# Retail Customer Analytics Using PostgreSQL and MongoDB

*Final Project Report*

## Executive Summary

In today’s competitive grocery retail market, understanding customer behavior is key to driving growth. This project analyzed two years of grocery transaction data for 2,500 households, integrating both relational (PostgreSQL) and document-oriented (MongoDB) database approaches to derive insights. We cleaned and loaded the **dunnhumby Complete Journey** dataset – which includes household transactions, product details, customer demographics for 801 households, promotional campaigns, coupon redemptions, and in-store promotion data – into both database systems for a comparative analytics study. Our goal was to answer four strategic questions commonly posed by retail executives: (1) how customer spending trends change over time, (2) how demographics influence purchasing, (3) which product categories are growing or shrinking, and (4) whether direct marketing campaigns (coupons, mailers, in-store displays) boost customer engagement and sales.

We approached the analysis twice – first using a normalized schema in PostgreSQL and then using a denormalized document model in MongoDB – to evaluate the strengths and weaknesses of each platform for retail analytics. Key findings include: **roughly half of households increased their spending from Year 1 to Year 2 while the others spent less**, indicating a bifurcation where some customers became more engaged as others drifted away. Those who spent more tended to expand their purchases in staple grocery categories (e.g. pantry goods, produce, dairy) and even broaden into non-food departments like personal care and fuel, signaling deeper loyalty and one-stop shopping behavior. Meanwhile, households that spent less cut back primarily on non-essentials (snacks, treats, general merchandise) while continuing to buy basic necessities like milk and bread, with some extreme cases dropping out entirely.

Demographic analysis revealed that **household size (especially the presence of children) is the strongest driver of spend** – larger families spend far more in total, particularly on categories like milk, cereal, and baby products. Income and age had nuanced effects: higher-income households did spend more on premium products, but middle-income families with kids contributed more to overall sales than the wealthiest segment, underscoring that family size often outweighs income in grocery spend. Additionally, “DINK” households (dual-income, no kids) and single adults tended to spend a bit more on indulgences or premium categories (like specialty foods or alcohol) compared to budget-conscious families, highlighting distinct segment behaviors. These insights suggest targeted marketing opportunities – for example, promotions on bulk family essentials for larger households, versus premium or convenience items for smaller households.

We also found **direct marketing campaigns had a measurable positive impact on customer engagement**. Households that redeemed targeted coupons generally maintained or increased their spending relative to those who received no offers or ignored them. For instance, one household that responded to multiple campaigns by using five coupons subsequently sustained higher purchase levels. Across the customer base, those who participated in campaigns were less likely to decrease spend and often showed boosts in the categories of the coupons, indicating that well-designed mailer campaigns can drive incremental sales and loyalty. Furthermore, analysis of in-store promotions (display ads and weekly flyer features) showed that products featured in these promotions tended to enjoy a lift in sales during the promotion period, confirming the value of merchandising efforts in driving short-term purchase spikes.

**Technology-wise, the project highlighted complementary strengths of SQL and NoSQL for analytics.** PostgreSQL’s structured schema and SQL queries excelled at complex joins and ensured data integrity through foreign keys, which was vital for combining transaction, product, and promotion tables reliably. MongoDB’s document model, on the other hand, allowed us to answer certain questions with fewer lookups – for example, computing basket (transaction) totals and item counts in MongoDB was extremely fast by leveraging embedded arrays, avoiding the multi-table join that PostgreSQL required. However, for multi-faceted analyses (e.g. tying transactions to product attributes or large promotional data), PostgreSQL’s set-based operations were more straightforward, whereas MongoDB needed careful pipeline design (using $lookup and pre-aggregation) and extensive indexing to achieve similar performance.

In summary, **PostgreSQL provided robustness and simplicity for cross-table analytics, while MongoDB offered speed and flexibility for aggregated, self-contained data like “basket”-level metrics**. We recommend a hybrid strategy: use PostgreSQL as the source of truth and for heavy relational crunching (especially where data integrity and complex joins matter), and use MongoDB for fast real-time analytics on pre-joined “views” (e.g. a household’s basket history as a document) to power applications that benefit from quick one-to-many aggregations. The concluding section of this report provides concrete recommendations on which analytical tasks suit each system best and how to leverage both in a complementary fashion.

## Introduction

Retailers are constantly looking to **analyze customer behavior** to tailor marketing strategies and improve sales. In grocery retail, understanding *who* your customers are, *how* their spending changes, *what* products they buy, and *whether* marketing efforts keep them engaged can inform everything from inventory management to personalized promotions. In this project, I set out to explore these questions using a rich transactional dataset and to evaluate how two different database technologies – **PostgreSQL (relational)** and **MongoDB (NoSQL document-oriented)** – perform in supporting such analytics. By conducting the same analyses on both systems, we can identify the practical trade-offs in data modeling, query patterns, and performance, thereby guiding data architects and business stakeholders on the best tool for each job.

**Business Questions:** The analysis was guided by four key questions posed by retail strategists:  
- *Customer Spending Trends:* Are customers spending more or less over time? What are the weekly or seasonal patterns in total sales?  
- *Demographic Influences:* How do customer demographics (family size, income, age, etc.) affect how much and how often they shop? Do different groups have different category preferences?  
- *Category Growth:* Which product categories or departments are growing the fastest, and which are stagnating or declining, especially among customers who increase or decrease their spend?  
- *Marketing Campaign Impact:* Do direct marketing efforts (like coupon mailers and in-store promotions) actually lift sales or basket sizes? Can we see differences between households who redeem offers and those who don’t?

These questions reflect common retail business objectives: tracking customer lifetime value, segmenting customers for targeted marketing, optimizing product mix, and measuring ROI on promotions. Answering them requires integrating data from multiple sources – purchases, customer attributes, products, and marketing touchpoints – in a reliable and efficient manner.

