DegreeLive

Bachelor of Commerce (B2001) | Level 1 • Part B Major | Monash University

Introduction to data analysis

Course Code: ETC1010 • Module 2 Notes

Module 2: Data Wrangling, Relational Joins, Dates, Strings & Scraping Principles

2.1 Grammar of Data Manipulation: Row, Column & Grouping Verbs

Once raw data have been imported into a tidy tabular representation, analytical workflows rely on a composable algebra of data manipulation verbs. Rather than inventing custom imperative loops for every filtering or aggregation task, modern tabular frameworks (exemplified by `dplyr` in R and `polars`/`pandas` in Python) decompose single-table transformations into five orthogonal verbs chained sequentially via a forward pipe operator (`|>`):

- **Row Filtering (`filter()`):** Retains observations satisfying one or more logical predicates while preserving column schema. Relational operators include equality (`==`), inequality (`!=`), inequalities (`<`, `<=`, `>`, `>=`), and set membership (`%in%`).
- **Column Selection (`select()`):** Subsets, reorders, or renames variables using explicit names, contiguous ranges, or predicate helpers (`starts_with()`, `ends_with()`, `where(is.numeric)`).
- **Feature Engineering (`mutate()`):** Appends newly derived columns or transforms existing columns while preserving exact row count and ordering.
- **Row Ordering (`arrange()`):** Sorts observations lexicographically or numerically by one or more key columns (ascending by default, or descending via `desc()`) for ranking, lead/lag window calculations, and presentation.
- **Aggregation (`summarise()`):** Collapses multiple rows into single-row summaries using vector-to-scalar reduction functions (`sum()`, `mean()`, `median()`, `sd()`, `n()`, `n_distinct()`).

Three-Valued Logic and Missing-Value Traps

A subtle source of silent data corruption in row filtering arises from **three-valued Kleene logic** (`TRUE`, `FALSE`, `NA`). Because `NA` represents an unknown quantity, comparing `x == NA` evaluates to `NA` rather than `TRUE`, causing `filter(x == NA)` to return zero rows. Analysts must instead invoke the explicit predicate `is.na(x)` or `!is.na(x)`.

Expression	Evaluated Result	Logical Interpretation	Row Action in `filter()`
`TRUE & NA`	`NA`	Outcome depends on whether unknown operand is TRUE or FALSE	Row dropped
`FALSE & NA`	`FALSE`	Conjunction with FALSE is always FALSE regardless of unknown value	Row dropped
`TRUE | NA`	`TRUE`	Disjunction with TRUE is always TRUE regardless of unknown value	Row retained
`NA == NA`	`NA`	Two unknown values cannot be verified as equal	Row dropped
`is.na(NA)`	`TRUE`	Explicit test for missingness sentinel	Row retained

Split-Apply-Combine and Grouped Window Mutations

Pairing `group_by()` with `summarise()` or `mutate()` implements the **split-apply-combine** paradigm. Conceptually, the table is partitioned into disjoint sub-tables defined by unique combinations of grouping keys, an identical calculation is applied independently to each partition, and the results are row-bound back into a unified table.

Critically, `group_by() |> summarise()` reduces each group to one row and peels off the innermost grouping level, whereas `group_by() |> mutate()` retains every original row and computes group-wise summaries or window functions (such as within-group z-scores, cumulative sums, or period-over-period lags via `lag()`). Failing to call `ungroup()` (or using per-operation `.by = group_col`) after a grouped `mutate()` leaves latent grouping metadata attached, causing subsequent row-wise proportions or global means to compute silently within subgroups rather than across the full dataset.

