**Bidfood EDA – Project Handover Documentation**

**Repository:** <https://github.com/Together-NZ/Bidfood-EDA>

**1. Purpose & Scope**

This document is a handover guide for the Bidfood EDA (Exploratory Data Analysis) project. It explains the end-to-end logic implemented in the Python script, highlights key nuances embedded as in-code comments, and provides a runbook so the next team can maintain and extend the work.

The codebase is hosted on GitHub: https://github.com/Together-NZ/Bidfood-EDA

**2. Analysis Window & Visual Branding**

The script defines a standard color palette for consistent visuals and sets a default analysis window using start and end years for time-based filtering.

Key points:

* Color palette variables (e.g., green, blue, indigo, etc.) are used to style charts consistently.
* Analysis period variables: start\_year=2023, end\_year=2025. Adjust as required for new cycles.

**3. Data Sources (BigQuery) & Access**

The script uses Google BigQuery as the primary data source. It initializes a BigQuery client with the project ID and runs parameterized SQL queries. You must have appropriate IAM permissions and have authenticated via Google Cloud to run the queries locally or in a notebook environment.

Primary datasets referenced include:

* MenuIQ / gaps feed (e.g., menudata04062025.customer\_gaps\_v2) – used to identify category gaps at customer level.
* Raw landing tables – branch, customer, product, sales – used for building the consolidated sales dataset.

*Note: Ensure local or environment credentials are set (gcloud auth application-default login) and that the GCP project bid-x-marketing-47tt is available and accessible.*

**4. MenuIQ Gaps – Cleaning Rules**

The gaps dataset is filtered to exclude national accounts and to focus on non-purchased categories. A small set of audit/ingestion-related columns is dropped during cleaning.

* Filter: NatAcc\_Code != 'Y'
* Filter: purchased\_cat != 'purchased'
* Drop columns: inner\_quantity, count\_rows, no\_invoices, no\_products

**5. Raw Landing (Branch, Customer, Product, Sales) – Loading & Preparation**

The script loads core raw landing tables (branch/customer/product/sales) from the rawlanding04062025 dataset. Ingestion metadata columns (dss\_update\_time, upload\_date, upload\_time) are dropped from each table where present.

Sales data is validated for invoice date integrity, and then filtered to the defined analysis period.

* Null/invalid invoice\_date rows are reported for inspection.
* Deduplication of sales: rows are de-duplicated using a composite key across order/invoice and line columns.
* Customer filter: Wellington branches only (A40, 40, 74). National accounts excluded (NatAcc\_Code != 'Y').

**6. Customer Unification Logic (unified\_customer\_key)**

The script harmonizes customer identities and then constructs a unified key that joins downstream to sales. This step is critical to reduce duplication arising from case, missing values, or multi-warehouse flags.

* Case normalization: Customer\_Name/Email/Address\_1 strings are lowercased for deduping, then proper-cased for presentation.
* Deduping steps: unique on Customer\_Name, then unique on Address\_1 (non-null only), then unique on Email (non-null only).
* unified\_customer\_key is defined as Customer\_Code + '\_' + MultiWH\_Cust (string cast).
* If duplicates persist, a numeric suffix (\_2, \_3, …) is appended to guarantee uniqueness.

*Note: After the unified key is created, it is stitched back onto the original customer records (matching on Customer\_Name, Email, Address\_1).*

*Note: This unified key is then used to merge with cleaned sales.*

**7. Customer-Sales Merge & Data Quality Checks**

Customers are merged with the de-duplicated sales table on Customer\_Code. The merge step also captures unmatched customers and unmatched sales for diagnostics.

* customer\_sales (inner join) holds matched records for analysis.
* unmatched\_customers (left-only) and unmatched\_sales (right-only) aid in tracking mapping issues.

*Note: Downstream metrics (e.g., Pareto, histograms, retention) are computed from customer\_sales.*

**8. Yearly KPIs & New Customer Inference**

Yearly metrics include Total Sales, Number of Transactions (unique orders), Average Order Value, and Unique Customers. New customers are inferred by the year of their first observed transaction in the analysis window.

* KPI aggregation by invoice year; Average Order Value = Total Sales / Number of Transactions.
* New customer counts are derived from the minimum invoice\_date per unified\_customer\_key (i.e., first observed purchase).

*Note: Important nuance: Unique customers per year is not net-new customers—customers returning in later years are counted in each year they transact.*

**9. Value Distribution & Pareto**

Customer-level sales are aggregated and sorted descending to produce cumulative (Pareto) curves and distribution histograms.

* Cumulative % of sales vs. cumulative % of customers (20/50/80 reference lines).
* Histogram annotations include mean and median values in thousands.
* Decimal monetary values are cast to float for plotting and summary statistics.

**10. Frequency Distributions (Orders & Transactions)**

The script derives distribution charts for number of orders per customer (both raw Customer\_Code and unified key views).

These reveal long-tail dynamics: many customers with low order count vs. few high-frequency customers.

**11. Time Between Purchases (Intervals)**

Inter-purchase time is computed per customer, binned into interpretable intervals, and plotted with both count labels and share labels.

Cumulative insight annotations (e.g., “Same day or less”, “1 week or less”) are rendered alongside the bar chart.

* Compute days\_between\_purchases using groupby(...).diff().dt.days after sorting by unified\_customer\_key and invoice\_date.
* Bins include Same day, 1–6 days, 1–2 weeks, 1–3 months, 6 months, 1 year. First purchases are excluded (NaN).
* Both absolute counts (in thousands) and percentages are annotated for each bin.

**12. Industry-Level KPIs & Monthly Activity**

At the industry dimension, the script calculates unique customer counts, revenue, and transactions, plus percentages of total. Monthly views compute average monthly active customers, transactions, and revenue for each industry.

**13. Branch-Level KPIs**

Aggregations by Branch\_Code\_customer and Branch\_Description produce total sales, average transaction value, unique customers, and transaction counts for quick comparative review.

**14. Coffee Shops & Cafés Focus (Industry\_Code = '17')**

For a focused vertical view, the script filters the consolidated dataset to Industry\_Code '17'. It then joins to the product table to add Class\_Code / Class\_Description and removes non-consumables (e.g., crates/cardboards).

This enables repertoire / class-level distribution and acquisition tracking specifically for the segment.

* Non-consumables filtered by Class\_Code in {'NONCTN','NONCRT2','NONCRT','CNTBOARD'}.
* Annual summaries: Active customers, Total sales, Transactions; Acquisition by first invoice year.

**15. Known Nuances & Gotchas**

Below are noteworthy implementation details added as code comments and worth preserving in future maintenance:

* Invoice date integrity: warn if time component is missing; treat inter-purchase computation at day granularity.
* Deduping logic intentionally preserves null values in Address\_1/Email during staged uniqueness checks to reduce over-merging.
* Histogram and Pareto axes use thousands/millions for readability; ensure consistent unit labeling in titles and annotations.
* Wellington branch filter (A40, 40, 74) is explicit—update this when expanding to other regions.

**16. Outputs & Artifacts**

* customer\_sales: main fact-level DataFrame after join.
* Derived tables: industry\_stats, monthly\_stats, branch\_stats, yearly\_metrics, customer\_values (for Pareto), etc.
* Plots: sales distribution, order distributions, Pareto, time-between-purchases, branch/industry bars.
* Optional CSV extracts for Sales Rep enablement (commented lines show how to save).