**Data Source:** We used the publicly available **dunnhumby “The Complete Journey”** dataset, which provides a comprehensive view of household-level grocery shopping over two years. This dataset covers **2,500 households** who are frequent shoppers at a grocery chain, recording every item they purchased over a 104-week period (approx. two years). For a subset of these households (801 of them), the data also includes demographic information (e.g. age group, income tier, family composition) and a history of **direct marketing campaigns** they received (mailers with coupons). Additionally, the dataset contains reference tables for all **products** sold (over 92,000 items with category hierarchies and brand info) and detailed logs of **coupon offerings and redemptions** during marketing campaigns. Finally, a large **“causal\_data”** table logs in-store promotions, indicating if a given product was on special display or featured in a weekly flyer for each store-week. The breadth of this data allowed us to connect *customer profiles* to *purchases* to *marketing exposures*, which is ideal for the holistic analyses we aimed to perform.

**Data Volume and Structure:** The Complete Journey dataset is sizable, comprising millions of transaction records and tens of millions of promotion records. Table 1 summarizes the core tables and their contents (after initial cleaning), along with how they relate in a relational schema:

* **Household Demographics (801 rows):** Each row is a household, with fields like age group, marital status, income range, homeowner/renter, and number of children. Primary key: household\_key. Only ~32% of households have demographic data (801 of 2,500), which we had to account for in schema design and analysis.
* **Transaction Line Items (~1.43 million rows after filtering):** This fact table (often called transaction\_data) records every item purchased in every shopping trip. Each record includes a household ID, a basket ID (transaction receipt identifier), the product ID, quantity, price (sales\_value), any discounts applied (loyalty card or coupon discounts), store ID, and the day and week of purchase. In the raw data, each line is one product on a receipt – we later transformed this into basket-level groupings.
* **Product Details (92,353 rows):** A lookup table mapping each product\_id to descriptive attributes: department (high-level category), commodity and sub-commodity (subcategory hierarchies), brand (private or national brand), manufacturer, and package size. Primary key: product\_id. This table enables category roll-ups (e.g., summing sales by department or commodity).
* **Campaign Description (30 rows):** A lookup table of marketing campaigns. Each campaign has an ID (1–30), a type (TypeA, B, or C) indicating the offer structure, and start and end days when the campaign was active. Primary key: campaign.
* **Campaign Assignments (4,213 rows after filtering):** Often called campaign\_table, this table lists which households were targeted with which campaigns. Each row is a combination of a household\_key and a campaign ID (composite key) indicating that household received that campaign. It also repeats the campaign type description for convenience. We see that not all households got every campaign – this is the “target” group for each marketing initiative.
* **Coupons (119,384 rows after de-duplication):** This table enumerates **coupon offers**: each unique coupon is identified by a coupon UPC code and is tied to a campaign, and it lists a product that the coupon can be redeemed for. If a single coupon code is valid for multiple products (e.g., a coupon for any item in a category), it appears in multiple rows (same coupon\_upc and campaign, different product\_id). The composite of (campaign, coupon\_upc, product\_id) is unique. This represents the pool of coupon offers made available in campaigns.
* **Coupon Redemptions (1,856 rows after filtering):** This table logs actual usage of coupons by households. Each row has a household\_key, campaign, coupon\_upc, and the day the coupon was redeemed in a store. It tells us which campaign’s coupon was used, by whom and when. A given coupon code appears at most once per household here (household can redeem a coupon only once). We found 434 distinct households that redeemed at least one coupon, implying many did not use any.
* **Causal Promotions (36.8 million rows):** This large table lists weekly in-store promotions. Each entry is a unique combination of product\_id, store\_id, week\_no, indicating a product was promoted in that store during that week. Flags indicate the type of promotion: a display code (non-zero if featured on special display in-store, with values 1–7 for types of displays) and a mailer code (non-zero if featured in the weekly mailed flyer/ad, various letter codes). This is used to see if a purchase coincided with a promotion.

All tables link through common keys: **household\_key** ties demographics, transactions, campaign assignments, and redemptions; **product\_id** links transactions, product details, coupons, and promotions; **campaign** and **coupon\_upc** link the campaign descriptions, coupon offers, and redemptions. Figure 1 below (in text form) shows a simplified entity-relationship outline of the core data model we used in PostgreSQL:

* *Household (household\_key PK)* –< *Transaction Basket (basket\_id PK, household\_key FK)* –< *Basket Item (basket\_id+product\_id PK, FKs to Basket and Product)* >– *Product (product\_id PK, FK to Subcommodity)* >– *Sub-Commodity (sub\_commodity\_id PK, FK to Commodity)* >– *Commodity (commodity\_id PK, FK to Department)* >– *Department (department\_id PK)*.
* *Household* –< *Campaign Assignment (household\_key+campaign PK)* >– *Campaign (campaign PK)*.
* *Campaign* –< *Coupon (campaign+coupon\_upc PK)* >– *Product* (each coupon links to eligible products).
* *Household* + *Campaign* –< *Coupon Redemption (household\_key+campaign+coupon\_upc+day PK)* >– *Coupon*.
* *Product* + *Store + Week* –< *Causal Promo (product\_id+store\_id+week\_no PK)*.