Worked Pipeline Example: GST Extraction and State Revenue Aggregation

Suppose a Melbourne retail group logs 6 point-of-sale transactions with GST-inclusive gross revenue (in AUD, where Australian Goods and Services Tax is 10% on taxable items):

Txn_ID	State	Category	Gross_AUD	Units	Returned
T01	VIC	Electronics	440.00	2	FALSE
T02	VIC	Fresh Food	120.00	4	FALSE
T03	VIC	Electronics	220.00	1	TRUE
T04	NSW	Electronics	660.00	3	FALSE
T05	NSW	Electronics	330.00	1	FALSE
T06	QLD	Fresh Food	NA	2	FALSE

We execute a deterministic five-verb pipeline: (1) `filter(!Returned, !is.na(Gross_AUD))` drops returned transaction `T03` and missing-price row `T06`, leaving 4 valid sales; (2) `mutate(Net_AUD = if_else(Category == "Fresh Food", Gross_AUD, Gross_AUD / 1.10))` removes the 10% GST only from taxable Electronics while keeping GST-free Fresh Food intact, yielding net values: `T01` $= 440/1.10 = 400.00$, `T02` $= 120.00$, `T04` $= 660/1.10 = 600.00$, `T05` $= 330/1.10 = 300.00$; (3) `group_by(State)` partitions rows into `VIC` and `NSW`; (4) `summarise(Total_Net = sum(Net_AUD), Total_Units = sum(Units), Rev_Per_Unit = Total_Net / Total_Units)` yields `VIC` (Total Net $= 520.00$ AUD, 6 units, 86.67 AUD/unit) and `NSW` (Total Net $= 900.00$ AUD, 4 units, 225.00 AUD/unit); and (5) `arrange(desc(Total_Net))` places `NSW` first.

Multi-Branch Recoding (case_when) and Column-Wise Iteration (across)

Real-world wrangling rarely operates on one column or two binary branches at a time. Multi-tier business rules—such as classifying Australian enterprises under ABS headcount thresholds into Micro (< 5), Small (5–19), Medium (20–199), and Large (≥ 200) tiers—require sequential conditional evaluation via `case_when()`, where conditions are tested top-to-bottom and the first matching

predicate determines the assigned category (with an explicit `.default` fallback catching unexpected outliers or NAs). Ordering conditions from most restrictive to broadest prevents earlier broad predicates from masking later specialised tiers.

When applying identical transformations or summary metrics across dozens of columns (for example, computing both mean and standard deviation across 15 quarterly ABS expenditure categories, or replacing sentinel `-999` values across all numeric columns), copy-pasting expressions introduces maintenance errors. Column-wise selectors using `across(where(is.numeric), ...)` iterate cleanly across arbitrary column subsets using tidy selection predicates while systematically naming derived columns via templates such as `"{.col}_{.fn}"`.

2.2 Reshaping and Restructuring Data

Analytical workflows frequently alternate between *long (tidy)* representations required for plotting and regression and *wide (matrix)* representations required for correlation tables, principal component analysis, or human-readable executive cross-tabulations.

- **Wide-to-Long Pivoting (`pivot_longer()`):** Gathers multiple value-bearing columns into a key-value pair of columns (`names_to` and `values_to`). When column headers encode multiple attributes (such as `retail_VIC`, `retail_NSW`), regular expression capture groups or delimiters (`names_sep = "_"`) split composite headers cleanly into multiple tidy columns in a single pass.
- **Long-to-Wide Pivoting (`pivot_wider()`):** Spreads unique values of a categorical column (`names_from`) across new column headers populated by `values_from`. If any key combination lacks an observation, `values_fill = 0` replaces structural NAs only when absence genuinely signifies zero activity (e.g., zero units sold in a store-week) rather than unrecorded data.
- **Implicit vs. Explicit Missingness (`complete()`):** A missing cell in a wide table becomes an *implicit* missing row when dropped during pivoting. Calling `complete()` restores the full Cartesian grid of expected factor/date combinations so rolling averages or time-series lags do not skip across missing calendar intervals.

Worked Numerical Example: Pivoting and Computing Weighted State Growth

Consider a quarterly macroeconomic extract tracking Australian Gross State Product (GSP) sector contributions (in AUD billions) across Victoria (VIC) and Western Australia (WA) in wide format:

Sector	VIC_2024	VIC_2025	WA_2024	WA_2025
Financial Services	62.0	65.1	18.0	18.9
Mining & Resources	4.0	4.2	140.0	147.0
Retail & Hospitality	34.0	35.7	22.0	23.1

Step 1 (Pivot Longer with Header Splitting): Applying `pivot_longer(cols = -Sector, names_to = c("State", "Year"), names_sep = "_", values_to = "GSP_AUD_B")` converts the 3×5 wide matrix into a 12×4 tidy table with columns `Sector`, `State`, `Year`, and `GSP_AUD_B`.

