the trial caps (section 3: at most 50,000 rows and 150 post-encoding dimensions).
- proxy strength / `--bias-lambda` / `--fair-noise-sd` - how strongly the proxies encode the protected attribute, and how noisy the fair signal is.
- `--seed` for reproducibility.

2. `reproduce-synthetic-downstream.py` - score A/B/C. It trains three downstream models and scores them all against the held-out `fair_outcome`: **Model A** on the raw (biased) data, **Model B** on the Rosa-debiased data, and **Model C** the parity-constrained "fairness by quota" baseline (Model A forced to equal approval rates per group). All three are scored against the known `fair_outcome`, and a working debiaser gives **C < A < B** on R2: the parity constraint C sacrifices the most predictive power, so Rosa's B beats both the biased A and the parity baseline at close accuracy. (The "oracle" is the `fair_outcome` you score against - not Model C.) The validator separates probability metrics (R2, plus a Brier calibration score for A and B) from threshold metrics (accuracy, group bias, and a per-group breakdown of selection rate, TPR, FPR, precision and balanced accuracy), and reports the non-linear protected-attribute recovery against its chance baseline; Model C's R2 is computed on a constructed parity-shifted score - a deliberately naive quota baseline, not a calibrated probability, so no Brier is shown for it. Three sources of randomness sit behind the numbers: Rosa's training is stochastic and exposes no seed (which is why you run ~10 independent Rosa replicates and take the mean), while the downstream-model and split seeds are fixed. Before it scores anything the validator fail-closes - it checks the debiased rows line up with the raw rows (the ignored pass-through label is compared row-by-row) and stops rather than silently mis-score if they do not.

Flow: generate the dataset, then split it into a **training part** and a **held-out part** with a fixed seed (exactly as in section 6). Run **`remove_bias_training` on the training part**, then **`remove_bias_inference` on the held-out part** with the model it produced - the generator's emitted config sets the column roles (both `outcome` and `fair_outcome` as `ignore_columns`) so they survive into the outputs. Then run the validator: Models A, B and C are all fit on the

training data and scored on the **held-out part** against `fair_outcome` - never on rows they were trained on. Because the fair answer is known, this shows not just that debiasing helped but how close it came to the known fair answer - the clearest demonstration available of the no-trade-off result, which is claimed only for runs at or above the 1,500-row floor stated below and does not extend to weak-signal data like COMPAS. It is a stronger check than the real-data test precisely because real data gives you only A versus B, while synthetic data lets you grade all three models against a known fair answer.

How much data this check needs, and why the validator may decline to give you a verdict.

> In the named synthetic benchmark, on Rosa 1.48.0 and the stated held-out A/B/C protocol, we observed 5 utility-ordering failures in 20 runs at approximately 1,200 training rows and no failures in 40 runs at 1,500 or more. We have not characterised the transition between those ranges. Accordingly, the "no fairness/accuracy trade-off" claim is made only for runs with at least 1,500 **eligible training rows**, subject to the stated acceptance test.

"Eligible training rows" means the rows that actually enter training after validation and exclusions - not the size of the file you generated, and not the row count before you split it. Note what this floor is and is not. It is a condition on **this specific utility claim on this benchmark**, not a product minimum and not a guarantee: Rosa accepts and runs jobs well below it, and at that size the engine still completes and still reduces its measured bias. What it does not do below the floor is deliver the A/B/C ordering with acceptable run-to-run consistency - at about 1,200 rows a single run is close to a coin flip on the ordering, and Model B's *mean* still beats Model A even there, but you do not run an average, you run once. Nor is 1,500 "safe": zero failures in forty runs **bounds** the failure rate (about 7.5% at 95% confidence), it does not prove failures are impossible above it.

Three consequences for how you use these tools:

- **The generator warns you before you spend a job.** If `--n-rows` would put your training split below the floor, it says so at generation time, which is the cheapest possible place to find out.
- **The validator is three-state.** Below the floor it prints **"No verdict - outside the characterised utility range"** rather than a pass or a fail, *even when the ordering comes back C < A < B* - a pass shape at that size is a coin flip that landed heads. At or above the floor it reports the actual held-out result and reminds you that one invocation is one Rosa run against a published protocol of ten. It never infers a pass from the row count in either direction.
- **It prints four per-run facts to report alongside any figure**: eligible training rows; protected-group counts; protected x outcome joint counts with the smallest cell; and whether Rosa accepted or declined. These are facts rather than a threshold on purpose - minimum cell count alone predicts neither outcome, since a balanced set with a 221-row minimum cell was accepted and failed the ordering 30% of the time while an imbalanced set with a 225-row minimum cell was declined outright. The first three are also on your Run Manifest; the joint counts are not and cannot be, because Rosa is never told which column is your outcome - your label sits anonymously inside `ignore_columns`.

Bias removal and utility preservation are independent axes, and this is the reason the acceptance test is not optional. In the runs that failed the ordering, the manifest residual read *not detected* every time, and post-debias recovery was flat across passes and failures alike. Nothing in the report or the manifest tells you which side of the utility gate a given run landed on. So for any consequential deployment, pre-register a held-out acceptance test before you run Rosa, and include a **recovery probe chosen by someone other than the party being assured** - your own choice of non-linear model, on your own held-out split. An externally chosen probe is what makes the check evidence rather than self-assessment.