*(Note: The above schema was further refined from raw data to ensure* *3rd normal form (3NF)* *and eliminate transitive redundancy. For example, we extracted product category hierarchies into separate tables for Department, Commodity, Sub\_commodity to enable flexible “category-level” analysis of growth.)*

## Data Preparation and ETL Process

Preparing the data for analysis involved an **ETL (Extract, Transform, Load)** pipeline implemented in Python. The same raw CSV data files were loaded and cleaned for use in both PostgreSQL and MongoDB, ensuring a fair comparison with identical data inputs. Key steps in the data preparation included:

* **Data Loading:** We used pandas to read the CSV files for each table (transactions, products, households, etc.) into memory. The raw row counts matched expectations: e.g. ~2.59 million transaction rows, ~92k products, ~36.8M causal promo records, etc.. Basic column renaming was done for clarity (e.g., renaming demographic column names like “classification\_1” to “age\_desc”) to make the data more understandable in analysis.
* **Cleaning Invalid References:** A major data integrity issue was that many transaction and campaign records referenced households that had no entry in the demographic table. Since only 801 households have demographics, about 1,700 households are “missing” from that table. In a relational database with foreign keys, those records would violate referential integrity. We decided to **filter out records for households lacking demographic data** to maintain consistency (alternatively we could have allowed null foreign keys, but then those households couldn’t be used in demographic analysis). This filter dropped a substantial number of rows: we removed ~1.17 million transaction lines (out of 2.59M) where household\_key wasn’t in the demographic table, as well as ~2,995 campaign assignment rows and ~462 coupon redemption rows. After this, all remaining transactions, campaign assignments, and redemptions refer to one of the 801 known households.
* **Removing Duplicates:** We checked for duplicate primary keys in each table. The product table and others had no duplicates by their natural keys (e.g., no duplicate product\_ids) as expected. However, the coupon table (coupon offer mappings) had some exact duplicate rows, likely due to the way the data was collected or combined. We dropped 5,164 duplicate rows from the coupon dataset (reducing from 124,548 to 119,384 records) to ensure each (campaign, coupon\_upc, product\_id) combination is unique. Other tables did not require deduplication beyond the removals already done in the household filter step.
* **Relational Schema Implementation (PostgreSQL):** With clean data, we created tables in PostgreSQL reflecting the normalized schema described above. We defined primary keys and foreign key constraints to enforce relationships (for instance, transaction\_data.household\_key references hh\_demographic.household\_key, etc.). Before loading, we truncated any existing tables in the database (since this was a re-run of the pipeline during development). We then **bulk-inserted** the data using PostgreSQL’s COPY functionality via SQLAlchemy, in chunks of 50k rows for efficiency. This resulted in the following row counts in Postgres: 1,427,303 transactions (all linked to demographics), 92,353 products, 801 households, 4,213 campaign assignments, 30 campaigns, 119,384 coupon mappings, 1,856 coupon redemptions, and 36,786,524 causal promo records. All foreign keys were satisfied by design after the cleaning, so the relational integrity was solid.
* **Document Schema Design (MongoDB):** Loading the data into MongoDB required a different approach: we chose to **denormalize and embed data** to leverage Mongo’s strengths. We designed the MongoDB collections to mirror logical aggregates (access patterns) in the data:
* **Households collection (801 documents):** Each document represents one household’s demographic profile (essentially one doc per hh\_demographic row). We could have embedded campaign info here but opted not to, keeping marketing data separate for flexibility.
* **Products collection (92,353 documents):** Each document is one product with all its attributes. In a further denormalization step, we decided to also **embed category descriptors (department, commodity, sub-commodity names)** inside each product document. This way, queries on product categories in MongoDB wouldn’t require a join to separate category tables (since MongoDB doesn’t enforce foreign keys, duplicating this static info is acceptable for faster reads).
* **Baskets collection (140,339 documents):** Instead of storing each transaction line as a separate document, we aggregated all items of a single basket (shopping trip) into one document. Each basket document has fields for basket\_id, household\_key, store\_id, day, week\_no, and an **items array** that contains all the products purchased in that transaction. Every element in items is an embedded sub-document with product\_id, quantity, sales\_value, and discount details (retail and coupon discounts). We essentially transformed the flat transaction\_data table into a hierarchical JSON structure keyed by basket. This eliminated the need to join items by basket\_id when analyzing basket-level metrics in MongoDB.
* **Campaigns collection (30 documents):** We combined the campaign description and the list of targeted households into each document. Each campaign document contains the campaign attributes (type, start\_day, end\_day) and an embedded list or array of household\_key that were targeted. This means a single query on a campaign doc can tell us exactly who received it, without joining to a separate assignment table.
* **Coupons collection (1,397 documents):** Rather than 119k documents for each coupon-product combination, we aggregated by coupon. Each document represents a unique coupon (defined by a composite of campaign ID and coupon\_upc code). It contains that coupon’s code, campaign, and an array of eligible product\_ids it can be redeemed for. This compaction dramatically reduced document count and made it easy to see all products a given coupon covers.
* **CouponRedemptions collection (1,856 documents):** We kept coupon redemption events as their own documents, similar to the relational table (each redemption has household, day, campaign, coupon\_upc). We could have embedded redemptions inside either the household or basket documents, but chose not to, since querying them in MongoDB via an index on (household\_key, day) with $lookup was sufficient for our analysis needs.
* **CausalPromotions collection (36,786,524 documents):** We left this as is – one document per product-store-week promotion entry. This dataset is too large and already fairly atomic; there was no obvious way to embed it without huge duplication, so we rely on indexing and aggregation to join it with sales as needed.

This document-oriented model sacrifices some of the **integrity enforcement** of the relational model (e.g., we had to ensure in code that every embedded product\_id existed in the products collection, etc.), but it **minimizes the number of joins/lookups** needed at query time by collocating related data. We loaded the data into MongoDB using Python’s PyMongo, again in batch inserts. We had to be mindful of MongoDB’s 16MB document size limit and 48MB batch insert limit – our largest basket docs (max 168 items) were well under 16MB, and we inserted in moderate batches to avoid hitting message size limits. We also built appropriate **indexes** in MongoDB to support our queries (more on that later). For example, we indexed baskets on fields like household\_key, day, store\_id, week\_no, and a multikey index on items.product\_id, to speed up lookups and aggregations involving those fields.

After ETL, we verified that both databases had identical totals and record counts for consistency. A quick check summed total sales\_value from all transactions in each system, confirming the numbers matched to the penny (after cleaning, total sales across all baskets was ~$4.497 million in both). With the data prepared in both PostgreSQL and MongoDB, we proceeded to the analysis phase, running comparable queries in each environment to address the business questions.