Step 2 (Group-Wise Share and Period Growth Calculation): Grouping by `State` and `Year` yields state totals, and grouping by `State` and `Sector` ordered by `Year` enables percentage growth calculation:

$$g_{s,j} = \left(\frac{Y_{s,j,2025} - Y_{s,j,2024}}{Y_{s,j,2024}}\right) \times 100, \qquad w_{s,j,2024} = \frac{Y_{s,j,2024}}{\sum_k Y_{s,k,2024}}$$

Where:

- $Y_{s,j,t}$ is sector j's GSP contribution in state s during year t.
- $g_{s,j}$ is the annual percentage growth rate of sector j in state s.
- $w_{s,j,2024}$ is sector j's baseline share of total state output in 2024.

Step 3 (Numerical Verification):

- In **VIC**, 2024 total is $62.0 + 4.0 + 34.0 = 100.0$ billion AUD; 2025 total is $65.1 + 4.2 + 35.7 = 105.0$ billion AUD. Sector growth rates are all $(65.1 - 62.0)/62.0 = 5.0\%$, $(4.2 - 4.0)/4.0 = 5.0\%$, and $(35.7 - 34.0)/34.0 = 5.0\%$. The share-weighted state growth is $0.62(5.0) + 0.04(5.0) + 0.34(5.0) = 5.0\%$.
- In **WA**, 2024 total is $18.0 + 140.0 + 22.0 = 180.0$ billion AUD; 2025 total is $18.9 + 147.0 + 23.1 = 189.0$ billion AUD. Total WA growth is $(189.0 - 180.0)/180.0 = 9.0/180.0 = 5.0\%$, matching the weighted sum of sector shares $(18/180)(5.0) + (140/180)(5.0) + (22/180)(5.0) = 5.0\%$.

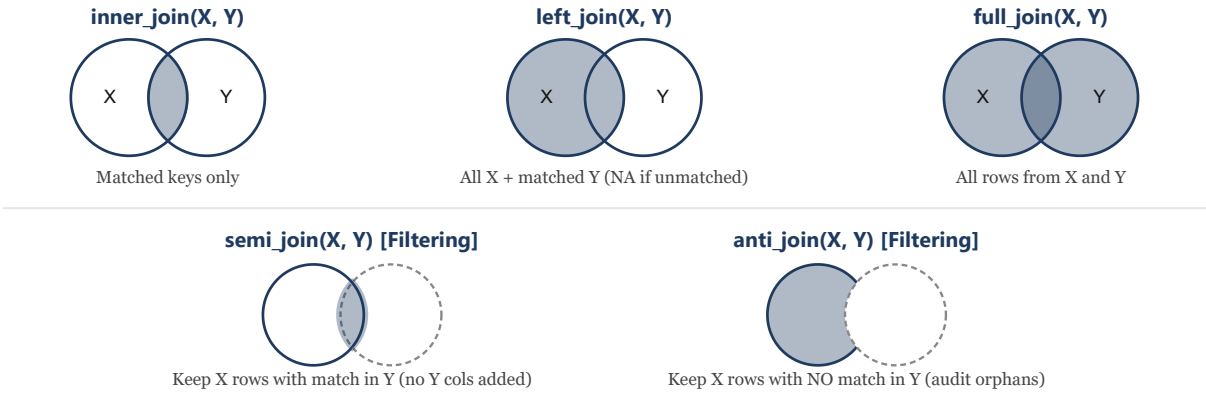
Pivoting the summary output back wider by `Year` or `State` produces an audit-ready comparative executive table without ever duplicating raw figures.

2.3 Relational Data Models, Keys, and Joins

Real-world analytical problems almost never reside in a single table. Business databases normalise entities across multiple linked tables—for instance, separating ASX daily equity prices, company GICS sector metadata, and RBA macroeconomic indicators—to avoid redundant storage and update anomalies. Recombining these tables safely requires explicit reasoning about **keys** and **join semantics**.

