

S-09, Anonymized sample from a real engagement

Response to the client's feedback document, after sprint 16

Point by point written answers to a feedback document with 17 points, with the hours for each option.

Anonymized sample from a real sprint for a US investment analytics platform, 2026

What this is. The client sent a feedback document with 17 numbered points after sprint 16. Some were bugs, some were questions and some were requests for estimates. I answered every point in writing the same week. Each question is restated in neutral words, and my answer follows as sent. Fund tickers, issuers, data vendors, people and amounts are replaced. Points 6, 10, 13 and 15 are left out for length. They were an acknowledgement, a note, a thank you and an ad network setup. The hours summary at the end is complete.

Good afternoon [Client],

Thank you for the document. Each of your questions is restated below, with my answer underneath. Every number comes from production, the deployed code or the issuers' own files, all pulled today.

Sprint 16 feedback

Point 1

Can the return cards show a 3 month return?

Yes, about 7 hours. Three months is the close 63 trading days back, measured the same way the card measures one week and one month today. It touches the two return tables with a migration, the calculation, the API, the card tile, the detail table and its CSV, the email and PDF layout, and the tests. Only the newest row gets a 3M value, and older dated rows stay empty.

Checking this turned up a fault in the one week and one month figures. Each is one session short, so the week covers 4 sessions and the month 20, where it should be 5 and 21. On Fund X today the card shows -1.34% for the week against -1.43% for a true 5 session week. For the month it shows -4.11% against -5.21%. The fix is about 1.5 hours, and I'd do it with 3M since both use the same function. It changes figures you already see, so I want your go ahead first.

Point 2

Can the individual stock rows match the return cards at the index level?

Yes, about 11 hours including 3M on the stock rows, assuming 3M on the fund cards is built first. Daily, week, month, YTD and 1Y already use the same formulas and the same YTD anchor. Three gaps remain.

- The fund cards carry 2Y, 5Y and 10Y annualised returns that the stock rows lack. Loading ten years of closes efficiently is the biggest piece, about 3 hours, since the table loads only 400 days today.
- The fund cards show total return because the price library hands the calculation dividend adjusted closes. The stock rows show price return from unadjusted closes, so the two never match exactly over a year or more. I'd label the stock table as price return. Storing dividend adjusted closes for every stock is possible, at 8 to 10 hours plus a full backfill.
- Format differences, N/A against a dash, and the total return note on one table but not the other.

Point 3

The YTD and 1Y returns look slightly off, maybe a day delayed. Can you check Fund X for [date]?

The card is correct and it isn't a day behind. Fund X reads 13.6558% YTD for that day. That's the total return with dividends reinvested, from the last close of the prior year to the close of [date]. The price only YTD is 12.89%, and the 0.77 point gap is the fund's three dividends this year. A card one day behind would show -0.36% daily and 14.11% YTD.

1Y follows the same rule, 16.02% total return against 14.85% price only. The pattern holds on every card I checked. Fund W shows 12.71% total against 11.84% price, and Fund V shows 20.83% against 20.42%. Each gap is the fund's yield. Most public quote pages show price only YTD, which is the likely source of the difference you saw. The recent decline is already in the figure. Fund X's YTD fell from 15.32% five days earlier to 13.66% on [date].

I'd make it unambiguous by labelling every card figure as total return and fixing the dividend adjustment in code, so a library update can't flip the method silently. That's about 1 hour, or 3 to 4 hours to show price return beside total return. The week and month fix in

1) is separate.

Point 4

How much extra work is it to add market cap as a field?

About 16 hours for the stocks in the drill down. No table stores market cap or shares outstanding today, apart from one exchange file, which carries it for its own names only. Issuer files give the fund's position, not the company's size. I tested the free sources today. One batch quote endpoint returned market cap for 48 of 50 symbols in 1.5 seconds, so about 7,000 names take roughly 4 minutes a night, against about 2 hours one ticker at a time.