## Analytical Approach and Key Insights

We conducted a series of analyses to answer the key questions, using SQL in PostgreSQL and aggregation pipelines in MongoDB to get identical results for comparison. In this section, we describe the findings of each analysis. (The next section will discuss how the implementations differed between SQL and MongoDB; here we focus on the insights themselves.)

### Customer Spending Trends Over Time

**Annual Spend Increase/Decrease:** First, we examined how each household’s total spending changed from Year 1 to Year 2. We aggregated each household’s transactions by year and then compared the two totals. This revealed a striking **split in customer behavior**: roughly half of the 2,500 households increased their total spend in Year 2, while the other half decreased their spend (and some nearly stopped shopping). In fact, our analysis found slightly more households decreased spend than increased, meaning a slight majority cut back in the second year. This bifurcation is important: it suggests we have one segment of customers becoming more engaged (buying more from the retailer) and another segment disengaging or buying less. In business terms, this is the classic scenario of growing loyalists versus at-risk customers. Figure 2 illustrates this split (e.g., a bar chart of number of households with spend up vs. down), which we found to be almost an even split. The data behind this showed, for example, that many of those who increased spending had relatively low Year 1 spend (leaving room for growth), whereas a number of those who decreased were previously high spenders who pulled back.

**Seasonal and Weekly Patterns:** We also looked at total sales aggregated by week to see seasonal trends. Plotting the total revenue per week (across all households) showed a fairly steady pattern with some periodic spikes. There was a slight uptick around weeks corresponding to holiday seasons (for instance, an increase in weeks 51-52 and again around week 102, which would correspond to year-end holidays in each year). Aside from seasonal peaks, weekly sales were relatively consistent, indicating no major seasonal slump beyond expected dips after holidays. The absence of dramatic seasonality is perhaps because grocery shopping, especially for staples, remains steady week to week, with only moderate increases for holidays. We also computed the distribution of basket sizes (in dollars and number of items). The **average basket (shopping trip) was about $32 with 10 items**, but with high variability. There were many small “fill-in” trips (min was essentially $0 purchases with just 1 item) and a few very large stock-up trips (max basket around $960 with 168 items – likely a big stock-up or combined transaction). This insight into basket distribution is useful for operations: e.g., staffing larger baskets might occur on weekends or paydays. (We indeed observed that the largest baskets often occurred at regular intervals, possibly indicating monthly stock-up routines). Additionally, we looked at **shopping frequency** by measuring the gap in days between each household’s shopping trips. The median inter-purchase gap was about a week, though it varied – some households shopped multiple times a week, while others had bi-weekly or irregular patterns. A cumulative distribution of inter-purchase days showed, for example, that about 50% of next trips occurred within 7 days and ~90% within 30 days, confirming that a majority of customers are at least making monthly visits, if not weekly.

**Interpretation:** The year-over-year trend highlights an area for concern: if half the customers are spending less, retention and re-engagement strategies are needed. However, the other half growing is a positive sign – possibly driven by successful onboarding or loyalty efforts. The weekly trends being relatively flat suggests the retailer does not have extreme seasonality to worry about (unlike, say, toy retailers or fashion), but can still capitalize on holiday spikes. The basket size distribution tells us that while typical trips are modest, there’s a significant tail of large baskets contributing to revenue – these might correspond to large families or special occasion stock-ups. Maintaining those large-basket shoppers (often the high-value customers) is crucial.

### Category Growth and Decline

We next delved into *what* categories customers were spending more or less on, especially among those who increased vs. decreased their total spend. Using the product hierarchy (department/commodity), we summed Year 1 vs Year 2 spend by category for each segment of households.

**Growth Categories (among Increasing-Spend Households):** Customers who spent more in Year 2 did so by **buying more of their everyday essentials and broadening into new categories**. The data showed significant sales growth in staple departments like **Grocery (dry goods/pantry)**, **Produce**, and **Dairy** for this group. For example, many increasing households roughly doubled their spending on Produce year-over-year. One specific insight: “Fluid White Milk” (a top-selling subcategory in Dairy) was highlighted as a consistently purchased item among large families and saw increased volume in Year 2. This confirms that as households become more engaged, they upsize their spend on basics like milk, bread, eggs, produce – likely shifting more of their regular grocery needs to this retailer.

Additionally, these households expanded spending in **non-food categories**. Departments such as **General Merchandise/Drug** (which includes toiletries, over-the-counter medicine, cosmetics) and **Fuel** showed faster growth rates for the increasing-spend segment. This suggests that as customers became more loyal, they started to take advantage of the retailer’s one-stop shop offerings – perhaps filling prescriptions, buying health and beauty items, or using the gas station associated with the store. The **Miscellaneous** category (which can include seasonal goods, small appliances, etc.) also saw upticks, indicating broader engagement. In summary, **increasing spenders deepened their wallet share with the store**, buying more groceries and also adding new categories (turning the store into more of a full-service solution for them). This pattern is encouraging for the retailer: it implies successful cross-selling and increased loyalty, as these customers rely on the store for more of their needs.

**Declining Categories (among Decreasing-Spend Households):** Households who spent less in Year 2 tended to **pull back across the board, but especially on non-essentials**. The largest declines were seen in categories that could be considered discretionary or occasional purchases: for instance, **snack foods, sweets, premium convenience items** and many **General Merchandise** items had steep drop-offs in this group. It appears that when tightening their budgets or disengaging, customers cut the “treats” and extras first – they might skip the snack aisle, forgo that extra bag of chips or cookies, and avoid impulse buys of general merchandise. These households generally *continued buying staple foods* (everyone still needs basics like milk, bread, rice), but often in smaller quantities or less frequently. For example, a customer might still purchase milk and bread weekly, but stop buying deli-prepared foods or reduce purchases of beverages and candies.

Notably, those who **churned** (households who virtually stopped shopping in Year 2) obviously dropped every category to zero. But even excluding full churners, spend-decreasing households showed the sharpest percentage declines in categories outside the core grocery basket. Departments like **Miscellaneous** and **Drug/GM**, which had grown in the loyal segment, saw disproportionate declines here. This likely means these customers either went elsewhere for those needs or just didn’t buy them. It underscores a typical behavior: when a customer starts to drift, they focus on just essentials and might shop those wherever is cheapest or most convenient, potentially splitting spend with competitors.