- **Primary Key:** A column or minimal composite tuple of columns that uniquely identifies every observation in its own table (for example, `ASX_Code` in a listed-companies table, or the composite `(ASX_Code, Trade_Date)` in a daily pricing table). Uniqueness must always be verified programmatically via `count(key_cols) |> filter(n > 1)` before joining.
- **Foreign Key:** A column (or tuple) in one table that references the primary key of another table (for example, `ASX_Code` inside a quarterly dividend announcements table).
- **Surrogate Key:** A synthetically generated integer or hash identifier added when no natural column combination is guaranteed unique or invariant over time (such as an internal transaction ID).

Degree Live



Joins fall into two functional families:

1. **Mutating Joins (`inner_join`, `left_join`, `right_join`, `full_join`):** Append columns from table Y onto table X by matching key values. In analytical pipelines, `left_join(primary_events, lookup_table)` is the workhorse because it preserves all observations in the primary event table while flagging missing metadata as `NA`.
2. **Filtering Joins (`semi_join`, `anti_join`):** Never add columns from Y and never duplicate rows in X. Instead, `semi_join(X, Y)` filters X to keys present in Y, while `anti_join(X, Y)` isolates unmatched orphan keys in X—indispensable for auditing referential integrity (such as finding customer postcodes absent from the Australia Post postcode directory).

Join Cardinality and Many-to-Many Row Explosion

Before executing a mutating join, analysts must anticipate its cardinality relationship. A **one-to-one (1:1)** or **many-to-one (m:1)** join (such as attaching GICS sector names from an ASX ticker reference table onto daily stock trades) preserves the exact row count of the left table. However, if the lookup table inadvertently contains duplicate keys—say, a firm switched GICS sectors mid-year and appears twice—a seemingly harmless `left_join()` silently triggers a **many-to-many (m:n) Cartesian expansion**. If a key appears a times in X and b times in Y, the joined output emits $a \times b$ rows for that key, silently inflating revenue totals and invalidating standard errors. Modern join implementations support explicit `relationship = "many-to-one"` assertions to halt execution immediately if duplicate right-hand keys violate design expectations.

Worked Example: Auditing ASX Portfolio Holdings via Mutating and Filtering Joins

Suppose a Melbourne superannuation fund tracks client equity holdings in table `Holdings` (4 rows) and sector metadata in table `ASX_Ref` (4 rows):

Table: `Holdings` (Portfolio Positions)	Table: `ASX_Ref` (Sector Reference)
`BHP` ($45,000), `CBA` ($60,000), `CSL` ($30,000), `WPL` ($15,000)	`BHP` (Materials), `CBA` (Financials), `CSL` (Health Care), `NAB` (Financials)

Step 1 (Pre-Join Key Uniqueness Audit): Checking `ASX_Ref |> count(ASX_Code) |> filter(n > 1)` returns zero rows, confirming `ASX_Code` is a valid primary key in the reference table.

DegreeLive

Step 2 (Orphan Detection via `anti_join`): Executing `anti_join(Holdings, ASX_Ref, by = "ASX_Code")` returns the single row `WPL` ($15,000). Why? Woodside Petroleum merged with BHP Petroleum and rebranded its ASX ticker from `WPL` to `WDS`, leaving an outdated foreign key in the portfolio table. An unguarded `inner_join()` would have silently dropped $15,000 (10.0% of total $150,000 portfolio value) from executive sector totals without raising an error.

Step 3 (Preserving Totals via `left_join`): Executing `left_join(Holdings, ASX_Ref, by = "ASX_Code")` retains all 4 portfolio rows and assigns `Sector = NA` to `WPL`, immediately exposing the missing sector classification while preserving the exact $150,000 portfolio valuation. Meanwhile, `semi_join(ASX_Ref, Holdings, by = "ASX_Code")` isolates only the 3 active reference rows (`BHP`, `CBA`, `CSL`), excluding unused benchmark stocks such as `NAB`.

