METCS777

Big Data Analytics

Term Paper

Short Report

Topic

Performance and Visualization of Big Data in Cloud Environments: A Comparative Study of Spark SQL (Databricks) and BigQuery

Team Members:

Tanvi Thopte-U35489362

Mehul Bisht-U93099406

1) Environment Setup

Component	BigQuery	Databricks
Cluster	BigQuery Free Tier (serverless, managed by Google)	One driver node and two worker nodes with 15 GB RAM total
Dataset	Stored in a Google Cloud Storage bucket linked to BigQuery	9 GB dataset split across multiple CSV files uploaded.
Libraries Tools	BigQuery Web Console using SQL	PySpark and Plotly for data processing and visualization
File Paths	glassy-ripsaw-4732233.ecommerce.events_clean	Volumes/ecom in the Databricks workspace
Visualization	Built-in BigQuery Charts and Data Studio (Looker)	Plotly and Matplotlib charts generated through notebooks

2) How to Run the Code

For BigQuery

- We first upload our raw dataset into BigQuery under a table named events_raw.
- Then, we run the CREATE OR REPLACE TABLE command to clean and partition the data, creating a new table called events_clean.
- After the table is ready, we execute all the EDA and analytics queries in sequence to perform our analysis.
- Once the queries finish, we check the Execution Details tab to view metrics like elapsed time, slot usage, and bytes processed to understand performance.
- Then,we connect Looker Studio to Big Query for visualisation purpose.

For Databricks

- We upload the CSV files containing the raw data into a Databricks workspace folder.
- Next, we run the create_table.sql script to create the cleaned events_clean table inside Databricks.
- After that, we run the eda_queries.sql and analytics_queries.sql scripts to perform the same exploratory and analytical steps as in BigQuery.
- Finally, we use plotly_charts.py to generate visualizations (like revenue trends and outlier detection). We make small adjustments in the script to match the actual dataset and query outputs.

3) DATASET DESCRIPTION:

The dataset used for this project is a 9 GB e-commerce events log containing detailed user activity and transaction information collected from an online retail platform. It captures all interaction users make on the website, from viewing products to completing purchases.

<u>Key Features:</u>

Total Size: 9 GB (split into multiple CSV files for Databricks; stored as a single table in BigQuery)

Time Period Covered: October 2019 – April 2020

Number of Records: Approximately 98 million rows

Each row represents a single user event (e.g., product view, cart addition, or purchase).

Main Columns:

- event_time – Timestamp of the user action
- event_type – Type of event (view, cart, purchase, etc.)
- product_id, category_id, category_code – Product details and hierarchy
- brand – Brand name of the product
- price – Product price at the time of the event
- user_id – Unique identifier of the user
- user_session – Session ID representing a continuous user visit

Purpose:
This dataset enables analysis of user behavior, product performance, sales trends, and anomalies over time. It is used for both exploratory data analysis (EDA) and performance analytics, including daily revenue tracking, brand/category aggregation, and anomaly detection using statistical methods.

What are we exactly doing:

Steps from start to end (no code, just what each query does):

1. Data Cleaning and Table Creation – Create a clean, partitioned table (events_clean) from raw events, remove rows with missing prices, and ensure proper data types.
2. Row Count and Time Coverage – Check total number of rows and event date range to verify data completeness.
3. Missing Data Overview – Identify how many rows have missing category codes, brands, or prices.
4. Cardinality Analysis – Count unique products, users, sessions, categories, and brands.
5. Price Statistics – Find minimum, maximum, average, and quartile distribution of prices.
6. Monthly Event Distribution – Analyze total number of events and total revenue per month.
7. Daily Revenue Trend – Calculate daily total orders, revenue, and average price; include day-over-day revenue change.
8. Category × Brand Aggregation – Compute total orders, revenue, and average price for each category-brand pair.
9. User-Level Monetization – Summarize each user’s activity: sessions, events, total and average spending.
10. Session Analytics – Evaluate basket size, number of items, and total value per session.

11. Trend and Anomaly Detection – Generate 7-day rolling averages and z-scores to detect abnormal revenue days.
12. Scalability Experiment – Run the same aggregation on 1-month, 3-month, and 7-month periods to compare performance.
13. Top-K Products by Revenue – Identify the top 100 products generating the highest total sales.
14. Brand-Level Price Outliers – Detect products whose prices deviate more than three standard deviations from their brand’s average.
15. Heavy Query (Category × Brand) – Repeat the main category-brand analysis for a fixed time window (Oct 2019–Apr 2020).
16. Heavy Query (User-Level) – Re-run the user-level aggregation for the same time window.
17. Heavy Query (Session Analytics) – Re-run the high-cardinality session analysis for the same time window.
18. Rolling-Window Trend (Fixed Period) – Compute 7-day moving average and z-score for daily revenue within a defined time frame.
19. Daily Metrics View Creation – Create a reusable view summarizing daily revenue, 7-day moving average, standard deviation, and z-score for dashboards or further analysis.

Codes:

BIG QUERY :

Final TermP Queries | Run | Save query | Download | Share | Schedule | Open in | More

```sql

-- Names: Tanvi Thopte, Mehul Bisht
-- Cleaning + analyzing ecommerce events in BigQuery

/*0) Create a clean, partitioned table from the raw events.
    PARTITION BY DATE(event_time) to speed up time-range scans.
    SAFE_CAST converts/validates types without failing the query.
    Filters out rows with NULL price, since price is central to revenue calcs. -*/

-- glassy-ripsaw-473223-u3 is project name with ecommerce_events_raw and later cleaned and has ecommerce_events_clean

CREATE OR REPLACE TABLE `glassy-ripsaw-473223-u3.ecommerce.events_clean`
PARTITION BY DATE(event_time)
AS
SELECT
  TIMESTAMP(event_time) AS event_time, -- normalize to TIMESTAMP
  event_type,  -- keeping as-is
  SAFE_CAST(product_id   AS STRING) AS product_id,    -- to string type
  SAFE_CAST(category_id  AS STRING) AS category_id,   -- to string type
  category_code,  -- may be NULL-used in group-bys
  brand,  -- may be NULL-pused in group-bys
  SAFE_CAST(price AS FLOAT64)AS price,         -- numeric price for math
  SAFE_CAST(user_id AS STRING) AS user_id,       -- user identifying
  SAFE_CAST(user_session AS STRING)AS user_session  -- session identifying
FROM `glassy-ripsaw-473223-u3.ecommerce.events_raw`
WHERE price IS NOT NULL;  -- remove rows without price

```

```sql
---EDA (Exploratory Data Analysis)

-- 0.1 Row count + time coverage (checking of table size and timeline)
SELECT
  COUNT(*) AS total_rows,
  MIN(event_time) AS first_event,
  MAX(event_time) AS last_event
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`;

-- 0.2 Missing data overview
SELECT
  SUM(CASE WHEN category_code IS NULL THEN 1 ELSE 0 END) AS null_category_code,
  SUM(CASE WHEN brand  IS NULL THEN 1 ELSE 0 END) AS null_brand,
  SUM(CASE WHEN price IS NULL THEN 1 ELSE 0 END) AS null_price  -- should be 0 by construction
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`;

-- 0.3 Cardinalities (uniques across main keys for sense of scale)
SELECT
  COUNT(DISTINCT product_id) AS unique_products,
  COUNT(DISTINCT user_id) AS unique_users,
  COUNT(DISTINCT user_session)  AS unique_sessions,
  COUNT(DISTINCT category_code) AS unique_categories,
  COUNT(DISTINCT brand) AS unique_brands
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`;

-- 0.4 Price stats (range, average, quartiles for distribution shape)
SELECT
  MIN(price) AS min_price,
  MAX(price) AS max_price,
  AVG(price) AS avg_price,
  APPROX_QUANTILES(price, 4) AS price_quartiles  -- min, Q1, median, Q3, max
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`;

-- 0.5 Monthly event distribution (volume + gross value per month)
SELECT
  FORMAT_DATE('%Y-%m', DATE(event_time)) AS month,
  COUNT(*) AS total_events,
  SUM(price) AS total_value
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
GROUP BY month
ORDER BY month;

```

```sql
-- PART B: Performance / Analyese
-- 1) Daily revenue trend with day-over-day change
--     Aggregates orders, revenue, avg ticket daily.
--    Uses LAG to compute day-over-day revenue delta.
WITH daily AS (
  SELECT
    DATE(event_time) AS day,
    COUNT(*) AS total_orders,
    SUM(price) AS total_revenue,
    AVG(price) AS avg_ticket
  FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
  GROUP BY day
)
SELECT
  day,
  total_orders,
  total_revenue,
  avg_ticket,
  total_revenue - LAG(total_revenue) OVER (ORDER BY day) AS revenue_change
FROM daily
ORDER BY day;

-- 2) Category into Brand aggregation (multi-dimensional performance view)
--    Filters out NULLs so results are cleaner.
--     Orders by revenue to surface top pairs.
SELECT
  category_code,
  brand,
  COUNT(*) AS total_orders,
  SUM(price) AS total_revenue,
  AVG(price) AS avg_price
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE brand IS NOT NULL AND category_code IS NOT NULL
GROUP BY category_code, brand
ORDER BY total_revenue DESC
LIMIT 1000;

-- 3) User-level monetization
--    Sessions = distinct sessions per user
--    Events = line-item or click-level counts.
--    total_spent, avg_spent = revenue view.
SELECT
  user_id,
  COUNT(DISTINCT user_session) AS sessions,
  COUNT(*) AS events,
  SUM(price)  AS total_spent,
  AVG(price)  AS avg_spent
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
GROUP BY user_id
ORDER BY total_spent DESC
LIMIT 1000;
```

```sql
-- 4) Session analytics (basket size/value)
--    unique_products = breadth of items in a basket.
--    items = total line count in session
--    session_value = revenue per session.
SELECT
  user_session,
  COUNT(DISTINCT product_id) AS unique_products,
  COUNT(*) AS items,
  SUM(price) AS session_value
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
GROUP BY user_session
ORDER BY session_value DESC
LIMIT 1000;

-- 5) Trend + anomaly detection (7-day rolling z-score)
--    ma7 = 7-day moving average revenue.
--    sd7 = 7-day rolling std dev.
--    zscore flags deviations from short-term trend.
WITH daily AS (
  SELECT DATE(event_time) AS day, SUM(price) AS revenue
  FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
  GROUP BY day
),
stats AS (
  SELECT day,revenue,
    AVG(revenue) OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS ma7,
    STDDEV_POP(revenue) OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS sd7
  FROM daily)

SELECT
  day, revenue, ma7, sd7,
  SAFE_DIVIDE(revenue - ma7, sd7) AS zscore
FROM stats
WHERE sd7 IS NOT NULL
ORDER BY day;

-- 6) Scalability experiment (compare scan size/latency by time window)
--    Same aggregation over 1 month, 3 months, and 7 months.
--    Measure performance externally (bytes processed, duration).

-- 6a: 1 month window
SELECT category_code, brand, SUM(price) AS revenue
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2019-11-01')
GROUP BY category_code, brand;

-- 6b: 3 month window
SELECT category_code, brand, SUM(price) AS revenue
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-01-01')
GROUP BY category_code, brand;

-- 6c: full 7 months window
SELECT category_code, brand, SUM(price) AS revenue
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-05-01')
GROUP BY category_code, brand;
```

```sql
-- 7) Top-K products by revenue (exact ranking)
SELECT product_id, SUM(price) AS revenue
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
GROUP BY product_id
ORDER BY revenue DESC
LIMIT 100;

-- 8) Brand-level price outliers (|z| > 3)
--      per-brand mean & stddev.
--     Join back to score each event's price.
WITH brand_stats AS (
  SELECT brand, AVG(price) AS avg_price, STDDEV_POP(price) AS sd_price
  FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
  GROUP BY brand
)
SELECT
  e.brand, e.product_id, e.price, b.avg_price, b.sd_price,
  SAFE_DIVIDE(e.price - b.avg_price, b.sd_price) AS z
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean` e
JOIN brand_stats b USING (brand)
WHERE b.sd_price IS NOT NULL AND ABS(SAFE_DIVIDE(e.price - b.avg_price, b.sd_price)) > 3
ORDER BY z DESC
LIMIT 100;


