Fuelling an Oil & Gas Giant

A diesel, haulage and SARS (South African Revenue Service) refund data story — 23.7 million rows from Kalahari Petroleum's upstream fleet-operations ERP (Enterprise Resource Planning system), modelled as a SQL Server star schema, shipped through Azure Data Factory, and served to Qlik Sense.

View source on GitHub →

Case-study note: Kalahari Petroleum is a fictional company invented for this portfolio piece. The underlying operational data is real (anonymised) mining/haulage fleet-fuel ERP data, presented here under a fictional oil & gas identity to demonstrate the data model and analytics pipeline without naming the source organisation.

290.6M L	**325 504**	**778 254**
fuel issued 2009–2022	AFS fuel transactions	equipment trips
R 385.2m	**23.7M**	
modelled diesel refunds	rows through the pipeline	

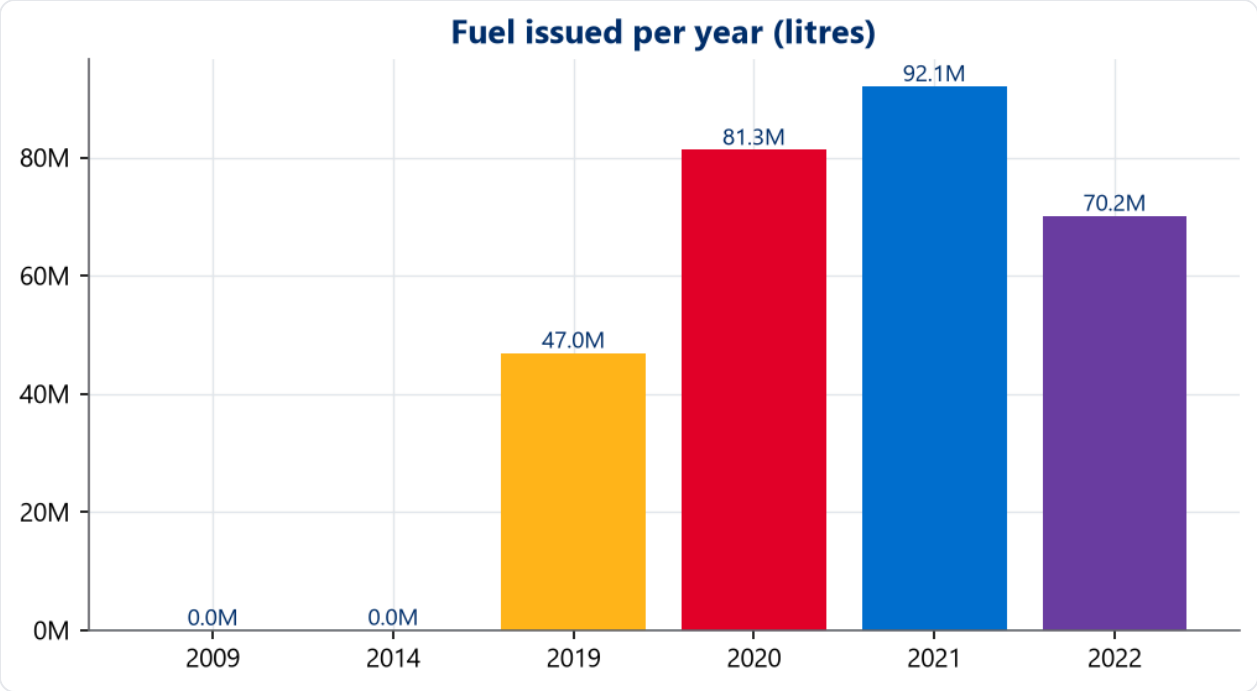
1 · More than fuel records

The source system is an operational ERP for Kalahari Petroleum's upstream fleet: an Automated Fuel System (AFS) issuing diesel to haul trucks and drill rigs, tank deliveries, odometer and hour-meter readings, haulage trips with material types, GPS trip traces over a 21.9-million-point coordinate grid, geofences, SAP integration staging and cost-centre accounting. Fuel is the connective tissue — every litre issued ties equipment, location, activity and money together. Kalahari Petroleum's fleet-management system classifies vehicles the way a mine would — drill rigs, haul trucks, waste removal — because well-pad

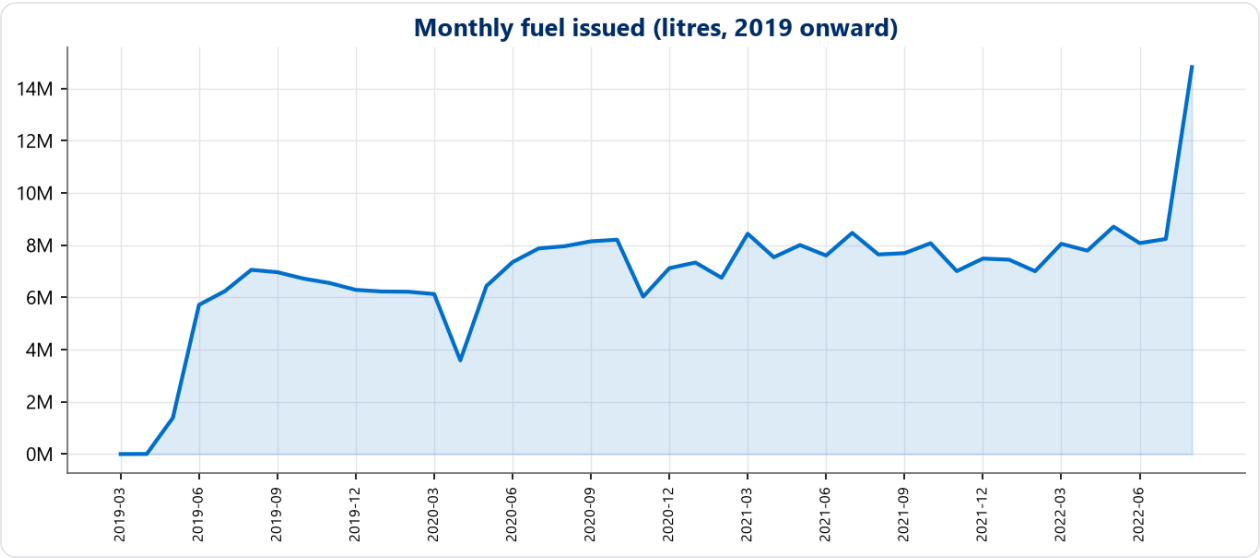
construction and haulage logistics on an oil & gas site mirror mining operations almost exactly.