Non-Equi, Rolling, and Cross Joins

Beyond exact equality matching (`x.id == y.id`), commercial pipelines frequently require **inequality (non-equi) joins** and **rolling joins**. Consider matching daily ASX corporate bond settlements against the prevailing Reserve Bank of Australia (RBA) cash rate target: because RBA board decisions occur on discrete meeting dates rather than every trading day, an exact equality join fails on non-meeting days. A rolling join (`join_by(closest(Trade_Date >= Effective_Date))`) pairs each trade with the most recent prior policy rate effective on or before the settlement date. Similarly, **interval overlap joins** match employee shifts against promotional campaign windows (`Start_Date <= Event_Date <= End_Date`), while `cross_join()` builds full Cartesian scenario grids (such as every Australian state paired with every financial quarter) prior to aggregation.

2.4 String Manipulation and Regular Expressions

Real-world administrative records frequently store structured facts inside messy free-text fields: suburb names concatenated with postcodes, inconsistent currency strings (`"A$1,250.50"` vs `"$ 1250.5"`), or multi-format Australian Business Numbers (ABNs). Systematic string pipelines combine deterministic character functions (trimming leading/trailing whitespace with `str_trim()`, collapsing internal spaces with `str_squish()`, case normalisation with `str_to_upper()` / `str_to_lower()`) with **regular expressions (regex)**.

A regular expression is a formal pattern language defining sets of matching character strings:

Regex Token	Pattern Meaning	Australian Parsing Example
`^` and `$`	Start-of-string and end-of-string anchors	`^\\d{4}$` matches exact 4-digit Australian postcodes
`\\d`, `\\w`, `\\s`	Digit `[0-9]`, word char `[A-Za-z0-9_]`, whitespace	`\\d{2}\\s\\d{3}\\s\\d{3}\\s\\d{3}` matches spaced 11-digit ABNs
`[A-Z]`, `[^0-9]`	Character class and negated character class	`[^0-9.]` strips currency symbols and commas before numeric parsing

Regex Token	Pattern Meaning	Australian Parsing Example
`*`, `+`, `?`	0 or more, 1 or more, 0 or 1 repetitions	`A?\\$` matches optional `A` prefix before literal dollar sign `$`
`{m,n}`	Quantifier between `m` and `n` repetitions	`[A-Z]{2,3}` matches state abbreviations (`WA`, `VIC`, `NSW`)
`(...)` and `|`	Capture group extraction and logical alternation	`(VIC|NSW|QLD|WA|SA|TAS|ACT|NT)` extracts state codes

Worked Example: Extracting Postcodes, States, and Clean Prices from Messy Strings

Suppose a Melbourne property listing export contains an uncleaned address column and an asking-price text column:

- Record 1: `"Unit 4, 18 Wellington Rd, Clayton VIC 3168"`, Price: `"A$685,000"`
- Record 2: `"2204/88 Southbank Blvd, Southbank Vic 3006 "`, Price: `"$1,120,500.00"`
- Record 3: `"15 Bay St, Brighton VIC 3186 (Auction)"`, Price: `"POA"`

Step 1 (Normalise Whitespace and Case): Apply `str_squish()` to collapse repeated internal spaces and trim edges, yielding `"15 Bay St, Brighton VIC 3186 (Auction)"`.

Step 2 (Extract State and Postcode via Anchored Capture Groups): Using case-insensitive regex pattern `"\\b(VIC|NSW|QLD|WA|SA|TAS|ACT|NT)\\s+(\\d{4})\\b"`:

- Word boundary `\\b` ensures digits inside street numbers (like `2204`) are not falsely matched unless immediately preceded by a state code and whitespace.
- Group 1 captures `VIC` (normalised via `str_to_upper()`); Group 2 captures `3168`, `3006`, and `3186`.

Step 3 (Sanitise Currency Strings Safely): First test whether the price string contains digits matching `^A?\\$\\s*[\\d,]+(\\.\\d+)?$`. Records 1 and 2 pass, and stripping `[^0-9.]` yields numeric doubles 685000 and 1120500.00. Record 3 (`"POA"`, Price on Application) fails the numeric regex and is mapped cleanly to `NA_real_` rather than throwing an unguarded coercion warning or silently converting to `0`.