Two traps here. That endpoint reports London market caps 100 times too large because London prices in pence. Market cap also arrives in local currency, so sorting needs a conversion to one currency. The batch request relies on an internal part of the library that an upgrade can break, so I'd build a one ticker fallback.

Part	Hours
Nightly fetch with fallback	4.5
Model, migration and schedule	2.5
API, column, sorting and CSV	3
Currency conversion	1.5
Missing and stale values	1
Tests	2.5
QA	1

A fund has no market cap. A fund level figure would be total assets, 1.5 hours more.

Point 5

Fund V's drill down is producing errors.

The issuer blocks automated requests from data centre addresses. Production holds 0 rows for this fund, and the page shows "Holdings request failed with HTTP 404" with a Retry button. I tested from a data centre address today and got the issuer's HTTP 406 refusal, so it's the issuer's policy, not a quirk of the cloud provider. No other US issuer runs a fund on the same index, and the index provider's own list has no weights. Weights derived from market cap miss by up to 6 points on single names.

The source that works is a second issuer's fund that tracks the same index from another market. It's daily and one trading day behind, with 101 stocks, weights summing to 99.93%, and the same 101 names as the index provider's list. It takes about 4 hours, and the page will state where the weights come from. The quarterly regulatory filing for Fund V is the fallback, official but 3 to 5 months old.

Point 7

What is required to raise the drill down from 600 to 1,800 rows?

The display limit is the easy part, a constant in three places plus a test. Pricing is the real work. Six funds hold more than 600 names, the largest 5,035. Showing 1,800 rows in all six needs 6,897 distinct priced names, 4,137 more than my measured count of the planned nightly list. At the production rate of 0.21 seconds a name, 0.27 at worst, that adds 14.5 to 18.3 minutes a night, inside the 60 minute window. It takes the list to about 8,700 names, though, over the 8,000 cap planned for sprint 17.

Part	Hours
Cap to 10,000	0.5
The three 600 limits and their test	1
Database change so the six funds price up to 1,800 names	1
Reading only the closes the returns need, about 45 times fewer rows per page load	3
Paging in the table and compressed responses, since the payload grows from 190 KB to about 570 KB	2.5
Database disk from 10 GB to 20 GB, since the extra history adds about 1.45 GB	0.5
Deploy, a one off history load of 1 to 2 hours, and checks	1.5

That's about 10 hours, after the sprint 17 price work is live. A bare 4 hour version skips the speed and paging work, but the page gets heavy on a phone. Since the foreign funds are parked, Fund X alone is the cheaper path. It adds at most 800 names, fits the planned cap and needs the same code changes.

Point 8

For a sector fund, can the platform return all holdings for a given day, and will a daily fetch pick up adds and drops?

Yes to both. Each day is stored as a full snapshot keyed by fund, date and ticker, and nothing is ever deleted. One sector fund has 124 daily snapshots in production with no gaps and 80 to 84 rows a day. The pipeline runs every day including weekends, so a Saturday row repeats Friday's file.

An add or drop shows up as a ticker present one day and absent the next. Across those 124 days the fund had 5 real changes, two from spin offs and three index replacements. The last one entered the index on a Monday and appeared in the data on the Tuesday.

Three things are worth tightening.

- The stored date is the scrape date, not the file's own as of date. Storing both is about 2 hours.
- The holdings endpoint returns nothing for a date whose scrape failed, while the lists endpoint falls back to the latest earlier date. Matching them is about 1 hour.
- A changes endpoint that lists adds and drops directly is 3 to 4 hours.

Storage runs about 2.4 GB a year at today's 16,880 list rows a day, affordable for years.

Point 9

If historic membership data is bought later, can it replace the stored history? Some analysis might give different results with different member lists.

Yes, and your instinct is right. The current history is survivorship biased in a way that changes results. The platform rebuilds past index membership from today's names plus a public change log, so a company removed in the past never comes back. As of the start of 2015 it returns 313 names where the index held 504, and as of 2010 it returns 264 against 509.