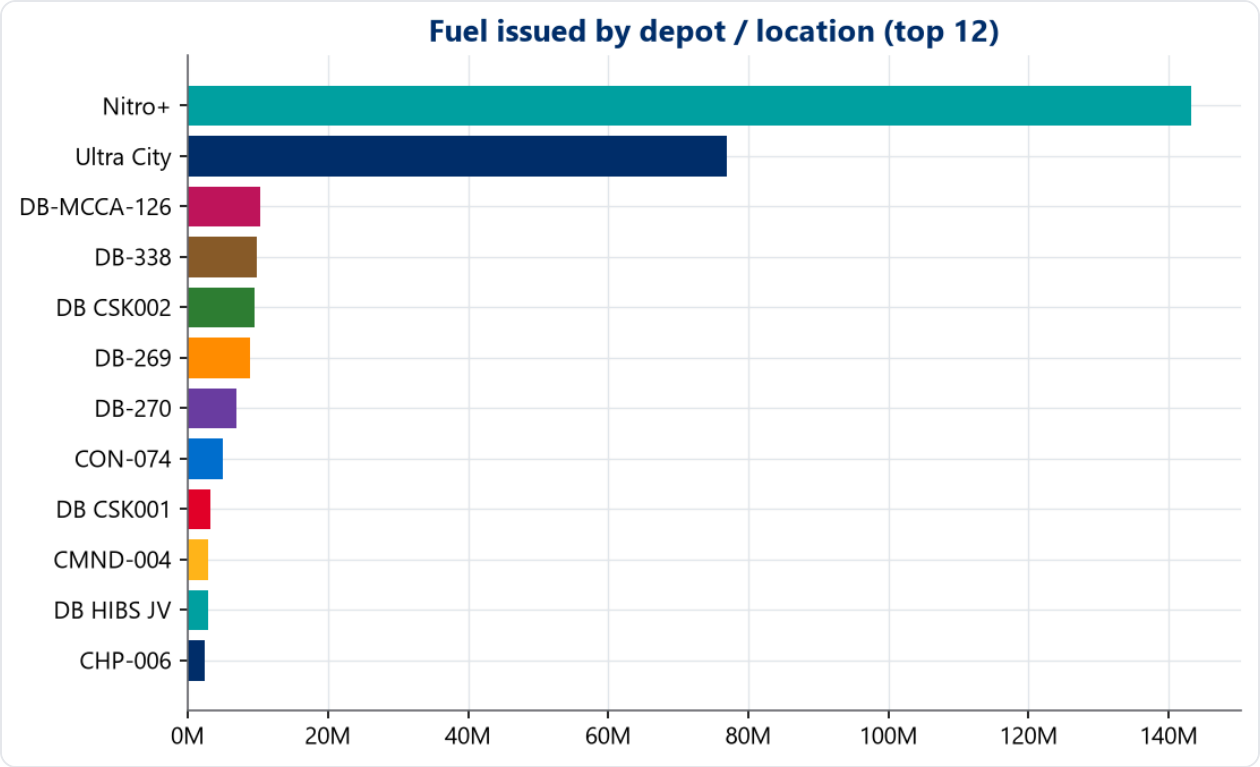
This project models that data as a Kimball star schema (8 dimensions, 8 facts) with one high-stakes business calculation at its centre: the **SARS diesel refund** under Rebate Item 670.04 of the Customs & Excise Act, where classifying litres as eligible or non-eligible is worth millions of rand per month.

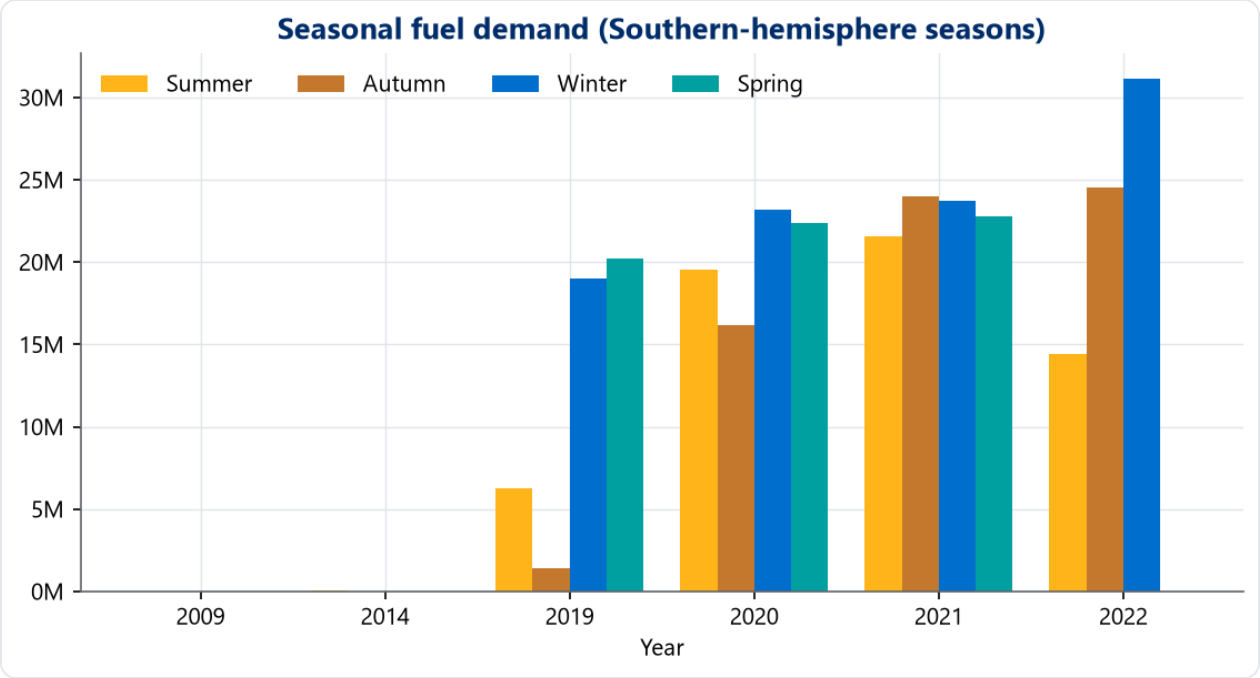
2 · The fuel story



Fuel issue volumes step up sharply from 2019 as the AFS rollout reaches full coverage, settling around 70–80 million litres a year across the operation.

