**Vanguard Funnel KPIs — Functions & Workflow (README)**

This README explains, in plain English, what each helper and pipeline function does, **step by step**, and why it’s needed.

**Overview**

This mini-library turns raw web traces into **process-level KPIs** for Vanguard’s online funnel.  
It:

* Standardizes step labels and timestamps.
* Groups events by **process** (client\_id, visitor\_id, visit\_id).
* Detects **back-steps** (users going to an earlier step).
* Keeps **only the last confirm** per process (per the business rule).
* Builds one **summary row per process** (reached flags, times, outcome).
* Aggregates **KPIs** (completion, drop-off, time, error rates).

You can run it for Control, Test, or any client subset.

**Data Inputs**

* df\_web\_data: clickstream-like table with at least  
  client\_id, visitor\_id, visit\_id, process\_step, date\_time
* df\_demo\_control: list of Control clients (at least client\_id)  
  *(Use the corresponding demo table for Test when you run the same pipeline.)*

**Process identity** is defined by the **triple**:  
(client\_id, visitor\_id, visit\_id).

**Functions (what each one does and why)**

**1) \_standardize\_steps(df)**

**What it does**

* Lowercases and normalizes the process\_step text into a canonical set:  
  start, step\_1, step\_2, step\_3, confirm.
* Filters out any rows whose step isn’t in that list.

**Why**  
Step names can vary (e.g., “Step 1”, “step1”). KPIs must compare apples to apples, so we unify labels before any logic (funnel flags, timing, etc.).

**2) \_to\_utc(series)**

**What it does**

* Parses date\_time values to pandas datetimes.
* Ensures all timestamps are **timezone-aware** in **UTC** (or converted to UTC if they already have a timezone).

**Why**  
Clickstream data often mixes tz-naive and tz-aware timestamps. Subtracting mixed types raises errors and can distort durations. Normalizing to UTC guarantees reliable time differences.

**3) prepare\_web\_for\_group(df\_web\_data, df\_demo\_group)**

**What it does**

1. **Filters** df\_web\_data to the client\_ids present in df\_demo\_group.
2. Calls \_standardize\_steps and \_to\_utc to clean process\_step and date\_time.
3. **Sorts** rows by (client\_id, visitor\_id, visit\_id, date\_time) so each process’s events are in the correct time order.

**Why**  
We need a clean, ordered sequence of events **per process** to correctly detect back-steps, completion, and durations. This function creates that reliable starting point for the pipeline.

**4) compute\_back\_jumps(w\_full)**

**What it does**

* Maps each process\_step to a numeric index (start=0 … confirm=4).
* For each process, computes consecutive **step changes** and flags **back-steps** where the index decreases (e.g., step\_3 → step\_2).
* Returns:
  + wf: the **full event-level** table with step\_idx, prev\_step\_idx, delta, and is\_back\_jump.
  + back: a **process-level** table with n\_back\_jumps (the count of back-steps per process).

**Why**  
Back-steps quantify **confusion/friction**. We need them to classify outcomes:

* *successful* (completed, **no** back-steps),
* *completed\_with\_errors* (completed **with** back-steps),
* *unsuccessful* (never reached confirm).

**5) collapse\_last\_per\_step\_and\_last\_confirm(wf)**

**What it does**

* **Drops intermediate confirms** and **keeps only the last** confirm per process (as per the experiment rule).
* For timing and reached flags, **keeps only the last occurrence of each step** in the same process.
* Returns a compact event table with **one (at most) row per step per process**.

**Why**  
Users may hit confirm multiple times, or reach the same step repeatedly. KPIs must reflect the **final state** and the **last time** each step was reached (e.g., to compute total time from first start to **final** confirm). This ensures consistent, business-aligned counting.

**6) summarize\_processes(wf2, back)**

**What it does**

* **Pivots** the compact table (wf2) into **one row per process** with columns holding the **last timestamp** for each step.
* Ensures timestamps are consistent and converts them to **naive UTC** for clean subtraction.
* Creates **reached flags** (reached\_start, reached\_step\_1, …, reached\_confirm) and a completed flag (same as reached\_confirm).
* Computes **durations (minutes)** between steps:  
  t\_start\_step1, t\_step1\_step2, t\_step2\_step3, t\_step3\_conf, and t\_total (confirm - start).
* **Merges** the n\_back\_jumps from back.
* Assigns an **outcome**:
  + successful → completed==1 and n\_back\_jumps==0
  + completed\_with\_errors → completed==1 and n\_back\_jumps>0
  + unsuccessful → completed==0

**Why**  
Most KPIs and plots want a **process-level** table. This function is the heart of the pipeline: it transforms step-level events into a clean **per-process** dataset with everything we need (flags, times, errors, outcome).