```

```sql
/*Heavy queries repeated with explicit global time filter (2019-10 to 2020-04)*/

-- Category × Brand
SELECT
  category_code,
  brand,
  COUNT(*)   AS total_orders,
  SUM(price) AS total_revenue,
  AVG(price) AS avg_price
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-05-01')
  AND category_code IS NOT NULL AND brand IS NOT NULL
GROUP BY category_code, brand
ORDER BY total_revenue DESC
LIMIT 1000;

-- User-level large group-by
SELECT
  user_id,
  COUNT(DISTINCT user_session) AS sessions,
  COUNT(*)  AS events,
  SUM(price) AS total_spent,
  AVG(price)AS avg_spent
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-05-01')
GROUP BY user_id
ORDER BY total_spent DESC
LIMIT 1000;

-- Session analytics (very high cardinality)
SELECT
  user_session,
  COUNT(DISTINCT product_id) AS unique_products,
  COUNT(*)  AS items,
  SUM(price)  AS session_value
FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-05-01')
  AND user_session IS NOT NULL
GROUP BY user_session
ORDER BY session_value DESC
LIMIT 1000;

-- Rolling-window trend + anomalies (7-day z-score)
WITH daily AS (
  SELECT DATE(event_time) AS day, SUM(price) AS revenue
  FROM `glassy-ripsaw-473223-u3.ecommerce.events_clean`
  WHERE event_time >= TIMESTAMP('2019-10-01') AND event_time < TIMESTAMP('2020-05-01')
  GROUP BY day
)
SELECT
  day,
  revenue,
  AVG(revenue)  OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS ma7,
  STDDEV_POP(revenue) OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS sd7,
  SAFE_DIVIDE(
    revenue - AVG(revenue) OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW),
    NULLIF( STDDEV_POP(revenue) OVER (ORDER BY day ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 0)
  ) AS zscore
```

Databricks Code:

We have run the same queries as BigQuery.

Outputs:

Both Platforms are giving almost the same outputs:

Creating new clean table

Query results

Job information | Results | Execution details | Execution graph

This statement replaced the table named events_clean.

Row count + time coverage

Row	total_rows	first_event	last_event
1	67501979	2019-11-01 00:00:00 UTC	2019-11-30 23:59:59 UTC

Missing data overview

Row	null_category_code	null_brand	null_price
1	21898171	9218235	0

Cardinalities (uniques across main keys for sense of scale)

Row	unique_products	unique_users	unique_sessions	unique_categories	unique_brands
1	190662	3696117	13776050	129	4201

Price stats (range, average, quartiles for distribution shape)

Row	min_price	max_price	avg_price	price_quartiles
1	0.0	2574.07	292.4593165646...	0.0
				68.47
				162.42
				360.09
				2574.07

Monthly event distribution (volume + gross value per month)

Row	month	total_events	total_value
1	2019-11	67501979	19741582645.09...

<u>1) Daily revenue trend with day-over-day change</u>

-- Aggregates orders, revenue, avg ticket daily.

-- Uses LAG to compute day-over-day revenue delta.

Query results | Save results | Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	day	total_orders	total_revenue	avg_ticket	revenue_change
1	2019-11-01	1445360	425699527.2600...	294.5283716582...	null
2	2019-11-02	1555538	454850035.9900...	292.4068945856...	29150508.73004...
3	2019-11-03	1567774	467094603.9000...	297.9349089218...	12244567.90996...
4	2019-11-04	1793128	531778362.6000...	296.5646415649...	64683758.70001...
5	2019-11-05	1717244	495287635.8198...	288.4200706597...	-36490726.7802...
6	2019-11-06	1694821	489674523.9398...	288.9240361902...	-5613111.88000...
7	2019-11-07	1796833	512657491.8200...	285.3117077769...	22982967.88016...
8	2019-11-08	1896402	554703837.0796...	292.5032968115...	42046345.25960...
9	2019-11-09	1877906	546360619.0097...	290.9414097457...	-8343218.06985...
10	2019-11-10	1940575	570583105.3198...	294.0278553109...	24222486.31010...
11	2019-11-11	2009390	579771948.0898...	288.5313194999...	9188842.770029...
12	2019-11-12	1987569	583480026.2198...	293.5646642808...	3708078.129974...
13	2019-11-13	2019165	602679110.2499...	298.4793764996...	19199084.03004...
14	2019-11-14	3069726	993883256.8096...	323.7693712109...	391204146.5597...
15	2019-11-15	6220416	1898027969.470...	305.1287839060...	904144712.6603...
16	2019-11-16	6502957	1998441656.939...	307.3127589402...	100413687.4696...
17	2019-11-17	6395377	1876480491.500...	293.4120211365...	-121961165.439...
18	2019-11-18	2021512	585124794.6899...	289.4490830081...	-1291355696.81...
19	2019-11-19	1728541	489910565.1600...	283.4243244215...	-95214229.5299...
20	2019-11-20	1700086	466821973.3798...	274.5872699263...	-23088591.7802...
21	2019-11-21	1677336	448364254.9000...	267.3073581560...	-18457718.4798...
22	2019-11-22	1568243	437934668.4498...	279.2517922605...	-10429586.4501...
23	2019-11-23	1561716	440199629.3599...	281.8691934769...	2264960.910097...
24	2019-11-24	1591765	447272961.949921	280.9918310491...	7073332.589924...
25	2019-11-25	1593582	440003052.7598...	276.1094520143...	-7269909.19003...
26	2019-11-26	1654879	455308302.8698...	275.1308723296...	15305250.10997...

2) Category into Brand aggregation (multi-dimensional performance view)

-- Filters out NULLs so results are cleaner.

-- Orders by revenue to surface top pairs.

Query results | Save results ▾ | Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	category_code	brand	total_orders	total_revenue	avg_price
1	electronics.smartphone	apple	4658729	4396550883.339...	943.7232
2	electronics.smartphone	samsung	5316962	1859341741.730...	349.7000
3	electronics.smartphone	xiaomi	3331784	762315172.0799...	228.8008
4	electronics.video.tv	samsung	771041	464426314.4800...	602.3367
5	electronics.smartphone	huawei	1237930	351044715.7599...	283.5739
6	computers.notebook	lenovo	598788	343124594.2699...	573.0318
7	computers.notebook	acer	536366	341795252.8000...	637.2425
8	computers.notebook	apple	169262	287483656.5799...	1698.453
9	computers.notebook	asus	389176	263873519.9399...	678.0313
10	electronics.smartphone	oppo	811698	243552014.1399...	300.0524
11	electronics.video.tv	lg	351790	207951920.5599...	591.1251
12	electronics.clocks	apple	424133	197039963.2399...	464.5711
13	computers.notebook	hp	319525	192047871.3400...	601.0417
14	appliances.kitchen.washer	samsung	425969	177984031.6199...	417.8332
15	electronics.audio.headphone	apple	813163	165488542.4099...	203.5121
16	appliances.kitchen.washer	lg	306091	131411032.9999...	429.3201
17	appliances.kitchen.refrigerators	lg	173244	122042147.1400...	704.4523
18	electronics.video.tv	sony	123282	100713806.2999...	816.9384
19	computers.desktop	pulser	135853	99488566.71999...	732.3251
20	appliances.kitchen.refrigerators	samsung	119826	92846469.95999...	774.8441
21	electronics.tablet	apple	121133	87504647.02000...	722.3848
22	computers.desktop	acer	89033	81908755.34999...	919.9815
23	electronics.clocks	garmin	88036	77337249.45000...	878.4730
24	electronics.smartphone	oneplus	110173	76271744.71000...	692.2907
25	electronics.video.tv	artel	311139	70713351.12000...	227.2725
26	appliances.kitchen.refrigerators	indesit	206983	64108196.75000...	309.7268
27	electronics.clocks	samsung	222906	59918188.32000...	268.8047
28	computers.desktop	lenovo	65947	55210916.11000...	837.2013
29	computers.notebook	dell	62062	54966179.54999...	885.66

3) User-level monetization

-- Sessions = distinct sessions per user

-- Events = line-item or click-level counts.

-- total_spent, avg_spent = revenue view.

Query results

Save results Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	user_id	sessions	events	total_spent	avg_spent
1	568778435	22542	22929	5187451.930000...	226.2397806271...
2	512365995	561	6042	2263807.200000...	374.6784508440...
3	569335945	14810	14810	1655685.840000...	111.7951276164...
4	569038711	2407	2407	1591572.03	661.2264353967...
5	568805468	2767	2847	1567113.060000...	550.4436459430...
6	513558661	74	1528	1557656.669999...	1019.408815445...
7	512845454	72	1729	1537491.579999...	889.2374667437...
8	568793129	4453	4771	1390417.869999...	291.4311192622...
9	566522251	131	1854	1235717.829999...	666.5144714131...
10	563907320	125	976	1201993.319999...	1231.550532786...
11	567475167	3617	3724	1199111.089999...	321.9954591836...
12	568833833	1441	1577	1156774.750000...	733.5286937222...
13	569625394	76	990	1145607.740000...	1157.179535353...
14	568818636	57	6171	1075817.049999...	174.3343137254...
15	529616034	47	1113	1055755.129999...	948.5670530098...
16	568804062	3290	4257	1054749.990000...	247.7683791402...
17	468703624	1205	1218	1002879.65	823.3823070607...
18	568797382	1289	1336	964422.9700000...	721.8734805389...
19	513778820	139	1671	956587.1699999...	572.4638958707...
20	571663686	14	789	953195.05	1208.105259822...
21	518433364	48	900	936563.2300000...	1040.625811111...
22	547556934	46	687	920568.3599999...	1339.983056768...
23	550781174	2050	2575	918893.8399999...	356.8519766990...
24	512558829	96	1166	914411.9999999...	784.2298456260...
25	568836063	1054	1072	903175.08	842.5140671641...
26	516365893	64	727	872393.0499999...	1199.990440165...
27	528026033	80	785	868511.0700000...	1106.383528662...
28	568512539	92	2080	850350.1599999...	408.8221923076...
29	512432656	209	1169	837250.2299999...	716.2106330196...

4) Session analytics (basket size/value)

-- unique_products = breadth of items in a basket.

-- items = total line count in session

-- session_value = revenue per session.