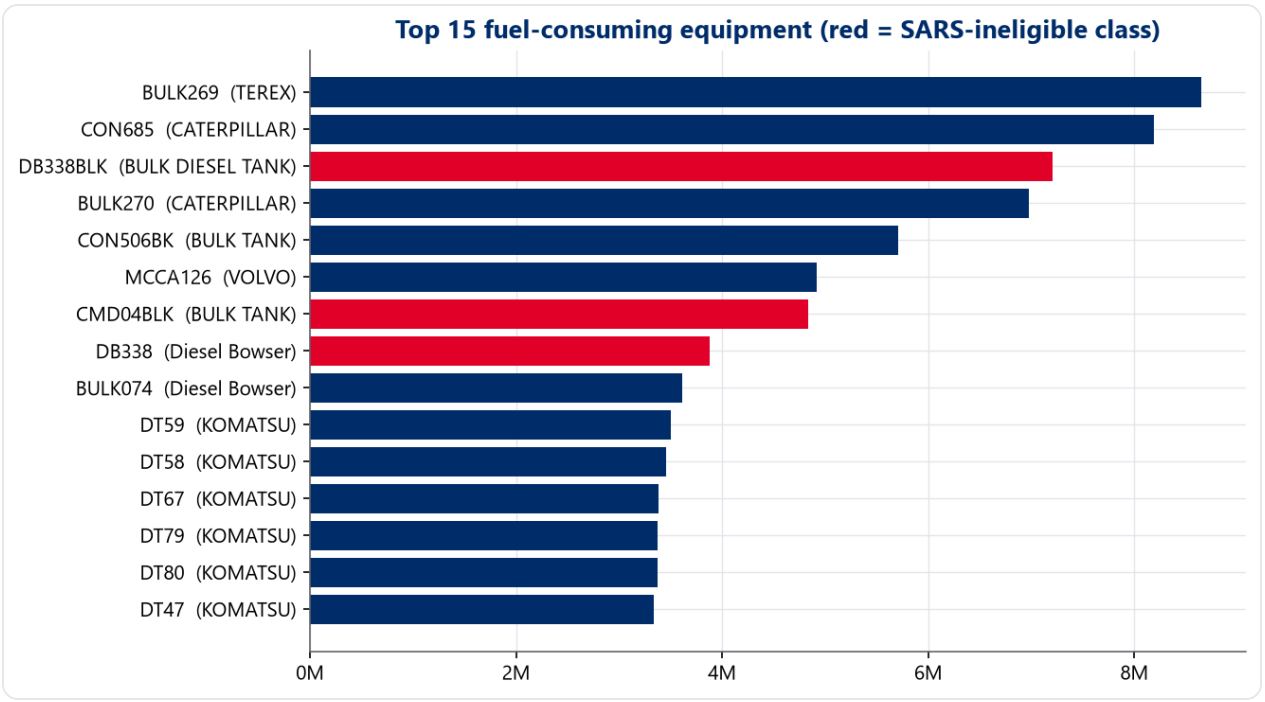

Monthly fuel issued (litres, 2019 onward)
14M
12M
10M
8M
6M
4M
2M
0M
2019-03
2019-06
2019-09
2019-12
2020-03
2020-06
2020-09
2020-12
2021-03
2021-06
2021-09
2021-12
2022-03
2022-06

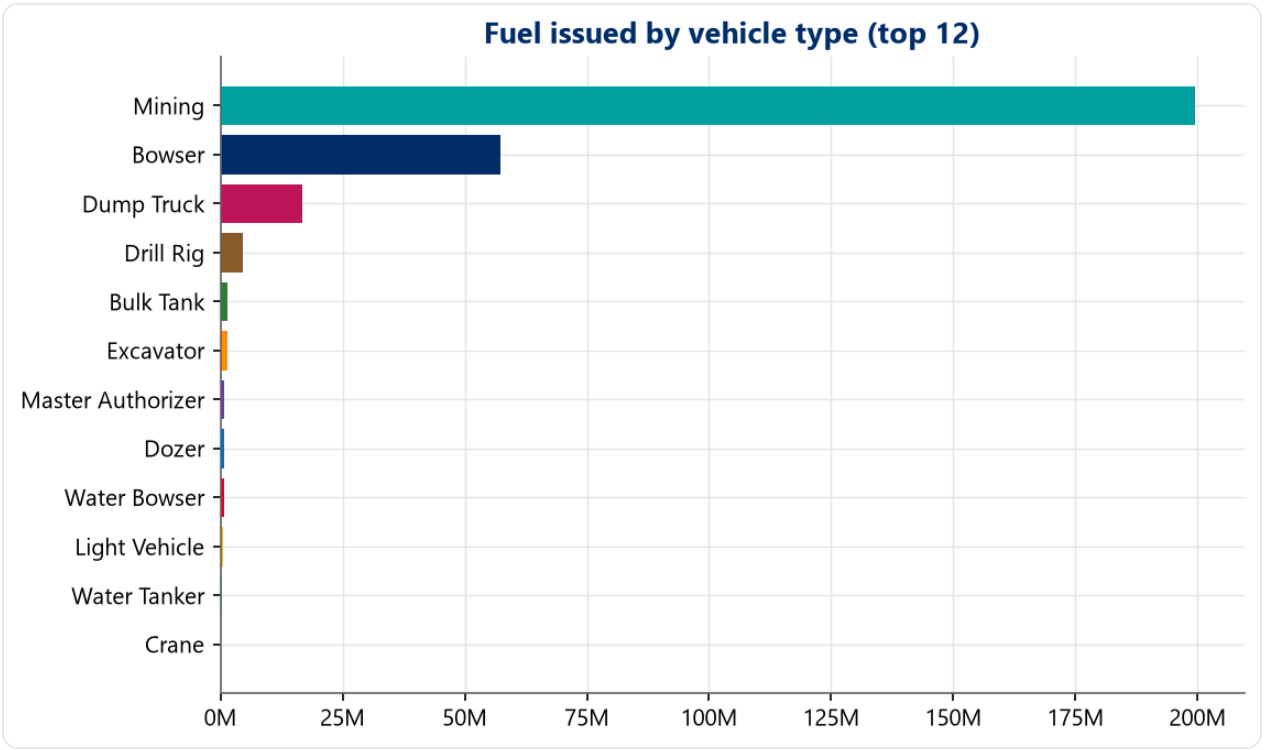

Fuel issued by depot / location (top 12)
Nitro+
Ultra City
DB-MCCA-126
DB-338
DB CSK002
DB-269
DB-270
CON-074
DB CSK001
CMND-004
DB HIBS JV
CHP-006
0M
20M
40M
60M
80M
100M
120M
140M