Greedy vs. Lazy Quantifiers, Lookarounds, and Unicode Normalisation

Three subtle regex and encoding traps frequently corrupt text extraction pipelines:

- **Greedy vs. Lazy Matching:** Quantifiers such as `.*` are *greedy*—matching as many characters as possible. On HTML markup `"<b>VIC</b> and <b>NSW</b>"`, the greedy pattern `<b>.*</b>` captures the entire string from the first opening tag to the final closing tag. Appending `?` (`<b>.*?</b>`) switches to *lazy* matching, isolating each tag individually.
- **Lookahead and Lookbehind Assertions:** Zero-width lookarounds test surrounding context without consuming characters in the returned match. Positive lookbehind `(?<=ABN:\\s)\\d{11}` extracts the 11-digit Australian Business Number immediately following `"ABN: "` without retaining the prefix label.

DegreeLive

- **Unicode Equivalence and Invisible Characters:** Data copied from web portals or PDFs frequently mix non-breaking spaces (`U+00A0`), zero-width spaces (`U+200B`), or decomposed accents (NFD vs. NFC) that cause exact string joins and `distinct()` checks to fail silently despite looking visually identical. Normalising strings via Unicode NFC and replacing non-standard whitespace before key matching eliminates phantom duplicates.

2.5 Date-Time Pipelines, Time Zones, and Temporal Aggregation

Temporal data introduce subtle ambiguities that regularly break commercial reporting pipelines. Under the international **ISO 8601** standard, calendar dates take the unambiguous descending big-endian format `YYYY-MM-DD` (for example, `2026-07-01`), while full timestamps specify date, time, and UTC offset (`2026-07-01T09:30:00+10:00` for Australian Eastern Standard Time).

In Australian commerce, parsing errors stem from three recurring sources:

1. **DMY vs. MDY Locale Collisions:** Australian administrative exports write `04/07/2026` to denote 4 July 2026, whereas US software defaults interpret the same string as 7 April 2026. Parsers must declare explicit order—`dmy()` rather than generic automatic guessing—and reject out-of-range components immediately.
2. **Australian Time Zones and Daylight Saving (DST):** Australia spans multiple time zones with differing DST rules: Victoria, NSW, Tasmania, and the ACT observe `Australia/Melbourne` / `Australia/Sydney` (switching between AEST `UTC+10` and AEDT `UTC+11` on the first Sunday of October and April), South Australia uses `UTC+9:30/+10:30`, whereas Queensland (`Australia/Brisbane`) and Western Australia (`Australia/Perth`) do not observe DST. Comparing intraday ASX timestamps or national retail transactions without converting to a unified IANA time zone (such as UTC or `Australia/Melbourne`) creates phantom one-hour gaps or duplicate timestamps during clock transitions.
3. **Durations vs. Periods vs. Intervals:** Adding 86,400 seconds (an exact physical *duration*) across a spring DST transition in Melbourne lands at 01:00 AM the next day rather than midnight, whereas adding a calendar *period* of `days(1)` respects local wall-clock time. Similarly, adding `months(1)` to 31 January requires explicit rollback policies because 31 February does not exist.

Time-Zone Conversion vs. Re-Labelling (`with_tz` vs. `force_tz`)

Handling national Australian datasets requires distinguishing two distinct time-zone operations. Changing display timezone via `with_tz()` preserves the exact physical instant in UTC while converting wall-clock labels—for instance, an ASX trade executed at `10:00:00 AEDT` in Sydney (`UTC+11`) occurred simultaneously at `09:00:00 AEST` in Brisbane (`UTC+10`) and `07:00:00 AWST` in Perth (`UTC+8`). By contrast, `force_tz()` keeps wall-clock digits unchanged while altering the underlying UTC instant—used solely to repair timestamps imported erroneously under a default UTC tag when the sensor actually recorded local Melbourne time.