**Key Takeaways:** For growing customers, the retailer should continue reinforcing staples (perhaps loyalty programs on frequently bought goods) and also recognize the opportunity to sell across departments (maybe promote the pharmacy or fuel discounts to keep that momentum). For declining customers, an intervention might be needed in the categories they’re cutting – e.g., targeted promotions on snacks or personal care could entice them back for those items. Since many in the declining segment are likely price-sensitive or have reduced needs, emphasizing value on essentials or bundling offers (like a discount on a non-food item with purchase of staples) might slow their disengagement. The analysis of category trajectories thus informs category managers where to focus retention efforts vs. growth investments.

### Demographic Influences on Spending

Using the demographic data available for 801 households, we explored how factors like family size, marital status, income, home ownership, and age correlate with spending behavior and category preferences. While the data is anonymized and categorical, clear patterns emerged: **family composition is the dominant factor in grocery spending**.

**Household Size & Children:** As expected, larger households spend more. Our data shows a near-linear relationship: for each additional person (or child) in the home, annual spend jumps significantly. Families with 3+ children were among the highest spenders overall. Interestingly, when we segmented by income as well, we found that **family size often trumped income level** in determining total spend. For example, a middle-income family of five typically spent more per year than a high-income couple with no kids. This makes intuitive sense – more mouths to feed drive up grocery bills more than upscale preferences do. In terms of category engagement: households with children bought more **milk, cereal, snacks, and baby products** (as one would imagine). Large families also tended to buy in larger pack sizes and likely more frequently. This insight is critical for targeting: families with kids are a key segment for promotions on bulk goods, family packs, and staples.

**Marital Status:** We noticed an interesting pattern where **unmarried households (especially couples without kids, i.e. dual-income no-kids – “DINKs”) actually spent more on average than married households with children** in certain categories. This at first seems counterintuitive since families have bigger needs; however, when adjusting for household size, it appears single adults and DINKs often have **higher per-capita or discretionary spend**. The theory is that without children, they have more disposable income for themselves – they might splurge on premium groceries, gourmet items, nice alcohol, or health and beauty products. In contrast, a married-with-kids household has to budget for many necessities and may be more price-conscious, sticking to value brands and skipping luxury items. Our data hinted at this: single/couple households spent relatively more in categories like premium organic foods, wine/beer, and personal care, whereas big families spent more in absolute dollars but mostly on child-focused or value categories. This suggests segmentation opportunities – e.g., promote high-end or novelty products to single professionals, and budget-friendly bulk deals to large families.

**Income Level:** Income had an effect but perhaps less stark than expected. Higher income groups did have higher total spend on average (no surprise) and were more likely to purchase **premium brands or organic options** (trading up on quality). For instance, the top income bracket had a noticeable skew towards organic produce and high-end health items. However, the very highest income group was not the largest contributor to sales in aggregate – the *middle-income with large families* segment was. This underscores that a retailer’s core revenue can come from middle-class families buying lots of goods, even if each item is budget-friendly. Lower-income households in our data spent less overall, and their buying patterns showed more price sensitivity: they would respond to promotions, buy store brands, and likely redeem more coupons (if they had them) as compared to wealthier households. So while income is useful for predicting preference (premium vs value products), it interplays with household size in determining volume.

**Homeownership & Age:** These factors were included as well (homeowner vs renter, and an age code). We found subtle trends: homeowners might make slightly larger purchases of bulk goods (perhaps because they have storage, e.g. a pantry or freezer) – for example, homeowner households were a bit more likely to buy in bulk sizes in categories like paper goods or canned goods (though this is a soft correlation). Age-wise, younger singles/couples spent more on convenience foods and beverages (and alcohol), while older (senior) households had lower grocery spend overall (smaller households) and possibly more spend on pharmacy items (if available in store). Seniors and small households tend to buy smaller pack sizes and fewer impulse items. The demographic data was limited (age was given in broad groups), so we focused mostly on the clearer signals: family size, presence of kids, and income.

**Implications:** Demographic insights help tailor marketing. For example, knowing that families with kids drive milk and cereal sales means those are great products for loyalty discounts or exclusive promotions to retain that segment. Childless high-income shoppers might be swayed by artisanal or ready-to-eat offerings – a different approach. The fact that large families are the top spenders means retention efforts (like ensuring they get the best coupons or perks) could yield big returns, whereas lower-spending singles might be targeted with smaller-basket encouragements (like meal kits or recipe-themed marketing) to increase their basket size. Our analysis confirms that **“life stage” segmentation (e.g. young single, young family, empty nester) is highly relevant in grocery retail**, aligning with industry intuition but now backed by data.

### Impact of Marketing Campaigns and Promotions

Finally, we assessed whether the retailer’s direct marketing campaigns and in-store promotions were effective in driving sales and engagement. The dataset’s campaign and coupon tables allowed us to see which households got coupon offers and which redeemed them, while the causal data let us check if being on display or in a flyer affected product sales. We approached this both as a **quasi-experimental analysis** (comparing target vs. control groups for campaigns) and by looking at correlations between promotions and sales.

**Coupon Redemption and Customer Engagement:** We segmented households by whether they redeemed at least one coupon from the campaigns. About 434 households (out of 801 with campaigns info) redeemed coupons. Comparing their spend trajectories to those who never redeemed coupons, a pattern emerged: **households who used coupons were more likely to maintain or increase their spending** over the two years. In contrast, those who got campaigns but didn’t redeem any offer were more often in the declining spend group. This suggests that the act of engaging with a coupon is a proxy for engagement with the store. It could be that coupons enticed them to try new products or make extra trips, thereby boosting overall sales. For example, one case was household **208** (an example noted in the user guide and observed in our analysis) – this household received 7 campaigns and redeemed 5 coupons (from campaigns 8 and 18). They subsequently showed increased spending, especially in the categories of those coupons. The redeemed coupons included items the household had not bought before, indicating the campaign prompted new purchases.