Query results   Save results ▾   Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	user_session	unique_products	items	session_value
1	0c307610-aa79-bf12-4ada-323...	543	918	360819.7
2	ef2fd879-4b1a-4319-818c-567d...	41	246	331040.0500000...
3	c6be5380-8322-432e-b948-90b...	49	178	291139.89
4	123c2880-3db4-4f69-b2aa-9c6...	139	360	274889.9300000...
5	1d34878d-1a42-401b-90a4-d44...	9	251	272767.1499999...
6	84f2e900-9da5-c970-7a9e-6c57...	3	204	248544.5999999...
7	37738e78-398a-460c-bd3b-6b1...	100	305	245503.2899999...
8	eefe97d8-7901-4a45-9a75-616...	91	323	243561.6399999...
9	21bba45e-ecbf-437c-9cbf-17ca...	96	180	233097.01
10	c200e51b-7c35-4efa-9956-a5e3...	73	279	232081.6299999...
11	1b64f998-793f-4093-a6a8-c9a8...	102	231	231680.6299999...
12	62731828-1c87-484f-ad90-f59d...	39	190	230063.1899999...
13	4921edd9-3cbb-43a4-bb0d-d12...	161	196	224270.9100000...
14	ea8d0053-08f0-4661-b00d-06b...	30	227	220417.0799999...
15	fdd27028-5bb0-422f-aa32-4a02...	48	124	204457.8300000...
16	ca310f80-ce7f-4fa0-acbf-28c37...	93	162	200352.09
17	1f569f01-9836-4491-abfd-d2aa...	94	124	199504.2500000...
18	081a2ea6-cc00-4fb8-9e92-dc97...	89	156	199335.44
19	fe991bee-2200-454b-93c6-36c3...	50	120	199126.8600000...
20	babd114c-3846-4244-a386-02e...	61	159	198432.16
21	194d4a86-f76e-4756-ad00-02e...	178	285	198114.1999999...
22	0b48b16b-f8ce-4b3f-be2d-7d7d...	144	215	196698.8200000...
23	e2778db6-0aee-4d95-832c-e6b...	62	240	195241.9600000...
24	81ea3b1d-3929-480b-b3ac-9ea...	80	107	191142.13
25	44f0883f-2647-4784-a631-1bf2...	50	146	189344.3099999...
26	53c412c9-e864-d0dc-a036-012...	574	594	187928.5500000...
27	7226bcff-353b-4f8c-a4bc-e4d0...	63	182	187526.71
28	b5a0b4de-1c09-4065-bef9-54bf...	41	217	187218.72

5) Trend + anomaly detection (7-day rolling z-score)

-- ma7 = 7-day moving average revenue.

-- sd7 = 7-day rolling std dev.

-- zscore flags deviations from short-term trend.

Query results — Save results ▾ — Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	day	revenue	ma7	sd7	zscore
1	2019-11-01	425699527.2600...	425699527.2600...	0.0	null
2	2019-11-02	454850035.9900...	440274781.6250...	14575254.36502...	1.0
3	2019-11-03	467094603.9000...	449214722.3833...	17362904.17513...	1.029774819713...
4	2019-11-04	531778362.6000...	469855632.4375...	38784588.88062...	1.596580805667...
5	2019-11-05	495287635.8198...	474942033.1140...	36150814.03062...	0.562797913446...
6	2019-11-06	489674523.9398...	477397448.2516...	33454642.62831...	0.366976739957...
7	2019-11-07	512657491.8200...	482434597.3328...	33340107.24275...	0.906502617616...
8	2019-11-08	554703837.0796...	500863784.4499...	32530138.83757...	1.655082165450...
9	2019-11-09	546360619.0097...	513936724.8813...	29674072.46479...	1.092667484953...
10	2019-11-10	570583105.3198...	528720796.5127...	28406490.04634...	1.473688186708...
11	2019-11-11	579771948.0898...	535577023.0112...	33628893.21413...	1.314195052368...
12	2019-11-12	583480026.2198...	548175935.9255...	32681694.22397...	1.080240517898...
13	2019-11-13	602679110.2499...	564319448.2555...	27256825.71423...	1.407341500309...
14	2019-11-14	993883256.8096...	633065986.1112...	148311469.7630...	2.432834569537...
15	2019-11-15	1898027969.470...	824969433.5955...	461391328.3739...	2.325701568029...
16	2019-11-16	1998441656.939...	1032409581.871...	596222392.2173...	1.620254602440...
17	2019-11-17	1876480491.500...	1218966351.325...	626089449.9421...	1.050192013673...
18	2019-11-18	585124794.6899...	1219731043.697...	625311069.5385...	-1.01486488872...
19	2019-11-19	489910565.1600...	1206363977.831...	639605834.0118...	-1.12014833913...
20	2019-11-20	466821973.3798...	1186955815.421...	659384949.3427...	-1.09212963195...
21	2019-11-21	448364254.9000...	1109024529.434...	708040259.4459...	-0.93308292250...
22	2019-11-22	437934668.4498...	900439772.145661	658193309.5646...	-0.70268885595...
23	2019-11-23	440199629.3599...	677833768.2056...	491625270.8224...	-0.48336436906...
24	2019-11-24	447272961.949921	473661263.9842...	48502262.29646...	-0.54406332374...
25	2019-11-25	440003052.7598...	452929586.5656...	17596369.35134...	-0.73461368920...
26	2019-11-26	455308302.8698...	447986406.2384...	9520247.533333...	0.769086791675...
27	2019-11-27	456351290.0500...	446490594.3342...	6907307.833478...	1.427574382599...
28	2019-11-28	471908085.4800...	449853998.7028...	11322084.23613...	1.947882237693...

6) Scalability experiment (compare scan size/latency by time window)

-- Same aggregation over 1 month, 3 months, and 7 months.

-- Measure performance externally (bytes processed, duration).

Query results   Save results   Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	category_code	brand	revenue
1	apparel.shoes.keds	null	5366510.830000...
2	appliances.personal.massager	null	537759.3800000...
3	appliances.environment.water_...	null	918041.9999999...
4	stationery.cartrige	null	67380.45000000...
5	null	a-mega	65144.00999999...
6	electronics.audio.acoustic	adagio	76252.34999999...
7	appliances.kitchen.hood	akpo	874625.24
8	appliances.kitchen.refrigerators	almacom	1767210.120000...
9	auto.accessories.videoregister	alpine	80439.24999999...
10	construction.tools.welding	alteco	65194.86000000...
11	computers.components.memory	amd	31106.08999999...
12	apparel.shoes.keds	anta	523612.2499999...
13	computers.notebook	apple	287483656.5799...
14	null	arnica	19602.65999999...
15	appliances.kitchen.hood	artel	717065.2099999...
16	furniture.bedroom.bed	askona	215933.1400000...
17	accessories.bag	asus	5601.939999999...
18	appliances.kitchen.washer	atlant	4503965.73
19	sport.snowboard	atlant	125212.2299999...
20	apparel.shoes	atrai	53860.49999999...
21	sport.bicycle	author	1107202.439999...

Query results

Save results

Job information | Results | Visualization | JSON | Execution details | Exe

Row	category_code	brand	revenue
1	auto.accessories.player	null	18247747.96999...
2	appliances.environment.vacuum	null	1163649.059999...
3	appliances.kitchen.coffee_mac...	null	333720.4599999...
4	apparel.sock	null	35719.58999999...
5	apparel.shoes.keds	adidas	2924287.990000...
6	null	adile	214312.4599999...
7	kids.carriage	aimile	28619.04000000...
8	appliances.environment.vacuum	airline	33778.08000000...
9	electronics.audio.microphone	akg	183545.1599999...
10	electronics.audio.microphone	alctron	108037.7899999...
11	construction.tools.pump	alteco	99745.08000000...
12	furniture.bedroom.pillow	alvitek	7630.849999999...
13	null	am.pm	663982.7800000...
14	furniture.bathroom.toilet	am.pm	55960.94000000...
15	null	amatis	197572.9799999...
16	auto.accessories.videoregister	anytek	1135409.000000...
17	null	apacer	878882.2300000...
18	null	aplus	467317.7899999...
19	apparel.shoes.sandals	ara	7206.280000000...
20	electronics.video.tv	arg	5242354.089999...
21	null	arg	1008515.670000...

7) Top-K products by revenue (exact ranking)

Query results

Save results ▾ Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Row	product_id	revenue
1	1005115	619370131.6099...
2	1005105	414205784.4399...
3	1005135	253700985.780001
4	1004249	219218005.9900...
5	1005116	189324729.1200...
6	1004767	140070377.5600...
7	1002544	128290954.8099...
8	1003317	114420230.7499...
9	1004659	100994058.8900...
10	1005174	92020653.60999...
11	1004870	89571223.92000...
12	1005129	86360298.34999...
13	1005284	85956164.55999...
14	1002524	85351937.71000...
15	1005106	82279350.14000...
16	1004856	80997844.46999...
17	1004873	79871374.42000...
18	1005124	77456216.88999...
19	1005144	76219416.78000...
20	1004258	75131906.01000...
21	1005132	73550214.96999...
22	1005186	72367965.02000...
23	1004246	70121840.31000...
24	1005160	68146237.29000...
25	1005104	66355126.88999...
26	1005073	64122712.81000...
27	4804056	63645354.55999...
28	1005118	61673358.31999...

8) Brand-level price outliers (|z| > 3)

-- per-brand mean & stddev.

-- Join back to score each event's price.

Query results

Save results ▾ Open in ▾

Job information Results Visualization JSON Execution details Execution graph

Row	brand	product_id	price	avg_price	sd_price
1	fitbit	5100235	218.77	180.1720392890...	0.68168
2	goodride	100017570	688.05	49.43657006525...	13.3569
3	turboair	2401679	618.25	49.65299112139...	13.1420
4	turboair	2401679	618.25	49.65299112139...	13.1420
5	triangle	12720013	1536.74	51.66865925173...	35.8682
6	triangle	12720013	1536.74	51.66865925173...	35.8682
7	triangle	12720013	1536.74	51.66865925173...	35.8682
8	triangle	12720013	1536.74	51.66865925173...	35.8682
9	triangle	12720013	1536.74	51.66865925173...	35.8682
10	triangle	12720013	1536.74	51.66865925173...	35.8682
11	triangle	12720013	1536.74	51.66865925173...	35.8682
12	triangle	12720013	1536.74	51.66865925173...	35.8682
13	triangle	12720013	1536.74	51.66865925173...	35.8682
14	triangle	12720013	1536.74	51.66865925173...	35.8682
15	triangle	12720013	1536.74	51.66865925173...	35.8682
16	triangle	12720013	1536.74	51.66865925173...	35.8682
17	triangle	12720013	1536.74	51.66865925173...	35.8682
18	triangle	12720013	1536.74	51.66865925173...	35.8682
19	triangle	12720013	1536.74	51.66865925173...	35.8682
20	triangle	12720013	1536.74	51.66865925173...	35.8682
21	triangle	12720013	1536.74	51.66865925173...	35.8682
22	triangle	12720013	1536.74	51.66865925173...	35.8682
23	triangle	12720013	1536.74	51.66865925173...	35.8682
24	triangle	12720013	1536.74	51.66865925173...	35.8682
25	triangle	12720013	1536.74	51.66865925173...	35.8682
26	triangle	12720013	1536.74	51.66865925173...	35.8682
27	triangle	12720013	1536.74	51.66865925173...	35.8682

PERFORMANCE ANALYSIS:

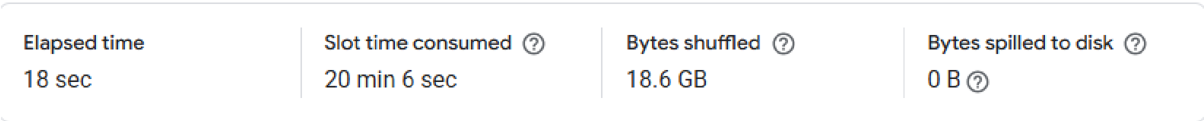
Following screenshots give:

- Elapsed time
- Bytes processed
- Bytes billed
- Bytes shuffled
- Bytes spilled to disk
- Slot time consumed
- Destination table

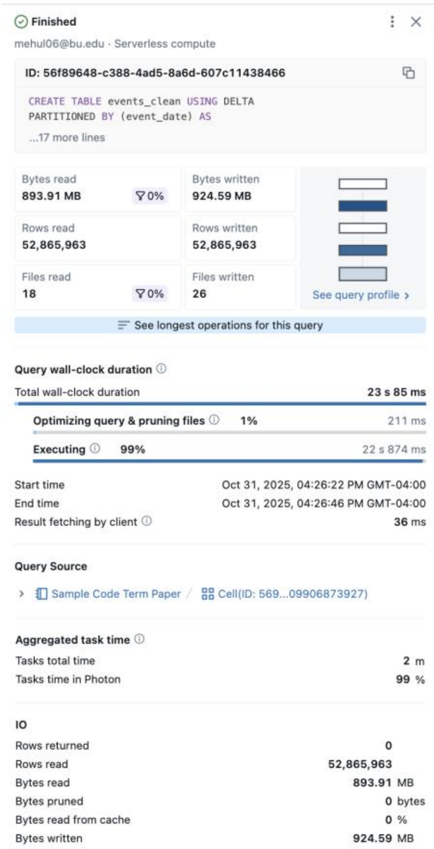
TO:

Clean and create new clean table:

Big Query:

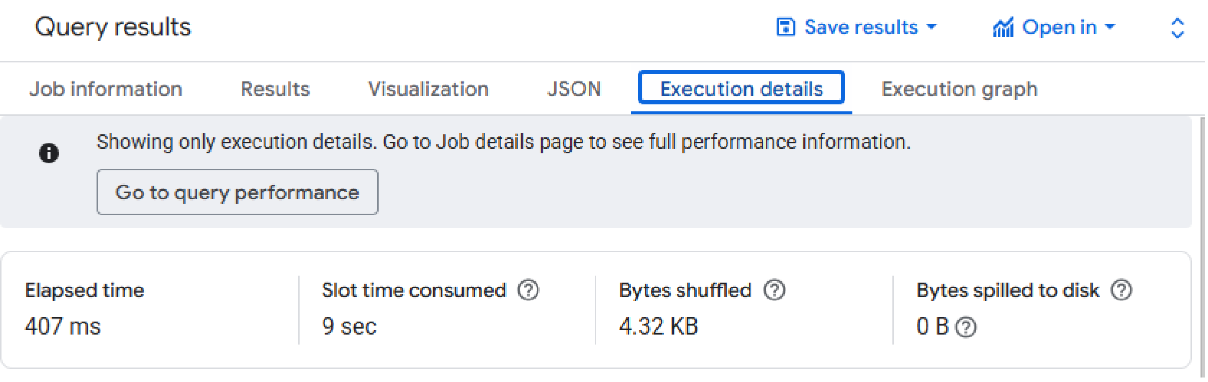


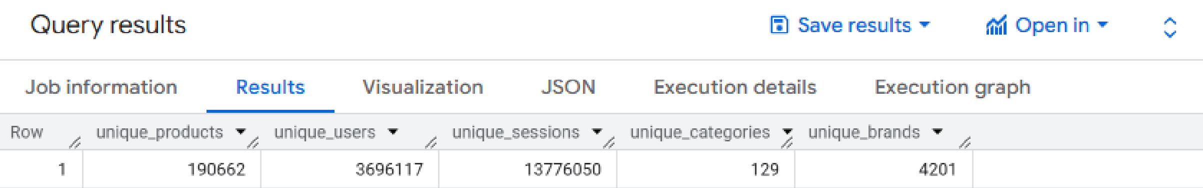
Databricks:



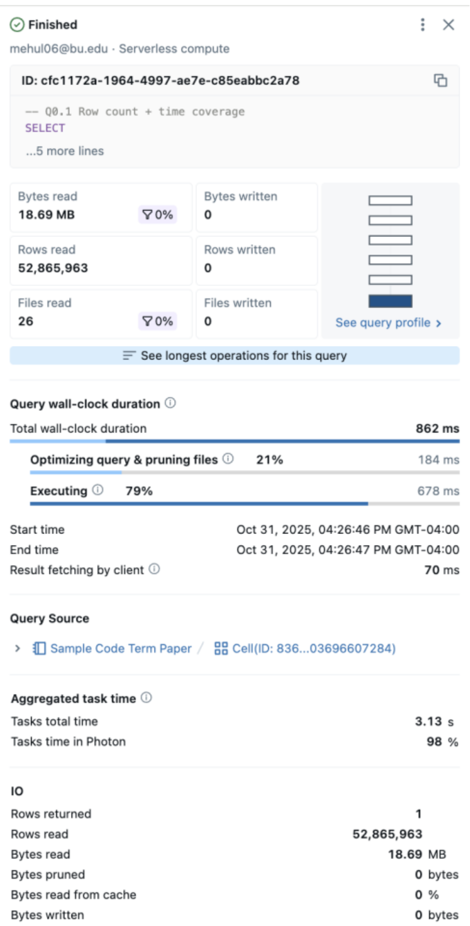
Row count and time coverage:

BigQuery:





Databricks:



Missing Data Overview:

BigQuery:

Query results

Save results ▾ Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Showing only execution details. Go to Job details page to see full performance information.

Go to query performance

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
6 sec	5 min 37 sec	5.16 GB	0 B

Query results

Save results ▾ Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_31e66071_19a37e83be2
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:35:57 PM UTC-4
Start time	Oct 30, 2025, 9:35:57 PM UTC-4
End time	Oct 30, 2025, 9:35:58 PM UTC-4
Duration	0 sec
Bytes processed	515 MB
Bytes billed	515 MB
Slot milliseconds	9854
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: d555afad-2ddf-4fac-ab30-6e94c044b152

```
-- Q0.2 Missing data overview
SELECT
```

...20 more lines

Bytes read 184.07 MB 0%	Bytes written 0	
Rows read 52,865,963	Rows written 0	
Files read 26 0%	Files written 0	See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **1 s 234 ms**

Optimizing query & pruning files **14%** 174 ms

Executing **86%** 1 s 60 ms

Start time	Oct 31, 2025, 04:26:48 PM GMT-04:00
End time	Oct 31, 2025, 04:26:49 PM GMT-04:00
Result fetching by client	**66** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607285)

Aggregated task time

Tasks total time	**3.77** s
Tasks time in Photon	**99** %

IO

Rows returned	**1**
Rows read	**52,865,963**
Bytes read	**184.07** MB
Bytes pruned	**0** bytes
Bytes read from cache	**3** %
Bytes written	**0** bytes

Cardinality Analysis:

BigQuery:

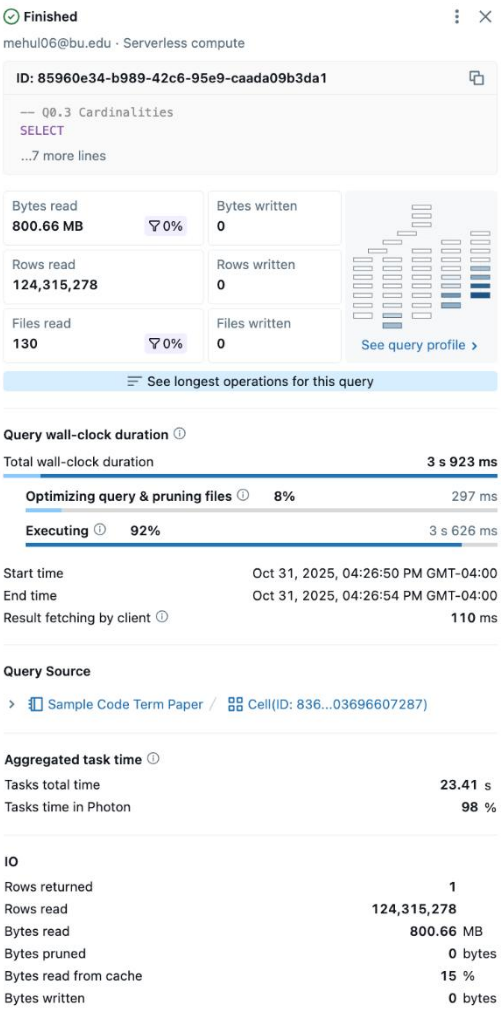
Query results

Save results ▾ Open in ▾

Job information | Results | Visualization | JSON | Execution details | Execution graph

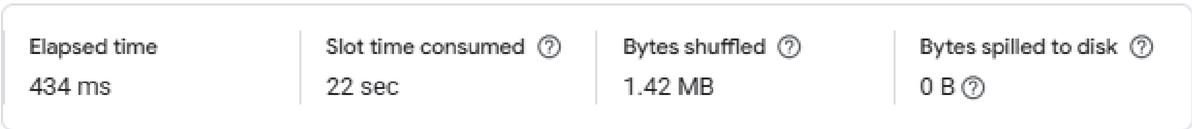
Job ID	glassy-ripsaw-473223-u3:US.bquxjob_20030245_19a37e9cf62
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:37:40 PM UTC-4
Start time	Oct 30, 2025, 9:37:40 PM UTC-4
End time	Oct 30, 2025, 9:37:47 PM UTC-4
Duration	6 sec
Bytes processed	5.14 GB
Bytes billed	5.14 GB
Slot milliseconds	337149
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:



Price Statistics

BigQuery:



Query results — Save results — Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_f341467_19a3bc4d251
User	tanvit@bu.edu
Location	US
Creation time	Oct 31, 2025, 3:35:46 PM UTC-4
Start time	Oct 31, 2025, 3:35:46 PM UTC-4
End time	Oct 31, 2025, 3:35:46 PM UTC-4
Duration	0 sec
Bytes processed	515 MB
Bytes billed	515 MB
Slot milliseconds	22786
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: 807c2e5e-1679-454f-b4c6-143a5b257e55

```
-- Q0.4 Price stats (Spark equivalent of APPROX_QUANTILE…
SELECT
```

...6 more lines

Bytes read: **96.36 MB** (0%)	Bytes written: **0**
Rows read: **52,865,963**	Rows written: **0**
Files read: **26** (0%)	Files written: **0**

See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration		**1 s 214 ms**
Optimizing query & pruning files	**18%**	215 ms
Executing	**82%**	999 ms

Start time	Oct 31, 2025, 04:26:54 PM GMT-04:00
End time	Oct 31, 2025, 04:26:56 PM GMT-04:00
Result fetching by client	**60** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607288)

Aggregated task time

Tasks total time	**4.46** s
Tasks time in Photon	**99** %

IO

Rows returned	**1**
Rows read	**52,865,963**
Bytes read	**96.36** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

Monthly Event Distribution

BigQuery:

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
633 ms	23 sec	4.86 KB	0 B

Query results

Save results

Job information | Results | Visualization | JSON | Execution details

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_24e0c4f4_19a3bc559d7
User	tanvit@bu.edu
Location	US
Creation time	Oct 31, 2025, 3:36:21 PM UTC-4
Start time	Oct 31, 2025, 3:36:21 PM UTC-4
End time	Oct 31, 2025, 3:36:21 PM UTC-4
Duration	0 sec
Bytes processed	1.01 GB
Bytes billed	1.01 GB
Slot milliseconds	23103
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: a92dea23-ac92-4295-94a4-5426aaf5ee42

```
— Q0.5 Monthly event distribution
SELECT
...9 more lines
```

Bytes read: 96.36 MB (0%)	Bytes written: 0
Rows read: 52,865,963	Rows written: 0
Files read: 26 (0%)	Files written: 0

See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration		694 ms
Optimizing query & pruning files	30%	207 ms
Executing	70%	487 ms

Start time	Oct 31, 2025, 04:26:56 PM GMT-04:00
End time	Oct 31, 2025, 04:26:57 PM GMT-04:00
Result fetching by client	76 ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607289)

Aggregated task time

Tasks total time	1.31 s
Tasks time in Photon	96 %

IO

Rows returned	1
Rows read	52,865,963
Bytes read	96.36 MB
Bytes pruned	0 bytes
Bytes read from cache	100 %
Bytes written	0 bytes

Daily Revenue Trend

BigQuery:

Query results   Save results   Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Showing only execution details. Go to Job details page to see full performance information.