Operation / City	UTC Instant	Local Wall-Clock String	Offset
Melbourne (AEDT, Oct–Apr)	`2026-11-01T00:00:00Z`	`2026-11-01 11:00:00 AEDT`	`+11:00`

DegreeLive

Operation / City	UTC Instant	Local Wall-Clock String	Offset
Brisbane (AEST, Year-Round)	`2026-11-01T00:00:00Z`	`2026-11-01 10:00:00 AEST`	`+10:00`
Adelaide (ACDT, Oct–Apr)	`2026-11-01T00:00:00Z`	`2026-11-01 10:30:00 ACDT`	`+10:30`
Perth (AWST, Year-Round)	`2026-11-01T00:00:00Z`	`2026-11-01 08:00:00 AWST`	`+08:00`

Worked Numerical Example: Financial-Year Aggregation and 7-Day Rolling Average

In Australia, the statutory financial year (FY) runs from **1 July to 30 June**, labelled by the ending calendar year (for example, `2025-07-01` through `2026-06-30` constitutes `FY2026`). Algebraically, for a transaction date with calendar year Y and month $M \in \{1, \dots, 12\}$:

$$\text{FY}(Y, M) = Y + \text{I}(M \geq 7)$$

Where:

- Y is the 4-digit calendar year of the transaction date.
- M is the calendar month integer from 1 (January) to 12 (December).
- $\text{I}(M \geq 7)$ is an indicator equal to 1 if the date falls between July and December, and 0 otherwise.

Thus, `2025-06-30` maps to $2025 + 0 = \text{FY}2025$, while `2025-07-01` maps to $2025 + 1 = \text{FY}2026$.

Now consider computing a 3-day trailing moving average (analogous to a 7-day retail foot-traffic smoother) over five daily observations in Melbourne (`Units`): $[120, 135, 110, 150, 145]$ from 1 to 5 July 2026. The trailing k-day moving average at day t is:

$$\bar{x}_t^{(k)} = \frac{1}{k} \sum_{j=0}^{k-1} x_{t-j}$$

Where:

- $\bar{x}_t^{(k)}$ is the trailing k-period simple moving average at time t.
- x_{t-j} is the observed value j periods prior to t.

For $k = 3$ with complete-window alignment (setting leading incomplete windows to `NA`):

- $t = 1, 2$: NA (fewer than 3 prior days available).
- $t = 3$: $(120 + 135 + 110)/3 = 365/3 = 121.67$.
- $t = 4$: $(135 + 110 + 150)/3 = 395/3 = 131.67$.
- $t = 5$: $(110 + 150 + 145)/3 = 405/3 = 135.00$.

DegreeLive

Crucially, before applying rolling functions, the pipeline must verify that the date series has no missing calendar days (using `complete(Date = seq(min(Date), max(Date), by = "day"))`); otherwise, a 3-row lag silently spans across closed public holidays.

2.6 Data Acquisition, APIs, Web Scraping & Ethical/Legal Governance

When authoritative CSV or Parquet releases are unavailable, analysts retrieve data programmatically from the web through two primary technical mechanisms: structured **Application Programming Interfaces (APIs)** and unstructured **HTML web scraping**.

Acquisition Channel	Data Structure	Reliability & Maintenance	Recommended Use Case
RESTful API (JSON / XML / SDMX)	Explicitly schema-typed key-value payloads	High; versioned endpoints and status codes	ABS Data API, Bureau of Meteorology feeds, ASX/financial APIs
Static HTML Scraping (DOM / CSS / XPath)	Presentational HTML markup tags (`<table>`, `<div>`)	Brittle; breaks whenever page CSS layout changes	Public government tables or historical archives lacking APIs
Headless Browser Automation	JavaScript-rendered dynamic DOM	High resource overhead; slower execution	Interactive dashboards where data arrive via XHR after page load

APIs, HTTP Protocols, and Polite Scraping Mechanics

RESTful APIs operate over HTTP requests (`GET` for retrieval, parameterized via query strings such as `?startPeriod=2024-Q1&format=json`) and return structured status codes (`200 OK`, `400 Bad Request`, `401 Unauthorized`, `429 Too Many Requests`). Responsible API clients store authentication tokens in environment variables (`.Renviron` or system secrets)—never hardcoded inside committed scripts—handle pagination loops cleanly, and respect rate limits via exponential backoff.