We performed an analysis comparing **“target vs. control”** for a specific campaign: taking campaign 22 as an example (which ran in the latter part of Year 2), we compared households who were targeted vs. similar households who were not, in terms of their Year 2 spending growth during that period. In PostgreSQL, this meant joining the campaign\_table with transactions and aggregating basket totals for targeted households versus non-targeted (control) households. We found that targeted households on average had slightly higher basket sizes during the campaign window than the control group. When visualized (e.g. a box plot of basket spend for each group), the median basket sales for the targeted group was higher, and there were more high-value baskets – suggesting a **lift** due to the campaign. However, variance was high and not every campaign had a clear uplift, implying that some campaigns were more successful than others. Overall, though, aggregating across all campaigns, we conclude that **direct marketing provided a modest but noticeable uptick in retention and spend** for those who engaged. This aligns with the business expectation that coupon mailers can drive incremental trips or larger baskets (even if just by reminding customers to shop or giving them a deal on something extra).

**In-Store Display and Mailer Impact:** We also turned to the causal\_data to see if a product being on display or in the weekly ad flyer corresponded with increased sales for that product. Using SQL, we joined transaction line-items with the causal\_data by product, store, and week, and compared average sales of items when they were on display vs. not. The results clearly showed that items had **higher sales when on promotion**. For example, when an item had a display code (meaning it was on some special display in the store that week), its average quantity sold and revenue were significantly higher than in weeks it wasn’t displayed. A simple comparison: products not on display (display code 0) might have an average weekly sales value X, whereas on display (e.g. code 4 for an endcap display), the average sales were substantially higher (often 1.5x to 3x, depending on the product and promotion) – a clear promotional lift. Similarly, features in the mailer (mailer code not “0”) tended to boost sales, though the effect varied by category. Some codes (like “A” presumably front-page features) had big impacts. We did a more advanced MongoDB pipeline where we pre-aggregated item-week-store sales and then looked up the promo flags, to avoid an explosion of data. That confirmed the SQL finding: promotion weeks see a jump in sales. For a concrete example, if a cereal was featured on display in week 30, its units sold that week were higher than its typical weekly sales when not featured.

**Takeaway:** Both targeted direct marketing and in-store promotions have positive effects on sales, though in different ways. **Coupons and mailer campaigns** help keep customers engaged over the long run and can introduce them to new products (which may lead to repeat purchases later). **In-store displays and features** drive immediate short-term spikes for the promoted products, catering to impulse buys or drawing attention to discounts. An integrated strategy is important: for instance, a campaign coupon could coincide with an in-store display of that item for maximum effect – the data suggests such synergy would likely yield a strong lift (though our dataset didn’t explicitly tie specific coupon items to displays, one could infer it). Importantly, the analysis quantitatively backs up the investment in these marketing efforts: we saw measurable lifts that justify campaigns as a tool to prevent churn and promotions as a tool to boost category sales.

## PostgreSQL vs. MongoDB – Analytical Workflows Compared

Throughout the project, we wrote equivalent queries in SQL and MongoDB’s aggregation framework to produce the same results. This exercise illuminated the **differences in how each database handles analytical workloads** on the same data. In this section, we discuss those differences, along with performance observations, to guide when to use each system for similar tasks.

**Schema Design and Query Complexity:** In PostgreSQL, we worked with a highly normalized schema with many tables, **requiring joins to gather information**. This design, backed by foreign keys and indexes, ensures data integrity and makes each query result exact and consistent. However, **analytical queries often had to join multiple large tables (e.g. transactions + products + promotions) which increased query complexity.** For instance, to compute weekly sales by department in SQL, we joined the basket\_item (transactions) with product and then grouped by week and department. **In MongoDB, because we had denormalized some data (like embedding product category info inside each items entry), we could sometimes avoid an explicit join** – but if needed, we used $lookup. The **MongoDB queries were generally more verbose** than SQL equivalents. **A SQL JOIN with a WHERE clause can be easier to read than a multi-stage Mongo aggregation** with $unwind and $group, especially for newcomers. That said, **when data was embedded, MongoDB could do in one stage what SQL did in two or three.** For example, summing sales per basket was a single aggregation on the baskets collection using $sum: "$items.sales\_value" to get basket\_total, whereas in SQL we grouped by basket\_id over millions of rows. **Result:** For queries confined within a natural document boundary (e.g. computing a property of a basket or household), MongoDB’s approach felt very efficient. For queries that needed to combine disparate entities (like linking transactions to an external lookup not embedded), SQL’s set-based joins were straightforward while Mongo required careful pipeline planning.

**One-to-Many Aggregations:** A common pattern was one-to-many relationships (one basket has many items, one household has many baskets, one product has many transactions, etc.). In PostgreSQL, one-to-many is handled via joins and grouping. MongoDB handles one-to-many nicely when the “many” are stored as an array in the “one” document. We saw this with basket analysis: calculating metrics per basket or per household was typically **faster and simpler in MongoDB** because the data was pre-joined. For example, to get each basket’s total and item count: SQL had to scan the basket\_item table and group by basket\_id (in memory or using disk if large), whereas MongoDB just read each basket document and summed the embedded array – effectively already grouped by basket by design. In our timing experiments, the Mongo query for basket totals was notably quick (single collection scan) compared to SQL which had to scan and group a larger number of rows. Similarly, **to get total spend per household, in Mongo we could sum over each household’s baskets if we had embedded baskets in household** (we didn’t embed baskets in household though – we did a $lookup between baskets and households when needed). If we had embedded, it would be one document read per household. Without embedding, the $lookup on household\_key and grouping was still reasonably efficient given proper indexes.