**7) kpis\_from\_processes(proc)**

**What it does**  
Aggregates the process table into **funnel KPIs** (denominator = number of processes that **reached start**):

* **Funnel rates** (% of starters):  
  step1\_rate\_%, step2\_rate\_%, step3\_rate\_%, completion\_rate\_%
* **Outcome mix** (% of starters):  
  successful\_%, completed\_with\_errors\_%, unsuccessful\_%
* **Time KPIs** (medians, minutes):  
  t\_total\_median\_min, t\_step1\_median\_min, t\_step2\_median\_min, t\_step3\_median\_min, t\_conf\_median\_min
* **Error level**: avg\_back\_jumps

**Why**  
These are the **minimum viable KPIs** requested: completion, time, and error rates—plus a practical split of *successful vs completed with errors vs unsuccessful*. Using **medians** makes the timing KPIs robust to outliers.

**8) step\_dropoff\_table(proc)**

**What it does**  
Builds a tidy **drop-off table** (great for Tableau/Power BI):

* Rows for each transition: start→step\_1, step\_1→step\_2, step\_2→step\_3, step\_3→confirm.
* For each, reports:
  + n\_from (how many reached the source step),
  + n\_to (how many reached the next step),
  + conv\_rate\_% (conversion),
  + dropoff\_% (100 − conversion).

**Why**  
Product teams want to see **where users leave the funnel**. This table quantifies step-wise leakage and compares segments (Control vs Test) at a glance.

**End-to-End Workflow (Control example)**

1. **Prepare web**  
   w\_control = prepare\_web\_for\_group(df\_web\_data, df\_demo\_control)  
   → filtered, normalized, time-ordered events.
2. **Detect back-steps**  
   (wf, back) = compute\_back\_jumps(w\_control)  
   → per-event back-step flags and per-process n\_back\_jumps.
3. **Keep only the last confirm & last occurrence of each step**  
   wf2 = collapse\_last\_per\_step\_and\_last\_confirm(wf)  
   → compact, step-level table.
4. **Process-level summary**  
   proc\_control = summarize\_processes(wf2, back)  
   → one row per process with flags, durations, outcome.
5. **KPIs**  
   kpis\_control = kpis\_from\_processes(proc\_control)  
   → completion, time, errors, outcomes.
6. **Drop-off**  
   dropoff\_control = step\_dropoff\_table(proc\_control)  
   → conversion + drop-off between steps.

Run the same 6 steps for your Test cohort to compare groups (gaps in percentage points, time medians, and error rates).

**KPI Definitions (quick reference)**

* **Completion rate**: % of processes that reached 'confirm' over those that reached start.
* **Step rates**: % of starters that reached each intermediate step.
* **Successful**: completed with **zero** back-steps.
* **Completed with errors**: completed with **≥1** back-step.
* **Unsuccessful**: never reached confirm.
* **Time KPIs**: **median** minutes for each segment and total.
* **Error level**: average number of back-steps per process.

**Sanity Checks & Troubleshooting**

* **All zeros?**  
  Most often due to step labels or timestamps. Ensure process\_step normalizes to start/step\_1/step\_2/step\_3/confirm and that **all timestamps** become UTC with \_to\_utc.
* **Mixed tz errors?**  
  Always run \_to\_utc *before* sorting and timing.
* **Multiple confirms counted?**  
  collapse\_last\_per\_step\_and\_last\_confirm drops **intermediate** confirms by design.
* **Outliers in time**?  
  We report **medians**; use means only if you need them for specific analyses.

**How to Use (quick start)**

# Control

w\_control = prepare\_web\_for\_group(df\_web\_data, df\_demo\_control)

wf, back = compute\_back\_jumps(w\_control)

wf2 = collapse\_last\_per\_step\_and\_last\_confirm(wf)

proc\_control = summarize\_processes(wf2, back)

kpis\_control = kpis\_from\_processes(proc\_control)

dropoff\_control = step\_dropoff\_table(proc\_control)

display(kpis\_control)

display(dropoff\_control)