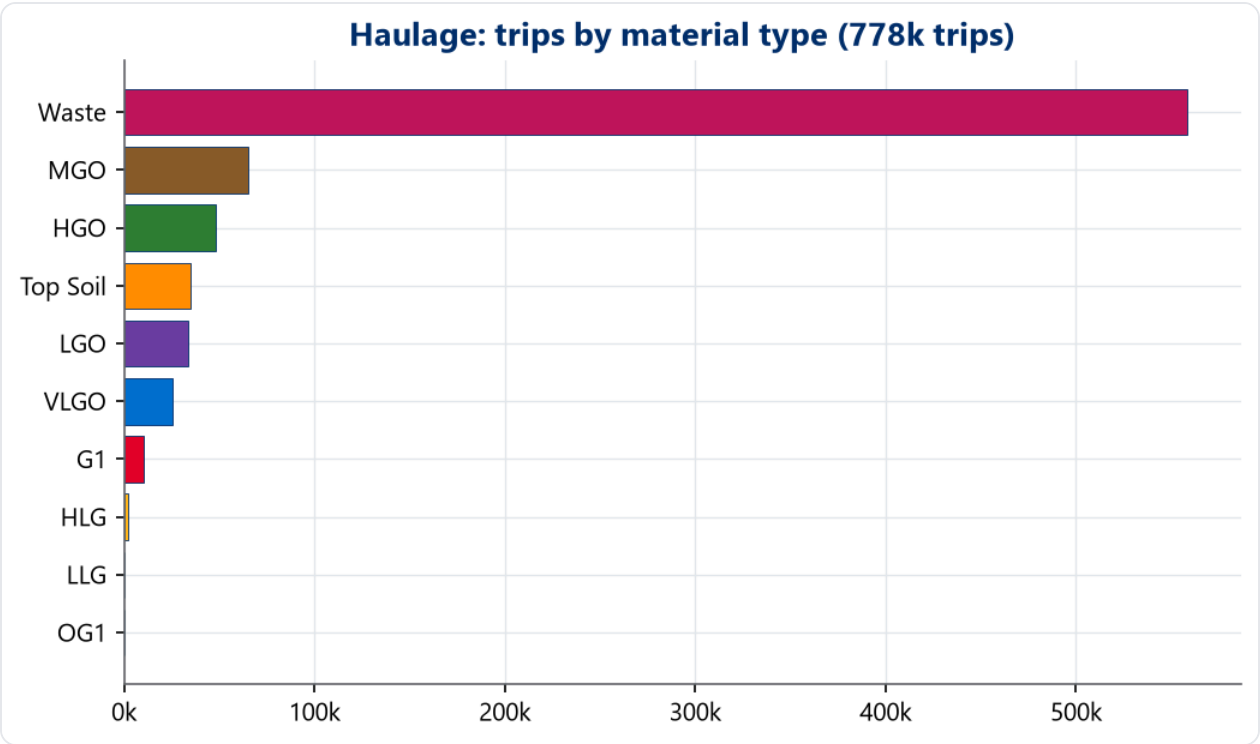


Demand is not smooth: maintenance shutdowns, production cycles and season shape the curve — one reason a refund forecast needs more than a flat average.

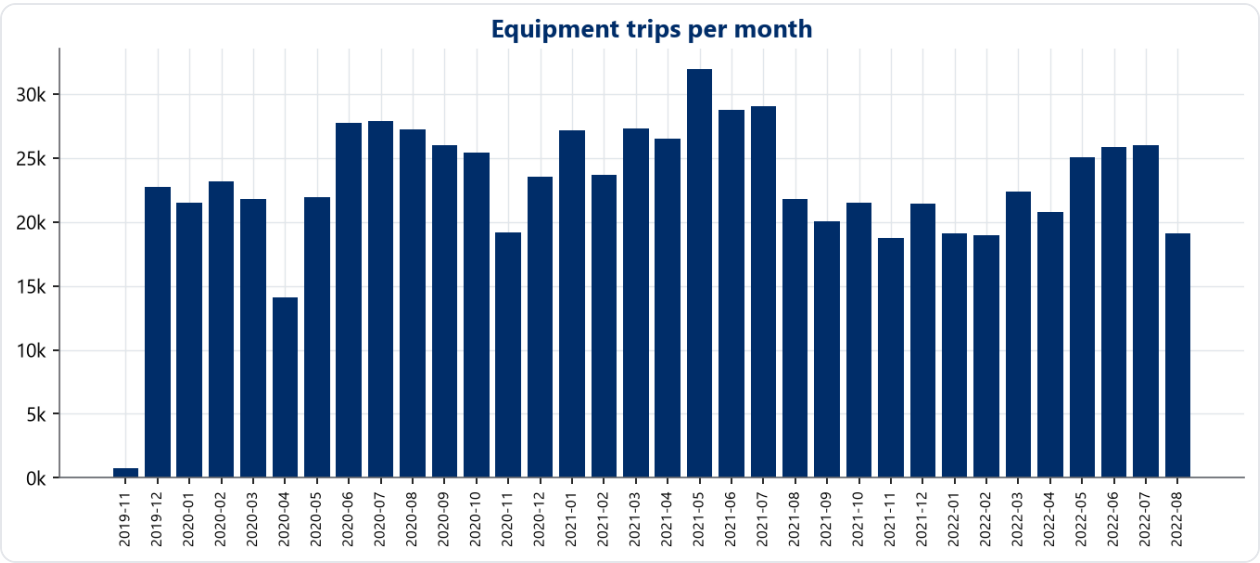
3 · Fleet & haulage


Top 15 fuel-consuming equipment (red = SARS-ineligible class)
BULK269 (TEREX)
CON685 (CATERPILLAR)
DB338BLK (BULK DIESEL TANK)
BULK270 (CATERPILLAR)
CON506BK (BULK TANK)
MCCA126 (VOLVO)
CMD04BLK (BULK TANK)
DB338 (Diesel Bowser)
BULK074 (Diesel Bowser)
DT59 (KOMATSU)
DT58 (KOMATSU)
DT67 (KOMATSU)
DT79 (KOMATSU)
DT80 (KOMATSU)
DT47 (KOMATSU)
0M
2M
4M
6M
8M


Fuel issued by vehicle type (top 12)
Mining
Bowser
Dump Truck
Drill Rig
Bulk Tank
Excavator
Master Authorizer
Dozer
Water Bowser
Light Vehicle
Water Tanker
Crane
0M
25M
50M
75M
100M
125M
150M
175M
200M



Trip records carry the material being moved — extracted product, waste, consumables — which is exactly the evidence SARS eligibility classification leans on: transport of extracted material on the production site is claimable, general road use is not.