**Many-to-Many and High Cardinality Joins:** Some analyses involved many-to-many relationships with lots of records – for example, joining transaction items with the huge causal promotions table (product-store-week) to see display effects. **PostgreSQL excelled at this kind of heavy lifting:** we created a composite index on (product\_id, store\_id, week\_no) in the causal table and the planner could handle the join across millions of rows. MongoDB, in contrast, struggled with a naive approach: if we unwound every basket’s items (creating millions of intermediate docs) and then did a $lookup to causal\_data, performance was poor. We had to implement a more clever pipeline: grouping items first by product-store-week to reduce duplicates, then looking up promotions once per unique key, then computing averages. This made the pipeline more complex, but once optimized with compound indexes in Mongo (and by avoiding unwinding full detail), we got acceptable performance. The key lesson is **MongoDB can do many-to-many, but it often requires pre-aggregation or denormalization to keep the intermediate result size manageable**. SQL, with its query optimizer and ability to use join algorithms (hash join, merge join, etc.) and statistics, handled these large-scale joins more gracefully without manual intervention.

**Use of Indexes:** Both systems required thoughtful indexing. PostgreSQL automatically indexes primary keys and foreign keys (if we choose), and we added indexes on common join columns (e.g., on basket\_item(product\_id) for joining with product, etc.). In some cases we used partial indexes or multi-column indexes (like on causal\_data as mentioned). **In MongoDB, we explicitly created indexes on fields used in $lookup or $match stages to avoid collection scans.** For example, to join baskets to coupon\_redemptions by household\_key and day, we indexed both collections on those two fields. After indexing, that join (lookup) in Mongo was efficient. **Without indexes, MongoDB would have been untenably slow for those operations.** The takeaway is in analytic workloads, **index strategy often matters more than the choice of SQL vs NoSQL** – **a well-indexed Mongo query can beat a poorly indexed SQL query and vice versa.** In our tests, once indexes were in place, most queries in both systems ran in seconds. Mongo queries benefitted immensely from compound indexes on the join keys and the use of its multikey indexes for array fields.

**Data Integrity and Cleaning Overhead:** One clear difference was how each system deals with data integrity. PostgreSQL, with its constraints, forced us to clean data upfront (dropping those household-less records) – which we did, as described. This meant that once data was in Postgres, we could trust relationships (no missing foreign keys). MongoDB had no such enforcement; indeed we had to implement those filters and de-duplications ourselves before inserting. If we had not, we could have ended up with documents referencing nonexistent households or duplicate coupon entries. The **ingestion code in MongoDB took on the work of the database schema** in a sense. This makes development a bit more complex and requires discipline in testing data quality. On the other hand, Mongo’s flexibility meant we could restructure data as needed without formal migrations – e.g., embedding new fields or changing document structure on the fly was easy, whereas in PostgreSQL adding a new column or table would require an ALTER TABLE and careful migration if done later. In this project, schema changes were planned from the start, so SQL’s rigidity was not a big issue. But in a scenario of evolving requirements, that could favor MongoDB.

**Query Readability and Development Productivity:** We experienced that **writing complex analytical queries in SQL was generally more straightforward than in MongoDB’s aggregation framework** for those used to SQL. SQL’s declarative style lets you express what you want (join these tables, group by that) fairly succinctly. The MongoDB aggregation pipeline is powerful (you can do almost anything, including things not as easy in SQL, like hierarchical transformations), but it’s more procedural in feel and can become verbose. For example, our campaign impact analysis in SQL used a WITH clause and a couple of joins with CASE expressions to flag target vs control, which was easy to follow. **In Mongo, it required multiple stages ($lookup, $group, $project with $filter) to replicate.** Debugging or validating intermediate results was also easier in SQL (thanks to being able to run subqueries or CTEs in isolation) versus in Mongo where we sometimes had to insert inspection stages or check documents manually. That said, those more familiar with JSON might feel more at home with Mongo’s syntax. Our takeaway: **for pure analytical querying, SQL has an edge in simplicity**. However, if the analysis needs to be embedded in an application or microservice, the document model might be more directly usable (no OR/M or complex query building, just a pipeline in code). In a collaborative environment, we found it easier to communicate the logic of a SQL query to others than the equivalent Mongo pipeline.

**Performance Summary:** In terms of raw speed, after optimization both databases performed well for our analysis. **Simpler aggregate queries (like computing distributions within a single collection/document) ran faster on MongoDB due to fewer joins.** **More complex multi-factor queries ran faster or more naturally on PostgreSQL.** Neither system “timed out” on us after indexing, but the initial attempt in MongoDB for the promotion join was very slow until we refactored it. This highlights that **MongoDB may require more data-specific optimizations for complex analytics**, whereas PostgreSQL can often brute-force via its engine. For example, computing an average sales lift for display vs non-display took a few seconds in Postgres with a proper index; doing the same in Mongo needed careful pipeline but then was also a few seconds. Memory-wise, both handled the data (PostgreSQL was running on a local instance with enough memory for aggregations; MongoDB’s aggregation pipeline also fit in memory after using indexes).

In summary, our experience reflects the general guidance: use **MongoDB for analytics that can exploit its document model (one-to-many aggregates, quick rollups within a doc)**, and use **PostgreSQL for heavy relational analytics involving lots of cross-entity relationships or where set-based operations and integrity are paramount**. Often, a combination is best – for instance, one might keep a normalized PostgreSQL warehouse but periodically dump or sync aggregated views (like the baskets or household rollups) into MongoDB for fast querying in applications.