Purchased data loads as its own list, backfilled only for dates before the first scrape, and it never touches the scraped rows after that. Storing a snapshot only on the dates membership changes keeps it to about 325,000 rows, and the existing as of lookup reads it with no API change.

The real cost is prices for removed companies. The current price source has none for most delisted stocks, and about 356 names have left the index since 2010.

- One vendor sells membership and weights, with no prices, so a backtest stays biased.
- A second vendor includes delisted prices and historical members back to 1990. Its licence is personal use only, so a business needs a commercial agreement.
- An importer with ticker mapping and delisted prices is 30 to 40 hours.
- A free middle step rebuilds a corrected point in time list from the public change table, 6 to 8 hours. It's reliable from about 2010, where the log carries 20 to 30 changes a year against 2 to 13 before.

Point 11

Why read index members from a public list? Is the fund issuer's own file easier?

Both are already scraped, and today they agree exactly, 503 against 503 with the same spellings. They answer different questions. The issuer file is the fund's actual portfolio, with weights, shares and security identifiers. It runs one to two trading days behind index changes, and its sector column is empty on all 503 rows. The public list carries the sector and sub industry, the date added and the change history, but volunteers edit it and it can change before an event takes effect.

I'd make the issuer file the source of truth for membership and weights and keep the public list for classification and the change log. A daily comparison would alert when the two disagree for more than 2 days. That's 9 to 11 hours, 4 to 6 for the switch, 3 for the comparison, and 1 to 2 to keep the sector columns the parser discards today.

Point 12

How much effort is it to split the exchange list into the main board and the junior board, without depositary receipts? If it's more than a few hours, go with your suggestion.

It's a few hours, well inside your limit, because the scraper already supports all three. Depositary receipts carry a flag on every row, so excluding them is one setting. The main board without receipts is 669 rows from 654 companies, down from 802. The junior board has 1,498 rows and no depositary receipts. 153 of those are shell companies with no business yet. Filtering them is 1 hour.

The main board alone without receipts is about 1.5 to 2 hours. Both boards as two separate lists, both without receipts, is 4 to 6 hours, or 6 with the shell filter. The junior board adds about 1,345 names to the nightly price fetch, about 5 minutes a night. I'll confirm the count on production before the cap changes. My recommendation is two lists, no receipts and no shells, about 6 hours.

Sprint 17 data pipeline

Point 14

When I have an API key for a new data source, what is the best way to share it so the site reads it securely?

Never by chat or email. Today every key reaches production as a plain environment variable. The task definitions hold 22 variables and none is stored as a secret, the database password, the Django key and a cloud access key included. The right home is the cloud provider's secrets store. I create one empty secret for data keys, and you sign in to the console, open it and paste the key in yourself, so the value never passes through anyone else. Each task reads it at start up and each script reads it from its environment. Rotating a key is an edit in the console and a redeploy, with no code change. If the console is awkward, a one time link that expires after a single view works, and I paste it in.

My cloud user has no access to the secrets store today, so that permission has to be granted first. Moving the existing secrets is 6 to 8 hours including rotating them, and each new key after that takes 0.5 to 1 hour. This ties back to the sandbox. Until scripts run in their own task, every script can read every key, so the sandbox should land before the new API keys do.

Point 16

How long would a function take that changes dates and reruns individual scripts?

About 20 hours for the platform, plus conversion per script. Today no date is passed into a script. Each script picks its own dates, mostly through `datetime.now()`, and every run overwrites the previous chart and table. The function adds these parts.