Go to query performance

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
761 ms	24 sec	12.7 KB	0 B

Show average time | Show maximum time

Query results   Save results   Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_4e028272_19a37f004af
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:44:27 PM UTC-4
Start time	Oct 30, 2025, 9:44:27 PM UTC-4
End time	Oct 30, 2025, 9:44:28 PM UTC-4
Duration	0 sec
Bytes processed	1.01 GB
Bytes billed	1.01 GB
Slot milliseconds	24648
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: 82866ba5-7267-4b07-94bb-86d01e5e6afb

```
-- PART B: PERFORMANCE / ANALYTICS (Q1–Q8)
-- Q1 Daily revenue trend (with day-over-day change)
```

...21 more lines

Bytes read 96.36 MB (0%)	Bytes written 0	
Rows read 52,865,963	Rows written 0	
Files read 26 (0%)	Files written 0	See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **648 ms**

- **Optimizing query & pruning files** **31%** 201 ms
- **Executing** **69%** 447 ms

Start time	Oct 31, 2025, 04:26:57 PM GMT-04:00
End time	Oct 31, 2025, 04:26:58 PM GMT-04:00
Result fetching by client	**200** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607290)

Aggregated task time

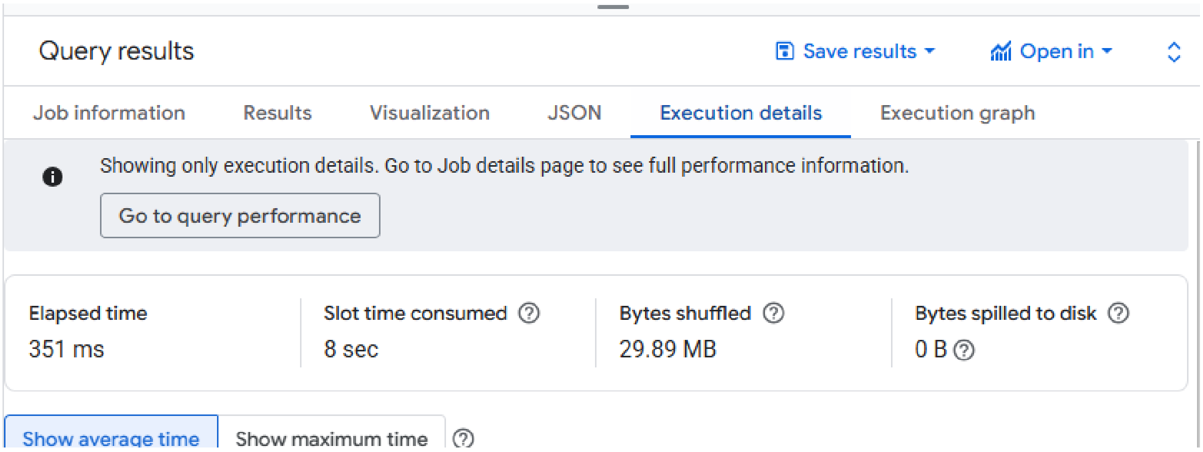
Tasks total time	**1.06** s
Tasks time in Photon	**94** %

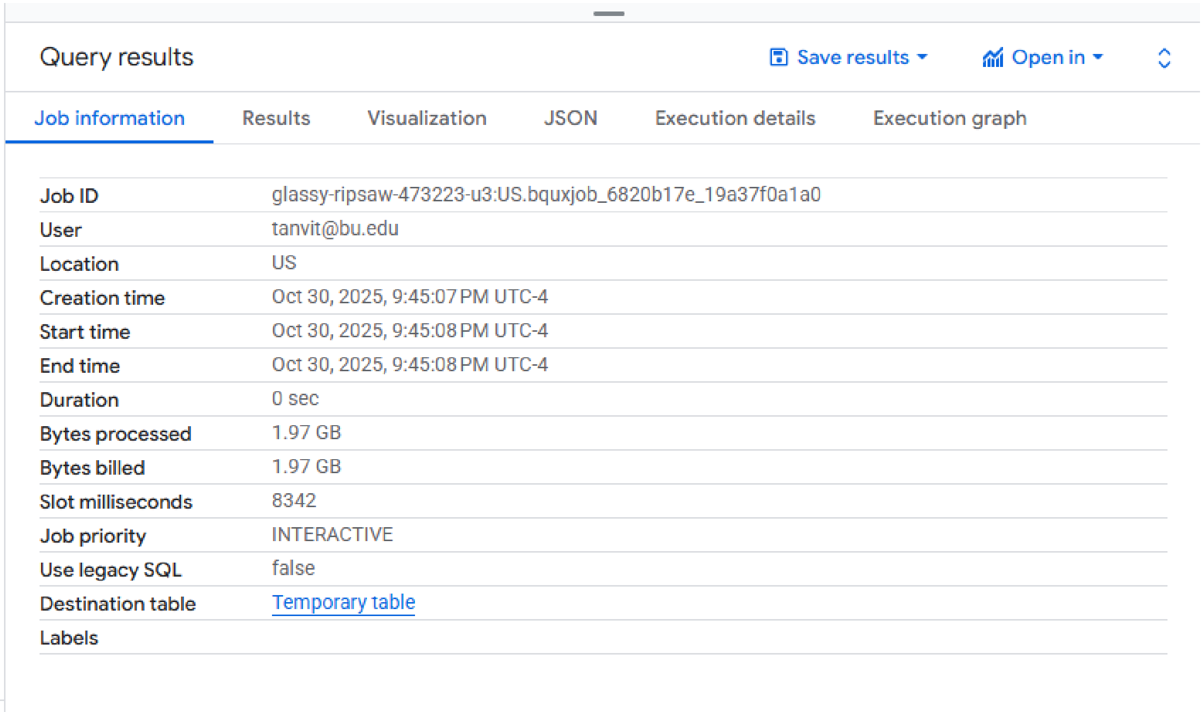
IO

Rows returned	**26**
Rows read	**52,865,963**
Bytes read	**96.36** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

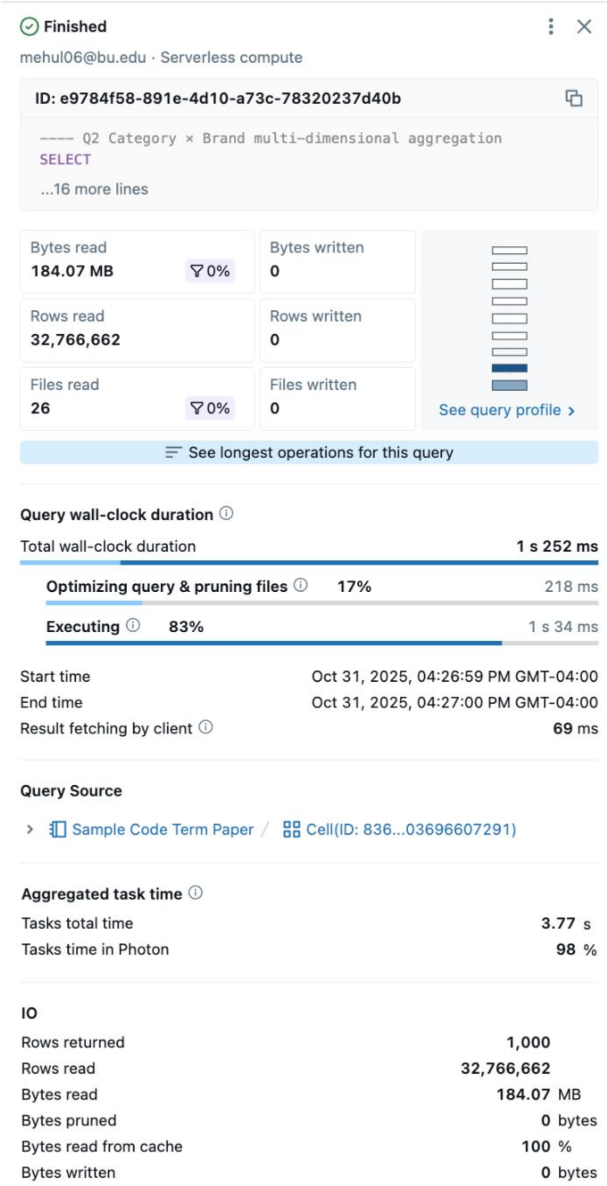
Category × Brand Aggregation

BigQuery:



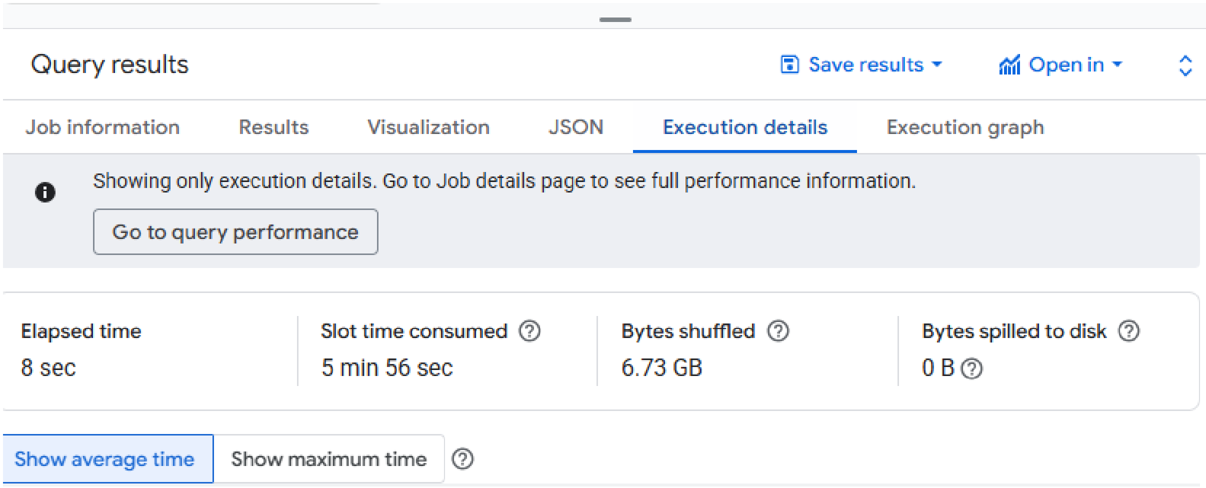


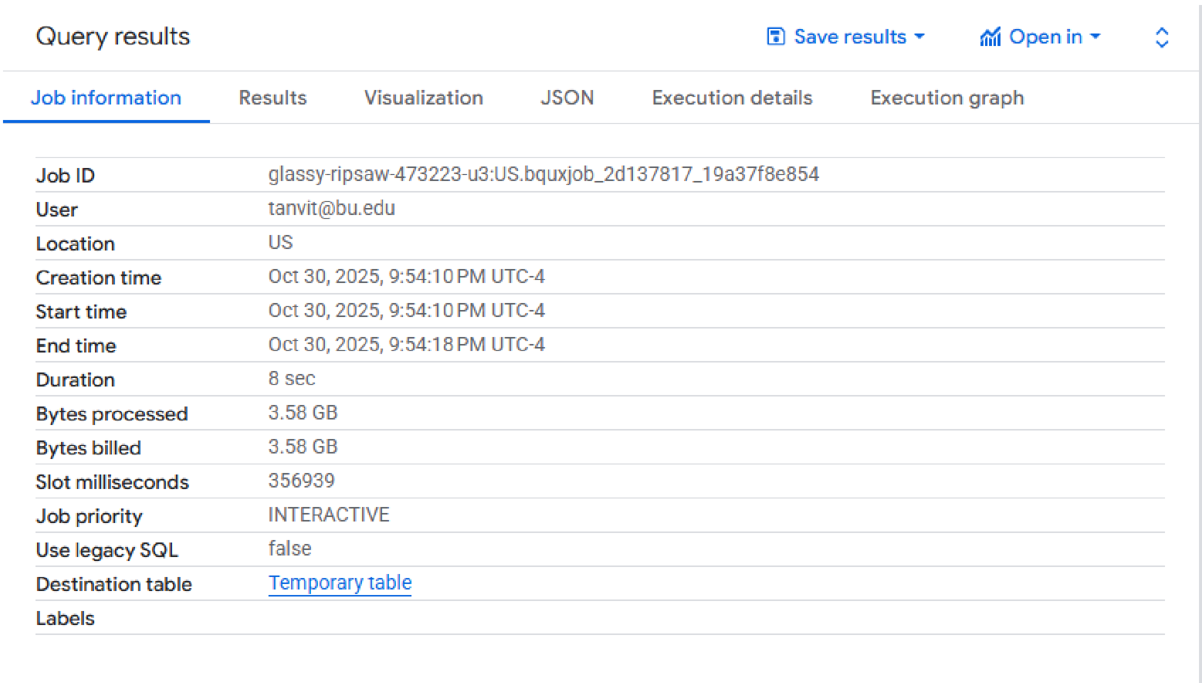
Databricks:



User-Level Monetization

BigQuery:





Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: c4a61f88-f47b-4e08-b9af-c1f7d95f280a

```
-- Q3 User-level monetization
SELECT
```

...12 more lines

Bytes read 658.28 MB 0%	Bytes written 0	
Rows read 52,865,963	Rows written 0	
Files read 26 0%	Files written 0	See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **3 s 762 ms**

Optimizing query & pruning files 5% 206 ms

Executing 95% 3 s 556 ms

Start time	Oct 31, 2025, 04:27:01 PM GMT-04:00
End time	Oct 31, 2025, 04:27:04 PM GMT-04:00
Result fetching by client	**81** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607292)

Aggregated task time

Tasks total time	**18.70** s
Tasks time in Photon	**99** %

IO

Rows returned	**1,000**
Rows read	**52,865,963**
Bytes read	**658.28** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

Session Analytics

Big Query

Query results | Save results | Open in

Job information | Results | Visualization | JSON | **Execution details** | Execution graph

Showing only execution details. Go to Job details page to see full performance information.

