Data Analytics

Version 2026-11 · verified 2026-11-01

Turn raw data into answers you can trust — grain, cleaning, summary stats, GROUP BY, joins, windows, rates, A/B tests and charts.

Contents

Mental model 2

From question to metric 2

Tidy data and types 3

Cleaning 4

Descriptive statistics 6

Aggregating with GROUP BY 7

Joins without double counting 8

Window functions and time 9

Rates and ratios 10

Correlation and experiments 12

Choosing a chart 13

Further reading 14

Mental model

Every analysis is a question, a table whose rows mean one known thing (its *grain*), and an aggregation that collapses those rows into an answer. Most wrong numbers are not math errors: they come from rows silently duplicated by a join, dropped by a filter or a missing value, or from averaging numbers that should have been summed first. Check the row count after every step and the arithmetic mostly takes care of itself.

From question to metric

Step	Ask
Question	What decision will the answer change?
Metric	Exact formula: numerator and denominator
Grain	What does one row represent?
Population	Which rows count, over which time window?
Comparison	Versus what: last period, control, target?

“Are users engaged?” cannot be computed. “What share of users who signed up in March placed an order within 7 days?” can, because it names the population, the event, the window, and the denominator. Write the definition down before querying, so the result cannot quietly reshape it.

```
Metric:  7-day activation rate
Numer.:  signups with >= 1 order in 7 days
Denom.:  all signups in the window
```

```
Grain:   one row per user
Window:  signups 2026-03-01 .. 2026-03-31
```

> **Gotcha:** Comparing a period in progress with finished ones — month-to-date against last full month — always shows a “drop”. Compare equal lengths (first 10 days vs first 10 days) or complete periods only.

Tidy data and types

Concept	Meaning
Tidy table	One variable per column, one observation per row
Dimension	A category you group by: region, plan
Measure	A number you aggregate: revenue, duration
Ordinal	Ordered labels: small < medium < large
Discrete / continuous	Counted (orders) vs measured (seconds)

Spreadsheets are often *wide* — one column per month — which reads well but analyzes badly: every new month is a new column. Reshape to *long* form, where the month is a value, so grouping, filtering and charting work the same for every period.

```
import pandas as pd


wide = pd.DataFrame({
  "region": ["EU", "US"],
```

```
  "jan": [100, 80],
  "feb": [120, 90],
})
long = wide.melt(
  id_vars="region",
  var_name="month",
  value_name="sales",
)
print(len(long))  # 4: one row per region-month
```

> **Gotcha:** Codes that look numeric — ZIP codes, account and phone numbers — get parsed as numbers: `02134` becomes `2134`, and they show up in sums. Read them as text: `pd.read_csv(path, dtype={"zip": "str"})`.

Cleaning

Problem	pandas fix
Missing values	`df.isna().sum()`, then drop, fill or keep
Duplicate rows	`df.drop_duplicates()`
Numbers stored as text	`pd.to_numeric(s, errors="coerce")`
Dates stored as text	`pd.to_datetime(s, errors="coerce")`
Inconsistent labels	`s.str.strip().str.lower()`

Profile before you clean: row count, nulls per column, distinct values, min and max. Every fix is a decision that changes the result, so keep it in

code rather than editing the source file by hand. `errors="coerce"` turns unparseable values into missing ones — count them afterwards instead of assuming there were none.

```
import pandas as pd


df = pd.DataFrame({
  "id": [1, 2, 2, 3],
  "amount": ["10", "n/a", "n/a", "7.5"],
})
df = df.drop_duplicates()
df["amount"] = pd.to_numeric(
  df["amount"], errors="coerce"
)
print(df["amount"].isna().sum())  # 1
print(df["amount"].mean())        # 8.75
```

> **Gotcha:** `mean()` skips missing values by default (`skipna=True`), and SQL `AVG` ignores `NULL` the same way. 8.75 is the average of the two known amounts. If missing means zero, `fillna(0)` first — the answer becomes 5.83.