When an API does not exist and public HTML pages are parsed via CSS selectors or XPath queries (for example, extracting `<table>` nodes with `rvest`), ethical engineering mandates:

1. **Inspecting `robots.txt` and Terms of Service:** Checking the target domain's `/robots.txt` file identifies disallowed paths and requested crawl delays.
2. **Throttling and Caching:** Inserting deliberate pauses between sequential requests (e.g., 2 to 5 seconds via polite wrappers) prevents overloading public servers, while caching raw downloaded HTML locally avoids repeated scraping during parser debugging.
3. **Identifying the Client:** Supplying an informative HTTP `User-Agent` header with project contact details rather than masquerading as an interactive desktop browser.

Rectangling Nested JSON Payloads into Tidy Tables

Unlike flat CSV exports, RESTful APIs return hierarchical JavaScript Object Notation (JSON) objects combining key-value dictionaries (`{}`) and ordered arrays (`[]`). Converting nested JSON into tidy relational tables—termed *rectangling*—alternates between two list-column operations:

DegreeLive

- **Horizontal Expansion (`unnest_wider()`):** Transforms a list-column of named dictionaries (for example, an address object containing `suburb`, `state`, and `postcode`) into separate named columns alongside the parent record.
- **Vertical Expansion (`unnest_longer()`):** Transforms a list-column of variable-length arrays (for example, multiple quarterly filings attached to a single firm object) into multiple rows, repeating parent identifiers so the resulting table obeys tidy observational-unit rules.

Whenever API responses mix scalar attributes at the top level with deeply nested sub-arrays (such as ABS SDMX-JSON data cubes linking dimension index arrays to observation keys), rectangling preserves full relational traceability while converting irregular tree structures into rectangular data frames ready for grouped summaries and joins.

Australian Legal, Privacy, and Re-Identification Safeguards

Technical capability to scrape or link datasets does not imply legal or ethical permission to publish or model them. Analysts operating in Australia must navigate three interlocking governance obligations:

- **Privacy Act 1988 (Cth) and Australian Privacy Principles (APPs):** Collecting, storing, or linking personal information (data from which an individual is reasonably identifiable) triggers APP obligations regarding consent, purpose limitation, security, and cross-border disclosure. Scraping personal profiles, emails, or social media posts without consent frequently violates APP 3 (collection of solicited/unsolicited personal information) and platform terms.
- **Copyright Act 1968 (Cth) and Licensing:** Automated extraction of proprietary commercial databases, real estate portals, or news archives for redistribution or commercial model training may infringe copyright or database rights. Conversely, Australian government datasets published under Creative Commons Attribution (CC BY 4.0)—such as ABS Census tables and data.gov.au releases—explicitly permit reuse with attribution.
- **De-Identification and Linkage Re-Identification Risk:** Stripping explicit names and Medicare numbers is insufficient to anonymise microdata. Research demonstrates that linking just three *quasi-identifiers*—4-digit postcode, full date of birth, and sex—uniquely identifies a substantial majority of Australians when joined against public electoral or marketing lists. Before sharing derived datasets, analysts apply disclosure-control safeguards such as top-coding extreme incomes, aggregating geographic units from SA1 to SA3/State, and verifying **k-anonymity** (ensuring every equivalence class of quasi-identifiers contains at least k indistinguishable individuals).

End-to-End Provenance Audit for External Acquisition

To keep scraped or API-retrieved datasets reproducible after upstream web pages change or disappear, production pipelines separate acquisition from parsing: (1) execute the HTTP fetch script once on a recorded UTC timestamp and save the raw JSON or HTML payload untouched in `data-raw/`; (2) compute and log the cryptographic SHA-256 checksum alongside the exact request URL, HTTP status code, and licensing terms; and (3) run all CSS/XPath parsing, regex cleaning, and relational joins deterministically against the archived local snapshot.

DegreeLive