Go to query performance

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
7 sec	9 min 52 sec	15.18 GB	0 B

Show average time | Show maximum time

Stages | Working timing

Query results | Save results | Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_21a0a8e8_19a37f9e462
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:55:14 PM UTC-4
Start time	Oct 30, 2025, 9:55:14 PM UTC-4
End time	Oct 30, 2025, 9:55:22 PM UTC-4
Duration	7 sec
Bytes processed	3.48 GB
Bytes billed	3.49 GB
Slot milliseconds	592892
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks

Finished

mehul06@bu.edu · Serverless compute

ID: 66ee892e-f3fe-4735-9f53-1f9c8513eeb5

```
-- Q4 Session analytics (basket size / value)
SELECT
```

...11 more lines

Bytes read **627.35 MB** 0%	Bytes written **0**
Rows read **52,865,963**	Rows written **0**
Files read **26** 0%	Files written **0**

See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **7 s 102 ms**

Optimizing query & pruning files **3%** 205 ms

Executing **97%** 6 s 897 ms

Start time	Oct 31, 2025, 04:27:05 PM GMT-04:00
End time	Oct 31, 2025, 04:27:12 PM GMT-04:00
Result fetching by client	**74** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607293)

Aggregated task time

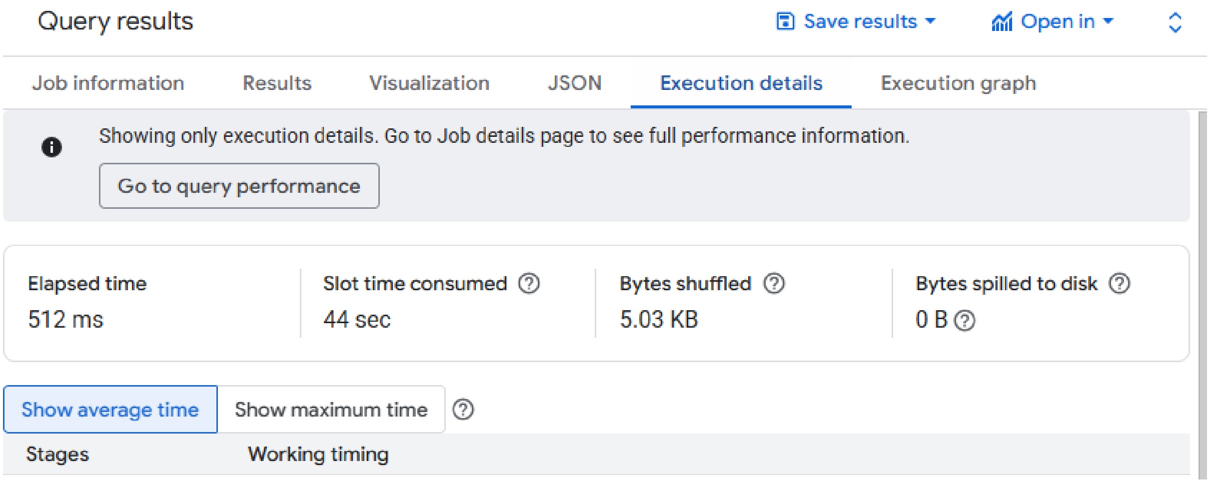
Tasks total time	**39.45** s
Tasks time in Photon	**100** %

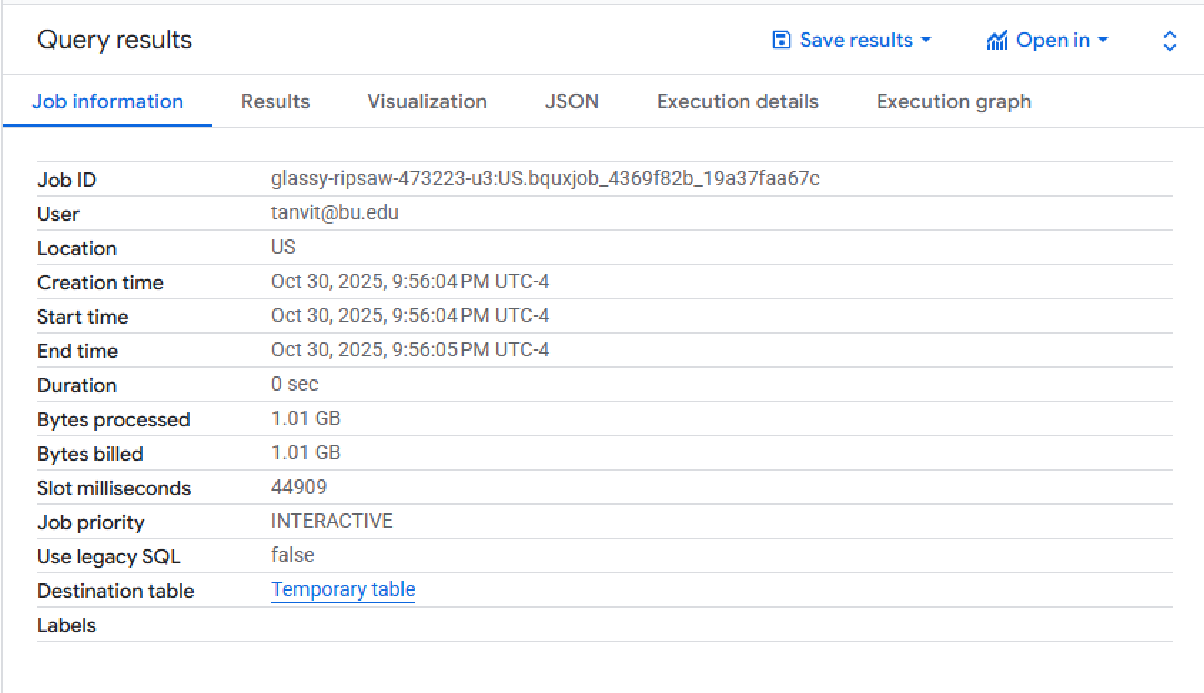
IO

Rows returned	**1,000**
Rows read	**52,865,963**
Bytes read	**627.35** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

Trend and Anomaly Detection

Big Query:





Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: 44ccdb28-623f-4faf-bf95-dbe987d86d9b

```
-- Q5 Trend + anomaly detection (7-day rolling z-score)
WITH daily AS (
```

...29 more lines

Bytes read **96.36 MB** 0%	Bytes written **0**
Rows read **52,865,963**	Rows written **0**
Files read **26** 0%	Files written **0**

See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **563 ms**

Optimizing query & pruning files **34%** 193 ms

Executing **66%** 370 ms

Start time	Oct 31, 2025, 04:27:13 PM GMT-04:00
End time	Oct 31, 2025, 04:27:13 PM GMT-04:00
Result fetching by client	**56** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607294)

Aggregated task time

Tasks total time	**673** ms
Tasks time in Photon	**93** %

IO

Rows returned	**26**
Rows read	**52,865,963**
Bytes read	**96.36** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

Scalability Experiment

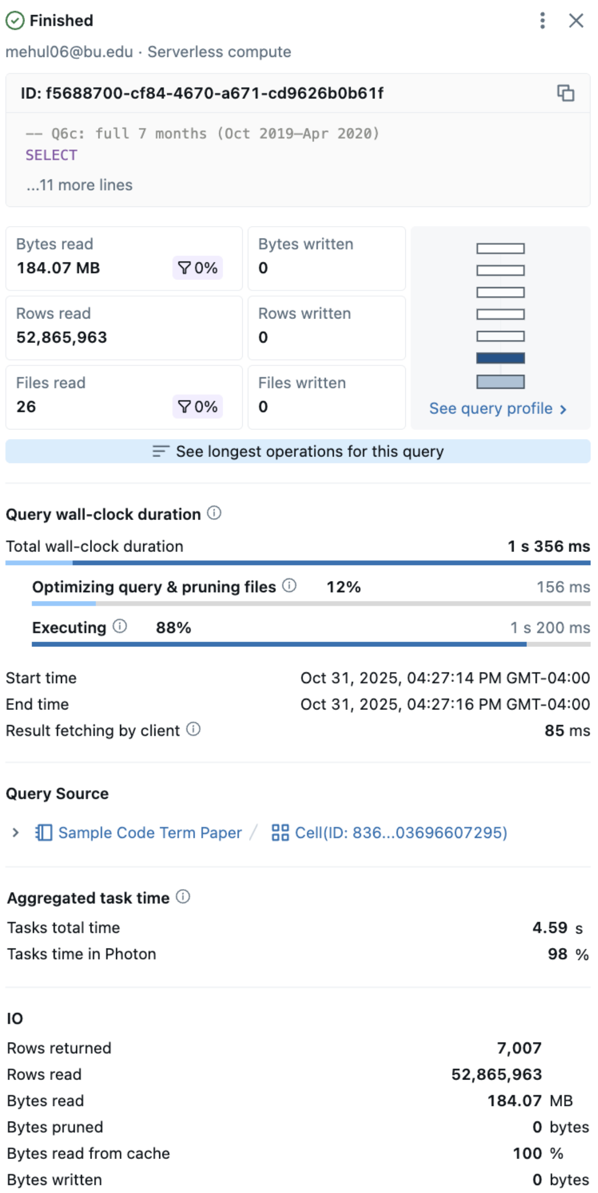
Big Query:

All results

Elapsed time	Statements processed	Job status
2 sec	3	SUCCESS

Status	End time	SQL	Action
	9:56 PM [2:1]	SELECT category_code, brand, SUM(price) AS revenue	View results
	9:56 PM [8:1]	SELECT category_code, brand, SUM(price) AS revenue	View results
	9:56 PM [14:1]	SELECT category_code, brand, SUM(price) AS revenue	View results

Databricks:



Top-K Products by Revenue

BigQuery:

Query results Save results Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Showing only execution details. Go to Job details page to see full performance information.