Descriptive statistics

Statistic	Tells you
Count, min, max	Sanity check — do this first
Mean	Typical value, but dragged by extremes
Median (p50)	Middle value; robust to outliers
p90, p99	The tail: slowest, largest, worst
Standard deviation	Typical distance from the mean

Business data is usually skewed: revenue per customer, latency, session length. When the mean sits far above the median, a few large values dominate it, so report the median plus a high percentile. `df.describe()` gives count, mean, std, min, quartiles and max in one call.

```
import pandas as pd


s = pd.Series([20, 22, 25, 30, 900])
print(s.mean())          # 199.4
print(s.median())        # 25.0
print(s.quantile(0.9))   # 552.0
```

> **Tip:** Plot a histogram before trusting any summary. Anscombe's quartet is four datasets with near-identical means, variances and correlations that look completely different when drawn.

Aggregating with GROUP BY

Need	SQL
Rows per group	`COUNT(*)`
Non-null values	`COUNT(col)`
Distinct entities	`COUNT(DISTINCT col)`
Filter rows before grouping	`WHERE`
Filter groups after grouping	`HAVING`

`GROUP BY` collapses the table to one row per group; every other column must be inside an aggregate. Pick the count deliberately: `COUNT(*)` counts rows, `COUNT(col)` skips `NULL`s, and `COUNT(DISTINCT user_id)` counts people rather than events.

```
CREATE TABLE sales
  (day TEXT, region TEXT, uid INT, amt REAL);
INSERT INTO sales VALUES
  ('2026-03-01', 'EU', 1, 50),
  ('2026-03-01', 'EU', 2, NULL),
  ('2026-03-02', 'US', 3, 30),
  ('2026-03-02', NULL, 1, 20);

SELECT region,
  COUNT(*) AS n,          -- EU: 2
  COUNT(amt) AS paid,     -- EU: 1
  SUM(amt) AS revenue     -- EU: 50
```

```
FROM sales
GROUP BY region;          -- NULL is its own group
```

> **Gotcha:** SQL keeps `NULL` keys as a group; pandas `groupby` drops them by default (`dropna=True`). The same data gives different totals in each tool — pass `dropna=False` to match SQL.

Joins without double counting

Join	Keeps
`INNER JOIN`	Only rows with a match on both sides
`LEFT JOIN`	Every left row; `NULL`s where no match
`LEFT JOIN … WHERE r.id IS NULL`	Left rows with no match (anti-join)
One-to-many	Each left row repeated once per match

Before joining, know whether the key is unique on each side. Joining users to sales gives one row per sale, so any user-level number summed afterwards is counted once per sale. Aggregate the “many” side to the grain you need first, then join. In pandas, `merge(..., validate="many_to_one")` raises `MergeError` if the right side has duplicate keys.

```sql
CREATE TABLE users (uid INT, plan TEXT);
INSERT INTO users VALUES
  (1, 'pro'), (2, 'free'), (3, 'pro');

-- Aggregate the "many" side first
SELECT u.plan, SUM(t.spend) AS spend
FROM users u
LEFT JOIN (
  SELECT uid, SUM(amt) AS spend
  FROM sales GROUP BY uid
) t ON t.uid = u.uid
GROUP BY u.plan;  -- free: NULL, pro: 100
```

> **Gotcha:** A `WHERE` condition on the right table of a `LEFT JOIN` turns it back into an inner join: unmatched rows have `NULL` there and fail the test. Put that condition in the `ON` clause instead.