Part	Hours
An as of date and an optional start date passed into each run	2
Data helpers that never return data after that date, so a rerun can't see the future	2
A shim that makes <code>datetime.now()</code> and <code>date.today()</code> return the as of date, so most scripts need no edit	3
A run record per script and date, so the daily chart stays untouched	4
A staff only rerun endpoint that rejects future dates	2
A date picker with a Rerun button and run history	4
Tests	3

Conversion is separate. A 2 hour scan of all 345 scripts gives exact counts. I'd convert only the scripts you want to rerun. Built together with the sandbox, the date passes in as a task setting and the shim gets simpler, which saves 3 to 4 hours.

Point 17

Before sprint 17 starts, can you explain the sandbox for uploaded scripts?

Uploaded scripts run sealed off from the platform, about 12 hours. Today the platform reads an uploaded file and runs it with Python's `exec()`, inside a thread of the same Django process that serves the site, or inside the nightly pipeline task. Nothing separates the script from the platform. It inherits the process's environment, a database connection as the master database user, and the container's cloud permissions. 345 scripts are active. Yesterday they made 429 run attempts counting retries, and 173 of those raised an exception. None of them has a working time limit.

In practice any script, buggy or hostile, can read the database password and every user's email and password hash. It can also drop tables, read the Django signing key and forge an admin login, send email from your domain, and read or delete files in storage. A runaway loop can fill the web container's 3 GB of memory and take the site down.

One finding is more urgent than the sandbox itself. Every container carries a permanent cloud access key with full rights across identity, storage, database, compute and mail. A script that reads its environment holds the keys to the whole account. I'd remove that key from the containers and retire it before anything else, about 1 hour.

Here's the sealed design. Each run starts its own small container task, 0.5 vCPU and 1 GB, with a hard 300 second limit. The task gets no cloud role and no keys. The platform hands it short lived signed links, one to read the script's code and one each to write its chart, table and result file, so a script can write only its own output. For market data it connects as a new read only database user that can read the 9 market data tables and nothing else, with a 60 second query limit. The network allows the database and outbound HTTPS only. The platform then reads the result, checks it and saves the chart and table itself, so a script never writes to the platform's database.

Outbound HTTPS stays open in this first step because 98 of the 228 scripts I could measure over 48 hours reach outside sites. Cutting the internet outright would break 43% of them. A domain allowlist comes next, once your data pipeline moves those feeds into the database.

Nothing changes for your analyst. The same data helpers work, and `plt.savefig()` and `to_csv()` still produce the chart and table. What stops working is reading environment variables, importing Django models and writing to the database, and no script should do those anyway.

Part	Hours
Terraform for the task definition, a separate execution role, the security group and the log group	2
The read only database user and its grants	1
The runner image with the same library versions and the time limit	3
The platform side behind an on and off switch, covering the Run button, reports and the nightly run	2.5
A shadow run of all 345 scripts compared against today's results	2
Rollout, rollback by switching it off, and a note to your analyst	1.5

The shadow run is the step that protects you. 124 of the 228 scripts show no recorded data source, so that's where breakage will appear, and I see it before anything changes for users.

Hours at a glance

Item	Hours
Cloud key removal and retirement	1
Script sandbox	12
Week and month fix	1.5
Total return label	1
3M on the cards	7
Stock rows matched to the cards	11
Market cap	16
Fund V holdings from a second issuer	4
1,800 rows	10, or less for Fund X alone
Sector fund as of date, fallback and changes endpoint	6 to 7
Issuer file as source of truth with a cross check	9 to 11
Point in time index history	6 to 8 free, or 30 to 40 with purchased data
Exchange list as two boards without receipts or shells	6
Secrets store move	6 to 8, then 0.5 to 1 a key
Ad network, my side	8 to 10
Date change and rerun	about 20 plus conversion, 3 to 4 less if built with the sandbox

My order would be the cloud key first, then the sandbox, then the secrets store, since the new API keys depend on both. Tell me which of the rest go into sprint 17 and I'll size the sprint around them.

Michael

Axdale, Michael Lip. Warsaw, Poland (CET). michael@axdale.com. www.axdale.com. Code record github.com/theluckystrike. Names, tickers and amounts are replaced.