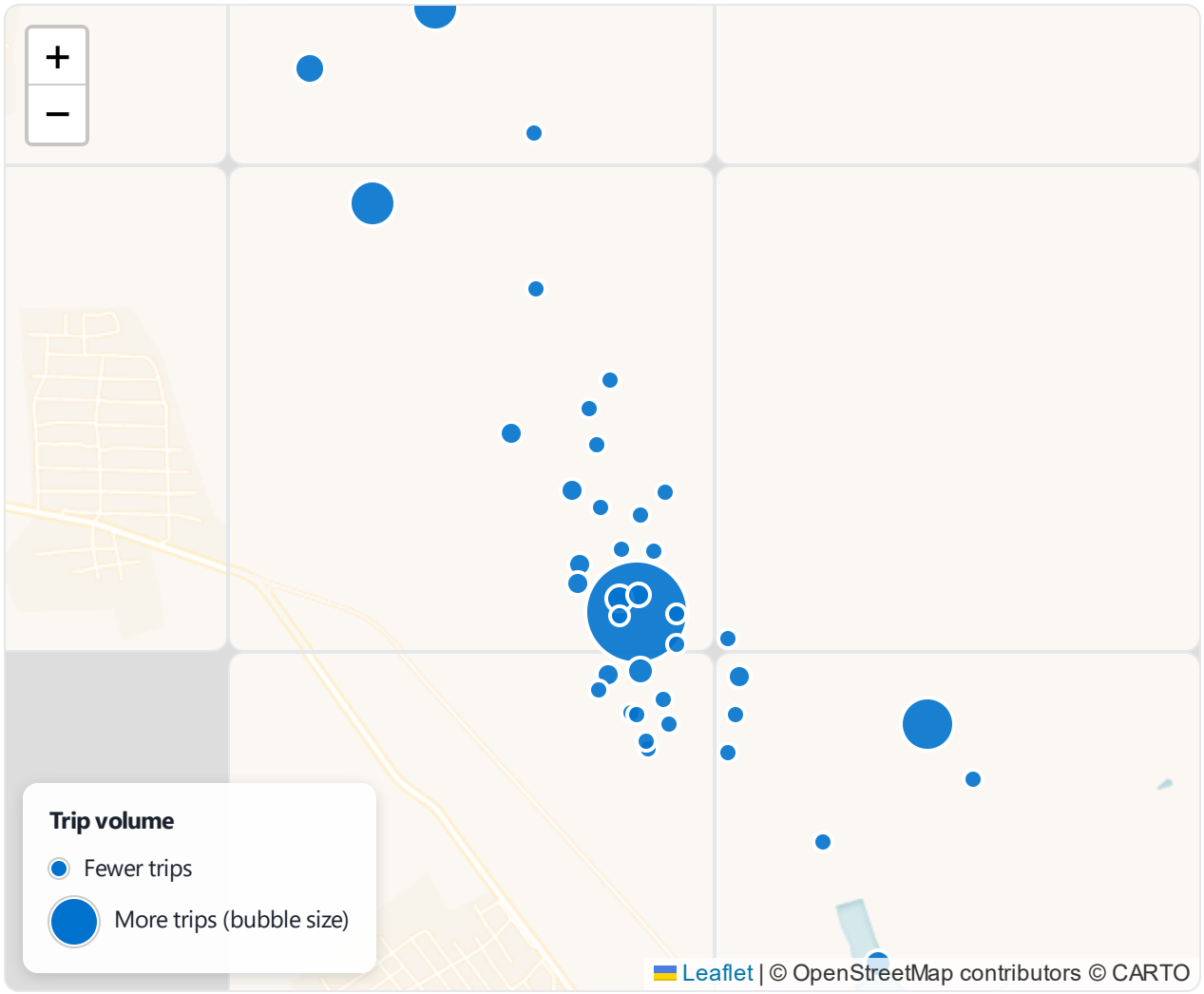


4 · Haulage & site activity map

Real GPS trip data plotted against a label-free basemap — three toggleable layers built for three different audiences: **dispatch** (where the traffic actually is), **production** (what's moving out of each site), and **compliance** (which activity supports a SARS eligible-activity claim). The top 40 haul routes overlay shows road-maintenance priority by traffic volume.

Trip Volume | Material Type | SARS Eligibility

☐ Show top 40 haul routes

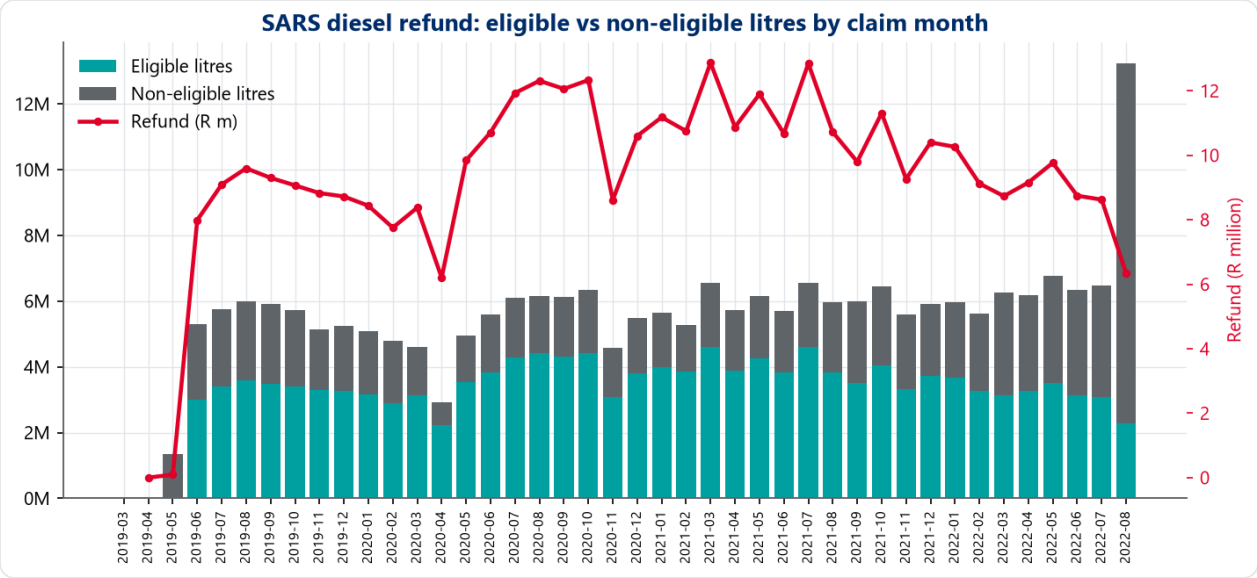


45 haulage sites · 694,217 trips · real GPS coordinates (site/company identity kept fictional — see case-study note — so the basemap below is intentionally label-free: no place names, only terrain/road geometry for genuine ops/dispatch use). Hover a site for a quick summary, click for full detail. **Data-quality note:** the source system's Lat/Long columns are transposed (the "Lat" column holds longitude values and vice versa) — corrected here. Material-type codes (HGO/MGO/LGO/VLGO) are the raw source values — no lookup table defines them in the source database, so they're shown as-is rather than guessed at. Map requires an internet connection to load street tiles.