And a decline is a controlled non-result, not evidence that your data are fair. If Rosa returns `declined_no_bias`, its test found nothing above chance at your sample size and estimator; that is a finding about the test, not a clean bill of health for the dataset.

Keep a trial-scale synthetic dataset within the section 3 caps. A synthetic check at production scale and width uses the same two tools with a dedicated instance's higher limits. Note: these two tools are for datasets **you generate**. The portal's own **Test 1** and **COMPAS** worked examples (on the **Data Validation** page) each have their *own* reproduction script - use those, not `reproduce-synthetic-downstream.py`, for the portal's published figures.

Reading the COMPAS worked example. When you run the COMPAS example, look at the full metric panel, not just overall accuracy. On this dataset the fairness gaps shrink sharply - the between-race false-positive-rate gap drops by about 70% on average, and a single run varies widely - while the model's overall ranking power (balanced accuracy, ROC AUC) falls toward the majority-class baseline (about a 14-point overall-accuracy cost), because on weak-signal data the protected attribute carries much of the model's predictive power, so removing it costs ranking power. One gap widens rather than shrinks: between-race precision. That is expected - Rosa equalises selection and false-positive rates, and precision parity is a separate criterion that generally cannot be satisfied at the same time. This whole pattern is specific to weak-signal data like COMPAS, and is not what happens on strong-signal data like the Test 1 credit experiment, where the debiased model matches or beats the original. Measured on Rosa v1.59.0 (image `sha256:99ab92b8`), 82 independent runs, range 30.0-99.9%.

The Test 1 feature set. The Test 1 credit benchmark reports a downstream R-squared of about 0.670 for the Rosa-debiased model on a defined feature set. That set excludes two columns that are present in the data - ethnicity and age band - because they are themselves raw protected characteristics, not the target of this benchmark, which debiases gender only. The headline model is built from legitimate credit features plus the engineered gender proxies, and deliberately does not consume two further protected attributes that Rosa was not asked to remove. Using the fuller "every feature except the protected attribute and the label" rule folds those two columns back in and gives an even stronger result - a downstream R-squared of about 0.729. We report the narrower set as the headline because it is the more conservative benchmark, and disclose the stronger all-feature result alongside so the choice of feature set is explicit. Measured on Rosa v1.59.0 (image `sha256:99ab92b8`), 10 independent runs, range 0.631-0.715.

8. A recipe for any dataset

1. Check the shape: 1,000 to 50,000 rows, at most 100 columns.
2. Choose the protected attribute (one low-cardinality categorical; band it first if it is continuous). This is `bias_columns`.
3. Choose the outcome label and put it in `ignore_columns`, along with weights and free text. Strip or pseudonymise direct identifiers before you submit rather than relying on `ignore_columns`, which controls how a column is modelled, not whether it is uploaded.
4. List every categorical column, including the bias column and the label if categorical. This is `cat_columns`.
5. Estimate the post-encoding width (sum of distinct values in `cat_columns`, plus the count of numeric columns); keep it at most 150.
6. Pre-check recoverability (section 6): can a simple model predict the protected attribute from the other columns above chance? If not, expect Rosa to decline, and record that.
7. Run `diagnose` (to see the bias and the proxies), then `remove_bias_training` (before and after bias, plus debiased train data), then `remove_bias_inference` on your held-out test part (debiased test data).
8. Run the Model A versus Model B check in section 6.
9. For the strongest check - a comparison against a known fair answer (Model A/B/C) - run a synthetic validation as well (section 7).

If a dataset has no sensitive attribute, no outcome to predict, or is just a lookup table, it is not a meaningful debiasing case. And if the protected attribute is present but Rosa's test cannot recover it above chance, Rosa will correctly decline; that is a finding, not a fault.

9. A good summary to keep

For each dataset, record: the config you used and why; whether the protected attribute was recoverable; Rosa's reported bias before and after, with the top few proxy features before versus after; Model A versus Model B accuracy and the three fairness gaps, measured held out; the synthetic A/B/C result if you ran one; and any advisories or anomalies. This is a clean, defensible record of what you tested and what you found, and it doubles as evidence for your own governance and audit needs.

10. Good practice

- Always evaluate on held-out data. A fairness number measured on the same rows a model was trained on is not meaningful.
- Rosa preserves each column's distribution and debiases the dataset as a whole, so do not expect an individual row's values to be reproducible; do expect the distribution and the fairness property to hold.
- A "no bias detected" decline is correct behaviour when Rosa's test cannot recover the protected attribute above chance. Treat it as a valid result, and pre-check recoverability before spending jobs.
- The trial allows 20 jobs per month. A full evaluation of one dataset is about three jobs: diagnose, training, and inference.
- If you cancel a job, a follow-up call that races the cancellation (an immediate status check or a second cancel) can briefly surface a transport error while the engine is being stopped. Treat a just-cancelled job as already terminal rather than retrying.
- Your portal's FAQ covers more edge cases than this guide - check it for anything unusual in your data or your results.
- If a run behaves unexpectedly, read the Run Manifest and the job's `error` field before drawing a conclusion. The manifest is the record of what actually happened.