Go to query performance

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
534 ms	14 sec	182.23 MB	0 B

Show average time | Show maximum time

Stages | Working timing

Query results Save results Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_16200b9_19a37fbb66a
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:57:14 PM UTC-4
Start time	Oct 30, 2025, 9:57:14 PM UTC-4
End time	Oct 30, 2025, 9:57:14 PM UTC-4
Duration	0 sec
Bytes processed	1.1 GB
Bytes billed	1.1 GB
Slot milliseconds	14198
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks:

Finished

mehul06@bu.edu · Serverless compute

ID: a1b0f40b-6c28-4434-b456-edd3dc7e9b50

```
-- Q7 Top-K products by revenue (exact)
SELECT
```

...9 more lines

Bytes read 214.90 MB 0%	Bytes written 0	
Rows read 52,865,963	Rows written 0	
Files read 26 0%	Files written 0	See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration **1 s 253 ms**

Optimizing query & pruning files **16%** 197 ms

Executing **84%** 1 s 56 ms

Start time	Oct 31, 2025, 04:27:16 PM GMT-04:00
End time	Oct 31, 2025, 04:27:18 PM GMT-04:00
Result fetching by client	**110** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607297)

Aggregated task time

Tasks total time	**4.25** s
Tasks time in Photon	**98** %

IO

Rows returned	**100**
Rows read	**52,865,963**
Bytes read	**214.90** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

Brand-Level Price Outliers

BigQuery

Query results | Save results | Open in

Job information | Results | Visualization | JSON | **Execution details** | Execution graph

Showing only execution details. Go to Job details page to see full performance information.

Go to query performance

Elapsed time	Slot time consumed	Bytes shuffled	Bytes spilled to disk
564 ms	49 sec	23.02 MB	0 B

Show average time | Show maximum time

Query results | Save results | Open in

Job information | Results | Visualization | JSON | Execution details | Execution graph

Job ID	glassy-ripsaw-473223-u3:US.bquxjob_22188527_19a37fc617e
User	tanvit@bu.edu
Location	US
Creation time	Oct 30, 2025, 9:57:58 PM UTC-4
Start time	Oct 30, 2025, 9:57:58 PM UTC-4
End time	Oct 30, 2025, 9:57:58 PM UTC-4
Duration	0 sec
Bytes processed	1.53 GB
Bytes billed	1.53 GB
Slot milliseconds	49768
Job priority	INTERACTIVE
Use legacy SQL	false
Destination table	Temporary table
Labels	

Databricks

Finished

mehul06@bu.edu · Serverless compute

ID: bb9cca4b-bc0d-4777-b312-7e99ef8a901d

```
-- Q8 Brand-level price outliers (z > 3)
WITH brand_stats AS (
```

...24 more lines

Bytes read	Bytes written
423.89 MB 0%	**0**

Rows read	Rows written
91,267,040	**0**

Files read	Files written
52 0%	**0**

See query profile ›

See longest operations for this query

Query wall-clock duration

Total wall-clock duration		**2 s 111 ms**
Optimizing query & pruning files	**11%**	239 ms
Executing	**89%**	1 s 872 ms

Start time	Oct 31, 2025, 04:27:18 PM GMT-04:00
End time	Oct 31, 2025, 04:27:20 PM GMT-04:00
Result fetching by client	**62** ms

Query Source

› Sample Code Term Paper / Cell(ID: 836...03696607298)

Aggregated task time

Tasks total time	**7.91** s
Tasks time in Photon	**99** %

IO

Rows returned	**100**
Rows read	**91,267,040**
Bytes read	**423.89** MB
Bytes pruned	**0** bytes
Bytes read from cache	**100** %
Bytes written	**0** bytes

VISULIZATION PART:

Daily Revenue Trend

What we did: Used Day as the Dimension and Revenue + Orders as Metrics.

What it shows: Daily revenue and order count across the month.

Insight: Identifies sales spikes or drops over time (helps track performance trends).

Category × Brand Revenue

What we did: Used category_code as Dimension, brand as Breakdown, and Revenue as Metric.

What it shows: How much revenue each brand contributes within different product categories.

Insight: Reveals top-performing brand–category combinations driving sales.

Top Users by Total Spending

What we did: Used user_id as Dimension and Revenue + Avg Ticket as Metrics.

What it shows: The highest-spending customers and their average order values.

Insight: Highlights your most valuable users and potential VIP customers.

Spending Distribution Histogram

What we did: Used Revenue as Dimension and COUNT_DISTINCT(user_id) as Metric.

What it shows: The number of users grouped by spending range.

Insight: Shows how spending is distributed whether most users spend small amounts or a few spend a lot.

Session Analytics

What we did: Used user_session as Dimension and Unique Products, Orders, and Session Value as Metrics.

What it shows: The value of each user session number of products viewed, items bought, and total spend.

Insight: Measures engagement efficiency per session and identifies sessions with the highest revenue.

Anomalies (Revenue vs Rolling Average)

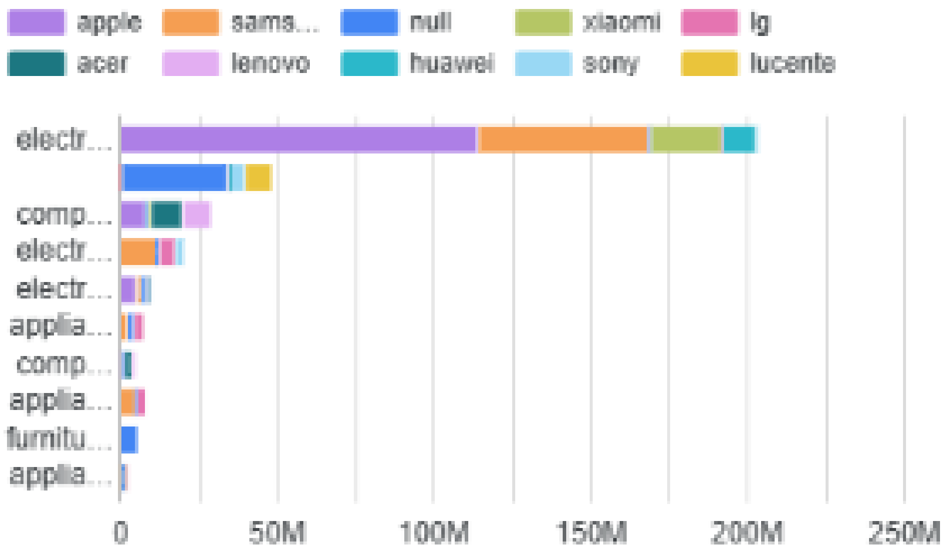
What we did: From the BigQuery daily_metrics view plotted revenue and ma7 (7-day moving average).

What it shows: Actual revenue compared to its smoothed 7-day trend.

Insight: Highlights unusual spikes or drops in daily revenue useful for anomaly detection and demand forecasting.

LOOKER STUDIO RESULTS:

Category × Brand Revenue

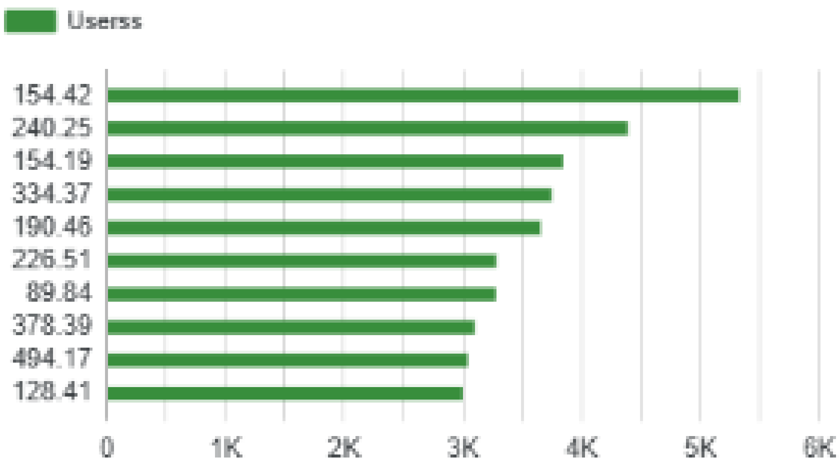


Top users by total spending

	user_id	Revenue ▾	Avg Tic...
1.	568797382	212,631.41	1,012.53
2.	516054872	191,874.31	1,148.95
3.	569335945	172,308.04	121.09
4.	562850008	168,034.34	852.97
5.	568793129	164,944.46	420.78
6.	554501441	158,196.68	1,068.9
7.	546635249	154,455.24	718.4