5 · The SARS diesel refund

For on-land primary producers (mining included), the refund formula from the policy evidence is:

```
eligible_litres    = total_litres − non_eligible_litres
qualifying_litres  = eligible_litres × 80%
refund_rand        = qualifying_litres × refund_rate (c/L) ÷ 100
```



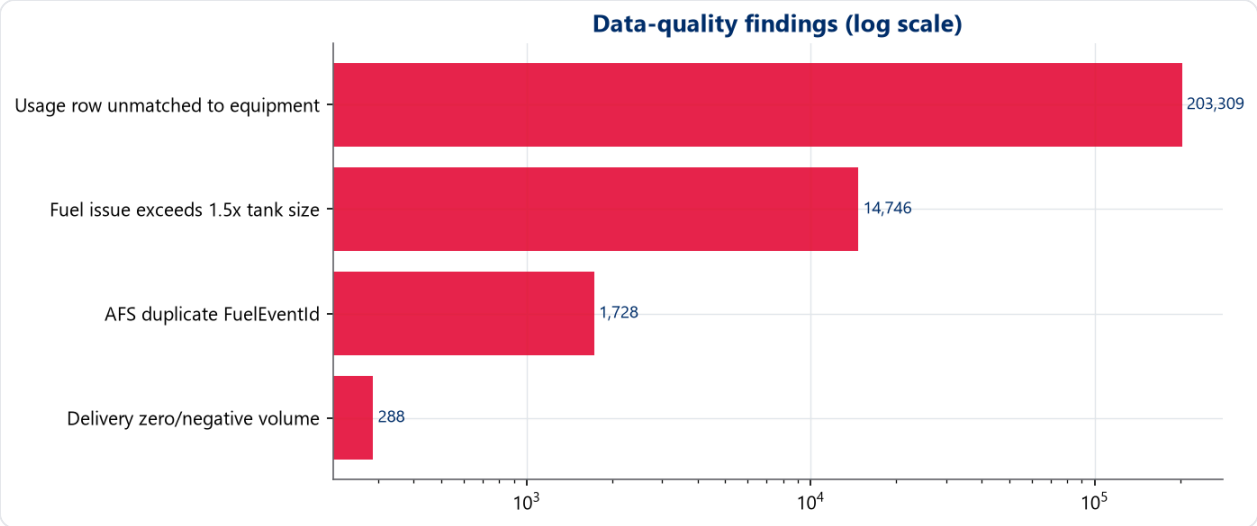
Last 12 claim months

Claim month	Total L	Non-eligible L	Eligible L	Qualifying L	Refund
2021-09	5 982 077	2 471 924	3 510 153	2 808 122	R 9 800 347.18
2021-10	6 455 776	2 405 818	4 049 958	3 239 966	R 11 307 482.74
2021-11	5 604 740	2 285 870	3 318 870	2 655 096	R 9 266 285.04
2021-12	5 922 270	2 197 315	3 724 955	2 979 964	R 10 400 075.48
2022-01	5 972 874	2 297 767	3 675 107	2 940 086	R 10 260 899.86
2022-02	5 612 826	2 345 327	3 267 500	2 614 000	R 9 122 859.44
2022-03	6 272 003	3 143 907	3 128 096	2 502 477	R 8 733 643.19
2022-04	6 193 891	2 914 930	3 278 961	2 623 169	R 9 154 858.83
2022-05	6 778 019	3 279 961	3 498 059	2 798 447	R 9 766 579.61
2022-06	6 344 588	3 212 236	3 132 352	2 505 881	R 8 745 525.95

2022-07	6 473 722	3 384 942	3 088 781	2 471 025	R 8 623 875.99
2022-08	13 224 056	10 952 680	2 271 376	1 817 101	R 6 341 680.95

> **Caution:** rates are the 2020 SARS policy examples found in the project's evidence pack (349 c/L on-land after 1 Apr 2020). They are implementation evidence, not tax advice — update `dw.DimRefundRate` with current rates before relying on any figure.

6 · Data quality — where the money leaks



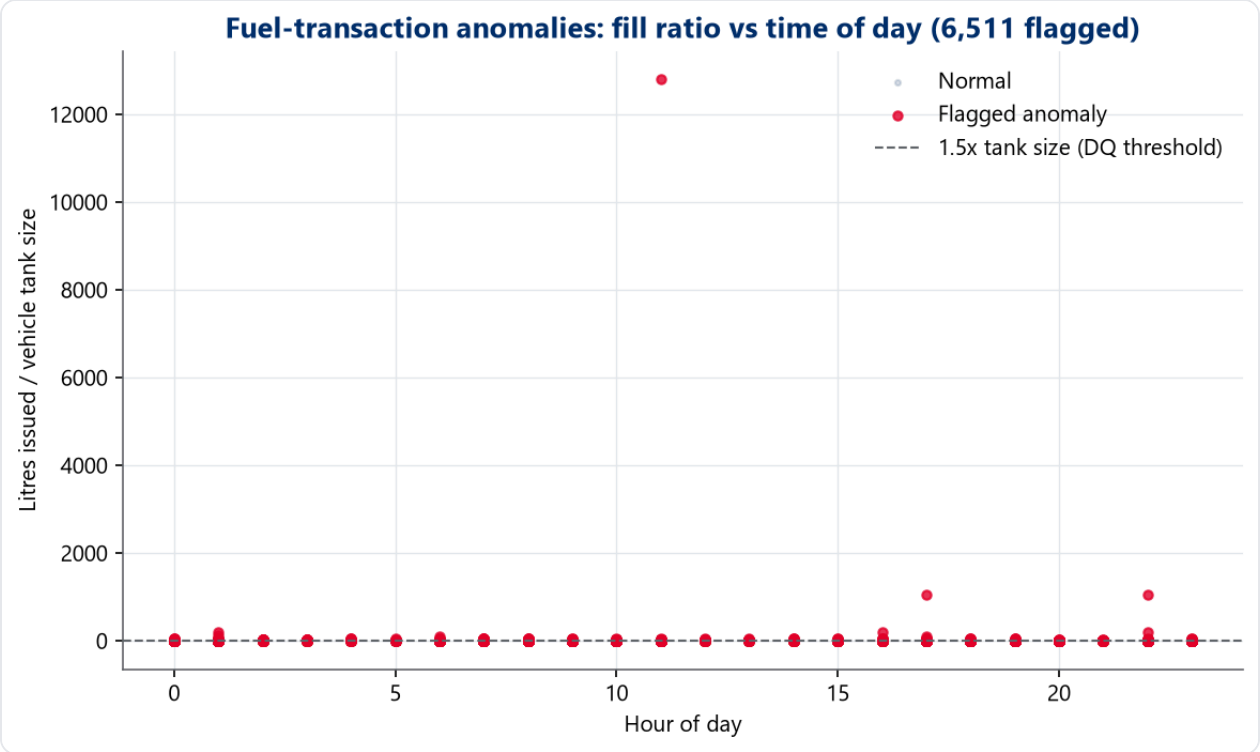
Finding	Rows
Usage row unmatched to equipment	203 309
Fuel issue exceeds 1.5x tank size	14 746
AFS duplicate FuelEventId	1 728
Delivery zero/negative volume	288