## Conclusion and Recommendations

This project demonstrated that **both PostgreSQL and MongoDB can successfully handle retail customer analytics**, but each shines in different aspects. By leveraging the two in tandem, a business can enjoy the reliability of relational data and the agility of document-based queries. Based on our analysis, here are our recommendations for how to best utilize each system for retail analytics tasks:

* **Use PostgreSQL for complex, multi-table analytical queries** – especially those involving **many-to-many relationships or large detailed data**. PostgreSQL’s SQL engine excels at joining facts like transactions with reference data (products, campaigns) and large promotional tables efficiently. Tasks like full category trend analysis, correlating sales with promotions across hundreds of products, or performing window functions (e.g. running totals, rankings) are more straightforward in SQL. The query planner and indexes handle these scenarios well, and the data integrity ensures clean joins. In our case, evaluating **promotion lift** and comparing **target vs. control groups** for campaigns were easier and safer to do in PostgreSQL, with clear, auditable SQL queries yielding accurate results.
* **Use MongoDB for single-entity or nested aggregations (one-to-many rollups)** that can leverage **pre-joined documents**. Calculations constrained within a natural data boundary (e.g., per basket, per household, per campaign) run faster with Mongo’s document model. For instance, computing **basket metrics (total value, number of items)** or **household summary statistics** can be done with one aggregation pipeline reading each document, avoiding expensive join operations. We observed MongoDB to be very efficient for these kinds of aggregations – it “shards” the problem by each document. So for **real-time dashboards** or microservices that need quick stats (like “show this customer’s last 6 months spend grouped by month” or “get the distribution of basket sizes in the last hour”), MongoDB is a great choice. It provides speed via its denormalized design and can simplify application logic by storing data in the shape that the application queries it.
* **Prefer PostgreSQL when data consistency and governance are crucial.** For any **system-of-record** use – e.g., maintaining the master list of products, customers, and transactions – a relational database is recommended. PostgreSQL’s enforcement of foreign keys and unique constraints prevented bad data in our project (we had to clean data before insert, catching issues early). In a production setting, this means fewer downstream errors and the confidence that analyses reflect reality (no missing references, no duplicate entries). If the analysis requires precision and the ability to easily **validate or debug results**, SQL’s transparency is valuable – one can write intermediate queries or examine relational schemas more easily than unwinding nested documents. Thus for **regulatory reporting, financial roll-ups, or any mission-critical analytics**, PostgreSQL is the safer bet.
* **Leverage MongoDB for flexibility and faster development in certain analytics pipelines.** If the data model is expected to evolve (say new attributes are added frequently) or if you need to ingest semi-structured data (like varying campaign content), MongoDB can adapt without strict migrations. In exploratory analytics, one might create a new derived field and store it in documents as needed. Also, when powering an application that requires aggregated data, storing that data already aggregated in Mongo (like our coupon documents containing all eligible products) can cut down on query logic. So for use-cases like **personalized recommendations**, where you might store a precomputed list of recommended products per user as a document, Mongo fits well. In our project, while we didn’t face schema changes, we note that if tomorrow marketing wanted to tag campaigns with a new property, adding it in Mongo would be trivial, whereas in Postgres it requires an alter table and backfill.
* **Combine both in an analytics ecosystem:** One practical strategy is to **build your core warehouse in PostgreSQL** for correctness and comprehensive analysis, and then **publish key result sets or aggregates to MongoDB** for quick serving. For example, keep all raw transactions in SQL, but have a MongoDB collection for “customer profiles” that embed recent purchases, top categories, and response scores. This hybrid approach gives analysts the power of SQL for deep dives, and applications the speed of MongoDB for quick lookups. Our findings suggest that basket-level analysis is a prime candidate to offload to Mongo (since each basket is independent), whereas cross-basket or cross-customer comparisons benefit from SQL.

In conclusion, **PostgreSQL and MongoDB are not mutually exclusive – they are complementary**. PostgreSQL offered reliability, easy QA of queries, and strong performance for relational joins in our customer analytics, making it ideal for **enterprise-wide BI and historical trend analysis**. MongoDB provided agility and speed for aggregated views, making it suitable for **real-time analytics and microservice-driven data applications**. By using each for what it does best, a retailer can ensure both accuracy in their analytical insights and responsiveness in their customer-facing applications.

**Executive Recommendations (Summary):**

* **Customer Spending & Trends:** Use PostgreSQL to compute long-term spending trends and complex cohort analyses (e.g., year-over-year comparisons, retention rates), where SQL’s ease with joins and filtering shines. Use MongoDB to quickly retrieve or display individual customer’s spending history or basket distributions (one document per customer or basket allows instant aggregation) for operational dashboards.
* **Category and Basket Analysis:** Use PostgreSQL for category growth analysis across the whole population (joining transactions with product hierarchies). Use MongoDB for on-the-fly basket analytics and basket content insights (since each basket is a self-contained document, queries like “largest baskets this week” or “basket composition” are extremely fast with Mongo’s aggregations on arrays).
* **Demographics & Segmentation:** Use PostgreSQL to join demographic data with purchasing patterns for robust statistical analysis (e.g., spending by income group, requiring multiple table joins and perhaps window functions). PostgreSQL ensures all households considered have valid demographic links. MongoDB can be used to store precomputed segmentation results or to quickly fetch a particular segment’s profile (embedding segment labels or scores within customer documents for fast querying).
* **Marketing Campaign Impact:** Use PostgreSQL for rigorous A/B evaluations of campaigns (joining target vs control households, computing lift with precise filtering by campaign dates) – the clarity of SQL helps ensure methodological correctness. Use MongoDB to monitor ongoing campaign engagement in real-time (e.g., as coupon redemptions stream in, a Mongo collection of campaign docs with embedded stats can be updated and queried quickly to see current uptake).
* **Performance and Scaling:** Ensure heavy analytical queries in MongoDB are optimized via proper indexes and, when necessary, pre-aggregations to avoid large unwinds. For PostgreSQL, leverage indexes and consider summary tables or materialized views for frequently run analytics to speed them up. In operational use, a **tiered architecture** could be ideal: PostgreSQL as the source of truth and deep analysis repository, and MongoDB as a high-performance serving layer for specific aggregated results and application-facing queries.

By aligning each workload with the most suitable data store, the retailer can maximize insight generation while maintaining performance. In essence: **use PostgreSQL for depth and accuracy, and MongoDB for speed and flexibility**, to fully unlock the value in retail customer data.

**References – Used chatGPT for compiling my reports.**