1 - 100 / 268643

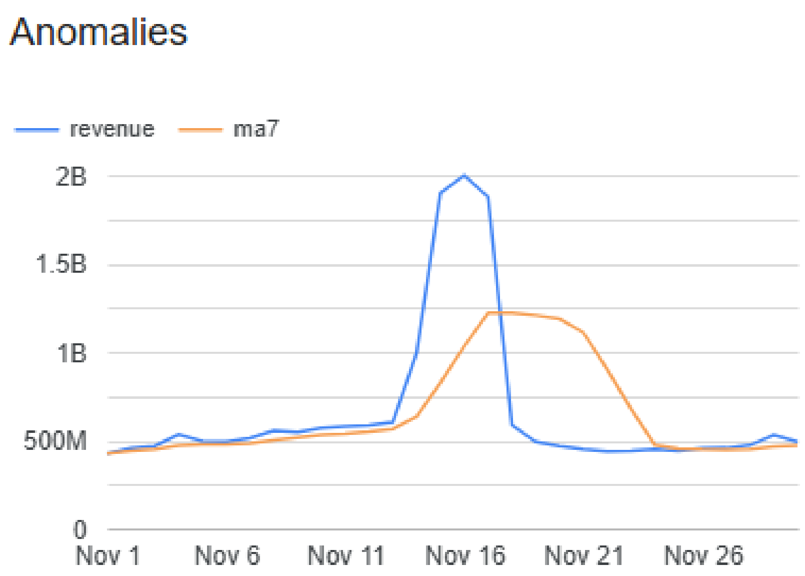
Spending distribution histogram



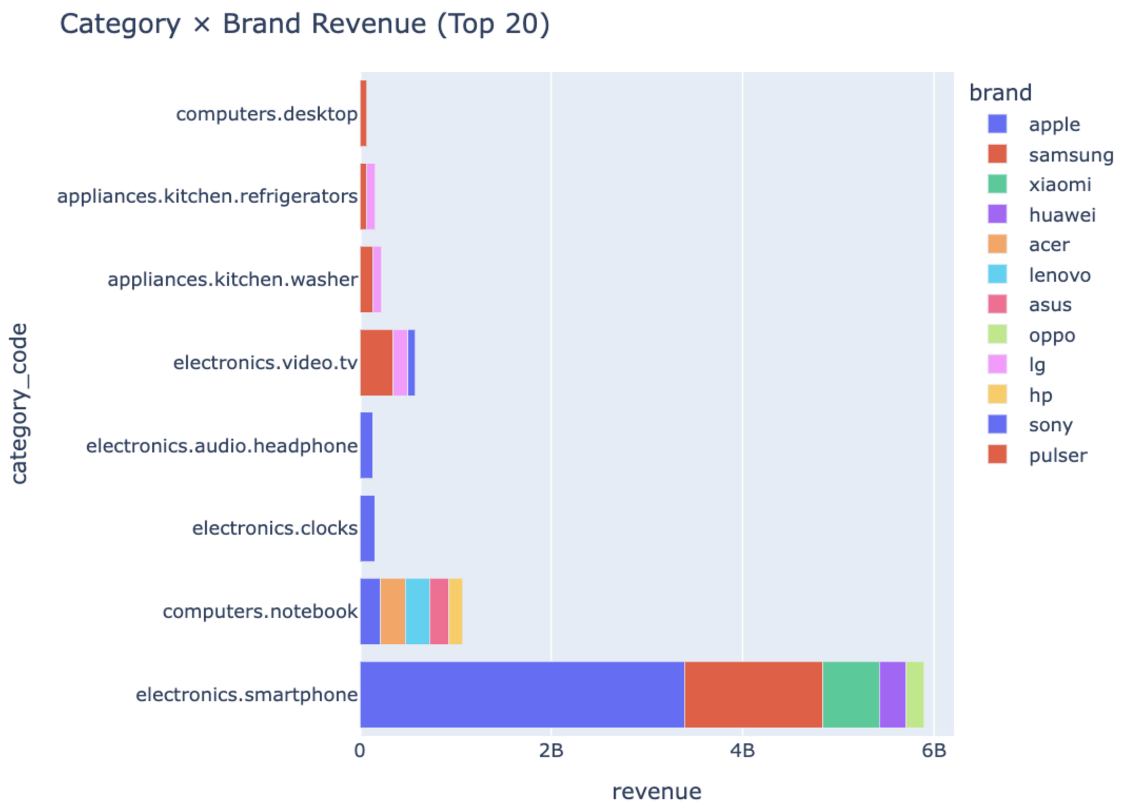
Session Analytics

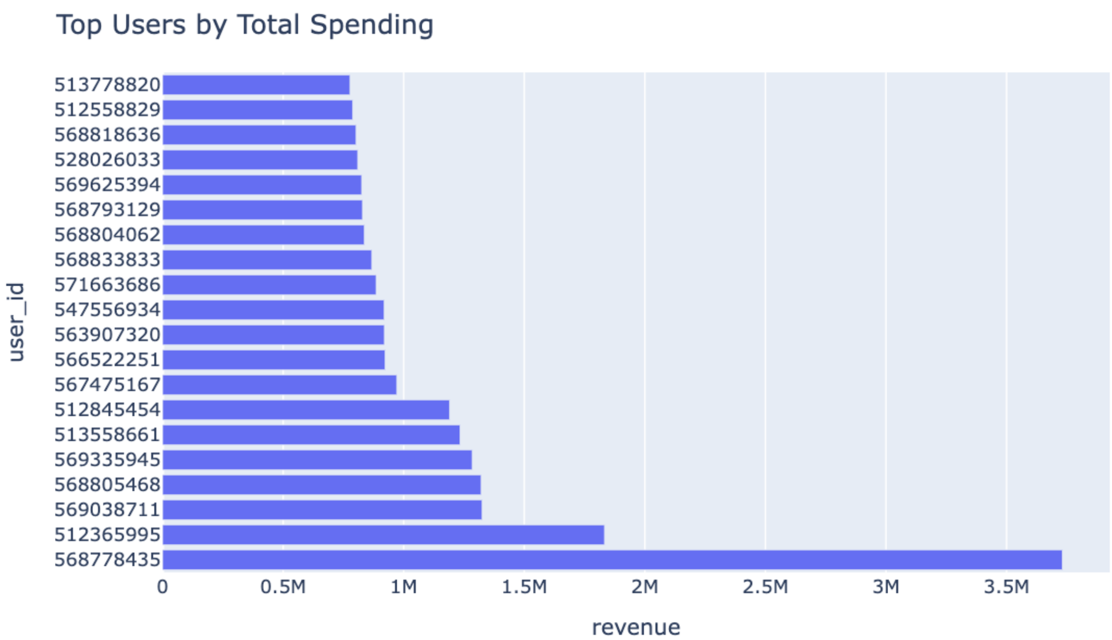
	user_session	Unique Prod...	Orders	Ses... ▾
1.	0c307610-aa79...	543	918	360,819.7
2.	ef2fd879-4b1a-...	41	246	331,040.05
3.	c6be5380-8322...	49	178	291,139.89
4.	123c2880-3db4...	139	360	274,889.93
5.	1d34878d-1a42...	9	251	272,767.15
6.	84f2e900-9da5-...	3	204	248,544.6
7.	37738e78-398a...	100	305	245,503.29

1 - 100 / 2000000+ ‹ ›



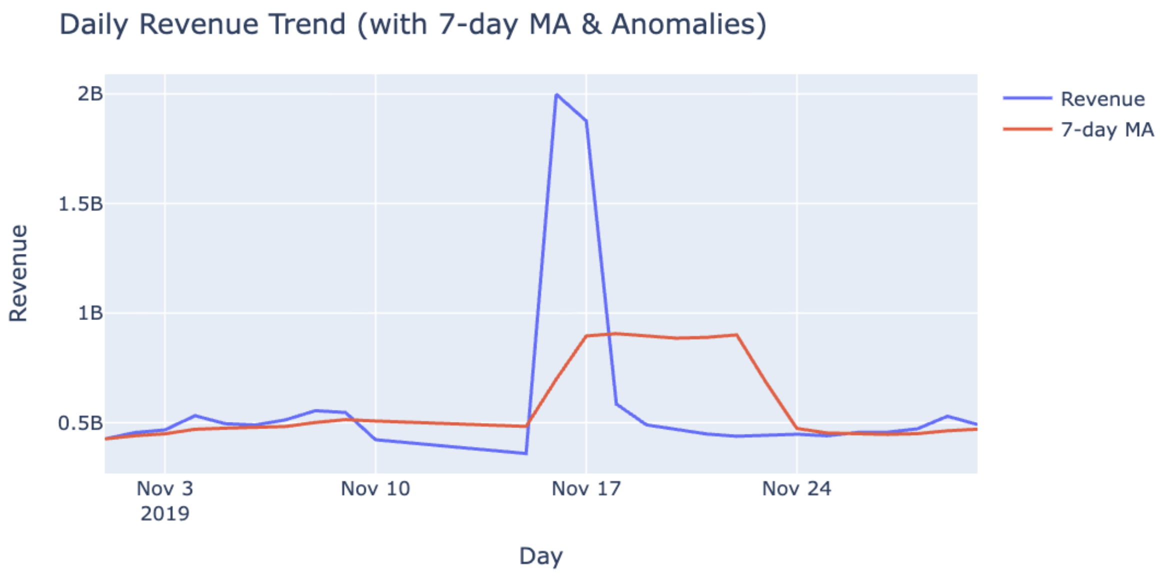
PYTHON RESULTS (Plotly):





Session Analytics (Top 100 by Value)

user_session	unique_products	orders	session_value
0c307610-aa79-bf12-	543	918	360819.6999999998
ef2fd879-4b1a-4319-	41	246	331040.05000000005
c6be5380-8322-432e	49	178	291139.88999999996
1d34878d-1a42-401b	9	251	272767.15000000014
84f2e900-9da5-c970-	3	204	248544.5999999997
1b64f998-793f-4093-	102	231	231680.63000000003
62731828-1c87-484f-	39	190	230063.1900000001
4921edd9-3cbb-43a4	161	196	224270.9099999999
fdd27028-5bb0-422f-	48	124	204457.83000000007
081a2ea6-cc00-4fb8-	89	156	199335.43999999997
fe991bee-2200-454b-	50	120	199126.86
194d4a86-f76e-4756-	178	285	198114.2000000001
e2778db6-0aee-4d95	62	240	195241.96
44f0883f-2647-4784-	50	146	189344.31
3f48e024-c25e-40eb-	59	97	184565.29999999996
1ad07d6d-05ff-4ebb-	42	133	183670.61000000007



RESULT EXPLANATIONS:

Query / Task	Big Query Execution Time	Databricks Execution Time
Table Creation (events_clean)	~18s	~23 s
EDA Queries (0–4)	12sec total	9sec total
Daily Revenue Trend (5)	~761ms	~648s
Category × Brand Aggregation	~351ms	~1s
User-Level Monetization	~8 s	~3s
Session Analytics	~7s	~7.2s
Anomaly Detection	~513ms	~563ms
Scalability Test (1-Month Data)	~2s	~1 s
Scalability Test (3-Month Data)	~534ms	~2s
Scalability Test (7-Month Data)	~564ms	~3s

VISUALIZATION RESULTS:

Chart	What It Shows	Key Insight
Daily Revenue Trend	Revenue & order volume per day	Sales spikes or slow periods
Category × Brand Revenue	Revenue split by brand within each category	Top-performing brand-category pairs
Top Users by Spending	Total and average spend per user	Identifies VIP customers
Spending Distribution Histogram	Number of users per spending range	Reveals concentration of low vs. high spenders
Session Analytics	Products viewed, items bought, and session value	Measures engagement efficiency
Anomalies (Revenue vs Rolling Avg)	Daily revenue vs 7-day trend line	Detects unusual revenue fluctuations

Detailed Comparison and Explanation of Results:

Both BigQuery and Databricks were tested on the same 9 GB e-commerce dataset. The queries included table creation, EDA (exploratory data analysis), trend analysis, user/session analytics, anomaly detection, and scalability testing. The results confirm that both platforms delivered accurate outputs, but their performance varied depending on the workload type.

1. **Table Creation (events_clean)**

BigQuery: ~18 seconds

Databricks: ~23 seconds

BigQuery completed table creation slightly faster. It directly pulls data from Google Cloud Storage and automatically optimizes schema and partitioning, while Databricks required CSV reading and transformation into a Delta table, adding a few extra seconds.

2. **EDA Queries (Q0–Q4)**

BigQuery: 12 seconds total

Databricks: 9 seconds total
Databricks outperformed here because once data was loaded into memory, it executed smaller, repeated queries faster. Spark's in-memory processing provided an edge over BigQuery's on-demand query execution.

3. **Daily Revenue Trend (Q5)**

BigQuery: ~761 milliseconds

Databricks: ~648 milliseconds

Both systems were efficient. Databricks performed marginally better because the operation involved calculating aggregates over limited time windows, which suited Spark's caching and partition-based parallelism.

4. **Category × Brand Aggregation**

BigQuery: ~351 milliseconds

Databricks: ~1 second

BigQuery handled large group-by operations faster due to its Dremel engine, which parallelizes across thousands of slots automatically. Databricks needed more shuffle operations to reorganize data before aggregation, slightly slowing it down.

5. **User-Level Monetization**

BigQuery: ~8 seconds

Databricks: ~3 seconds

Databricks was faster in this query since user-level aggregation benefited from Spark's distributed joins and cached intermediate results. It's more efficient for iterative or session-based workloads once data is loaded in memory.

6. **Session Analytics**

BigQuery: ~7 seconds

Databricks: ~7.2 seconds

Both platforms performed almost identically. This query required grouping by session IDs, which is high in cardinality but evenly distributed, making it well-optimized on both systems.

7. **Anomaly Detection**

BigQuery: ~513 milliseconds

Databricks: ~563 milliseconds

BigQuery's analytical functions (LAG, STDDEV, AVG) were slightly faster because it executes window functions natively at scale. Databricks performed very closely, showing Spark SQL's efficiency for analytical workloads.

8. **Scalability Tests**

For smaller ranges (1 month), Databricks performed faster as data was already cached. However, as data size increased (3–7 months), BigQuery scaled more efficiently — it automatically distributed queries across multiple nodes without manual cluster tuning.

9. **Key Insights**

BigQuery Strengths:

Consistently faster for large scans and heavy aggregations.

Auto-scaling ensures stable performance even as data grows.

Best suited for ad-hoc analytics, dashboards, and large-scale reports.

Built-in cost and performance transparency (bytes processed, slot time).

Databricks Strengths:

Better for iterative, in-memory analytics and custom visualization (Plotly, PySpark).

Ideal for data science, ML pipelines, and streaming workloads.

Cluster resources can be adjusted for fine-grained control

10. **Overall Conclusion**

Aspect	BigQuery	Databricks
Performance on Large Data	Faster (auto-optimized, scalable)	Slower for very large queries
Performance on Cached Data	Slightly slower	Faster (Spark in-memory advantage)
Ease of Use	No setup, simple SQL	Requires cluster configuration
Visualization	Limited (Looker/Charts)	Excellent (Plotly, Matplotlib)
Scalability	Automatic and seamless	Manual tuning needed
Best For	Large-scale analytics, BI dashboards	Interactive data science, ML workflows

In this experiment:

- BigQuery excelled in speed, scalability, and simplicity, best for analytics at scale.
- Databricks excelled in flexibility, caching, and visualization, best for exploration and experimentation.

Both platforms are powerful, choosing between them depends on whether your goal is rapid analytics.