The stand-outs: over 200 000 usage-logbook rows carry registration numbers that don't match the equipment master (an audit-trail gap for refund claims), and ~14 700 single fuel issues exceed 1.5× the receiving vehicle's tank size — classic candidates for meter faults or leakage/theft review. One equipment-master row even had a tab character embedded in its

registration number, found because it was the only row in 23.7 million that broke a rectangular file export.

7 · Machine learning: fuel-anomaly detection

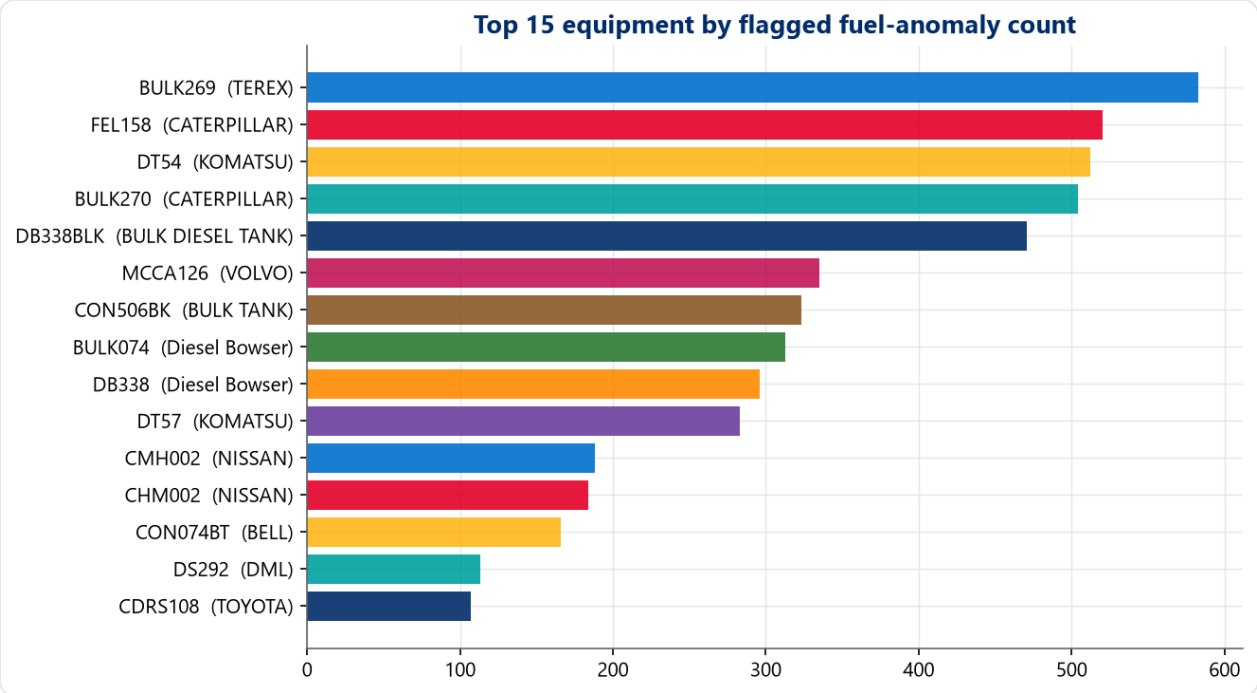
An **Isolation Forest** (a scikit-learn algorithm that isolates unusual data points by how few random splits it takes to separate them from the rest — the fewer splits, the more anomalous; 300 trees, 2% contamination setting) scores every fuel transaction on five features: litres issued, tank-fill ratio (litres ÷ that vehicle's tank size), that vehicle's own fill-ratio z-score (how far this fill is from *that specific vehicle's* normal pattern, so a naturally large tanker isn't penalised for being large), hour of day, and day of week. This catches what a fixed 1.5×-tank-size rule misses — a fill that's unremarkable in isolation but anomalous in combination (e.g. an odd hour *plus* a fill well above that vehicle's own history).



6 511 of 325 504 transactions flagged (2.0%). They split into three distinct categories with three different explanations and three different actions:

Category	Count	Litres involved	What it looks like	Most likely cause
Extreme (fill ratio > 50×)	35	8.86M L	Single transactions of 100,000+ litres against 100–600 L tanks — physically impossible	Decimal-point data-entry error at the pump terminal (e.g. 159,015.1 L almost certainly means 15.9 L)
Moderate (fill ratio 1.5×–50×)	4 216	20.2M L	A real, plausible fill — just larger than that vehicle normally takes	Genuine theft/leakage candidates, meter faults, or a vehicle swap not reflected in the equipment master
Behavioural (fill ratio normal)	2 260	—	Fill size is unremarkable, but timing/pattern is off for that vehicle	Off-hours or off-schedule fuelling worth a second look, not necessarily theft

Timing is a real signal here, not noise: flagged transactions happen overnight (22:00–05:00) 32% of the time, against a 22% overnight share for all transactions — a meaningfully higher overnight rate among the flagged group.



Business insight — this isn't spread evenly across the fleet. Ultra City alone accounts for 3 564 of the 6 511 flagged transactions (55% of all anomalies) — one depot, not the whole operation.

Root-cause refinement — that concentration is partly a modelling artefact, not partly fraud. 48% of Ultra City's flagged transactions belong to bulk-storage equipment (Diesel Bowser / BULK DIESEL TANK / BULK TANK) — mobile tankers and depot infrastructure that *are* the fuel supply, not vehicles being refuelled from it. Diesel Bowser equipment gets flagged at 11.9% of its transactions, versus 2.0% fleet-wide — six times the baseline rate, which is the signature of a category the model wasn't built for (their `TankSize` field doesn't reflect real bulk-carrying capacity), not a fleet-wide theft pattern. Excluding bulk-storage equipment, Ultra City still accounts for a disproportionate share of *genuine* vehicle anomalies — that residual, not the raw 3 564 figure, is the number worth investigating on site. **Recommended fix:** route bulk-storage transactions through delivery-style logic (comparable to `dw.FactFuelDelivery`) instead of scoring them against vehicle fill-ratio thresholds, then re-run the model.