Window functions and time

Expression	Gives
`SUM(x) OVER (ORDER BY day)`	Running total
`LAG(x) OVER (ORDER BY day)`	Previous row, for growth
`ROW_NUMBER() OVER (PARTITION BY …)`	First or latest row per entity
`AVG(x) OVER (… ROWS 6 PRECEDING)`	7-row moving average

A window function computes across related rows without collapsing them, so each row keeps its detail and gains context. `PARTITION BY` splits rows into groups; `ORDER BY` sets the sequence inside each. They run in SQLite (3.25+), PostgreSQL, and every major warehouse.

```
WITH daily AS (
  SELECT day, SUM(amt) AS rev
  FROM sales GROUP BY day
)
SELECT day, rev,
  LAG(rev) OVER (ORDER BY day) AS prev,
  SUM(rev) OVER (ORDER BY day) AS running
FROM daily;
-- 2026-03-01 | 50 | NULL | 50
-- 2026-03-02 | 50 | 50   | 100
```

> **Gotcha:** `LAG` returns the previous *row*, not the previous *day*. A day with no sales has no row, so "day over day" silently compares with an older day. Left-join onto a complete calendar first.

Rates and ratios

Do	Don't
`SUM(num) / SUM(den)`	Average the per-group ratios

Do	Don’t
Show the denominator beside every rate	Quote a % from a tiny group
Compare rates within like segments	Compare raw counts across unequal groups

An average of ratios weights a group of 10 the same as a group of 1,000. Segment mix matters too: a rate can improve in every segment yet fall overall when traffic shifts toward a weaker segment (*Simpson’s paradox*). When totals and segments disagree, the segments usually tell the truth.

```
CREATE TABLE visits
  (src TEXT, n INT, conv INT);
INSERT INTO visits VALUES
  ('ads', 10, 5), ('seo', 1000, 100);

SELECT
  AVG(conv * 1.0 / n) AS avg_ratio,  -- 0.3
  SUM(conv) * 1.0 / SUM(n) AS rate   -- 0.104
FROM visits;
```

> **Gotcha:** In SQLite and PostgreSQL, dividing two integers truncates: `SELECT 3 / 4` returns `0`. Multiply by `1.0` (or cast) before dividing, as above, or every conversion rate comes out as zero.

Correlation and experiments

Concept	Meaning
Correlation (r)	Linear co-movement, from −1 to 1
Confounder	A hidden variable driving both
A/B test	Random assignment isolates the cause
p-value	Chance of a gap this large if there were no effect
Confidence interval	Plausible range for the true effect

Observational data shows association: ice-cream sales and drownings both rise in summer. Only random assignment supports “B caused the lift”. Fix the metric, sample size and duration before starting, and report the effect size with its interval, not just whether $p < 0.05$.

```
from scipy import stats


# [converted, not converted] per variant
table = [
  [120, 880],  # A: 12.0%
  [150, 850],  # B: 15.0%
]
res = stats.chi2_contingency(table)
print(round(res.pvalue, 3))  # 0.058
```

> **Warning:** Checking results daily and stopping the first time $p < 0.05$ (“peeking”) pushes the false-positive rate far above 5%. Decide the sample size up front and read the result once.

Choosing a chart

Question	Chart
How did it change over time?	Line
How do categories compare?	Bar, sorted, axis from zero
How are values distributed?	Histogram or box plot
Do two measures move together?	Scatter
What makes up the whole?	Stacked bar, few parts

One message per chart, stated in the title (“Revenue up 2.1× since January”), with units on the axis. Label series directly rather than by color alone, so the chart survives grayscale printing and color-blind readers.

```
import pandas as pd


df = pd.DataFrame({
  "month": ["Jan", "Feb", "Mar"],
  "rev": [100, 140, 210],
})
ax = df.plot.bar(
  x="month", y="rev", legend=False,
```

```
  title="Revenue up 2.1x since Jan",
)
ax.set_ylabel("Revenue, $k")
```

> **Gotcha:** A bar's length encodes the value, so a bar axis that starts above zero exaggerates differences — 102 vs 100 can look doubled. Line charts may zoom; bars may not.

Further reading

- pandas 3.0 — What's new
- pandas User Guide — Group by
- SQLite — Window Functions
- PostgreSQL — Aggregate Functions
- SciPy — chi2_contingency
- NIST/SEMATECH e-Handbook of Statistical Methods