Top 5 equipment to investigate first

- `BULK269` — 583 flagged transactions
- `FEL158` — 520 flagged transactions
- `DT54` — 512 flagged transactions
- `BULK270` — 504 flagged transactions
- `DB338BLK` — 471 flagged transactions

Top 10 highest-risk transactions

Date/time	Fleet ID	Make	Litres	Tank (L)	Fill ratio	Anomaly score
11/08/2020 00:07:04	DB338	Diesel Bowser	30 202.0	500	60.4x	0.801
05/30/2019 01:01:03	MOG1475	TOYOTA	15 901.5	130	122.3x	0.800
05/29/2019 17:36:20	MOG1410	TOYOTA	159 015.1	150	1 060.1x	0.800
08/11/2022 11:52:33	CON685	CATERPILLAR	8 000 000.0	625	12 800.0x	0.799
02/17/2020 16:38:34	CON093	CATERPILLAR	21 436.0	532	40.3x	0.799
05/30/2019 06:24:46	MOG1396	TOYOTA	15 901.5	150	106.0x	0.799

09/26/2020 05:57:43	DB338	Diesel Bowser	28 659.5	500	57.3x	0.797
05/29/2019 22:08:25	MOG1420	TOYOTA	159 015.1	150	1 060.1x	0.797
05/30/2019 01:13:18	CNL349L	TOYOTA	15 901.5	80	198.8x	0.797
07/22/2019 02:02:38	BULK270	CATERPILLAR	32 096.6	750	42.8x	0.796

Recommendations

1. **Prevent, don't just detect:** add input validation at the Automated Fuel System (AFS) terminal that rejects or holds any single transaction exceeding ~2× the vehicle's registered tank size. This alone would have caught all 35 extreme cases before they ever reached the ledger.
2. **Fix the model before the site visit:** exclude bulk-storage equipment (Diesel Bowser / BULK DIESEL TANK / BULK TANK) from the vehicle-style anomaly scoring and re-run. Only then is Ultra City's remaining anomaly count a trustworthy audit target — right now it's inflated by a data-model mismatch, not necessarily a site-specific problem.
3. **Wire this into the refund workflow:** run the anomaly score against `dw.FactFuelUsageClassification` before each monthly SARS claim, so flagged litres are excluded or held for review rather than claimed on potentially bad data.
4. **Quantify exposure before acting:** the 4 216 moderate-tier transactions (20.2M litres) are the ones worth a real investigation — most are probably legitimate, but at scale even a small leakage/theft rate here is a material rand figure.

Full 200-row review queue: `data/analysis/ml_review_queue.csv` and the "ML Anomaly Review Queue" sheet in the Excel workbook. Model: `sql/10_ml_anomaly_detection.py`.

8 · Azure in production

The pipeline ran for real, not just as a diagram: resource group `rg-anglo-mining-dw` (ADLS Gen2 storage `stanglominingdw01`, Data Factory `adf-anglo-mining-dw`) was deployed, exercised, and validated end to end before being torn down — the cost-control step this project's own architecture doc recommends. The evidence below is real command output captured while the pipeline was live, not a mock-up.

```
$ az datafactory pipeline-run show --run-id 3381ce57-... --query "{status,durationMs,mes
{
  "status": "Failed",
  "durationMs": 159644,
  "message": "DelimitedTextIncorrectRowDelimiter ... cp1252 vs UTF-8 encoding mismatch"
}
> root cause: bcp -c writes cp1252, ADF's Copy activity expected UTF-8 - fixed at source
> (02_export_dw_to_tsv.ps1, -C 65001) and by re-encoding existing exports.

$ az datafactory pipeline-run show --run-id 06be42ab-... --query "{status,durationMs,mes
{ "status": "Succeeded", "durationMs": 132855, "message": "" }    <- 16 warehouse tables

$ az datafactory pipeline-run show --run-id e09e391f-... --query "{status,durationMs,mes
{ "status": "Succeeded", "durationMs": 219681, "message": "" }    <- CoordRef, 21.9M row

$ azcopy list "https://stanglominingdw01.blob.core.windows.net/curated?<sas>"
dw/FactFuelTransaction/FactFuelTransaction.parquet          15.25 MiB
dw/FactEquipmentTrip/FactEquipmentTrip.parquet               37.90 MiB
dw/FactFuelUsageClassification/FactFuelUsageClassification.parquet  14.29 MiB
dw/FactStorageLogbook/FactStorageLogbook.parquet              8.67 MiB
dw/CoordRef/CoordRef.parquet                                346.23 MiB
dw/DimEquipment/DimEquipment.parquet                         88.24 KiB
... (16 tables total, all present, all reconciled against source)
```

The first run's failure is left in on purpose — a real build hits real bugs (here, a Windows bcp encoding default that ADF's Copy activity didn't expect), and the fix is what turned it into a working pipeline, not a lucky first attempt.

> **Why no portal screenshots:** the resource group was already deleted (per this project's own kill-switch policy) by the time this section was written, and re-deploying just to capture a screenshot didn't seem worth the cost for a result the command output above already proves. See `azure/AZURE_ARCHITECTURE.md` to redeploy from scratch if you want to reproduce it.

9 · How it's built

```
SQL Server (source ERP, QA copy)            ── local, always works offline
  └─ dw star schema  · 8 dims, 8 facts, 5 views   (01_create_load_dw_full.sql)
       └─ TSV export (bcp, UTF-8)                 (02_export_dw_to_tsv.ps1)
            ├─ curated_local/ Parquet             (03_build_local_parquet.py)
            └─ azcopy → ADLS (Azure Data Lake Storage) Gen2 raw/  ── Azure mirror
                 └─ Data Factory pl_raw_to_curated (ForEach Copy, TSV → snappy Parquet)
                      └─ curated/dw/<Table>/*.parquet
                           └─ Qlik Sense Cloud (SAS web-files or Azure Storage connector
```

The warehouse reconciles exactly: 290 557 288.29 litres issued in both source and fact table, to the hundredth of a litre. Everything regenerates from five scripts, and the Azure resource group tears down with one command when it has served its purpose.

Full source, the SQL build script, both Qlik load scripts (cloud + local), and the Azure Data Factory definitions are in the repository: https://github.com/anthonyapollis/KalahariPetroleum_DW_Azure_Qlik. See qlik/QLIK_APP_GUIDE.md for the six-sheet Qlik Sense app design, and azure/AZURE_ARCHITECTURE.md for the deployed-resource inventory and kill switch.